1. Общий обзор архитектуры UNIX систем. 5

Краткий обзор UNIX подобных операционных систем. 5

Основные причины популярности UNIX. 5

Структура операционной системы. 5

Задачи выполняемые ядром операционной системы 6

Подсистема управления процессами. 7

Подсистема управления памятью. 7

Файловая подсистема. 8

Подсистема ввода-вывода. 8

2. Шеллы и основные команды HP-UX. 9

Общее знакомство с шелами 9

Bourne Shell. 10

C Shell . 42

Korn Shell и POSIX shell 57

Key Shell (keysh). 65

3. Администрирование системы X Window 69

Базовые концепции X Window 69

Компоненты системы X Window 69

Предварительное конфигурирование 70

Имена и классы клиентов 73

Имена и классы ресурсов 74

Типы ресурсов 74

Управление шрифтами 75

2. Программирование на HP-UX 82

Этап линкирования. 85

Работа с библиотеками 85

Создание архивной библиотеки. 89

Открытие Библиотек Распределеного доступа 91

Создание Общедоступной Библиотеки с ld 92

Модифицирование Общедоступной Библиотеки 93

Применение make 94

Отладчик ADB 103

Отладчик XDB 109

3. Примеры команд 110

VARIABLES 117

LOCATIONS 117

FORMATS 118

4. Системные вызовы и взаимодействие с UNIX. 119

Время в UNIX. 124

Сигналы. 131

Деления просесса 135

Пайпы и FIFO-файлы. 135

Нелокальный переход. 138

Разделяемая память 139

Семафоры 140

Очереди сообщений 142

5. Старт системы. 146

6. Run-levels. 150

Остановка системы 155

Конфигурирование ядра системы 158

7. Изменение системных параметров 161

Инсталирование периферии 164

Системная конфигурация 165

Управление процессами 173

Управление процессами и ядро системы 182

8. 1. Файловая система HFS. 185

Структура файловой системы HFS 186

Главный суперблок 186

Группы цилиндров 187

Размер 187

Блоки данных 189

Доступ к блокам данных 190

Модификация файлов в HP-UX 193

Менджер логических дисков LVM 194

Создание корневой VG и корневого LV 201

Резервное копирование и свосстановление конфигурации Volume Groups 202

Перемещение и переконфигурирование дисков 203

3. Особенности файловой системы VxFS 206

Монтирование и демонтирование файловых систем 208

Проверка файловых систем 209

10. Мониторинг использования дискового пространства 212

11. . Организация веб-сайта 214

Выбор операционной системы. 214

Выбор программного обеспечения сервера. 215

Анализ веб-серверов. 216

Инсталляция веб-сервера. 217

Стратегическое планирование. Определение объема работы. 221

Тактическое планирование сайта. Разработка структуры. 226

Дизайн интерфейса. 228

Программирование. 230

Публикация и Маркетинг. 233

12. Веб-страницы и веб-приложения. 235

Классификация веб-объектов. 235

Спецификация DTD. Понятие ортогональности и методы ее реализации. 236

Веб-страницы. Языки разметки. (HTML, XML) 237

Веб-страницы. Программирование (JavaScript, CSS, SSI, CGI, PHP) 261

Модульность и ортогональность с использованием существующих технологий. 288

Веб-приложения. 289

Общие требования к страницам сайта. 290

Совместимость с различными браузерами. 292

13. Конфигурация и управление веб-серверами. 295

Встроенные средства управления сервером. (apachectl, apxs) 295

Глобальные разделы конфигурации. 296

Вспомогательные скрипты – просмотр и ротация логов, статистика посещений. 299

Безопасность веб-сервера. 299

Организация доступа и разграничение прав пользователей. 303

. Подключение новых модулей и апгрейд программного обеспечения веб-сайта. 303

14. Администрирование веб-сервера. 304

Логгирование и поиск ошибок. 305

Обеспечение безопасности . 311

Создание резервных копий. Технологии: Backup, mirroring. 313

15. Система безопасности HP-UX 314

16. Политика и планирование системы безопасности 314

Установка Trusted Системы 315

Управление паролями и системным доступом 316

17. Управлением доступом к файлам и каталогам 317

Контроль безопасности сети (networks) 319

# Общий обзор архитектуры UNIX систем.

## Краткий обзор UNIX подобных операционных систем.

В даный момент, не существует стандартной системы UNIX, вместо этого вы столкнетесь со множеством операционных систем, имеющих свои названия и особенности. Но за этими особенностями и названиями прослеживается общая архитектура, интерфейс и среда программирования. Все эти системы так или иначе являются родственными Из-за своей простоты, ясности, легкой способности к расширению и модификации UNIX стали переносить на множество платформ. Однако несмотря на множество реализаций базовой системы, среди всех них можно четко выделить две основные ветки: System V UNIX и BSD UNIX. Различия между ними не носят принципияльный характер и зачастую сказать к какой из веток принадлежит та или иная реализация операционной системы бывает сложно. К основным различиям между System V и BSD подобными системами можно отнести терминальную инициализацию, имена конфигурационных файлов и файлов инициализации системы, стандартный размер блоков файловой системы, управление терминалами, различное отображение информации о процессах и.т.п. Одним словом принципиальных различий с точки зрения пользователя между разными ветками операционной системы UNIX не существует. Рассматриваемая нами операционная система HP-UX 10.20 является одной из реализаций UNIX выпущенной фирмой Hewlett-Packard. По своей структуре это чистая System V подобная 32-х разрядная операционная система, включающая поддержку симметричных многопроцессорных систем (SMP), файловых систем большего объема (до 128Гб) и расширенного виртуального адресного пространства (до 3.75 Гб).

## Основные причины популярности UNIX.

Каковы же причины популярности этой операционной системы ? В первую очередь это более чем трех десятилетний возраст. За этот период она полностью прошла проверку временем. Во вторых код системы практически полностью написан на языке высокого уровня С, что сделало ее простой для понимания, внесения изменений и переноса на другие аппаратные платформы. Некоторые из версий UNIX поставляются вместе с исходными текстами, однако даже несмотря на то что большинство UNIX поставляется в виде бинарных файлов, система все равно остается легко расширяемой и настраиваемой. Так же следует отметить тот факт что UNIX в изначально создавалась как мно­го­поль­зо­ва­тель­ская и многозадачная система ориентированная в первую очередь на выполнение серверных функций. Следует отметить и тот факт что UNIX практически изначально создавалась как сетевая операционная система (даже графическая оболочка UNIX система X Window является полностью сетевой), что позволило ей занять лидирующие позиции на рынке серверов для Интернет приложений и дало мощные встроенные средства удаленного администрирования. Не маловажную роль в популярности UNIX сыграла ее единая иерархическая файловая система с унифицированным доступом не только к файлам данных но и к аппаратным ресурсам таким как диски, терминалы, принтеры, сеть, память и.т.п.

## Структура операционной системы.

В задачу операционной системы UNIX входит непосредственное управление ресурсами компьютера, распределение их между пользователями, скрывая от последних внутреннюю архитектуру аппаратного обеспечения, путем предоставления унифицированного интерфейса доступа к аппаратным ресурсам. К аппаратным ресурсам компьютера относится в первую очередь вычислительные ресурсы процессора, память и дисковое пространство, а также ряд периферийных устройств, таких как накопители на магнитных лентах, принтеры, терминалы, сетевые адаптеры и.т.п. Самый общий взгляд позволяет увидеть двухуровневую модель системы в том виде как она представлена на рис. 1.1.

Пользовательские программы

Ядро системы

Аппаратные ресурсы

Системные вызовы

 Рис 1.1

В центре находятся аппаратные ресурсы компьютера с которыми непосредственно взаимодействует ядро операционной системы изолируя прикладные программы пользователя от особенностей аппаратной архитектуры. Ядро имеет определенный минимальный набор услуг представляемых прикладным программам. В первую очередь это операции ввода-вывода (открытие, закрытие, чтение, запись и управление файлами), создание и управление процессами, организация синхронизации и обмена данными между процессами, управление памятью (реальной и виртуальной).

Второй важнейшей функцией выполняемой ядром является защита операционной системы от разрушения со стороны пользовательских программ и реализация механизмов защиты данных в многопользовательской среде. Все пользовательские приложения пользуются услугами ядра посредством системных вызовов.

 На втором уровне находятся приложения, как пользовательские, обеспечивающие интерфейс с пользователем так и системные, управляющие работой системы. Несмотря на различные выполняемые задачи, схемы их взаимодействия с ядром одинаковы.

## Задачи выполняемые ядром операционной системы

Остановимся более подробно на структуре ядра операционной системы. Функционально его можно представить состоящим из трех основных подсистем: подсистемы управления процессами и памятью, подсистемы ввода-вывода и файловой подсистемы. Все современные микропроцессоры поддерживают виртуальную память, защищенный и многозадачный режим работы. Последний подразумевает выделение кванта процессорного времени определенной задачи с последующим переключением на другую задачу

Каждая задача имеет идентификатор уровня защиты, некоторые команды из системы команд процессора могут выполняться на любом уровне защиты, но есть привелигированные команды выполнение которых возможно лишь задачей имеющей нулевой уровень привилегии. Ядро операционной системы работает на нулевом уровне защиты, только оно имеет непосредственный доступ к физической памяти, системным регистрам процессора и портам ввода вывода. Пользовательские программы общаются с ядром посредством системных вызовов, представляющих собой команду приводящую к переключению процесса в контекст ядра, передачей параметров ядру. Затем ядро проверяет корректность параметров, права пользовательского процесса на возможность выполнения данного системного вызова и лишь после этого переходит к непосредственному выполнению всех низкоуровневых действий необходимых для исполнения пользовательского запроса. Благодаря этому достигается защита критически важных данных ядра от случайного или преднамеренного разрушения со стороны пользователя.

Второй из важнейших функций ядра является обработка исключительных ситуаций возникающих в результате работы операционной системы и представляющим собой програмные прерывания. К последним относятся ошибки защиты (на пример попытка прикладной программы получить доступ к портам ввода-вывода или чужой области памяти), ошибки в работе оборудования, а также системные события возникающие при нормальной работе операционной системы. Некоторые из исключительных ситуаций приводят к аварийному завершению системы (ошибки оборудования или исключительная ситуация возникшая во время обработки исключительной ситуации), некоторые к аварийному завершению пользовательской программы при этом возможен сброс на диск в файл образа процесса вызвавшего исключительную ситуацию c целью дальнейшего анализа отладчиком, а некоторые просто жизненно необходимы для нормального функционирования операционной системы (одна из таких исключительных ситуаций будет рассмотрена ниже).

## Подсистема управления процессами.

Запущенная на выполнение программа порождает в системе один или больше процессов. Подсистема управления процессами отвечает за создание и уничтожение процессов, распределения ресурсом между процессами, синхронизацю и межпроцессное взаимодействие. Так как в данный момент времени на одном процессоре может выполняться лишь одна задача ядро операционной системы в зависимости от приоритета задачи выделяет ей определенный квант процессорного времени по истечению которого происходит переключение на следующую задачу. При этом исключается возможность захвата всех ресурсов процессора одной задачей и создается эффект параллельного выполнения нескольких задач.

## Подсистема управления памятью.

Подсистема управления памятью управляет выделением, размещением и освобождением памяти для прикладных задач. Прикладные программы никогда не используют физическую память напрямую, т.к. все современные операционные системы реализуют так называемую виртуальную память объем которой может превышать объем физической памяти. При этом задействуется механизмы страничной адресации памяти в которых все виртуальное адресное пространство разделяется на небольшие блоки – страницы. Размер страницы варьируется в зависимости от архитектуры, для архитектуры HP PA-RISC это 2К, для Intel это 4K. Каждая страница имеет специальные атрибуты которые определяют права доступа к ней, факт присутствия в физической памяти, частоту обращений и.т.п. Преобразование из виртуального адреса в физический осуществляется аппаратно. Данная схема адресации дает ряд неоспоримых преимуществ которые используются всеми современными операционными системами. А именно: возможность экономии физической памяти путем совместного использования одних и тех же страниц виртуальной памяти разными процессами, реализация разделяемой памяти, а также возможность использования вторичных устройств в качестве устройств памяти что позволяет задачам задействовать виртуальной памяти больше чем есть реально физической памяти в системе. Реализуется это следующим образом: в случае нехватки физической памяти, ядро сбрасывает на внешний носитель (как правило диск) страницы к которым наиболее долго не было обращений, а так же при обращении к странице которой реально нет в физической памяти процессор генерирует исключительную ситуацию обработчик которой загружает страницу обратно с внешнего носителя в физическую память. Этот процесс носит название пейджинг или свопинг.

## Файловая подсистема.

Файловая подсистема ядра предоставляет унифицированный интерфейс для доступа к данным находящимися на дисках и других внешних устройствах. Она обеспечивает контроль прав доступа к файлам со стороны прользователя т.к. каждый файл имеет атрибуты доступа определяющие права доступа к нему со стороны определенного прользователя или групп пользователей.

## Подсистема ввода-вывода.

Подсистема ввода-вывода обслуживает запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам (сетевые адаптеры, диски, терминалы …). Также она обеспечивает буферизацию и кеширование данных и взаимодействует с драйверами устройств – специальными модулями ядра непосредственно работающими с внешними устройствами.

#  Шеллы и основные команды HP-UX.

 **2.1.1 Общее знакомство с шелами**

 **2.1.2 Смена шела**

 **2.2.1 Bourne Shell**

 **2.2.1.1 Основные возможности Shell.**

* + - 1. **Shell скрипты**
			2. **Основы программирования на языке shell**
			3. **Некоторые специальные команды**

## Общее знакомство с шелами

Шелл это интерфейс между операционной системой и пользователем. Шелл интерпретирует пользовательский ввод и дает указания операционной системе выполнить те или иные действия. Шелл можно также рассматривать как язык программирования.

 **Bourne Shell.** Это самый старейший из шелов который был написан Стэфаном Борном в Лаборатории Беэлла. Этот шелл является шелом по умолчанию для HP-UX пользователей и долгое время был стандартом де факто.

Bourne Shell не имеет в своем арсенале ни интерактивных возможностей ни сложных программных конструкций в отличии от С и Korn шеллов.

 **C Shell.** Этот шел был разработан Биллом Джоем в Калифорнийском Университете Беркли. Его синтаксис имеет сходство с языком программирования С. Он также имеет интерактивный интерфейс например историю команд и раскрытие имен файлов.

 **Korn Shell.** Он является относительно новым шелом разработанным Девидом Корном в Лаборатории Бэлла и является вверх совместимым с большинством возможностей Bourne Shell. Так же как и С shell он имеет интерактивные возможности, но выполняется быстрее имеет расширенные возможности редактирования командной строки.

 **POSIX shell.** Этот шелл базируется на стандарте определенном в Portable Operation System Interface (POSIX) – IEEE P1003.2. Этот стандарт был разработан для прикладных и системных программистов. Он фактически определяет стандарт на интерфейс операционной системы. Большинство возможностей POSIX Shell очень сильно схожи с аналогичными возможностями Korn Shell-a. Мы будем рассматривать оба этих шела едино, указывая лишь небольшие различия между ними. POSIX Shell имеет тоже имя что и Bourne Shell поэтому он помещен в /usr/bin/posix директорию в отличии от Bourne Shell, который находится в директории /usr/bin.

 **Key Shell.** Это оболочка для Kourn Shell-a разработанная фирмой Hewlett-Packard. Она позволяет использовать меню и онлайн помощь помогая в построении команд и выполнению ряда часто встречаемых задач, таких как просмотр, редактирование и печать файлов, просмотр содержимого директории и.т.п. Построена она таким образом что пользователь может сам в дальнейшем

расширять ее возможности.

 **Bash.** Название этого шела расшифровывается как Bourne Again Shell. Он бы разработан консорциумом Free Software Foundation и несмотря на то что по умолчанию он отсутствует в стандартной поставке HP-UX 10.20, в следствии своих мощных функциональных возможностей он пользуется огромной популярностью среди пользователей и администраторов HP-UX. Его интерпретатор команд совместим с Bourne Shell. Также он вместил в себе полезные возможности Korn C шеллов. Он разрабатывался в сооответствии со спецификациями IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2).

### 2.1.2 Смена шела

Для того чтобы определить Ваш системный шелл достаточно сразу после логина выполнить команду:

 echo $SHELL

она показывает содержимое переменной *SHELL* в которую система прописывает Ваш шелл установленный по умолчанию. Для временной смены шела достаточно выполнить запуск желаемого шела в текущем:

$ ksh *запуск Kourne Shell*

$ ps *печать списка процессов*

 PID TTY TIME COMMAND

 12320 pts/2 0:00 sh *Bourne shell*

 12322 pts/2 0:00 ksh *Korn Shell*

 12323 pts/2 0:00 ps

$ exit *выход из Korn Shell-a*

Для постоянной смены шела устанавливаемого системой после входа (login shell) необходимо выполнить команду:

chsh *<ваш\_логин> <полное\_имя\_шела>*

*Замечание:* список шелов доступных в системе для пользователей находится в файле /etc/shells

## Bourne Shell.

* + - 1. **Основные возможности Shell.**

**Запуск шелла и выход из шелла**

Для запуска Bourne Shell достаточно ввести sh в текущем шеле. Выход из шела возможен либо по команде exit либо введя в терминале символ коца файла Ctrl-D.

**Последовательное выполнение команд.**

Несколько команд можно последовательно выполнять разграничивая точкой с запятой. Например, фрагмент

 $ who

$ ps –ef

$ ls –l

и

 $ who; ps –ef; ls -l

дадут полностью одинаковый результат.

**Фоновое выполнение.**

Запуск программы в фоновом режиме (без блокировки текущего шела) достигается добавлением в конец команды знака **&**

**Перенаправление ввода-вывода.**

Каждая запущенная программа имеет три ассоциированных канала: стандартный ввод, стандартный вывод и стандартный канал диагностики ошибок. По умолчанию стандартный ввод закреплен за клавиатурой а стандартный вывод и канал ошибок закреплены за монитором. Однако шелл позволяет связать эти каналы с файлами абсолютно прозрачно для выполняемой программмы, т.к. изменения вывода и ввода осуществляются на системном уровне. Примеры перенаправлений ввода-вывода:

|  |  |  |
| --- | --- | --- |
| **Символ** | **Функция** | **Пример** |
| < | Перенаправление ввода из файла  | program < in\_file |
| > | Перенаправление вывода в файл | program > out\_file |
| >> | Перенаправление вывода в файл с добавлением | Program >> out\_file |

**Пайпы.** Две или более программ могут быть объединены таким образом что результат одной программы попадет на вход другой. При этом данные от одной программы к другой следуют через програмный канал, например:

program1 | program2 | program3

или более реалистичный пример:

 ls –l /etc | more

*Примечание:* пайпы всегда работают лишь в одном направлении – со стандартного выхода одной к стандартному входу другой программы.

**Расширение имен файлов.**

В целях уменьшения количества набираемых символов при вводе имен файлов шел поддерживает метасимволы. Нпример для того чтобы вывести листинг всех файлов в текущей директории имена которых начинаются с буквы ***a*** достаточно воспользоваться командой **ls –l a\***

ниже приведена таблица основных метасимволов

|  |  |
| --- | --- |
| Метасимвол | Описание |
| \* | Означает любой символ в любом количестве |
| ~ | Означает путь к домашнему каталогу |
| ? | Любой символ в количестве один или больше |
| [ … ] | Равенство любого одного из символов заключенных в скобки. Пара символов разделенная знаком минус означает любой символ из промежутка между ними. Например [a-zA-Z] – любаялатинская буква |

* + - 1. **Shell скрипты**

**Выполнение скриптов.**

Несколько команд обьедененных одной последовательностью выполнения называется скриптом. Например последовательное выполнения команд, пайпы являются простейшими скриптами. Обычно команды из которых состоят скрипты сохраняют в файлах. Для запуска скрипта можно воспользоваться двумя методами:

sh <*имя\_файла\_скрипта*>

или установить атрибут выполняемости на файл командой

chmod +x <*имя\_файла\_скрипта*>

После чего можно выполнять скрипт:

 ./<*имя\_файла\_скрипта*>

**Вывод текста.**

Для формирования вывода текстовой информации в скриптах используют команду

echo “*строка*”

***Файл .profile*** *.*

Каждый раз во время входа в систему Bourne Shell автоматически запускает файл .profile (скрипт) находящийся в вашей домашней директории. Этот скрипт устанавливает “окружение” в котором Вам предстоит работать. Это различные переменные окружения отвечающие за вид системной подсказки, путь поиска исполняемых файлов, тип терминала и.т.п. Приведем список основных переменных окружения.

* PATH устанавливает путь поиска исполняемых файлов и представляет собой набор директорий разделенных двоеточием
* MAIL определяет имя файла почтового ящика для уведомления о приходе новой электронной почты
* MAILCHECK параметр показывающий как часто (в секундах) следует проверять почтовый ящик на предмет прихода новой почты
* HOME определяет домашний каталог (каталог по умолчанию). Команда **cd** без параметров выполняет переход в этот каталог.
* PS1 основная системная подсказка (по умолчаию $)
* PS2 вторичная системная подсказка (по умолчанию >)
* SHELL имя логин шелла пользователя
* TERM тип терминала пользователя
* EDITOR имя текстового редактора по умолчанию

Устанавливать переменные окружения можно по разному. Например команды

### PATH=/bin:/usr/bin:/usr/sbin:/usr/contrib/bin:/usr/local/bin

и

 PATH=/bin:/usr/bin:/usr/sbin

 PATH=$PATH: :/usr/contrib/bin:/usr/local/bin

полностью эквивалентны. Для того чтобы эти переменные вошли в системное окружение (кроме шела, стали доступны и другим программам) необходимо выполнит команду

export *<имя\_переменной>*

**Комментарии***.*

Текст следующий за символом # рассматривается шелом как комментарий и не оказывает никакого влияния на работу скрипта.

##### Основы программирования на языке shell

**Параметры.**

В дополнение к стандартным параметрам шела можно создавать свои параметры, кторые затем можно использовать в командах. Например

$ x=test

$ echo $x

test

$ aaa=/

$ ls $aaa

cdrom home nsmail tcb var

SD\_CDROM dev lib opt tmp

TT\_DB etc lost+found sbin tmp\_mnt

Bin export net stand usr

При включении параметров в строку или в выражение знак $ должен предшествовать параметру. Также необходимо пользоваться следующими правилами:

${parameter} значение параметра заключенного в скобки будет использовано в выражении. Скобки {} используются когда за параметром следуют буквы или цифры не относящиеся к парамаметру. Например если значение параметра *xxx* равно test то значение выражения *${xxx}*123 будет равняться test123

${parameter:-word} если параметр установлен и не пустой то результатом выражения будет являться значение параметра, в противном случае в результате будет использовано значение параметра *word*. Например: *${xxx:-/usr/bin/sh}*

 если *xxx* пустой то результатом выражения будет являться строка */usr/bin/sh*

${parameter:=word} если параметр неустановлен или пустой, тогда значение *word* будет результатом выражения.

${parameter:+word} если параметр установлен и не пустой, тогда результатом выражения является *word*, в противном случае результат пустой.

**Аргументы командной строки.**

Когда Вы запускаете на выполнение программы, Вы можете передавать ей один или больше аргументов. Шелл скрипты имеют доступ к этим аргументам посредством параметров *$0, $1, $2 …$9*. Если аргументов больше девяти, их значения помещаются в буффер и могут быть доступны с использованием команды *shift* которая будет обсуждаться ниже. Параметр $0 всегда принимает значение имени выполняемого файла, $1 – первого аргумента командной строки, $2 – второго, и.т.п.

Количество агрументов командной строки всегда можно узнать проанализировав параметр *$#*. В том случае если Вам необходим один параметр содержащий все аргументы командной строки разделенные пробелом нужно воспоьзоваться параметром *$\**.

Для получения доступа к аргументам с номером больше девяти необходимо воспользоваться командой *shift.* После очередного выполнения команды *shift* происходит сдвиг на один элемент в буффере параметров в результате которого значение параметра $2 получает параметр $1, значение параметра $3 получает $2, и.т.п. Используя цикл (цыклы будут рассвотрены ниже) в сочетании с командой *shift* можно получать доступ ко всем аргументам командной строки с номерами превышающими 9.

**Квотинг.**

Очень часто случаются ситуации когда необходима особая трактовка тех символов которые шелл воспринимает как спец символы. Например случай когда строку текста включая пробелы нужно передать как один аргумент или знак *$* должен не быть воспринят как указатель на параметр. Для этих целей используется квотинг (quoting).

Например символ бекслэша ( **\** ) можно использовать для квотинга символа *$*.

$ param=aaa

$ echo $param

aaa

$ echo \$param

$param

В сочетании с бекслэшем для квотинга можно использовать двойные кавычки. Например:

 echo "$param is a \"new directory\""

aaa is a "new directory"

При этом знак $ внутри двойных кавычек интерпретируется как указатель на параметр а бэкслэш используется для “экранирования” внутренних кавычек.

Одинарные кавычки также можно использовать для квотинга но в отличии от двойных они “экранируют” все что находится внутри них. Разница сразу становится понятной после замены в предыдущем примере двойных кавычек однираными:

$ echo '$param is a \"new directory\"'

$param is a \"new directory\"

**Подстановка команд.**

*Символ обратного ударения (* ***`*** *)* используемый в скриптах указывает на подстановку команд. Это подразумевает что результат вывода команды подставляется в шелл как параметр. Нпример:

$ echo "The current date is `date`"

### The current date is Sat Jan 6 04:16:35 GMT 2001

или

$ users=`who`

$ echo "Users currentrly logged in the system:\n $users"

Users currentrly logged in the system:

 root console Jan 6 03:57

 roman pts/0 Jan 5 23:12

**Условия.**

Очень часто случаются ситуации при написании скриптов когда необходимо выполнять те или иные команды в зависимости от конкретных условий.

Например, нужно отсортировать файл, а в случае его отсутствия вывести сообщение об ошибке. На этот случай шелл имеет условный оператор, и указанная задача решается с его использованием следующим образом:

if test –f $1

then

sort $1

 else

echo “file $1 doesn’t exist”

fi

Условный оператор имеет следующий синтаксис:

if *<список\_комманд1>*

then *<список\_комманд2>*

elif *<список\_комманд3>*

then *<список\_комманд4>*

…..

else *<список\_коммандN>*

fi

Оператор ifпроверяет статус выполненной команды (в данном примере test –f)

И в случае успеха (программа возвращает 0) выполняет команды стоящие после then, в противном случае выполняются команды else/elif.

Наиболее часто используемой командой в оператореусловия является команда **test**. Она имеет множество опций, полный список которых можно получить обратившись к ее документации выполнив команду man test.Очень часто вместо команды test используют команду **[** которая является ее фунуциональным аналогом. Здесь приводится лишь краткий список основных опций команды test.

-r file возвращает успех если файл существует и доступен для чтения.

 -w file возвращает успех если файл существует и доступен для записи.

 -x file возвращает успех если файл существует и выполняемый.

 -f file возвращает успех если это регулярный файл.

 -d file возвращает успех если это директория.

 -c file возвращает успех если это специальный файл.

 -s file возвращает успех если файл имеет ненулевой размер.

-h file возвращает успех если это симлинк.

 -z s1 возвращает успех если длинна строки s1 нулевая.

-n s1 возвращает успех если длинна строки s1 ненулевая.

 s1 = s2 проверка двух строк на равенство.

s1 != s2 проверка двух строк на неравенство

 s1 возвращает успех если строка s1 непустая

n1 -eq n2 Алгебраическая проверка двух чисел n1 и n2 на

равенство. Помимо –eq есть еще ряд опций для алгебраического

сравнения такие как -ne, -gt, -ge, …

Все вышеперечисленные условия могут обьеденяться с использованием логических операторов:

 ! отрицание

 -a Бинарный И

 -o Юинарный ИЛИ (-а имеет приоритет больший чем –о)

 ( expr ) Группировка выражений скобками

***Ввод данных.***

Для ввода иданных в скрипт можно воспользоваться командой

 read [parameter …]

которая в качестве аргументов принимает список из одного или более параметров.

*Примечание:* знак **$** перед именем параметра в команде read ставить не нужно.

**Циклы.**

Очень часто при составлении скриптов простого последовательного выполнения бывает недостаточно. Необходимы механизмы которые позволяли бы выполнять команды с изменяющимися параметрами. Для этих целей шелл предоставляет в распоряжение три цикличекие конструкции: циклы **for,while** и **untill**.

**Цыкл *for***

Этот тип циклов позволяет выполнять один и тот же набор команд каждый раз с новыми значениями полученными из списка параметров. Он имеет следующий формат:

for *parameter* [ in *wordlist*  ]

do *command-list*

done

где parameter это любое имя параметра, wordlist – один или несколько значений последовательно присваиваемых параметру, command-list – набор команд выполняемых при каждом проходе цикла. wordlist может быть либо просто набором аргументов разделенных пробелом, либо командой шелла которая генерирует сама аргументы. Например следующий скрипт:

for i in 1 2 3 4 5

do

 if mkdir $i

 then

 echo “directory $i was created”

 fi

 done

создает последовательно директории с именами 1 2 3 4 5 и в случае успешного создания директории выдает сообщение. Следующий скрипт:

 for i in `ls a\*`

 do

 cp $i /tmp

 echo “$i was copied”

 done

копирует все файлы начинающиеся с буквы *a*  из текущего каталога в каталог /tmp.

**Цикл *while***

while *command-list1*

do *command-list2*

done

этот цикл запускает команды из списка *command-list1,* и если последняя команда из списка выполнилась успешно (код возврата равен 0) то начинают выполняться команды из списка *command-list2,* в противном случае цикл заканчивается. Цикл из следующего примера:

 while [ -r $1 ]

 do

 echo “processing $1”

 cat $1 >> summary

 shift

 done

по очереди считывает аргументы командной строки, в случае когда в текущем каталоге присутствует файл доступный на чтение с именем совпадающим с этим аргументом, содержимое этого файла дописывается к файлу с именем summary. В противном случае, скрипт прекращает свою работу.

**Цикл *util***

until command-list1

do *command-list2*

done

конструкция этого цикла полностью аналогична циклу while за исключением того что тело цикла (набор команд *command-list2)* выполняется как минимум один раз не зависимо от успешности выполнения условия цикла (набор команд *command-list1).*

**Оператор *case***

Оператор case является расширением стандартного условного оператора **if**. Если есть условие при котором может реализовываться множество вариантов то вместо серии операторов **if** лучше использовать один **case**.

case *parameter* in

*pattern1* [ | *pattern2* …] ) *command-list1* ;;

*pattern2* [ | *pattern3* …] ) *command-list2* ;;

…

esac

шаблоны *pattern* определяют варианты параметра *parameter.* При совпадении параметра с одним из шаблонов будет выполнен соответствующий список команд *command-list.* В квадратных скобках ([ **|** *pattern2*…] ) указаны дополнительные варианты разделенные символом ( **|** ) которые может принимать *parameter* помимо основного. Заканчивается список шаблонов скобкой. Более понятней структура этого оператора станет после рассмотрения примера:

case $i in

 -d | -c ) mkdir dir1

 echo “directory dir1 was created” ;;

 -r ) rmdir dir1

 echo “directory dir1 was removed” ;;

 \* ) echo “invalid option” ;;

 esac

В этом примере скрипт получив при запуске один из аргументов –d или –c попытается создать директорию с именем dir1 , получив аргумент –r попытается ее стереть. Во всех остальных случаях (шаблон \*) он выдаст предупреждение о неправильном аргументе.

*Примечание:* обратите внимание на то что порядок следования гшаблонов в операторе case имеет большое значение. Так если строку

\* ) echo “invalid option” ;;

поставить в самое начало, то на ней будет все время терминироваться скрипт.

**Команда *. (точка****)*

Каждый раз когда Вы запускаете шелл программы, создается еще одна копия шелла в которой они выполняются. Поэтому если Вы написали шелл скрипт, то все переменные “живут” только во время выполнения скрипта. Если вы хотите чтобы все переменные из шелл программы были в вашем текущем окружении запускать скрипт нужно в текущем шеле, это достигается использованием команды . (точка).

 . scriptname

**Команда *eval***

eval [arg ...]

Аргументы читаються и соединяются в одну команду. Затем эта команда выполняется шеллом и статус выхода команды возвращается как результат команды eval. Если аргументы в команде отсутствуют или пустые то команда возвращает нулевой статус.

В качестве примера рассмотрим два скрипта:

 d=’date &’ ; $d

и

 d=’eval date &’ ; $d

первый из них не выполнится так как команда date воспримет символ & как аргумент а не как признак запуска в фоновом режиме, в результате чего первый скрипт не выполнится в отличии от второго.

**Использование метасимволов.**

Во всех конструкциях циклов и операторе case возможно использование метасимволов. Например скрипт из примера оператора цикла for

for i in `ls a\*`

 do

 cp $i /tmp

 echo “$i was copied”

 done

можно переписать в более простом виде

for i in a\*

 do

 cp $i /tmp

 echo “$i was copied”

 done

результат будет один и тот же.

**2.3.1 Некоторые специальные команды**

***Разделители && и ||***

Это условные разделители. При разделении двух команд с помощью && вторая команда выполнится лишь в том случае когда первая завершится успешно. При разделении команд с помощью || вторая выполнится лишь тогда когда первая закончилась неуспешно (код возврата не равен нулю).

Например при выполнении скрипта:

#### test –d /usr/tools && cd /usr/tools

#### test –d /usr/tools || echo “directory doesn’t exist”

вход в директорию /usr/tools будет произведен только при ее наличии, в противном случае будет выдано сообщение о ее отсутствии.

###### Определение функций

Для сокращения обьема шелл программ и упрощения их понимания и сопровждения шелл допускает введение и использование функций. Для опредения функции используется следующий синтаксис:

name () { *list*; }

где name – это имя функции, а list – список команд из которых состоит тело функции. Ниже приведен пример функции возвращающей 0 если аргумент переданный ей является директорией и 1 в противном случае.

dir\_test () {

 if [ -d $1 ]

 then

 echo “$1 is a directory”

 return 0

 else

 echo “$1 is not directory”

 return 1

 fi;

}

вызов функции осуществляется следующим образом

 name [ *parameter* … ]

например в нашем случае это можно сделать так

 dir\_test /usr/bin/sh

###### Перенаправление ввода-вывода

Как было сказано выше для перенаправления ввода-вывода используються символы (> перенаправление вывода, >> перенаправление вывода с добавлением в файл и < перенаправление ввода). Кроме этого существует еще ряд конструкций, одна из которых:

 << [-] *word*

при этом все линии от первой и до содержащей строчку word будут использоваться как входные данные. Например:

 $ cat<<mark

> These words will be printed

> the cat command until the "mark"

> word is found

> mark

These words will be printed

the cat command until the "mark"

word is found

$

Если перед строчкой word стоит минус то все символы табуляции в начале строк будут вырезаны.

Другой часто используемой конструкцией при перенаправлении ввода-вывода является:

 <& цифра

 >& цифра

При этом задействуется файловый дискриптор ассоциированый с указаной цифрой. В большинстве программ со стандартным вводом связан дискриптор 0, со стандартным выводом дискриптор 1, и со стандартным потоком ошибок дискриптор 2. Все программы которые работают друг с другом через пайпы по умолчанию пользуються дискрипторами 0 и 1. Наиболее часто используемые перенаправления это 1>&2 и 2>&1. Рассмотрим следующий пример из которого станет понятен смысл этих конструкций:

 $ ls /no/such/file > out

/no/such/file not found

$ cat out

$

$ ls /no/such/file >out 2>&1

$ cat out

/no/such/file not found

$

первая команда пытается вывести листинг не суцествующего файла перенаправив стандартный вывод в файл out. Файл out при этом оказывется пустым т.к. сообщение об ошибке выводится в стандартный поток ошибок и появляется на терминале. Вторая команда объединяет стандартный поток ошибок со стандартным выводом который перенаправлен в файл out. При этом сообщения об ошибке попадают в файл, о чем свидетельствует команда cat.

**Команда *exec***

exec [*arg* …]

Эта команда выполняет замещение текущего шела новым шелом или программой. Разница между простым запуском шела и запуском через exec становится очевидной на следующем примере:

$ ksh

$ ps

PID TTY TIME COMMAND

 2125 pts/0 0:00 ksh

 2094 pts/0 0:00 sh

 2126 pts/0 0:00 ps

 $

$ exec ksh

$ ps

 PID TTY TIME COMMAND

 2127 pts/0 0:00 ps

 2094 pts/0 0:00 ksh

$

**Команда *expr***

 expr *expression* { +, -, \\*, /, \*, =, \>, \>=, \<, \<=, != } *expression*

Это очень полезная команда для выполнения арифметических операций в шелл скриптах. Например:

 x=10

 expr $x + 5

 y=`expr $x – 10`

 if expr $x \<= $y

 then

 echo “$x is less or equal than $y”

 fi

**Команда *set***

Эта команда используется во многих модификациях. Основное назначение – это устанавливать значение параметров. Если Вы просто запустите эту команду без аргументов то увидите все параметры вашего окружения, большинство из которых было установлено при входе в систему из файла .profile.

Наример команда:

 set bob brr kab ram

установит параметры $1, $2, $3, $4 следующим образом $1=”bob”, $2=”brr”, $3=”kab”, $4=”ram”. Команда set имеет множество опций, полный список которых можно получить обратившись к man-странице (man set).

**Команда *trap***

Команда trap ожидает приход сигналов посланных шелу (от внешних процессов или сигнал от шела вследствии неуспешного запуска программы) и выполняет их обработку.

 trap [*command\_list*] [*s1* …]

Когда trap получает сигнал s1 (сигналы s2 …) она выполняет список заранее предопределенных в *command\_list* команд. Если s1 равен 0, то команды запускаются когда шелл заканчивает свою работу. Команда trap запущенная без аргументов выдает на печать список команд ассоциированных с каждым из номеров сигналов. Ниже приведен краткий список наиболее часто используеиых сигналов.

|  |  |  |
| --- | --- | --- |
| Номер сигнала | **Описание** | **Возможность перехвата** |
| 1 | сигнал перезапуска | Есть |
| 2 | Прерывание процесса | Есть (если процесс не в фоне) |
| 3 | Сигнал выхода | Есть (если процесс не в фоне) |
| 6 | Сигнал завершения | Есть |
| 9 | “Убиение” процесса | Не може быть перехвачен в принцыпе |
| 14 | Сигнал таймера | Нет  |
| 15 | Програмный сигнал завершения процесса | Есть |

Попытка перехватить 11-й сигнал (memory fault) приводит к ошибке. Сигнал с номером 9 не может быть перехвачен т.к. он не перехватывается в принципе.

Очень часто возникает ситуация когда скрипт в процессе работы создает временные файлы которые затем нужно удалить, но в случае прерывания процесса какимто из сигналов эти файлы не будут удалены. Эта проблема решается в следующем примере:

trap “echo ‘removing temporaty file’ rm /tmp/temp” 0 1 2 3 15

**Команда *pwd***

Эта команда показывает текущую рабочую директорию.

**Команда *type***

type [ *prog\_name* …]

Эта команда ищет каталог в котором находится запускаемая программа указанная в аргументе команды.

Команда *times*

Эта команда выдает информацию о времени выполнения процесса запущеного из шелла.

##### Параметры устанавливаемые шеллом

|  |  |
| --- | --- |
| Параметр | **Описание** |
| $# | Количество аргументов командной строки |
| $? | Значение кода возврата предыдущей команды |
| $$ | Номер процесса шела в системе |
| $! | Номер процесса в системе последней команды запущеной в фоне |

**Основные опции для запуска *Bourn Shell***

|  |  |
| --- | --- |
| Опция | **Описание** |
| -с string | Выполнять команды из файла string |
| -s | Выполнять команды полученные со сотандартного ввода (в случе запуска шела без опций он ведет себя также) |
| -r | Запустить шелл в ограниченном режиме (restricted mode) |

При запуске шела в restricted mode запрещается:

* Изменение директории командой cd
* Устанавливать переменную окружения PATH
* Запускать программы содержащие в имени символ /
* Перенаправлять ввод-вывод

## C Shell .

**2.3.1 Общие сведения.**

**2.3.2 История команд**

**2.3.3 Алиасы, подстановка команд, метасимволы**

**2.3.4 Переменные *csh***

**2.3.5 Задания**

**2.3.6 Скрипты**

**2.3.7 Управляющие структуры скриптов.**

**2.3.1 Общие сведения.**

csh это командный интерпретатор HP-UX и язык программирования высокого уровня используемый для трансляции команд вводимых пользователем в системные действия такие как запуск программ, перемещение по директориям файловой системы, управление информационными потоками между программами. csh в отличии от Bourne Shell имеет ряд дополнительных полезных возможностей

* Буфер истории команд
* Механизм алиасов
* Расширеный, С-подобный командный язык

Выход из шелла может осуществляться несколькими способами: по команде exit (либо logout если это логин шелл) и также в зависимости от переменной окружения ignoreoff если она не установлена, то по комбинации клавиш Ctrl-D.

**Запуск шелла**

В зависимости от того является ли csh Вашим логин-шеллом возможны различные сценарии его запуска. При запуске csh проверяет файлы:

 /etc/csh.login Если csh это логин-шелл и этот файл существует, то он

выполняется

 .cshrc Если этот файл существует в вашем домашем каталоге,

то он выполняется в любом случае.

 .login Если csh это логин-шелл и этот файл существует в

домашнем каталоге то он выполняется

 .logout Если этот файл существует в домашнем каталоге, то он

выполняется всякий раз при выходе из шелла если это

логин-шелл

**Установка переменных**

Существуют два типа переменных которые могут быть устаовлены в .cshrc и .login, это локальные переменные, которые не передаются дальше порождаемым из шелла процессам и носят название локальных переменных и глобальные переменные, которые становяться доступными всем процессам порожденным из шелла и носят название переменных окружения. Локальные переменные устанавливаются командой set , глобальные – командой setenv.

**2.3.2 История команд**

csh поддерживает так называемый буфер команд в котором храняться последние введенные команды называемые событиями. Размер буфер определяет переменная history. Переменная savehistory определяет сколько команд буфера подлежит сохранению перед выходом из шела. Наиболее оптимальным размером буфера считается буфер на 10 … 20 команд. Просмотр буфера возможен по команде history. Подстановка команд из буфера осуществляется с использованием символа восклицательного знака ( ! ). Переменная prompt отвечает за вид системоной подсказки.

Например, выполнив команду:

% set prompt='[\!] % '

[2] %

можно изменить стандартную системную подсказку на новую в которой в скобках отображается номер последнего события. Повторный запуск событий из буфера может осуществляться по:

* По номеру события: !n где n – это номер события в буфере
* Относительному номеру события относительно текущего события: !-n
* Тексту события: !*word*где word – первые несколько символов события. Например набрав один раз команду history, в следующий раз можно использовать вместо этого команду !h.

Как специальный случай следует отметить команду !! которая запускает повторно самое последнее событие.

csh также допускает использование аргументов команд находящихся в буфере при посторении новых команд. Каждый аргумент в событии хранящемся в буфере нумеруется начиная с нуля (нулевой аргумент – имя самой команды). Последний аргумент еще представляется знаком ( $ ), первый аргумент знаком ( ^ ). Для ссылки на на аргумент события нужно после номера события через двоеточие ( : ) ввести номер аргумента или его обозначение. Например:

 cruiser 4: ls -l .cshrc

-rw-r--r-- 1 roman users 814 Jan 2 23:08 .cshrc

cruiser 5: history

 1 ls -l .profile

 2 history

 3 cat .profile

 4 ls -l .cshrc

 5 history

cruiser 6: cat !4:$

cat .cshrc

#

# Default user .cshrc file (/usr/bin/csh initialization).

set path=( $path )

# Set up C shell environment:

 if ( $?prompt ) then # shell is interactive.

 set history=20 # previous commands to remember.

 set savehist=20 # number to save across sessions.

 set system=`hostname` # name of this system.

 set prompt = "$system \!: " # command prompt.

endif

**2.3.3 Алиасы, подстановка команд, метасимволы**

csh предоставляет дополнительные удобства в виде командных алиасов. Например, вместо того чтобы каждый раз при необходимости вывести полный листинг директории или при выходе из шелла набирать набирать полные команды, можно создать их сокращенные псевдонимы (dir и x):

alias dir ls –als

alias exit x

для снятия алиаса нужно воспользоваться командой unalias :

 unalias dir

чтобы просмотреть список всех алиасов имеющихся в данный момент нужно ввести команду alias без параметров.

**Подстановка команд**

Подстановка команд в csh выполняется полностью аналогично подстановке в Bourne Shell рассмотреной перед этим:

cruiser 7: set dir=`pwd`

cruiser 8: echo $dir

/home/roman

cruiser 9:

**Метасимволы**

Все етасимволы включая синтаксические ( ; | () & || && ) а так же файловые ( ? \* [] ~ ) и метасимволы квотинга ( \ ‘ “ ) в csh имеют тот же смысл что и рассмотренные перед этим в Bourne Shell.

**2.3.4 Переменные *csh***

|  |  |
| --- | --- |
| **Имя переменной** | **Описание** |
| **$argv** | Список аргументов командной строки |
| **$autologout** | Авто логаут если шеллом не пользовались спустя количеству секунд указанному в этой переменной. Неустановленное или нулевое значение отключает авто логаут. |
| **$cwd** | Указатель текущей рабочей директории |
| **$home** | Домашняя директория |
| **$ignoreeof** | Если переменная установлена, то гнорировать символ конец файла (Ctrl-D) как символ завершения работы |
| **$noclobber** | Если переменная установлена, то запретить перенаправление вывода в существующий файл (операция > ). Перенаправление возможно лишь с использованием операции >! |
| **$notify** | Если переменная установлена то посылать немендленные уведомления после окончания фоновых процессов |
| **$path** | Путь для поиска выполняемых файлов. При изменении этой переменной нужно уведомить шелл выполнив команду rehash |
| **$prompt** | Вид системной подсказки |
| **$status** | Код возврата самой последней команды |

**Цифровые переменные *csh***

Команда at ( @ ) назначает цифровой переменной арифметическое значение, точно так же как и команда set назначает значение стороковой переменной. Значением цифровых переменных являются десятичные целые.

% @ sum=( 1 + 2 )

% echo $sum

3

**Основные арифметические операции в *csh***

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| ( ) | Скобки изменяют порядок выполнения операций |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| % | Остаток от деления |

**Основные логические операции в *csh***

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| == | Проверка на равенство |
| != | Пооверка на неравенство |
| ! | Логическое отрицание |

Кроме этих операций есть еще ряд логических операций которые должны быть взяты в скобки и их операнды должны ьыть разделены пробелами в виде ( operand1 >= operand2 )

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| > | Больше |
| < | Меньше |
| >= | Больше или равно |
| <= | Меньше или равно |
| >> | Правый битовый сдвиг |
| << | Левый битовый сдвиг |
| & | Битовое И |
| | | Битовое ИЛИ |
| && | Логическое И |
| || | Логическое ИЛИ |

Например:

% @ r= ( 2 << 4 )

% echo $r

32

**Основные операции присвоения в *csh***

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| = | Простое рисвоение а = b |
| += | a = a + b |
| -= | a = a - b |
| \*= | a = a \* b |
| /= | a = a / b |
| %= | a = a % b |

**Постфиксные операции**

К последним относятся операции ++ и --

 % @ a=10

 % @ a++

% echo $a

11

**Основные файловые операции**

Виражения в csh могут возвращать значение в зависимости от наличия или отсутствия файла, прав доступа к нему, и.т.п. Для этого используется следующий синтаксис:

 *-file\_test filename*

где *file\_test и filename* могут принимать следующие значения

|  |  |
| --- | --- |
| **file\_test**  | **описание** |
| d | Является ли файл директорией ? |
| e | Существует ли файл ? |
| f | Это обычный файл ? |
| o | Являюсь ли я его собственником ? |
| r | Имеются ли права на чтение из файла ? |
| w | Имеются ли права на запись в файл |
| x | Можно ли исполнять этот файл ? |
| Z | Пустой ли файл |

**2.3.5 Задания**

Каждый раз когда одна или больше команд (например связанные через пайп, или последовательно) выполняються шелл создает один блок команд назывемый заданием. Фактически каждая строка введенная в строке шелла является заданием. Задание может исполняться также в фоновом режиме если при запуске в конце стоял символ &. В этом случае после запуска шелл выдаст номер задания в своей таблице заданий и номер процесса. Просмотреть таблицу активных в данный момент заданий можно воспользовавшись командой jobs . После завершения фонового процесса шелл проинформирует пользователя об этом.

 % sleep 10 &

 [1] 73059

 % jobs

 [1] + Running sleep 10

 % ps

 PID TTY TIME COMMAND

71453 pts/2 0:00.03 -sh (csh)

73059 pts/2 0:00.00 sleep 10

73061 pts/2 0:00.00 ps

…. Спустя 10 секунд ….

%

[1] Done sleep 10

**2.3.6 Скрипты**

сsh крипты могут как и Bourne Shell скрипты быть запущены двумя способами. Либо

 csh *script\_file arg1 arg2 ….*

либо установив атрибут исполняемости на файл и запустив его

 chmod +x *script\_file*

 ./*script\_file arg1 arg2 ….*

При этом первая строчка должна содкржать полный путь к интерпретатору с префиксом #! , в нашем случае она должна выглядеть так: #!/usr/bin/csh . Без этой строчки система в качестве интерпретатора скрипта запустит Bourne Shell что приведет к ошибке.

При запуске csh помещает аргументы скрипта в массив argv доступ к которым возможее через argv[1], argv[2], … Аргумента argv[0] не существует, вместо этого имя скрипта помещается в аргумент $0.

Для проверки установлена ли данная переменная можно воспользоваться конструкцией

 $?*variable*

Чтобы узнать сколько значений хранит переменная нужно использовать следующий синтаксис:

 $#*variable*

Для доступа к компонентам переменной необходимо следовать следующему правилу:

 $*variable*[*componet\_number*]

Также как и в Bourne Shell возможно использование переменных $n вместо argvn и $\* вместо $argv. В отличии от команды read в Bourne Shell csh использует конструкцию $< для интерактивонго ввода.

Все вышесказанное иллюстрируется на следующих примерах:

% echo $#var1

var1: Undefined variable.

% set var1=a

% echo $#var1

1

% set var1=(a b c)

% echo $#var1

3

% echo $var1[1]

a

% echo $var1[1-3]

a b c

% set a=($<)

xxx

% echo $a

xxx

Еще одно отличие csh от Bourne Shell заключается в том что с помощью фигурных скобок { } можно проверять завершилась ли нормально команда помещенная в эти скобки. Например, следующий скрипт:

#!/usr/bin/csh

if ({ cat /tmp/aaa }) then

 echo OK

endif

в случае успешного вывода содержимого файла /tmp/aaa напечатает в конец фразу OK.

**2.3.7 Управляющие структуры скриптов.**

В виду того что все управляющие структуры csh очень похожи на аналогичные в Bourne Shell кратко остановимся лишь на их синтаксисе. По своему синтаксису они очень близки с аналогисными командами языка программирования С.

**Условный оператор**

if *( expression* ) then

 *command1*

 *command2*

 *…*

else

 *command1*

 *command2*

 *…*

endif

Если требуется запуск лишь одной команды то endif можно не ставить:

 if ( *expression* ) *command*

**Цикл *foreach***

Этот цикл является полным аналогом цикла for Bourne Shell.

foreach *index\_variable* (*loop\_values*)

 *command1*

 *command2*

 ….

end

Если в качестве команды внутри цикла встречается команда break то выполнение цикла прерывается, если команда continue то продолжается но уже со следующим из значений параметра взятым из *loop\_values*.

**Цикл *while***

while (*expression* )

 *command1*

 *command2*

 *…*

end

полный аналог цикла while в Bourne shell.

**Команда *switch***

Очень похожа на аналогичный оператор в языке С, является аналогом команды case в Bourne shell

switch ( *word* )

case *string1*:

 *command1*

 *command2*

 ….

 breaksw

case *string2*:

 *command1*

 *command2*

 *….*

 breaksw

…..

default:

 *command1*

 *command2*

 *…*

endsw

**Команда безусловного перехода *goto***

*label*:

 *command1*

 *command2*

 …

 goto *label*

осуществляет переход на команду следующую за меткой указаной в команде (в данном случае на *command1*).

Обработка прерываний

onintr *label*

Эта команда позволяет обрабатывать прерывания скриптов. При выполнении прерывания она выполняет переход на команду стоящей непосредственно за меткой label (аналогично выполнению goto label)

## Korn Shell и POSIX shell

**2.4.1 Общее знакомство с шеллами**

**2.4.2 Старт шеллов**

**2.4.3 Грамматика шеллов**

**2.4.4 Алиасинг**

**2.4.5 Возможности подстановок**

**2.4.6 Командная строчка и история команд**

**2.4.6 Управление заданиями**

**2.4.7 Программирование скриптов**

**2.4.8 Дополнительные команды**

**2.4.1 Общее знакомство с шеллами**

Оба этих шелла базируються на Bourne Shell, но помимо этого они унаследовали много полезных функций Csh. Они рактически на 95% совместимы вверх с Bourne Shell и большинство программ написанных на Bourne Shell будут исполняться на них без изменений. Также следует отметить тот факт что по скорости исполнения они оперережают Bourne Shell.

Основные возможности унаследованные от Csh

* Буфер истории команд.
* Алиасинг
* Массивы
* Целочисленная математика
* Управление заданиями

**2.4.2 Старт шеллов**

В том случае если Korn или POSIX Shell являються логин-шеллами, после входа в систему и запуска они используют файл /etc/profile и файл .profile если последний существует в домашней директории пользователя. Независимо от того запускается ли шелл после логина или в любой другой момент времени, при старте он анализирует переменную окружения $ENV, которая обычно указывет на файл .profile или .kshrc. Если файл с именем указанным в этой переменной существует то он выполняется.

**2.4.3 Грамматика шеллов**

Как и Bourne Shell новые шеллы поддерживают пайпы,перенаправления ввода-вывода, последовательное выполнение команд используя разделители а также запуск программ в фоновом режиме. Единственное что стоит отметить это поддержка двунаправленных пайпов (co-process) которые будут рассмотрены позже.

Механизмы квотинга спецсимволов и расширения имен файлов посредством метасимволов в новых шеллах полностью аналогичны рассмотренным в Bourne Shell. Единственное что следует упомянуть, так это метасимволы ~ , # и % , свойства которых не были описаны раньше.

* метасимвол # используется для задания комментариев. Все что следует за ним игнорируется интерпретатором комманд.
* метасимвол ~ используется при раскрытия путей к файлам и директориям. При этом используются следующие правила:

тильда сама по себе или тильда до слеша / обозначает путь к домашнему каталогу (переменная окружения $HOME)

тильда до знака + раскрывается в текущий каталог (переменная окружения $PWD)

тильда до знака – раскрвывается в предыдущий рабочий каталог (переменная $OLDPWD)

и наиболее полезное правило: если после тильды идет строчка а затем слеш / то выполняется проверка вхождения этой строки в файл пользователей системы ( /etc/passwd ). В случае если эта сточка совпадает с именем пользователя из этого файла то результатом такого расширения пути станет домашний каталог пользователя взятый из этого файла.

* Метасимвол % используется при управлении заданиями и будет рассмотрен позднее.

**2.4.4 Алиасинг**

Механизм алиасинга в новых шелах аналогичен Csh.

 alias -выводит список всех установленных алиасов alias *word*=*command* - устанавливает алиас *word* на команду *command*

 unalias *word* - снимает алиас *word*

 unalias -a - снимает все алиасы

alias –x *word*=*command* - устанавливает алиас *word* на команду *command* с экспортом (только для *ksh*)

Отличительной особенностью ksh от POSIX Shella является возможность экспорта алиасов в другой шелл который не вытесняет текущий (для запуска используется системный вызов fork() на не exec как например в случае если скрипт начинается с #!/bin/…)

**2.4.5 Возможности подстановок**

К новым подстановкам параметров не рассмотреным при рассмотрении Bourne Shell можно отнести

${*parameter*#*pattern*} Если шаблон *pattern* равен началу параметра

${*parameter*##*pattern*} *parameter* , то вырезать из параметра этот шаблон и результат присвоить выражению. В первой форме шаблон вырезается до первого совпадения, во второй – до последнего.

${*parameter*%*pattern*} Если шаблон *pattern* равен окончанию параметра

${*parameter*%%*pattern*} *parameter* , то вырезать из параметра этот шаблон и результат присвоить выражению. В первой форме шаблон вырезается до первого совпадения, во второй – до последнего.

**2.4.6 Командная строчка и история команд**

Установив переменную окружения VISUAL на один из системных текстовых редакторов (vi, emacs, gmacs) можно получить возможность редактировать строку ввода используя команды выбраного редактора. При этом помимо редактирования текущей команды, можно выполнять перемещение по истории команд (клавиши ESC j ,ESC k). По умолчанию история команд хранится в файле .sh\_history однако его можно изменить установив переменную окружения $HISTFILE . Количество хранимых команд определяется переменной $HISTSIZE. Двойное нажатие на клавишу ESC помогает раскрывать имена файлов в каталогах.

**2.4.6 Управление заданиями**

Задания в ksh и POSIX Shell имеют тот же смысл что и рассмотренные ранее в csh. Для управления заданиями испльзуется команда jobs . Запущенная без параметров она показвывает список заданий. Иногда возникает необходимость отложить на некоторое время выполнение текущего задания. Этого можно добиться послав сигнал SIGSTOP текущему процессу. Этого можно добиться комбинацией клавиш: Cntrl-Z.

*Примечание:* предварительно (обычно это ставиться в startup скриптах шеллов) должна быть выполнена команда stty susp <Cntr-Z> которая связывает эту комбинацию клавиш с посылкой сигнала SIGSTOP процессу (генерация символа suspend).

Для перевода отложеного (приостановленого задания) в фоновый режим нужно воспользоваться командой bg

 bg %*job\_number* перевод задания с номером *job\_number* в фоновый режим

 bg %%перевод текущего задания фоновый режим

 bg %+перевод текущего задания фоновый режим

 bg перевод текущего задания фоновый режим

 bg %-перевод предыдущего задания фоновый режим

Для возврата задания в интерактывный режим нужно использовать команду fg. Синтаксис ее полностью аналогичный команде bg.

Встроенная в шелл команда kill допускает использования тех же аргументов что и bg и fg для посылки сигналов заданиям.

**2.4.7 Программирование скриптов**

Все основные управляющие структуры ksh и POSIX Shell полностью повторяют аналогичные в Bourne Shell. Поэтому остановимся более подробно на новых возможностях предоставляемых этими шелами.

**Команда *select***

select *parameter* in *words*

do

 *command\_lines*

done

Эта команда помогает организовать интерактивный диалог. Она выводит список слов *words* в виде меню с возможностью выбора, а после выбора пользователем элемента списка она запускает команды из списка *command\_lines* , при этом выбранное пользователем слово доступно командам из списка как параметр *parameter*.

**Двунаправленные пайпы**

Korn Shell имеет в своем арсенале механизм позволяющий порождать процессы-потомки соединенные пайпом с родительским шеллом. Стандартный ввод и стандартный потомка может быть доступен из родительского шела. Для создания двунаправленого пайпа нужно использовать конструкцию |& после запускаемой команды.Доступ к пайпу из скрипта может быть получен посредством команд print –p и read –p . Рассмотрим пример использования двунаправленых пайпов:

 #!/usr/bin/ksh

bc |&

read a b

print -p "$a \* $b"

read -p mul

echo $mul

*Замечания:*

Использование двунаправленых пайпов оправдано лишь с командами которые работают с устройствами стандартного ввода и вывода, но не напрямую с терминалом (как например редактор vi).

Не существует способа закрыть двунаправленный файл, поэтому в них нельзя использовать команды требующие вначале получения символа конца файла EOF (например программа sort) прежде чем выдать результат.

**Команда *typeset* (только в Korn Shell)**

 typeset [-option ] [*name*[=*value*]] …]

Эта команда создает переменную, назначает ей значение и определяет ее тип. Основные опции команды:

 -i переменная *name* имеет тип integer

 -r переменная *name* является readonly

 -l конвертировать все символы с верхнего регистра в нижний

 -u конвертировать все символы с нижнего регистра в верхний

 -x автоматический експорт переменной

 -R выравнивание текста по правому краю

 -L выравнивание текста по левому краю

**Команда *let***

Эта команда практически аналогична команде expr в Bourne Shell и используется для вычисления в простых математических выражениях. Пример использования:

 $ x=10

$ let x=2\*x+5-3/x

$ echo $x

25

**Команда *ulimit* (только в Korn Shell)**

ulimit [-f *n* ]

Эта команда задает ограничение *n* блоков на файлы записанные на диск порожденными процессами. Запущеная без аргументов показывет текущий лимит.

**Массивы**

Для доступа к елементам массива используют следующий синтаксис:

*array\_name* [*subscript*]=*value*

например:

 $ a[0]=10

$ a[1]=xxx

$ a[2]=tt

$ echo ${a[1]}

xxx

$ echo ${a[\*]}

10 xxx tt

## Key Shell (keysh).

 Это меню-ориентированный интерактивный шелл разработанный компанией Hewlett-Packard для создания дружественного интерфейса к Korn Shell. Шелл содержит последовательности горячих клавиш для быстрого просмотра списка файлов, задач принтера и просмотра файлов, которые он автоматически транслирует в команды HP-UX.

Key Shell содержит все необходимые свойства и возможности Korn Shell (см. Korn Shell). Дополнительные возможности – шелл содержит интерактивную справку, горячие клавиши (видимые) , невидимые последовательности горячих клавиш, строку состояния, возможность конфигурирования.

**Запуск и работа в Key Shell.**

Шелл использует стандартные переменные среды, не требует дополнительной настройки поэтому, если ваша система сконфигурирована под какой-либо другой шелл вам достаточно набрать keysh для его запуска. При запуске шелла сверху вы видите приглашение $ (командная строка). Далее идет строка меню горячих клавиш, на клавиатуре для них соответственно, используются функциональные клавиши F1 – F8. Следует иметь ввиду – что показывается несколько горячих клавиш, а меню разбито на несколько строк по вертикали и переключение между ними осуществляется клавишей F8. Настройки шелла хранятся в /etc/keyshrc или .keyshrc общие и для каждого пользователя соответственно. Нажатие соотвествующей клавиши соответственно приводит к тому что в промпте появляется командная строка, например, F3 выдает команду Change\_dir которая соответствует стандартной cd. Убрать соответствующую команду можно ключом Delete\_line. Выход из всех интерактивных редакторов осуществляется клавишей q. Использование видимых последовательностей горячих клавиш достаточно просто и значительно облегчается благодаря наличию интерактивной справки.

**Неявные (невидимые) ключи**

Кроме всего прочего, keysh позволяет обращаться к стандартным командам HP-UX с использованием своего формата и горячих ключей. Допустим нам необходимо посмотреть календарь на определенный месяц. Команда cal выдает текущий месяц. Cal for\_month позволяет выдает приглашение нажать горячую клавишу для выбора желательного месяца и, таким образом, позволяет избежать изучения формата команды cal. Многострочное мменю подчиняется тем же законам, что и для видимых последовательностей. Кроме этого, вы так же можете пользоваться стандартными командами HP-UX, если вас не устраивает то что выдается в горячих клавишах или вам нужен другой формат, вы просто ингнорируете то, что выводит подсказка и жмете возврат каретки в конце своей командной строки.

**Настройка keysh**

Любые ключи так же можно добавить, фактически это есть соответствие алиасам, например, для шелла bash. Пример: Keysh\_config softkey add hostname with\_label hname from\_user mav

Пример в объяснении не нуждается, поскольку он был создан с использованием интерактивной подсказки шелла и может быть легко восстановлен пошагово. Для невидимых горячих клавиш можно создать такую же командную строку с ключвым словом invisible, например, Keysh config softkey add invisibles.

Перед тем как добавлять свой ключ следует все-таки посмотреть стандартніе последовательности, следует заметить, ято в keysh описаны практически все стандартные последовательности и команды HP-UX поэтому, если вы не настроили какое-нибудь специфическое ПО от третьего производителя, то редактированием последовательностей и не стоит заниматься. Ошиюочно созданный ключ можно удалить последовательностью Keysh config softkey delete.

**Иерархия горячих клавиш, файлы определений.**

Когда создаются новые ключи в keysh, фактически это есть добавление узла в иерархию узлов (с которыми связаны функциональные клавиши). В системе файл настроек и иерархии находится здесь /usr/keysh/C/softkeys.

Каждый узел ключа имеет следующие свойства:

*name*  (обязательное) это командная строка для доступа к невидимому ключу, для видимого ключа это его метка

*type* (обязательное для подменю) свойство поределяет то, является ли подменю командной строкой или параметром

*attributes* (необязательное) определяет поведение горячей клавиши и то как она интерпретируется.

*editrules* (необязательное) это часть поля атрибутов, которая определяет то, как горячая клавиша интерпретируется в команду HP-UX

Общий формат горячей клавиши, таким образом, будет таким

 softkey name

 attributes

 editrules ;

Пример (определение последовательности Copy\_files)

softkey Copy\_files command

 editrule { append("cp"); }

Подменю будет соответствовать более низкому уровню иерархии и определяется таим образом:

{

 type name

 attributes

 editrules

 ;

 .

}

Если узел (клавиша) ассоциирован с подменю, то в фигурных скобках идет определение дочерних узлов. Дочерние узлы при этом могут быть двух типов – литерал или метка, второй – строка – которая содержит команду или опции команды.

Пример – горячие клавиши для вызова последовательности Copy\_files]

 {

 string <files> disable -1 enable all

 editrule { append(argument); }

 required "Enter the name of the file(s) to copy."

 ;

 option to disabled

 required "Enter the name of the file(s) to copy;

 then select \"to\"."

 {

 string <dest>

 editrule { append(argument); }

 required

 "Then, enter the name of the file or directory to

 copy the file(s) to."

 ;

 }

 }

# Администрирование системы X Window

* 1. Базовые концепции X Window
	2. Компоненты системы X Window
	3. Предварительное конфигурирование
	4. Старт X Window
	5. X ресурсы
	6. Управление шрифтами

## Базовые концепции X Window

В HP-UX 10.20 X server это программа которая стартует автоматически после старта системы непосредственно перед тем как будет запущено окно логина. Она захватывает управление графическим дисплеем, клавиатурой и мышью и все дальнейшее взаимодействия между программами и этими устройствами происходит с участием Х сервера.

Х клиент – это программа которая написана для взаимодействия с Х сервером. Учитывая сетевые возможности сисметы Х Window клиент и сервер могут быть физически разнесены и общаться через сеть.

Несмотря на то что Х сервер использует мультизадачные свойства ОС в заданный интервал времени только одно графическое окно может обрабатывать пользовательский ввод, это окно называется активным.

## Компоненты системы X Window

Среди компонент системы можно в первую очередь выделить:

* Компьютерное оборудование
* Операционную систему
* Х сервер обеспечивающий взвимодействие меджу клиентами и дисплеем
* Клиентские программы которые включают
* Оконный менеджер для управления поведением окон
* И непосредственно прикладные программы

В системе X Window очень часто употребительным термином является термин “дисплей”. Чтобы не путать его с монитором (телевизор к которому подкючается компьютер) раскроем его смысл. Дисплей это принципиально устройство вывода информации. Дисплей – это логическое понятие и один дисплей может включать в себя несколько физических мониторов.

Экран это графическое битмап устройство которое после старта Х сервера становиться корневым окном. Оно содержит все остальные окна и прочие визуальные элементы. Одним из вспомагательных но необязательных элементов системы X Window является Font Server, позволяющий передавать по сети шрифты на Х сервера.

## Предварительное конфигурирование

Для работы в системе X Window прежде всего необходимо наличие переменной окружения DISPLAY, файла */etc/X0.host* (в системах X11R4/R5) а также работу механизма резолвинга хостов (файл /etc/hosts и/или DNS и/или NIS). Переменная окружения DISPLAY устанавливается в соответствии с правилом:

export DISPLAY=*Host*:*Display*.*Screen*

Файлы /etc/X**n**.hosts представляют собой текстовые файлы содержащие имена хостов которым разрешено подключаться к Х серверу с номером n (запускаемым как /usr/bin/X11/X :n). При попытке подсоединения к Х серверу **n** необходимо в качестве номера screen в переменной окружения DISPLAY использовать **n**. Конфигурация X серверов храниться в файлах X**n**screens которые соответствуют файлам /etc/X**n**.hosts. При не настроенном механизме резолвинга хостов или в случае сетевых проблем возможно возникновение проблем или задержек в работе X Window. Для контроля доступа к Х серверу используется программа xhost.

xhost + разрешает установку соединения с любым клиентом

xhost - запрещает установку соединения со всеми клиентом

xhost +*hostname* разрешает установку соединения от хоста *host*

xhost -*hostname* запрещает установку соединения от хоста *host*

Система X Window использует следующие конфигурационные файлы:

 .Xdefaults файл содержит ресурсы общие для всех программ

 .x11start файл содержит список клиентов вызывающихся при старте X11

 .mwmrc файл определяющий настройки оконного менеджера Motif.

 .app-defaults/\* индивидуалные настройки клиентов

Если HOME каталог не содердит этих файлов их можно скопировать из системных файлов:

 cp /usr/lib/X11/sys.Xdefaults ~/.mwmrc

 cp /usr/lib/X11/sys.Xdefaults ~/.Xdefaults

Х сервер вначале ищет файлы в HOME каталоге и лишь не нашев их переходит к системным. По умолчанию конфигурация Х сервера хранится в файле /etc/X11/X\*screens. В нем определяются такие параметры как файл устройства, апаратно независимые параметры, глубина цвета, прозрачность окон, а также апаратно зависимые параметры. Полное описание всех параметров можно получить из файла

/usr/lib/X11/Xserver/info/screen/hp

Конфигурациооный файл для устройств ввода Х сервера имеет имя /etc/X11/X\*devices.

* 1. **Старт X Window**

Для старта системы X Window нужно запустить команду x11start. Он стартует программу xinit которая запускает Х сервер и клиентов а также оконный менеджер в зависимости от конфигурационного файла .x11start.

* 1. **X ресурсы**

Ресурсы определяют поведение программ (геометрия, цвета, шрифты, поведение клавиатуры, ….). Существует несколько способов позволяющих их изменять:

* Опции командной строки
* .Xdefaults файл
* загрузка ресурсов через менеджер ресурсов
* application resource файлы (app-delaults файлы)

**Менеджер ресурсов *xrdb***

xrdb [*option*] *filename*

основные опции:

 -load ресурсы загружаемые из файла перезаписывают сущестивующие

 -merge ресурсы загружаемые из файла обьеденяются с сущестивующими

 -remove ресурсы указаные в файле удаляются из собственности менеджера

ресурсов.

 -edit поместить ресурсы собственности менеджера в файл

Для задания ресурсов используют строку вида:

 [client\_name] \* resource: value

или

 [client\_сlass] \* resource: value

## Имена и классы клиентов

Каждый Х клиент имеет имя и класс. Имя определяет специфического клиента а класс категорию клиента. Ресурсы определенные через имя клиента имеют больший приоритет перед теми что определены через класс клиента. Для лучшего понимания ресурсы определенные через класс пишуться с большорй буквы. Назначить имя клиенту можно при его старте:

xterm –name myTerminal

чтобы получить к ним доступ достаточно загрузить в ресурс менеджер следующие строки:

 Xterm.name: myTerminal

 MyTerminal\*background: green

## Имена и классы ресурсов

Ресурсы как и классы имеют имена и классы. Индивидуальные пишутся с маленькой буквы. Ресурс класса всегда ссылается на класс ресурсов. Поэтому если ресурс \*background: red сделает все тоновые цвета красными, то \*Background: red сделает красными те ресурсы которые принадлежат к классу Background к которым могут принадлежать cursorColor, pointerColor … Индивидуальные ресурсы всегда имеют приоритет перед ресурсами классов. Это демонстрирует следующий пример ресурсного файла:

 \*Foreground: red

 Xterm\*Foreground: gray

 Xterm\*foreground: yellow

 Xterm\*CursorColor: green

В именах ресурсов возможно употребление символа \* на первом месте. Нарример ресурс \*foreground будет обозначать цвет тона всех приложений, в отличии от xterm\*foreground только xterm-a.

## Типы ресурсов

Помимо того что каждое приложение может использовать свои собственные ресурсы, существует огромное количество стандартизированых ресурсов. Напрример рассмотренные выше цвета которые можно задавать и в RGB представлении (соответствие между именем цвета и RGB содержится в файле /usr/X11/lib/rgb.txt). Помимо них к стандартным ресурсам относятся геометрические (width, height, column, row), шрифты (Font, FontList, FontSet) и.т.п.

## Управление шрифтами

В системе X11 шрифты бывают двух видов: bitmap (растровые) шрифты и scalable typeface (векторные). Растровый шришт состоит из набора файлов в каждом из которых хранятся символы определенного размера. С растровыми шрифтами напрямую может работать Х сервер и Font сервер. Векторный шрифт перед тем как должен быть выведен на дисплей проходит серию математических обработок после которых он превращается в растровый определенного размера, начертания и.т.п. Если Х сервер желает работать с векторными шрифтами, он должен их получить от Font сервера.

**Настройка путей поиска шрифтов. Команда *xset*.**

 В качества источника шрифтов для Х сервера может выступать либо директория, либо Font сервер который принимает соединения на определенный TCP порт (tcp/*<hostname>*:*portnumber).*  Путь поиска может быть установлен командой *xset*.

xset *option*

где *option* может принимать следующие значения:

 q выводит информацию о системе X11 включая fontpath

-fp source[,source …] удаляет источник с начала (-fp) или с конца (fp-) пути

fp- source [,source] поиска шрифтов

+fp source[,source …] добавляет источник к началу (+fp) или к концу (fp+) пути

fp+ source [,source] поиска шрифтов

fp= source [,source] назначает fontpath

fp default сбрасывает fontpath в значение по умолчанию

fp rehash заставляет сервер перечитать базу шрифтов (это нужно в тех

случаях когда изменялось содержимое директорий со

шрифтами)

**Получение списка шрифтов доступных в системе. Команда *xlsfont***

xlsfont [-*option*]

Где *option*

-display host:display Х сервер с которого нужно получить листинг шрифтов

-l генерировать маленький листинг

-ll генерировать большой листинг

-lll генерировать очень большой листинг

-n количество колонок для листинга

-w ширина каждой колонки для листинга

В выводимом листинге могут быть как шрифты так и алиасы, последние имеют короткое название.

**Использование Font сервера.**

Фонт сервер позволяет иметь одну единственную точку с которой все Х сервера будут загружать шрифты, что существенно облегчит конфигурирование систем с большим количеством Х серверов. Кроме того как было сказано выше, если Х сервер хочет работать с векторными шрифтами – он должен это делать исключительно через фонт сервер.

Запуск фонт сервера (если он не был запущен предварительно) осуществляется командой:

/usr/X11/bin/fs –daemon –port <*TCP\_port*>

Для того чтобы фонт сервер запускался во время старта, нужно разрешить его автостарт выполнив:

 /etc/set\_parms font\_c-s

Конфигурационный файл фонт сервера находится в каталоге /etc/X11/fs/config . По сигналу SIGUSR1 он перечитывает свою конфигурацию.

**Описание шрифтов. XLFD формат.**

Стандарт Х11 предусматривает язык описания шрифтов XLFD (X Logical Font description). Согласно ему имя шрифта состоит из 15 позиций разделенных минусами:

FontNameRegistry-Foundry-FamilyName-Weightname-Slant-SetwidthName-AddStyle-Name-PixelSize-PointSize-ResolutionX-ResolutionY-Spacing-AverageWidth-CharSetRegistry-CharSetCoding

Каким именно будет результирующее имя в XLFD формате зависит от типа фонт-запроса к Х серверу. Возможные типы запросов:

 Reference XLFD Этот запрос идет при выполнениии команды xlsfonts и имя

берется из файла fonts.dir . Векторные шрифты при этом имеют PixelSize и PointSize нулевыми.

 Request XLFD В результате этого запроса будет получено XLFD имя

запрашиваемого шрифта или его алиас из файла font.alias. При этом поля содержащие \* (обозначающие любое значение) будут заменены на ?

resolved XLFD Это уже точное имя шрифта которое выддается сервером в ответ на запрос. Все поля являются заполненными, но результат может и не совпадать с исходным запросом.

**XLFD синтаксис**

FontNameRegistry авторитетный источник который зарегистрировал шрифт. Обычно пустое поле ☺

Foundry имя “оцифровщика” шрифта

FamilyName трейд-марка или коммерческое имя шрифта

WeightName[ext] относительный вес шрифта (жирность). Для векторных шрифтов может обозначать темность или светлось (параметр ext)

Slant[ext] напрвление шрифта (roman, italic, oblique, …) для векторных шрифтов параметр задает наклов в угловых единицах.

SetwithName ширина юнита (сжатый или растянутый)

AddStyleName[ext] название для уникальной идентификации шрифта (serife, cursive, …) В векторных шрифтах определят степень поворота или зеркальность шрифта.

PixelSize[ext] высота шрифта в пикселях. Для векторных шрифтов параметр указывает дополнительную растяжку по горизонтали.

PointSize[ext] размер кубика в поинтах.

ResolutionX Разрешение (горизонтальное и вертикальное) шрифта в

ResolutionY пикселях на инч. Если не указано сервер выбирает сам в зависимости от разрешения дистплея.

Spacing расстояние между юнитами в шрифте. (М – фиксированное,

Р – пропорциональное)

AverageWidth Cредняя ширина шрифта

CharacterSetRegistry имя закрепленное X консорциумом за CharacterSetEncoding

CharacterSetEncoding определяет кодировку

Файл font.dir

Этот файл находится в каталогах файлов фонтов, он создается либо при инсталяции системы либо после выполнения команды mkfontdir для растровых или stmkdirs для векторных шрифтов. Он содержит в первой строчке количество шрифтов в директории а в последующих XLFD названия шрифтов.

**Файл font.alias**

Этот файл так же как и font.dir содержится в каталогах шрифтов и служит для лиасинга

длинных имен шрифтов в короткие которые затем легче использовать. После правки этого файла обязательно нужно выполнять команду

 xset fp rehash

а также рестартовывать по сигналу SIGUSR1 фонт сервер.

**Администрирование растровых шрифтов.**

Для добавления растрового шрифта в систему нужно выполнить следующие действия:

* Если шрифт не в .pcf формате сконвеертировать его с помощью программы bdftopcf
* Скомпрессировать шрифт утилитой compress
* Скопировать в нужную директорию
* Запустить mkfontdir для модификации fonts.dir файла.
* Если директория со шрифтами используется только Х сервером то выполнить xset fp rehash , если она используется еще и фонт сервером то рестартовывать по сигналу SIGUSR1 фонт сервер.

Для удаления растрового шрифта из системы Х11 необходимо:

* Удалить фонт файл.
* Запустить mkfontdir для модификации fonts.dir файла.
* Если директория со шрифтами используется только Х сервером то выполнить xset fp rehash , если она используется еще и фонт сервером то рестартовывать по сигналу SIGUSR1 фонт сервер.

Для создания fonts.dir файла достаточно запустить mkfontdir указав ей в качестве аргумента директорию со шрифтами.

**Компилирование .BDF шрифтов в .PCF шрифты.**

Растровые шрифты в системе Х11 могут быть представлены в нескольких формах:

* .pcf Переносимый бинарный формат описания шрифта
* .pcf.Z компрессированный .pcf
* .bdf текстовый формат
* .bdf.Z компрессированный .bdf
* .bcf компрессированный .bdf
* .snf не переносимый бинарный формат шрифта (использовался до X11R5)
* .snf.Z компрессированный .snf (использовался до X11R5)
* .scf компрессированный .snf (использовался до X11R5)

Предпочтительным форматом для Х сервера является компрессированый .pcf.

Для конвертации .bdf в .pcf с одновременной компрессией можно воспользоваться командой:

bdftopcf font\_file.bdf | compress > font\_file.pcf.Z

# Программирование на HP-UX

Для создания выполняемых програм, нужно скомпилировать исходный код где содержиться главная програма.

 Расмотрим пример компиляции.

$ cc –Aa myprog.c

 Процес компиляции покадет все сообщения (статус,предупреждения, ошибки) на стандартный поток вывода ошибок (**stderr**). После этого компилятор создаст файл **a.out** который уже можно запускать. Аналогично можно скомпиларовать Фортрановскую прогамму командой **f77**. Если програма состоит из несколько файлов, то омпиляция будет выглядет следующем образом:

**$ cc –Aa main.c myfunc.c**

**main.c:**

**myfunc.c:**

после этого можно будет запускать **a.out**.

Можно сказать что процес компиляции похожий как на рисунке:

Исходный код

Всмогогатель-ные коды

**КОМПИЛЯТОР**

**a.out**

На самом деле процесс компиляции намноого сложнее. Этот процес компиляции занимает несколько этапов.

1. Для каждого исходного файла запускаеться компилятор который создает обьектный файл (если исходные коды написаны на разных языках програмирования, то для каждого запускаеться тот соотвествующий компилятор)
2. После компиляция (этап создания) обьектных файлов запускаеться линковщик (HP-UX linker (**ld**))

На картинке можете увидеть более детальный процес компиляции:

Исходный код

Дополнительные кода

**Компилятор**

Обьектный файл (**.о**)

Обьектный файл (**.о**)

# Линкеровщик (ld)

**a.out**

Процес компиляции

Для более детального просмтотра этапов прохождения компиляции, можно посмотреть задав опцию **–v** (verbose)

**$ cc -Aa -v main.c myfunc.c**

**cc: CCOPTS is not set.**

**main.c:**

**/opt/langtools/lbin/cpp.ansi main.c /var/tmp/ctmAAAa16327 -D\_\_hp9000s700 -D\_\_hp9000s800 -D\_\_hppa -D\_\_hpux -D\_\_unix -D\_PA\_RISC1\_1**

**cc: Entering Preprocessor.**

**/opt/ansic/lbin/ccom /var/tmp/ctmAAAa16327 main.o -Oq00,al,ag,cn,Lm,sz,Ic,vo,lc,mf,Po,es,rs,sp,in,vc,pi,fa,pe,Rr,Fl,pv,pa,nf,cp,lx,st,ap,Pg,ug,lu,dp,fs,bp,wp! -Aa**

**myfunc.c:**

**/opt/langtools/lbin/cpp.ansi myfunc.c /var/tmp/ctmAAAa16327 -D\_\_hp9000s700 -D\_\_hp9000s800 -D\_\_hppa -D\_\_hpux -D\_\_unix -D\_PA\_RISC1\_1**

**cc: Entering Preprocessor.**

**/opt/ansic/lbin/ccom /var/tmp/ctmAAAa16327 myfunc.o -Oq00,al,ag,cn,Lm,sz,Ic,vo,lc,mf,Po,es,rs,sp,in,vc,pi,fa,pe,Rr,Fl,pv,pa,nf,cp,lx,st,ap,Pg,ug,lu,dp,fs,bp,wp! -Aa**

**cc: LPATH is /usr/lib/pa1.1:/usr/lib:/opt/langtools/lib:**

**/usr/ccs/bin/ld /opt/langtools/lib/crt0.o -u main main.o myfunc.o -lc**

**cc: Entering Link editor.**

Из этого примера можно посмотреть такие этапы

**cpp.ansi** это С препроцесор после этого запускаеться **/lib/ccom** – эта програма(компилятор) уже создает **.о** файлы. Последний этап это этап создания исполняемого кода, это Линкер, которые связывает все обьекты .

**Что такое Обьектный файл ?**

Обьектный файл содержет машиные инструкции а данные с которых линкеровщик создает исполняемую програму. Каждий обьектный файл содержит НАЗВАНИЕ (symbol name) и ссылку на это название.

Названия делятся на 3 категории:

1. Локальные обьявления (local definition) – это коды или данные которые могут использоваться только в том обьектном файле где они обьявленые.
2. Глобальные обьявления (global definition) – это обьявления прорцедур,функций,данных котоые могут быть доступны из других обьектных файлов
3. Внешние ссылки (extern references) – это обьявления которые глобальный инаходяться в других обьектных файлах.

Для просмотра обявлений успользуеться програма **nm**.

## Этап линкирования.

Линкирование это последний этап создания запускания файлов, он в включает в a.out файл все ссылки обьявлений и их реализации , которые встречаються в програме. Если например есть сылка а нет реализаци то линкировщик скажет что не находит внешнего обявления и выдаст следюющее:

**$ cc main.c**

**/bin/ld: Unsatisfied symbol:**

 **my\_func (code)**

## Работа с библиотеками

Очень полезным средсвом для хранения сылски реализаций внешних обьявлений есть библиотеки. Стандартная библиотека libc которая содержит «основные» функции для C,Fortran

Библиотеки называються **lib**name.sfx

Name – название библиотеки, которая идентефецирует библиотеку

Sfx - если .а – архив, .sl – общедоступная библиотека.

Для того что б указать компилятору библиотеку то указываеться через опцию –l. Например

--lm (подключает стандартну математическую библиотеку libm.a).

По умолчанию подключаються библиотеки **libcl**,**libisamstub**,**libc.**

По умолчанию библиотеки ищатся по путям /lib,/usr/lib,lib/libp. Можно задать пути где искать:

1. Переменой коружения LPATH
2. Опция линкера -L

Также для каждой програмы входит обьектный файл **/lib/crt0.o** В этом файле содержаться таочки входа в програму,простомтр аргументов и прочее.

Можна прочитать о фунциях которые есть в стандартных библиотеках исполюзую **man-page**

Вызовы (функции) описываються следующим образом

Name(nL)

Name – название

N – 2-системные вызовы, 3-другие библиотеки

L – буква которая означает к которой библиотеке вызов относиться

Вот примеры:

|  |  |
| --- | --- |
| Група | Описание |
| (2) | Системные вызовы, низкоуровневый доступ до системных ресурсов. (работа с файлами,сигналы,управление процесами). Все вызовы содержаться в **libc** |
| (3C) | Стандартные С вызовы . Находяться в **libc** |
| (3S) | Стандартные вызова input/output (stdio(3S)) Находяться в **libc** |
| (3M) | Математические фунции. Для подключения используеться –lm или -lM |
| (3G) | Графические фунции |
| (3I) | Библиотека инструментариев |
| (3X) | Разные специализированые библиотеки |

|  |
| --- |
| Сравнительные оценки Архивных и Общедоступных библиотек |
| Расширение | .a | .sl |
| Обьектный код | Делаеться с обьектного кода | Делаеться с независимо-позиционого (PIC) обьектного кода.Делаеться компилятором с опцией +z или +Z. |
| Создание | Составляеться обьектные файлы **ar** командой | Составляеться PIC обьекты с ld командой |
| Связывание адресса вызова | Адресс определяется при линкировании програмы | Адресс определяеться при выполнении програмы |
|  a.out | Содержит в себе все вызовы и даные | Содержит только таблицу где где содержаться адреса иназвание библитек |
| При запуске | Каждая програма содержит собственую копию библиотеки | Все програмы используют одну бублиотеку, которая в памяти присутствующая только один раз |

Опции компилятора cc

cс [option] files

* -Amode
* mode=c По умолчанию, стандартный компилятор С (по Керниган, Риттчи)
* mode=a ANSI C (ISO 9899:1990)
* mode=e Расшириное ANSI C
* -c Отменить фазу редактирования связей и создавать об'ектный файл даже в случае программы, состоящей только из одного модуля.
* -p Сгенерировать дополнительные команды для подсчета числа обращений к каждой функции. Кроме того, если имеет место фаза редактирования связей, стандартная подпрограмма инициализации заменяется на такую, которая автоматически вызывает функцию [monitor(3C)](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5CMONITOR.3.shtml) и обеспечивает запись файла mount.out при нормальном завершении об'ектной программы. Профиль выполнения программы может быть затем получен при помощи команды [prof(1)](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5CPROF.1.shtml).
* -Dname=def определяет макрос для препроцесорра (эквивалентно #define )
* -E посылает на стандартный поток вывода (по умолчанию на stderr)
* -g содержит дополнительную информацию для отладки
* -Idir Изменить алгоритм поиска включаемых (посредством директивы #include) файлов, имена которых не начинаются с символа /, а именно: сначала искать в указанном каталоге, а затем уже в каталогах стандартного списка. Так, включаемые файлы, чьи имена заданы в двойных кавычках, сначала ищутся в каталоге, содержащем файл, затем в каталогах, указанных с помощью опции -I, а затем уже в каталогах стандартного списка. Включаемые файлы, чьи имена заданы в угловых скобках, не ищутся в каталоге, содержащем файл.
* -lname включает библиотеку
* -L dir dir= Дополнить каталогом список каталогов, которые содержат об ектные библиотечные модули [для редактирования связей посредством ld
* -v расширынай информация о процессе компиляции
* -w не показывает предупреждений
* -Wx,arglist передает аргументы (опции) arglist для процеса. x

может принимать значения:

* d Driver
* p Preprocessor
* c Compiler
* a Assembler
* l Linker
* +z,+Z Опция создает PIC код
* -O Включить оптимизацию обьектного кода

## Создание архивной библиотеки.

1. Для открытия библиотеки необходимо создать оььектные файлы. (в основном каждая функция представляет свой обьектный файл)
2. Соеденить все обьекты в один архив командой ar с ключем r

### Описание команды ar

 ar [-][d][r][q][t][p][m][x][v][c][l][s] [позиционирующее\_имя]

 а\_файл [имя ...]

Команда ar предоставляет средства обслуживания группы файлов, об единенных в один архивный файл. Применяется главным образом для создания и изменения библиотечных файлов, используемых редактором связей. Может применяться и для других подобных целей. Магические цепочки и заголовки файлов состоят из печатаемых ASCII-символов, так что если в состав архива входят только печатаемые файлы, то и архив в целом окажется печатаемым.

При создании архива командой ar заголовки файлов строятся в формате, не зависящем от конкретной машины. Формат и структура мобильного архива подробно описаны в [ar(4)](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5CAR.4.shtml). Таблица имен архива (описанная там же) используется редактором связей [ld(1)] для сокращения числа проходов по библиотекам об ектных файлов. Команда ar создает и поддерживает таблицу имен только при наличии в архиве хотя бы одного об ектного файла. Таблица имен в случае ее создания помещается в начале архива в качестве файла с особым именем. Ни ссылка на этот файл, ни доступ к нему для пользователя невозможны. При создании или изменении архива командой [ar(1)](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5CAR.1.shtml) таблица имен всякий раз перестраивается. Таблицу имен можно перестроить принудительно, воспользовавшись описанной ниже опцией s.

В отличие от командных опций командный ключ составляет обязательную часть командной строки ar. Ключ (которому может предшествовать символ -) представляет собой один из символов набора drqtpmx. Аргументами же ключа могут служить один или несколько символов из набора vuaibcls. Позиционирующее\_имя - это имя элемента архива, которое используется в качестве указателя конкретного места архива, куда должны помещаться другие файлы. А\_файл - это имя архивного файла. Под именами подразумеваются имена файлов, входящих в архив. Символам, образующим ключ, приписан следующий смысл:

|  |  |
| --- | --- |
| d  | Удалить указанные файлы из архива. |
| r  | Заменить указанные файлы в архиве. Если в ключе наряду с r присутствует необязательный символ u, то замена будет произведена только для тех из указанных файлов, у которых дата последней модификации превышает соответствующую дату у одноименных файлов, хранящихся в архиве. Если ключ содержит признак позиционирования, т.е. один из необязательных символов abi, то в команде должен присутствовать аргумент позиционирующее\_имя и в этом случае все новые файлы будут помещаться перед (b или i) или вслед за (a) файлом с таким именем. При отсутствии признака позиционирования новые файлы будут помещаться в конец архива. |
| q  | Быстро поместить указанные файлы в конец архива. Использование символов позиционирования недопустимо. Проверка, имеются ли уже в архиве указанные файлы, командой не осуществляется. Данная возможность полезна только для того, чтобы избежать квадратичного роста временных затрат при наращивании больших архивов. Отказ от проверок может, напротив, повести к росту размеров архивного файла. |
| t  | Вывести оглавление архива. Если имена не указаны, перечисляются все файлы архива; если имена указаны, выводятся только они. |
| p  | Напечатать указанные файлы из архива. |
| m  | Переместить указанные файлы в конец архива. Если ключ содержит признак позиционирования, то в команде должен присутствовать аргумент позиционирующее\_имя, и тогда место, куда перемещаются файлы, будет определяться так же, как и для опции r. |
| x  | Извлечь указанные файлы из архива и поместить в текущий каталог. Если имена не указаны, извлекаются все содержащиеся в архиве файлы. Операция не изменяет архивный файл. |

Аргументам ключа приписан следующий смысл:

|  |  |
| --- | --- |
| v  | Вывести подробное, файл за файлом, описание процедуры создания нового архивного файла из старого архива и указанных в команде файлов-компонентов. При совместном использовании ключа t и аргумента v выводится подробная информация о каждом файле. При совместном использовании x и v по мере извлечения файлов будут выводиться их имена. |
| c  | Подавить сообщение, выдаваемое обычно при создании а\_файла. |
| l  | Помещать временные файлы в локальный (текущий рабочий) каталог, а не в подразумеваемый временный каталог TMPDIR. |
| s  | Принудительно регенерировать таблицу имен архива, даже если вызов не предусматривает модификации содержимого архива. Эта команда полезна при восстановлении таблицы имен после применения к архиву команды  |

## Открытие Библиотек Распределеного доступа

Первый шаг в создание общедоступной библиотеки должен создать объектные файлы,

cодержащий переместимый код (PIC). Имеются два способа создать

PIC объектные файлы:

* Компилировать исходные файлы с + z или + Z опция компилятора, описанная ниже.
* Записать программы на языке ассемблера, которые используют соответствующее адресование режимы

+ z и + Z параметры вынуждают компилятор генерировать PIC объектные файлы.

#### Пример

Предположите, что Вы имеете некоторые функции C, сохраненные в length.c, которые конвертируют(преобразовывают) между Английскими и Метрическими модулями длины. Для компилиции эти подпрограммы и создайние PIC объектных файлов с компилятором C, Вы можете бы использовать эту команду:

 $ cc -Aa -c +z length.c

+z опция создает PIC.

## Создание Общедоступной Библиотеки с ld

Чтобы создавать общедоступную библиотеку от одного или большее количество PIC объектные файлы, используйте линкер ld, с -b опцией. По умолчанию, ld назовет библиотеку **а.out**. Вы можете изменять название с -o опцией.

Например, предположите, что Вы имеете три исходных файла C, содержащие подпрограммы, чтобы делать длину, объем, и массовые преобразования модуля. Они названы length.c, volume.c, и mass.c, соответственно. Делать общедоступную библиотеку от этих исходных файлов, сначала компилируют все три файла, использующие +z опцию, затем комбинируют заканчивающиеся .o файлы с ld. Показаны ниже команды, которые Вы использовали бы, чтобы создать общедоступную библиотеку, названную libunits.sl:

 **$ cc -Aa -c + z length.c volume.c mass.c**

 **length.c:**

 **volume.c:**

 **mass.c:c:**

 **$ ld -b -o libunits.sl length.o volume.o mass.o**

Как только библиотека создана, убедитесь наличия прав читения и выполнения.Но можна выставить права такой командой

 **$ chmod +r+x libunits.sl**

Например, если Вы имеете программу c названным convert.c, который вызываетподпрограммы с libunits.sl, Вы могли бы компилироватьИ связь это с командой cc:

 **$ cc -Aa convert.c libunits.sl**

Как только выполнимая программа создана, библиотека не должна быть перемещена потому что

абсолютное имя пути библиотеки сохранено в выполнимой программе

## Модифицирование Общедоступной Библиотеки

Команда ld не может заменять или удалять объектные модули в общедоступной библиотеке. Поэтому, чтобы модифицировать общедоступную библиотеку, Вы должны повторно связать библиотеку со всеми объектными файлами, которые Вы хотите, чтобы библиотека включила. Например, предположите, что Вы устанавливаете некоторые подпрограммы в length.c (от предыдущего раздела) которые давали неправильные результаты. Чтобы модифицировать libunits.sl библиотеку, чтобы включить эти изменения(замены), Вы использовали бы этот ряд команд:д:

  **$ cc -Aa -c + z length.c**

 **$ ld -b -o libunits.sl length.o volume.o mass.o**

Любые программы, которые используют эту библиотеку, будут теперь использовать новые версии подпрограмм. То есть Вы не должны повторно связать никакие программы, которые используют эту общедоступную библиотеку. Это - то, потому что подпрограммы в библиотеке приложены к программе во время выполнения.я.

Это - одно из преимуществ общедоступных библиотек по библиотекам архива: если Вы изменяете(заменяете) библиотеку архивов, Вы должны повторно связать любые программы, которые используют библиотеку архивов. С общедоступными библиотеками, Вы должны только освежить библиотеку.

## Применение make

Создание программы частенько начинается с маленького однофайлового проекта. Проходит некоторое время и проект, как снежный ком, начинает обрастать файлами, заголовками, подключаемыми библиотеками, требуемыми опциями компиляции... и для его сборки становится уже недостаточным сказать "cc -o file file.c". Когда же, через пару дней, однажды набранная магическая строчка, содержащая все необходимые для сборки проекта параметры компилятора, таинственно исчезает в недрах истории вашего командного интерпретатора, рождается естественное желание увековечить свои знания в виде, к примеру, шелл скрипта. Затем, возможно, захочется сделать этот скрипт управляемым параметрами, чтобы его можно было использовать для разных целей... Однако, чудо юникса состоит в том, что если вам что-то понадобилось, значит кто-нибудь это уже сделал, и пришло время вспомнить о существовании команды make.

Рассмотрим несложную программу на C. Пусть программа prog состоит из пары файлов кода main.c и supp.c и используемого в каждом из них файла заголовков defs.h. Соответственно, для создания prog необходимо из пар (main.c defs.h) и (supp.c defs.h) создать объектные файлы main.o и supp.o, а затем слинковать их в prog. При сборке вручную, выйдет что-то вроде:

cc -c main.c defs.h

cc -c supp.c defs.h

cc -o prog main.o supp.o

Если мы в последствии изменим defs.h, нам понадобится полная перекомпиляция, а если изменим supp.c, то повторную компиляцию main.о можно и не выполнять. Казалось бы, если для каждого файла, который мы должны получить в процессе компиляции указать, на основе каких файлов и с помощью какой команды он создается, то пригодилась бы программа, которая во-первых, собирает из этой информации правильную последовательность команд для получения требуемых результирующих файлов и, во-вторых, инициирует создание требуемого файла только в случае, если такого файла не существует, или он старше, чем файлы от которых он зависит. Это именно то, что делает команда make! Всю информацию о проекте make черпает из файла Makefile, который обычно находится в том же каталоге, что и исходные файлы проекта.

Простейший Makefile состоит из синтаксических конструкций всего двух типов: целей и макроопределений.

Цель в Makefile - это файл(ы), построение которого предполагается в процессе компиляции проекта. Описание цели состоит из трех частей: имени цели, списка зависимостей и списка команд интерпретатора sh, требуемых для построения цели. Имя цели - непустой список файлов, которые предполагается создать. Список зависимостей - список файлов, из которых строится цель. Имя цели и список зависимостей составляют заголовок цели, записываются в одну строку и разделяются двоеточием. Список команд записывается со следующей строки, причем все команды начинаются с обязательного символа табуляции. Возможна многострочная запись заголовка или команд через применение символа "\" для экранирования конца строки. При вызове команды make, если ее аргументом явно не указана цель, будет обрабатываться первая найденная в Makefile цель, имя которой не начинается с символа ".". Примером для простого Makefile может послужить уже упоминавшаяся программа prog:

prog: main.o supp.o

 cc -o prog main.o supp.o

main.o supp.o: defs.h

В прведенном примере можно заметить ряд особенностей: в имени второй цели указаны два файла и для этой же цели не указана команда компиляции, кроме того, нигде явно не указана зависимость объектных файлов от "\*.c"-файлов. Дело в том, что команда make имеет предопределенные правила для получения файлов с определенными суффиксами. Так, для цели - объектного файла (суффикс ".o") при обнаружении соответствующего файла с суффиксом ".c", будет вызван компилятор "сс -с" с указанием в параметрах этого ".c"-файла и всех файлов - зависимостей. Более того, в этом случае явно не указанные ".c"-файлы make самостоятельно внесет в список зависимостей и будет реагировать их изменение так же, как и для явно указанных зависимостей. Впрочем, ничто не мешает указать для данной цели альтернативную команду компиляции.

Вы вероятно заметили, что в приведенном Makefile одни и те же объектные файлы перечисляются несколько раз. А что, если к ним добавится еще один? Для упрощения таких ситуаций make поддерживает макроопределения.

Макроопределение имеет вид "ПЕРЕМЕННАЯ = ЗНАЧЕНИЕ". ЗНАЧЕНИЕ может являться произвольной последовательностью символов, включая пробелы и обращения к значениям уже определенных переменных. В дальнейшем, в любом месте Makefile, где встретится обращение к переменной-макроопределению, вместо нее будет подставлено ее текущее значение. Обращение к значению переменной в любом месте Makefile выглядит как $(ПЕРЕМЕННАЯ) (скобки обязательны, если имя переменной длиннее одного символа). Значение еще не определенных переменных - пустая строка. С учетом сказанного, можно преобразовать наш Makefile:

OBJS = main.o supp.o

prog: $(OBJS)

 cc -o prog $(OBJS)

$(OBJS): defs.h

Теперь предположим, что к проекту добавился второй заголовочный файл supp.h, который включается только в supp.c. Тогда Makefile увеличится еще на одну строчку:

supp.o: supp.h

Таким образом, один целевой файла может указываться в нескольких целях. При этом полный список зависимостей для файла будет составлен из списков зависимостей всех целей, в которых он участвует, однако создание файла будет производиться только один раз.

В нашем примере мы группировали цели по принципу общих зависимостей, однако существует и альтернативный способ - группировать зависимости по одной цели. В этом случае Makefile будет выглядеть немного иначе, однако его суть не изменится.

OBJS = main.o supp.o

prog: $(OBJS)

 cc -o prog $(OBJS)

main.o: defs.h

supp.o: defs.h supp.h

Обычно Makefile пишется так, чтобы простой запуск make приводил к компиляции проекта, однако, помимо компиляции, Makefile может использоваться и для выполнения других вспомогательных действий, напрямую не связанных с созданием каких-либо файлов. К таким действиям относится очистка проекта от всех результатов компиляции, или вызов процедуры инсталляции проекта в системе. Для выполнения подобных действий в Makefile могут быть указаны дополнительные цели, обращение к которым будет осуществляться указанием их имени аргументом вызова make (например, "make install"). Подобные вспомогательные цели носят название фальшивых, что связанно с отсутствием в проекте файлов, соответствующих их именам. Фальшивая цель может содержать список зависимостей и должна содержать список команд для исполнения. Поскольку фальшивая цель не имеет соответствующего файла в проекте, при каждом обращении к ней make будет пытаться ее построить. Однако, возможно возникновение конфликтной ситуации, когда в каталоге проекта окажется файл с именем, соответствующим имени фальшивой цели. Если для данного имени не определены файловые зависимости, он будет всегда считаться актуальным (up to date) и цель выполняться не будет. Для предотвращения таких ситуаций make поддерживает "встроенную" переменную ".PHONY", которой можно присвоить список имен целей, которые всегда должны считаться фальшивыми.

Теперь можно привести пример полного Makefile, пригодного для работы с проектом prog и принять во внимание некоторые часто применяемые приемы:

OBJS = main.o supp.o

BINS = prog

PREFIX = /usr/local

INSTALL = install

INSOPTS = -s -m 755 -o 0 -g 0

CC = gcc

.PHONY = all clean install

all: $(BINS)

prog: $(OBJS)

 $(CC) -o prog $(OBJS)

main.o: defs.h

supp.o: defs.h supp.h

clean:

 rm -f $(BINS)

 rm -f $(OBJS)

 rm -f \*~

install: all

 for $i in $(BINS) ; do \

 $(INSTALL) $(INSOPTS) $$i $(PREFIX)/bin ; \

 done

Итак, у нас появились три фальшивых цели: all, clean и install. Цель all обычно используется как псевдоним для сборки сложного проекта, содержащего несколько результирующих файлов (исполняемых, разделяемых библиотек, страниц документации и т.п.). Цель clean используется для полной очистки каталога проекта от результатов компиляции и "мусора" - резервных файлов, создаваемых текстовыми редакторами (они обычно заканчиваются символом "~"). Цель install используется для инсталляции проекта в операционной системе (приведенный пример расчитан на установку только исполняемых файлов). Следует отметить повсеместное использование макроопределений - помимо всего, этот прием повышает читабельность. Обратите также внимание на определение переменной $(CC) - это встроенная переменная make и она неявно "сработает" и при компиляции объектных файлов.

**Внутренние макросы**
Мake поддерживает пять внутренних макросов, полезных при написании правил построения целевых файлов:

$\*

Этот макрос является именем файла без расширения из текущей зависимости; вычисляется только для подразумеваемых правил (см. Суффиксы).

$@

Этот макрос заменяется на полное имя целевого файла; вычисляется только для явно заданных зависимостей.

$<

Вычисляется только для подразумеваемых правил или для правила .DEFAULT. Этот макрос заменяется на имя файла, от которого по умолчанию зависит целевой файл. Так, в правиле .c.o макрос $< будет заменен на имя файла с расширением .c. Например, правило для изготовления оптимизированного об ектного файла из файла с расширением .c может быть таким:

 .c.o:

 cc -c -O $\*.c

или

 .c.o:

 cc -c -O $<

$?

Макрос $? можно использовать в явных правилах make-файла. Этот макрос заменяется на список файлов-источников, которые изменялись позднее целевого файла.

$%

Этот макрос применяется только тогда, когда целевой файл указан в виде библ(файл.o), что означает, что он находится в библиотеке библ. В этом случае $@ заменяется на библ (имя архива), а $% заменяется на настоящее имя файла, файл.o.

Четыре из этих макросов имеют альтернативную форму. Если к любому из этих макросов добавлено F, то он заменяется на соответствующее макросу имя файла без имени каталога; если же добавлено D, то макрос заменяется на остальную часть значения первоначального макроса без последнего символа /, то есть на имя каталога. Так, $(@D) соответствует каталогу из $@. Если каталог не указан, то генерируется текущий каталог (.). Только макрос $? не имеет альтернативной формы.

**Библиотеки**

Если целевой файл или имя из списка зависимостей содержит скобки, то оно рассматривается как имя архивной библиотеки, а цепочка символов в скобках - как имя элемента библиотеки. Так, и библ(файл.o), и $(БИБЛ)(файл.o) обозначают библиотеку, содержащую файл.o (предполагается, что макрос БИБЛ был предварительно определен). Выражение $(БИБЛ)(файл1.o файл2.o) недопустимо. Правила обработки библиотечных файлов имеют вид .XX.a, где XX - суффикс, по которому будет получен элемент библиотеки. К сожалению, в текущей реализации требуется, чтобы XX отличался от суффикса элемента библиотеки. Например, нельзя, чтобы библ(файл.o) зависел от файл.o явно. Наиболее общее использование интерфейса работы с библиотеками следующее (предполагается, что исходными являются файлы на языке C):

 lib: lib(file1.o) lib(file2.o) lib(file3.o)

 @echo lib is now up-to-date

 .c.a:

 $(CC) -c $(CFLAGS) $<

 $(AR) $(ARFLAGS) $@ $\*.o

 rm -f $\*.o

Фактически, правило .c.a, приведенное выше, встроено в make. Более интересный, но более ограниченный пример конструкции, поддерживающей работу с библиотеками:

 lib: lib(file1.o) lib(file2.o) lib(file3.o)

 $(CC) -c $(CFLAGS) $(?:.o=.c)

 $(AR) $(ARFLAGS) lib $?

 rm $? @echo lib is now up-to-date

 .c.a:;

Здесь используется режим подстановки расширений макросов. Список $? определен как множество имен об ектных файлов (в библиотеке lib), чьи исходные C-файлы были изменены. Подстановка заменяет .o на .c. (К сожалению, нельзя еще трансформировать в .c~; однако, это может стать возможно в будущем). Заметим также, что запрещается правило .c.a:, создающее каждый об ектный файл один за другим. Эта конструкция значительно ускоряет обновление библиотек, но становится весьма громоздкой, если библиотека содержит как программы на C, так и на ассемблере.

## Отладчик ADB

#### Вызов ADB

Dызываетcz ADB, выполняя **adb** (1) команду. Синтаксис:

**adb [-w] [-k] [-Idir] [-Ppid ] [objfile [corefile]**

Где:

-w Разрешает запись в объектный файл.

-k Сообщает ADB, что объектные и основные файлы являются файлами ядра, так что ADB может исполнять соответствующее управление памятью.

-Idir Определяет каталоги , который содержит команды для ADB.

-Ppid "Принимают" уже процесс выполнения для отладки.

objfile Называет выполнимый объектный файл.

corefile Называет основной загрузочный модуль.

Обычно, вызывая ADB:

**adb a.out core**

Или более просто:

**adb**

Потому что настройка по умолчанию для объектного файла - a.out, и core файл - core.

Поставка знаку "минус" (-) для средств названия(имени) файла " игнорирует этот параметр, " как в:

**adb a.out -**

Чтобы записывать в объектный файл при игнорировании core файла, можна напечатать:

**adb -w a.out -**

Чтобы отлаживать выполняющийся в настоящее время процесс, вызовите ADB, печатая:

**adb -Ppid a.out**

Pid или " идентификатор процесса " может быть получен, используя ps (1) команда.

Потому что ADB прерывает нажатия клавиши, Вы не можете использовать сигнал выхода из, чтобы выйти от ADB. Вы должны использовать явный запрос ADB $q или $Q (или CONTROL D) чтобы выйти от ADB.

#### Использование ADB В интерактивном режиме

Вы работаете в интерактивном режиме с ADB, вводя запросы.

Общая форма для запроса:

[*address*] [*,count*] [*command*] [*modifier*]

ADB поддерживает текущий адрес, называемый "точкой". Этот адрес подобен в функции к текущему указателю в HP-UX редакторе, vi (1). Когда Вы указываете address, ADB устанавливает точку к тому расположению. ADB тогда выполняет команду command count раз. Вы можете вводить address и count как выражения. Вы создаете эти выражения от символов в пределах программы, которую Вы можете проверять и от десятичного числа, восьмеричных, и шестнадцатеричных целых чисел. Вот списки различных операторы для формирующихся выражений.

##### Формирующие выражение Операторы

Оператор Операция

+ Добавление

- Вычитание или Отрицание

\* Умножение

% Целочисленный раздел(деление)

~ Одноместный НЕ

& Поразрядный И

| Поразрядный Содержащий ИЛИ

\* Серия к следующему множителю

ADB исполняет арифметические операции на всех 32 битах.

ADB "помнит" последний(прошлый) набор оснований системы счисления. Вы можете изменять(заменять) текущее основание системы счисления с $o, $d, или $x команды. В течение запуска, заданное по умолчанию основание системы счисления шестнадцатерично. Если Вы изменяете(заменяете) основание системы счисления к десятичному числу, весь последующий ввод и вывод целых чисел интерпретируется как десятичное число, пока другой спецификатор основания системы счисления не используется.

Таблица 2 списка некоторые обычно использовала команды ADB и их значение.

##### Обычно Используемый ADB Команды

Команда Описание

? Печатает содержание от objfile.

/ Печатает содержание от corefile.

= Печатает значение "точки" (.) (адресс) .

: Контрольной точки останова .

$ Разные запросы.

; Разделитель команд.

! Выйти в Шелл.

CONTROL C Заканчивает любую команду ADB.

**Отображение Информации**

Вы можете запросить ADB расположения или в объектном файле или core файле. Запрос (?) показывает содержание объектного файла, в то время как / запрос исследует core файл. Как только Вы инициализируете процесс (использование или:r или команда:e), или ? или / обращаются к расположениям в адресном пространстве выполняющего процесса.

После любого ? или / запросов, Вы можете определить формат, что ADB должен использовать, чтобы печатать эту информацию. Таблица 3 списка некоторые обычно используемые команды формата.

##### Обычно Используемых Команды Формата

Команда Описание

c Один байт как символ.

b Один байт как шестнадцатеричное значение.

x Два байта в шестнадцатеричном.

X Четыре байта в шестнадцатеричном.

d Два байта в десятичном числе.

f Четыре байта в единственном(отдельном) с плавающей запятой.

F Восемь байтов в двойном с плавающей запятой.

i Команда Precision Architecture HEWLETT-PACKARD.

s Символьная строка С нулевым символом в конце.

a Печатать в символической форме.

n Печатать newline.

r Печатать пустое пространство.

^ Резервируют точку.

Например, чтобы печатать первый шестнадцатеричный элемент массива длинных целых чисел, названных ints, Вы напечатали бы запрос:

**ints/X**

Этот запрос устанавливает значение точки к значению таблицы идентификаторов ints. Это также устанавливает значение точечного приращения к четыре. " Точечное приращение " является числом байтов, которые ADB печатает в требуемом формате.

В другом примере, чтобы печатать первые четыре байта как шестнадцатеричный номер после последующих четыре байта как десятичное число, Вы напечатали бы запрос:

ints/XD

В этом случае, ADB все еще устанавливает точку к ints, но точечное приращение - теперь восемь байтов.

Команда newline - специальная команда, которая повторяет предыдущую команду. Команда newline также использует значение точечного приращения, но команда не может всегда иметь значение. В этом контексте, однако, это означает повторять предыдущую команду, используя индекс одних и адрес точки плюс точечное приращение. Так, в этом случае(регистре), команда newline устанавливает точку к ints + 0x8 и печатает два длинных целого числа: первый как шестнадцатеричный номер и второй как десятичное число. Вы можете также повторить команду newline так часто как хотите. Например, Вы могли бы использовать эту методику, чтобы просмотреть разделы памяти.

При использовании этого примера, чтобы иллюстрировать другой пункт, Вы можете печатать первые четыре байта в длинном шестнадцатеричном формате и следующих четырех байтах в байте шестнадцатеричный формат, печатая запрос:

ints/X4b

Как эти примеры, Вы можете предшествовать любой команде формата с десятичным символом повторения.

Кроме того, Вы можете использовать параметр индекса запроса ADB, чтобы повторить, полный формат командует определенным числом раз. Например, чтобы печатать три строки, использующие вышеупомянутый формат, Вы напечатали бы запрос:

ints, 3/X4bn

(n в конце команды печатает перевод каретки, который делает вывод более легким для чтения.)

В этом примере, ADB устанавливает значение точки к ints + 0x10, скорее чем ints. Это случается, потому что каждый раз ADB заново выполняет команду формата, это устанавливает точку к точке плюс точечное приращение. Поэтому, значение точки - значение, которое точка имела в начале последнего(прошлого) выполнения команды формата. Точечное приращение - размер требуемого формата (в этом случае(регистре), восемь байтов). Команда newline в это время установила бы точку в ints + 0x18 и печать только одно повторение(копия) формата, потому что значение индекса сброшено к одному.

Чтобы проверять текущее значение точки, Вы можете напечатать запрос:

. = a

= команда может печатать значение адреса в любом формате.

Вы можете также использовать = команду, чтобы конвертировать(преобразовать) от одного ядра до другого. Например, Вы можете печатать значение "0x32" в восьмеричном, шестнадцатеричном, и представление десятичных чисел, печатая:

0x32 = oxd

ADB "помнит" сложный формат просьбы о каждом из?, /, и = команды. Например, после ввода предыдущего запроса, Вы можете печатать значение "0x64" в восьмеричном, шестнадцатеричном, и представление десятичных чисел, печатая:

0x64 =

Тогда, потому что последний введенный / команда была ints/X4b, Вы можете напечатать:

ints/

Печатать четыре байта в длинном шестнадцатеричном формате и четырех байтах в байте шестнадцатеричный формат.

##### Дополнительные команды печати

Команда Описание

$b Печатают текущие контрольные точки.

$c Печать располагает в стеке след.

$d Основание системы счисления значения по умолчанию Набора, чтобы адресовать параметр.

$e Печатают внешние переменные.

$f Регистраторы С плавающей точкой как единственная(отдельная) точность.

$F Регистраторы С плавающей точкой как двойная точность.

$m Печатают карты сегмента ADB.

$r Печатают общих регистраторов.

$s Смещение Набора для соответствия символа.

$v Печатают ADB переменные.

$w Вывод Набора выравнивает ширину.

**Рекомендуеться также посмтотреть на лучший Отладчик под Unix:**

*http://www.kiarchive.ru/pub/gnu/gnu-mirror/Manuals/ddd/html\_mono/ddd.html*

## Отладчик XDB

**XDB –** отладчик для отлаживания програм написаных на языках C, HP FORTRAN, HP Pascal, and C++ и понимает следующие команды:

**xdb [-d dir] [-r file] [-R file] [-p file] [-P process\_ID] [-L] [-l library]**

 **[-i file] [-o file] [-e file] [-S num] [-s] [objectfile [corefile]]**

где

-d dir определяет дополнительный каталог где размещены исходный коды

-r file определяет рекордный файл

-R file определяет файл restore, который был определен перед –p но после –r опции

-p file определяет файл воспоизведения действий (playback)

-P process\_ID Определяет process\_ID до которого желаем «присоедениться» чтоб отправить в

режим отладки

 -L определяет строчно-ориентированный интерфейс.

 -l library определяет библиотеку (общедоступную) до которой желаете подсоедениться

-i file переопределяет поток ввода в файл или в устройство

-o file переопределяет поток вывода в файл или в устройство

-e file переопределяет поток вывода ошибок в файл или в устройство

-S num устанавливает размер кеширования строки (по умолчанию 1024 что есть минимальный) –s определяет все библиотеки (общедоступный) которые использует програма

Размер екрана будет в зависимости от переменой окружения TERM или можно установить используя переменный LINES и COLUMNS.

При запуске xdb имеет 3 окна:

* Окно кода (содержит исходный код)
* Окно информации (содержит значения параметров и прочее)
* Командное окно, окно упраления

# Примеры команд

r запускает програму с параметрами

R запускет програму без параметров

s пошаговый запусr (входит в функции)

S пошаговый запусr (не входить в функции)

к убить процесс

q выйти из отладчика

с продолжить выполнения програмы (continue)

v 11 посмотреть 11 строку кода

+5 посмотрить на 5 строк ниже

-5 ---------------------------- выше

v my\_function показать функцию

v test1.c просмотреть файл test1.c

v test1.c:40 просмотреть файл test1.c на 40 строке

V посмотреть текущуй стек (сотояний вызовов)

V 2 посмотреть текущей стек на 2 уровня глубже

w 12 установить размер окна кода на 12

td показать код на asssembly языке (что б возвратиться к коду то еще раз нужно набрать td)

ts показать код и asssembly

s 6 запустить 6 шагов

/ n=4 ищет код n=4 (снизу)

? n=4 ищет код n=4 (сверху)

b 42 устанавливает точку останова в 42 строке

b 32 \4 в 32 строке будет останавливаться програма 4 раза

bp устанавливает точку останова на точке входа в програму

bp my\_funс устанавливает точку останова на точке входа в процедуру my\_func

lb просмотреть точки останова

db 2 удалить 2ю точку остнаова

db \* удалить все точки останова

p count просмотреть значение count

p count\x просмотреть в шестнацатиричном виде

p num\D просмотреть переменую в long типе как десятеричную

p . показать преведущее значение

p \*(&.+42) показать значение на 42 байта дальше от преведуще-показаного значения

p my\_struct показывает даные в структуре

p my\_struct.name показывает значение в структуры поля name

p \*ptr значения указателя

p+ (p-) показывет следущий(преведущий) елемент

p num=num+20 увеличить значения на 20

t показывает стек

**Справочная таблица команд и флагов XDB**

**h [topic]** Print commands/syntaxes related to this topic.

 Help without a topic prints the complete help text. Available topics

 include command names (short form) which print the syntax for and a

 terse description of the command. Other topics are:

 assert assertions; macro macros;

 bpset set breakpoints; misc other commands, etc.;

 bpstat view & modify breakpoints; options xdb command line;

 C++ C++ features; proc procedure related;

 cmdlist command list features; record write & use log files;

 control process control; register registers;

 data view & modify data; screen window modes;

 disasm disassembly mode; signal signal handling;

 formats format specifiers; state global state switches;

 help this description; source view source;

 list list various items; trace trace stack or proc(s);

 locations location syntax; variables variable syntax.

**Process control:**

r [arguments] Run child process with arguments.

R Run child process with no arguments.

c [location] Continue from breakpoint with no signal, set temporary breakpoint at location.

C [location] Continue with current signal, set temporary breakpoint at location.

s [number] Single step, follow procedure calls.

S [number] Single step, step over procedure calls.

g (line | #label) Go to line in current procedure.

g (+|-) [lines] Go forward/back 1 or given number of lines.

k Kill child process, if any.

**Setting breakpoints:**

b [location] [\count] [commands] Set breakpoint.

ba [address] [\count] [commands] Set breakpoint at code address.

bb [depth] [\count] [commands] Set breakpoint at procedure beginning.

bi expr.proc [\count] [commands] Set an instance breakpoint.

bi [-c|-C] expr [commands] Set an instance breakpoint.

bp [commands] Set procedure breakpoints.

bpc [-c|-C] class [commands] Set a class breakpoint.

bpo [[class]::]proc [commands] Set breakpoints on overloaded functions.

bpt [commands] Set procedure trace breakpoints.

bpx [commands] Set procedure exit breakpoints.

bt [(depth | proc)] [\count] [commands] Trace procedure.

bu [depth] [\count] [commands] Set up-level breakpoint.

bx [depth] [\count] [commands] Set breakpoint at procedure exit.

bpg [commands] Set paragraph breakpoints. (MPE-only)

tpg [commands] Set paragraph trace breakpoints. (MPE-only)

txc Toggle the exception stop-on-catch state.

txt Toggle the exception stop-on-throw state.

**View and modify breakpoint status:**

lb List all breakpoints.

lx List exception stop-on-throw and -catch state.

db [number | \*] Delete one or all breakpoints.

dp Delete procedure breakpoints.

Dpx Delete procedure exit breakpoints.

Dpt Delete procedure trace breakpoints.

dpg Delete paragraph [trace] breakpoints.(MPE-only)

ab [number | \*] Activate one or all breakpoints.

sb [number | \*] Suspend one or all breakpoints.

tb Toggle overall breakpoints state.

abc commands Global breakpoint commands.

dbc Delete global breakpoint commands.

bc number expr Set a breakpoint count.

xcc commands Define the stop-on-catch command-list.

xtc commands Define the stop-on-throw command-list.

i expr {commands} [{commands}] Conditionally execute commands. (Also: if)

{ } Group commands.

; Separate commands.

Q Quiet breakpoint reporting.

 "any string" Print string.

**Source viewing:**

L Show current location and its source line.

v [location] View source at location in source window.

va [address] View address in disassembly window.

V [depth] View procedure at depth in source window.

top View procedure at top of stack.

up [number] View procedure number levels higher in stack.

down [number] View procedure number levels lower in stack.

+[number] Move forward in sourcefile.

-[number] Move backward in sourcefile.

/[string] Search forwards in sourcefile for string.

?[string] Search backwards for string.

n Repeat previous search.

N Repeat previous search in opposite direction.

apm old\_path [new\_path] Add (prefix) path map for source files.

dpm [index | \*] Delete path map(s) for source files.

lpm List path maps in order of precedence.

D "dir" Add a directory search path for source files.

ld List all directories.

lf [string] List all (or matching) files.

lsl List all shared libraries.

lp [[class]::][string] List all (or matching) procedures.

lo [[class]::][string] List all (or matching) overloaded functions.

lcl [string] List all (or matching) classes.

lct [string] List all (or matching) class templates.

ltf [string] List all (or matching) function expansions.

lft [string] List all (or matching) function templates.

**View and modify data:**

p expr [\format] Print value of expression using format.

p expr?format Print address of expression using format.

p -[\format] Print value of prev memory location using format.

p +[\format] Print value of next memory location using format.

p class:: Print static members of class.

l [[class]::][proc[:depth]] List all parameters and locals of proc.

t [depth] Trace stack.

T [depth] Trace stack and show local variables.

tst Toggle stub visibility. (PA-RISC only)

lr [string] List all (or matching) registers.

lc [string] List all (or matching) commons. (PA-RISC only)

lg [string] List all (or matching) globals.

ls [string] List all (or matching) special variables.

mm [string] Show memory-map of all (or matching) loaded shared -libraries.

f ["printf-style-format"] Set address printing format.

disp item [\format] Display Cobol data item value using format.(MPE-only)

move val to item Move value "val" to cobol data item "item" (MPE-only)

pq <<same as p>> Print quietly. Evaluate without printing.

ll [string] List all (or matching) labels.

lz List all signals.

z [number] [i][r][s][Q] Toggle flags (ignore, report, stop, Quiet) for signal.

**Screen modes:**

am Activate more (turn on pagination).

sm Suspend more (turn off pagination).

w number Set size of source window.

td Toggle disassembly mode.

ts Toggle split-screen mode.

fr Display floating point registers.

gr Display general registers.

tf Toggle float register display precision (PA-RISC only).

sr Display special registers. (PA-RISC only)

u Update screen.

U Refresh source & location windows on screen.

+r Scroll floating point registers forward.

-r Scroll floating point registers backward.

**Assertions:**

a commands Create a new assertion with a command list.

aa (number | \*) Activate one or all assertions.

da (number | \*) Delete one or all assertions.

la List all assertions.

sa (number | \*) Suspend one or all assertions.

ta Toggle overall assertions state.

x [expr] Exit assertion mode, possibly aborting the assertion command list.

**Macros:**

def name [replacement-text] Define a macro name.

lm [string] List all (or matching) macros.

tm Toggle the macro substitution mechanism.

undef (name | \*) Remove the macro definition for name or all.

**Record and playback:**

tr [@] Toggle the record [record-all] mechanism.

< file Playback from file.

<< file Playback from file with single stepping.

> file Record commands to file.

>> file Append commands to file.

>@ file Record-all debugger commands & output to file.

>>@ file Append all debugger commands & output to file.

 *">>" is equivalent to ">" for the next four commands.*

> Show status of current recording file.

>@ Show status of current record-all file.

>(t | f | c) Turn recording on (t), or off (f), or close the recording file (c).

>@(t | f | c) Turn record-all on (t), or off (f), or close the record-all file (c).

**Misc:**

ss file Save (breakpoint, macro, assertion) state.

tc Toggle case sensitivity in searches.

<carriage-return> Repeat previous command.

~ Repeat previous command.

! [command-line] Execute shell (with or without commands).

q Quit debugger.

$addr Unary operator, address of object.

$sizeof Unary operator, size of object.

$in Unary boolean operator, execution in procedure.

# [text] A comment.

I Print debugger status.

M [(t | c) [expr [; expr ...]]] Print or set (text or core) maps.

tM Toggle between default and modifiable core maps.

## VARIABLES

var Search current procedure and globals.

class::var Search class for variable.

[[class]::]proc:[class::]var Search procedure for variable.

[[class]::]proc:depth:[class::]var Search procedure at depth on stack.

:var or ::var Search for global variable only.

. Shorthand for last thing you looked at.

$var Define or use special variable.

$result Return value of last cmd line procedure call.

$signal Current child process signal number.

$lang Current language for expression evaluation.

$depth Default stack depth for local variables.

$print Display mode for character data.

$line Current source line number.

$malloc Debugger memory allocation (bytes).

$step Instr. count in non-debug before free-run.

$cplusplus C++ feature control flags.

$regname Hardware registers.

$fpa Treat fpa sequence as one instruction.(S300 only)

$fpa\_reg Address register for fpa sequences. (S300 only)

## LOCATIONS

line source line & code address (if any)

#label "

file[:line] "

[file:]proc[:proc[...]][:line|#label] "

[class]::proc[:line|#label] "

proc#line code address (if any)

[class]::proc#line "

name@shared\_lib Address of name in shared library

## FORMATS

 A format has the form [count]formchar[size]. formchar's are:

 a String at address.

(b | B) Byte in decimal (either way).

(c | C) (Wide) character.

(d | D) (Long) decimal.

(e | E) E floating point notation (as double).

(f | F) F floating point notation (as double).

(g | G) G floating point notation (as double).

 i Machine instruction (disassembly).

(k | K) Formatted structure display (with base classes).

 n "Normal" format, based on type.

(o | O) (Long) octal.

 p Print name of procedure containing address.

(r | R) Print template of object (with base classes).

 s String from pointer.

 S Formatted structure display.

(t | T) Print type of object (with base classes).

(u | U) (Long) unsigned decimal.

(w | W) Wide character string (at address).

(x | X) (Long) hexadecimal.

(z | Z) (Long) binary.

 Size can be a number or one of the following:

 b 1 byte (char)

 s 2 bytes (short)

 l 4 bytes (long)

 D 8 bytes (double - floating point formats only)

 L 16 bytes (long double - floating point only)

# Системные вызовы и взаимодействие с UNIX.

В этой главе речь пойдет о процессах. Скомпилированная программа хранится на диске как обычный нетекстовый файл. Когда она будет загружена в память компьютера и начнет выполняться - она станет **процессом**.

*UNIX* - многозадачная система (мультипрограммная). Это означает, что одновременно может быть запущено много процессов. Процессор выполняет их в режиме **разделения** **времени** - выделяя по очереди квант времени одному процессу, затем другому, третьему... В результате создается впечатление **параллельного** выполнения всех процессов (на многопроцессорных машинах параллельность истинная). Процессам, ожидающим некоторого события, время процессора не выделяется. Более того, "спящий" процесс может быть временно откачан (т.е. скопирован из памяти машины) на диск, чтобы освободить память для других процессов. Когда "спящий" процесс дождется события, он будет "разбужен" системой, переведен в ранг "готовых к выполнению" и, если был откачан будет возвращен с диска в память (но, может быть, на другое место в памяти!). Эта процедура носит название "своппинг" (**swapping**).

Можно запустить несколько процессов, выполняющих программу из **одного и того же** файла; при этом все они будут (если только специально не было предусмотрено иначе) независимыми друг от друга. Так, у каждого пользователя, работающего в системе, имеется свой собственный процесс-**интерпретатор команд** (своя копия), выполняющий программу из файла */bin/csh* (или */bin/sh*).

Процесс представляет собой изолированный "мир", общающийся с другими "мирами" во Вселенной при помощи:

a) Аргументов функции *main*:

 void *main*(int **argc**, char \***argv**[], char \***envp**[]);

Если мы наберем команду

 $ *a.out* **a1 a2 a3**

то функция *main* программы из файла **a**.**out** вызовется с

 **argc** = 4 /\* количество аргументов \*/

 **argv**[0] = "a.out" **argv**[1] = "a1"

 **argv**[2] = "a2" **argv**[3] = "a3"

 **argv**[4] = *NULL*

По соглашению **argv**[0] содержит имя выполняемого файла из которого загружена эта программа[\*](#b1).

b) Так называемого "окружения" (или "среды") char \***envp**[], продублированного также в предопределенной переменной

 extern char \*\**environ*;

Окружение состоит из строк вида

 "**ИМЯПЕРЕМЕННОЙ***=***значение**"

Массив этих строк завершается *NULL* (как и **argv**). Для получения значения переменной с именем **ИМЯ** существует стандартная функция

 char \**getenv*( char \***ИМЯ** );

Она выдает либо **значение**, либо *NULL* если переменной с таким именем нет.

c) Открытых файлов. По умолчанию (неявно) всегда открыты 3 канала:

 ВВОД В Ы В О Д

 *FILE* \* **stdin stdout stderr**

 соответствует **fd** 0 1 2

 связан с клавиатурой дисплеем

 #include <stdio.h>

 main(ac, av) char \*\*av; {

 *execl*("/bin/sleep", "Take it easy", "1000", NULL);

 }

Эти каналы достаются процессу "в наследство" от запускающего процесса и связаны с дисплеем и клавиатурой, если только не были перенаправлены. Кроме того, программа может сама явно открывать файлы (при помощи *open*, *creat*, *pipe*, *fopen*). Всего программа может одновременно открыть до определенное количество файлов в зависииости от настройки ядра.

d) Процесс имеет уникальный номер, который он может узнать вызовом

 int **pid** = *getpid*();

а также узнать номер "родителя" вызовом

 int **ppid** = *getppid*();

Процессы могут по этому номеру посылать друг другу сигналы:

 *kill*(**pid** /\* кому \*/, **sig** /\* номер сигнала \*/);

и реагировать на них

 *signal* (**sig** /\*по сигналу\*/, **f** /\*вызывать f(sig)\*/);

e) Существуют и другие средства коммуникации процессов: семафоры, сообщения, общая память, сетевые коммуникации.

f) Существуют некоторые другие параметры (контекст) процесса: например, его текущий каталог, который достается в наследство от процесса-"родителя", и может быть затем изменен системным вызовом

 *chdir*(char \***имя**\_**нового**\_**каталога**);

У каждого процесса есть свой **собственный** текущий рабочий каталог. К "прочим" характеристикам отнесем также: управляющий терминал; группу процессов (**pgrp**); идентификатор (номер) владельца процесса (**uid**), идентификатор группы владельца (**gid**), реакции и маски, заданные на различные сигналы; и.т.п.

g) Издания других запросов (системных вызовов) к операционной системе ("богу") для выполнения различных "внешних" операций.

h) Все остальные действия происходят внутри процесса и никак не влияют на другие процессы и устройства ("миры"). В частности, один процесс НИКАК не может получить доступ к памяти другого процесса, если тот не позволил ему это явно (механизм **shared memory**); адресные пространства процессов независимы и изолированы (равно и пространство ядра изолировано от памяти процессов).

Операционная система выступает в качестве **коммуникационной среды**, связывающей "миры"-процессы, "миры"-внешние устройства (включая терминал пользователя); а также в качестве распорядителя ресурсов "Вселенной", в частности - времени (по очереди выделяемого активным процессам) и пространства (в памяти компьютера и на дисках).

Уже неоднократно упоминали "системные вызовы". Что же это такое? С точки зрения Си-программиста - это обычные функции. В них передают аргументы, они возвращают значения. Внешне они ничем не отличаются от написанных нами или библиотечных функций и вызываются из программ одинаковым с ними способом.

С точки же зрения реализации - есть глубокое различие. Тело функции-сисвызова расположено не в нашей программе, а в резидентной (т.е. постоянно находящейся в памяти компьютера) управляющей программе, называемой **ядром операционной системы**[\*](#b2).

Поведение всех программ в системе вытекает из поведения системных вызовов, которыми они пользуются. Даже то, что *UNIX* является многозадачной системой, непосредственно вытекает из наличия системных вызовов *fork*, *exec*, *wait* и спецификации их функционирования! То же можно сказать про язык Си - мобильность программы зависит в основном от набора используемых в ней **библиотечных функций** (и, в меньшей степени, от диалекта самого языка, который должен удовлетворять **стандарту** на язык Си). Если две разные системы предоставляют все эти функции (которые могут быть по-разному реализованы, но должны делать одно и то же), то программа будет компилироваться и работать в обоих системах, более того, работать в них **одинаково**.

Сам термин "системный вызов" как раз означает "вызов системы для выполнения действия", т.е. вызов функции в ядре системы. Ядро работает в **привелегированном режиме**, в котором имеет доступ к некоторым системным таблицам[\*](#b3), регистрам и портам внешних устройств и диспетчера памяти, к которым обычным программам доступ аппаратно запрещен (в отличие от *MS DOS*, где все таблицы ядра доступны пользовательским программам, что создает раздолье для вирусов). Системный вызов происходит в 2 этапа: сначала в пользовательской программе вызывается библиотечная функция-"корешок", тело которой написано на ассемблере и содержит команду генерации **программного прерывания**. Это - главное отличие от нормальных Си-функций - вызов по прерыванию. Вторым этапом является реакция ядра на прерывание:

1. переход в привелегированный режим;
2. разбирательство, КТО обратился к ядру, и подключение *u-area* этого процесса к адресному пространству ядра (**context switching**);
3. извлечение аргументов из памяти запросившего процесса;
4. выяснение, ЧТО же хотят от ядра (один из аргументов, невидимый нам - это **номер** системного вызова);
5. проверка корректности остальных аргументов;
6. проверка прав процесса на допустимость выполнения такого запроса;
7. вызов тела требуемого системного вызова - это обычная Си-функция в ядре;
8. возврат ответа в память процесса;
9. выключение привелегированного режима;
10. возврат из прерывания.

Во время системного вызова (шаг 7) процесс может "заснуть", дожидаясь некоторого события (например, нажатия кнопки на клавиатуре). В это время ядро передаст управление другому процессу. Когда наш процесс будет "разбужен" (событие произошло) - он продолжит выполнение шагов системного вызова.

Большинство системных вызовов возвращают в программу в качестве своего значения признак успеха: 0 - все сделано, (-1) - сисвызов завершился неудачей; либо некоторое содержательное значение при успехе (вроде дескриптора файла в *open*(), и (-1) при неудаче. В случае неудачного завершения в предопределенную переменную *errno* заносится номер ошибки, описывающий причину неудачи (коды ошибок предопределены, описаны в include-файле <**errno**.**h**> и имеют вид *Eчтото*). Заметим, что при УДАЧЕ эта переменная просто **не изменяется** и может содержать любой мусор, поэтому проверять ее имеет смысл лишь в случае, если ошибка действительно произошла:

 #include <**errno**.**h**> /\* коды ошибок \*/

 extern int *errno*;

 extern char \**sys***\_***errlist*[];

 int **value**;

 if((**value** = *sys***\_***call*(...)) < 0 ){

 printf("Error:%s(%d)\n", *sys***\_***errlist*[*errno*],

 *errno* );

 *exit*(**errno**); /\* принудительное завершение программы \*/

 }

Предопределенный массив *sys***\_***errlist*, хранящийся в стандартной библиотеке, содержит строки-расшифровку смысла ошибок (по-английски). Посмотрите описание функции *per-* *ror*().

## Время в UNIX.

Ниже приведены примеры как узнавать время:

. В системе *UNIX* время обрабатывается и хранится именно в виде числа секунд; в частности текущее астрономическое время можно узнать системным вызовом

 #include <**sys**/**types**.**h**>

 #include <**time**.**h**>

 *time***\_***t* **t** = *time*(NULL); /\* *time*(&**t**); \*/

Функция

 struct *tm* \***tm** = *localtime*( &**t** );

разлагает число секунд на отдельные составляющие, содержащиеся в int-полях структуры:

 *tm***\_***year* год (надо прибавлять 1900)

 *tm***\_***yday* день в году 0..365

 *tm***\_***mon* номер месяца 0..11 (0 - Январь)

 *tm***\_***mday* дата месяца 1..31

 *tm***\_***wday* день недели 0..6 (0 - Воскресенье)

 *tm***\_***hour* часы 0..23

 *tm***\_***min* минуты 0..59

 *tm***\_***sec* секунды 0..59

Номера месяца и дня недели начинаются с нуля, чтобы вы могли использовать их в качестве индексов:

 char \***months**[] = { "Январь", "Февраль", ..., "Декабрь" };

 printf( "%s\n", **months**[ **tm**->tm\_mon ] );

Часто бывает нужда передавать значения времени в одной строке

Вот пример программы которая преобразовывает в ремя в такой формат:

/\* Mon Jun 12 14:31:26 2000 \*/

#include <stdio.h>

#include <time.h>

main() { /\* команда *date* \*/

time\_t **t** = *time*(NULL);

char \***s** = *ctime*(&t);

printf("%s", **s**);

}

*UNIX*-машины имеют встроенные таймеры (как правило несколько) с довольно высоким разрешением. Некоторые из них могут использоваться как "будильники" с обратным отсчетом времени: в таймер загружается некоторое значение; таймер ведет обратный отсчет, уменьшая загруженный счетчик; как только это время истекает - посылается сигнал процессу, загрузившему таймер.

Вот как, к примеру, выглядит функция задержки в микросекундах (миллионных долях секунды). Примечание: эту функцию не следует использовать вперемежку с функциями *sleep* и *alarm*.

 #include <sys/types.h>

 #include <signal.h>

 #include <sys/time.h>

 void **do\_nothing**() {}

 /\* Задержка на **usec** миллионных долей секунды (микросекунд) \*/

 void *usleep*(unsigned int **usec**) {

 struct itimerval new, old;

 /\* struct itimerval содержит поля:

 struct timeval it\_interval;

 struct timeval it\_value;

 Где struct timeval содержит поля:

 long tv\_sec; -- число целых секунд

 long tv\_usec; -- число микросекунд

 \*/

 struct sigaction new\_vec, old\_vec;

 if (usec == 0) return;

 /\* Поле tv\_sec содержит число целых секунд.

 Поле tv\_usec содержит число микросекунд.

 it\_value - это время, через которое В ПЕРВЫЙ раз

 таймер "прозвонит",

 то есть пошлет нашему процессу

 сигнал SIGALRM.

 Время, равное нулю, немедленно остановит таймер.

 it\_interval - это интервал времени, который будет загружаться

 в таймер после каждого "звонка"

 (но не в первый раз).

 Время, равное нулю, остановит таймер

 после его первого "звонка".

 \*/

 new.it\_interval.tv\_sec = 0;

 new.it\_interval.tv\_usec = 0;

 new.it\_value.tv\_sec = usec / 1000000;

 new.it\_value.tv\_usec = usec % 1000000;

 /\* Сохраняем прежнюю реакцию на сигнал SIGALRM в old\_vec,

 заносим в качестве новой реакции do\_nothing()

 \*/

 new\_vec.sa\_handler = do\_nothing;

 *sigemptyset*(&new\_vec.sa\_mask);

 new\_vec.sa\_flags = 0;

  *sigaction*(SIGALRM, &new\_vec, &old\_vec);

 /\* Загрузка интервального таймера значением new, начало отсчета.

 \* Прежнее значение спасти в old.

 \* Вместо &old можно также NULL - не спасать.

 \*/

 ***setitimer***(*ITIMER***\_***REAL*, &new, &old);

 /\* Ждать прихода сигнала SIGALRM \*/

 *sigpause*(SIGALRM);

 /\* Восстановить реакцию на SIGALRM \*/

 *sigaction*(SIGALRM, &old\_vec, (struct sigaction \*) 0);

 *sigrelse*(SIGALRM);

 /\* Восстановить прежние параметры таймера \*/

 ***setitimer***(ITIMER\_REAL, &old, (struct itimerval \*) 0);

 }

**Пример оспользования интервалов**

 #include <stdio.h>

 #include <unistd.h> /\* \_SC\_CLK\_TCK \*/

 #include <signal.h> /\* SIGALRM \*/

 #include <sys/time.h> /\* не используется \*/

 #include <sys/times.h> /\* struct tms \*/

 struct tms tms\_stop, tms\_start;

 **clock\_t** real\_stop, real\_start;

 **clock\_t** HZ; /\* число ticks в секунде \*/

 /\* Засечь время момента старта процесса \*/

 void hello(void){

 real\_start = **times**(&tms\_start);

 }

 /\* Засечь время окончания процесса \*/

 void bye(int n){

 real\_stop = **times**(&tms\_stop);

 #ifdef CRONO

 /\* Разность времен \*/

 tms\_stop.tms\_utime -= tms\_start.tms\_utime;

 tms\_stop.tms\_stime -= tms\_start.tms\_stime;

 #endif

 /\* Распечатать времена \*/

 printf("User time = %g seconds [%lu ticks]\n",

 tms\_stop.tms\_utime / (double)HZ, tms\_stop.tms\_utime);

 printf("System time = %g seconds [%lu ticks]\n",

 tms\_stop.tms\_stime / (double)HZ, tms\_stop.tms\_stime);

 printf("Children user time = %g seconds [%lu ticks]\n",

 tms\_stop.tms\_cutime / (double)HZ, tms\_stop.tms\_cutime);

 printf("Children system time = %g seconds [%lu ticks]\n",

 tms\_stop.tms\_cstime / (double)HZ, tms\_stop.tms\_cstime);

 printf("Real time = %g seconds [%lu ticks]\n",

 (real\_stop - real\_start) / (double)HZ, real\_stop - real\_start);

 exit(n);

 }

 /\* По сигналу SIGALRM - завершить процесс \*/

 void onalarm(int nsig){

 printf("Выход #%d ================\n", getpid());

 bye(0);

 }

 /\* Порожденный процесс \*/

 void dochild(int n){

 hello();

 printf("Старт #%d ================\n", getpid());

 signal(SIGALRM, onalarm);

 /\* Заказать сигнал SIGALRM через 1 + n\*3 секунд \*/

 alarm(1 + n\*3);

 for(;;){} /\* зациклиться в user mode \*/

 }

 #define NCHLD 4

 int main(int ac, char \*av[]){

 int i;

 /\* Узнать число тиков в секунде \*/

 HZ = **sysconf**(\_SC\_CLK\_TCK);

 setbuf(stdout, NULL);

 hello();

 for(i=0; i < NCHLD; i++)

 if(fork() == 0)

 dochild(i);

 while(wait(NULL) > 0);

 printf("Выход MAIN =================\n");

 bye(0);

 return 0;

 }

## Сигналы.

Процессы в *UNIX* используют много разных механизмов взаимодействия. Одним из них являются **сигналы**.

Сигналы - это **асинхронные** события. Что это значит? Сначала объясним, что такое **синхронные** события: я два раза в день подхожу к почтовому ящику и проверяю - нет ли в нем почты (событий). Во-первых, я произвожу **опрос** - "нет ли для меня события?", в программе это выглядело бы как вызов функции опроса и, может быть, ожидания события. Во-вторых, я знаю, что почта **может** ко мне прийти, поскольку я подписался на какие-то газеты. То есть я предварительно **заказывал** эти события.

Схема с синхронными событиями очень распространена. Кассир сидит у кассы и ожидает, пока к нему в окошечко не заглянет клиент. Поезд периодически проезжает мимо светофора и останавливается, если горит красный. Функция Си пассивно "спит" до тех пор, пока ее не вызовут; однако она всегда готова выполнить свою работу (обслужить клиента). Такое ожидающее заказа (события) действующее лицо называется **сервер**. После выполнения заказа сервер вновь переходит в состояние ожидания вызова. Итак, если событие **ожидается** в специальном месте и в определенные моменты времени (издается некий вызов для ОПРОСА) - это синхронные события. Канонический пример - функция *gets*, которая задержит выполнение программы, пока с клавиатуры не будет введена строка. Большинство ожиданий внутри системных вызовов - **синхронны**. Ядро ОС выступает для программ пользователей в роли сервера, выполняющего сисвызовы (хотя и не только в этой роли - ядро иногда предпринимает и активные действия: передача процессора другому процессу через определенное время (режим разделения времени), убивание процесса при ошибке, и.т.п.).

Сигналы - это асинхронные события. Они приходят неожиданно, в любой момент времени - вроде телефонного звонка. Кроме того, их не требуется заказывать - сигнал процессу может поступить совсем без повода. Аналогия из жизни такова: человек сидит и пишет письмо. Вдруг его окликают посреди фразы - он отвлекается, отвечает на вопрос, и вновь продолжает прерванное занятие. Человек **не ожидал** этого оклика (быть может, он **готов** к нему, но он не озирался по сторонам специально). Кроме того, сигнал мог поступить когда он писал 5-ое предложение, а мог - когда 34-ое. Момент времени, в который произойдет прерывание, не фиксирован.

Сигналы имеют **номера**, причем их количество ограничено - есть определенный список допустимых сигналов. Номера и мнемонические имена сигналов перечислены в includeфайле <**signal**.**h**> и имеют вид *SIG***нечто**. Допустимы сигналы с номерами 1..*NSIG*-1, где *NSIG* определено в этом файле. При получении сигнала мы узнаем его номер, но не узнаем никакой иной информации: ни **от кого** поступил сигнал, ни что от нас хотят. Просто "звонит телефон". Чтобы получить дополнительную информацию, наш процесс должен взять ее из другого известного места; например - прочесть заказ из некоторого файла, об имени которого все наши программы заранее "договорились". Сигналы процессу могут поступать тремя путями:

1. От другого процесса, который **явно** посылает его нам вызовом

 *kill*(**pid**, **sig**);

где **pid** - идентификатор (номер) процесса-получателя, а **sig** - номер сигнала. Послать сигнал можно только родственному процессу - запущенному тем же пользователем.

1. От операционной системы. Система может посылать процессу ряд сигналов, сигнализирующих об ошибках, например при обращении программы по несуществующему адресу или при ошибочном номере системного вызова. Такие сигналы обычно прекращают наш процесс.
2. От пользователя - с клавиатуры терминала можно нажимом некоторых клавиш послать сигналы *SIGINT* и *SIGQUIT*. Собственно, сигнал посылается **драйвером терминала** при получении им с клавиатуры определенных символов. Так можно прервать зациклившуюся или надоевшую программу.

Процесс-получатель должен как-то отреагировать на сигнал. Программа может:

1. проигнорировать сигнал (не ответить на звонок);
2. перехватить сигнал (снять трубку), выполнить какие-то действия, затем продолжить прерванное занятие;
3. быть убитой сигналом (звонок был подкреплен броском гранаты в окно);

В большинстве случаев сигнал по умолчанию убивает процесс-получатель. Однако процесс может изменить это умолчание и задать свою реакцию явно. Это делается вызовом *signal*:

 #include <**signal**.**h**>

 void (\**signal*(int **sig**, void (\***react**)() )) ();

Параметр **react** может иметь значение:

*SIG***\_***IGN*

сигнал **sig** будет отныне игнорироваться. Некоторые сигналы (например *SIGKILL*) невозможно перехватить или проигнорировать.

*SIG***\_***DFL*

восстановить реакцию по умолчанию (обычно - смерть получателя). **имя**\_**функции** Например

 void *fr*(**gotsig**){ ..... } /\* обработчик \*/

 ... *signal* (**sig**, *fr*); ... /\* задание реакции \*/

Тогда при получении сигнала **sig** будет вызвана функция *fr*, в которую в качестве аргумента **системой** будет передан номер сигнала, действительно вызвавшего ее **gotsig**==**sig**. Это полезно, т.к. можно задать одну и ту же функцию в качестве реакции для **нескольких** сигналов:

 ... *signal* (**sig1**, *fr*); *signal*(**sig2**, *fr*); ...

После возврата из функции *fr*() программа продолжится с прерванного места. **Перед** вызовом функции-обработчика реакция автоматически сбрасывается в реакцию по умолчанию *SIG***\_***DFL*, а после выхода из обработчика снова восстанавливается в *fr*. Это значит, что во время работы функции-обработчика может прийти сигнал, который **убьет** программу.

Приведем список **некоторых** сигналов; полное описание посмотрите в документации. Колонки таблицы: *G* - может быть перехвачен; *D* - по умолчанию убивает процесс (*k*), игнорируется (*i*); *C* - образуется дамп памяти процесса: файл *core*, который затем может быть исследован отладчиком *adb*; *F* - реакция на сигнал сбрасывается; *S* - посылается обычно системой, а не явно.

 сигнал G D C F S смысл

 *SIGTERM* + k - + - завершить процесс

 *SIGKILL* - k - + - убить процесс

 *SIGINT* + k - + - прерывание с клавиш

 *SIGQUIT* + k + + - прерывание с клавиш

 *SIGALRM* + k - + + будильник

 *SIGILL* + k + - + запрещенная команда

 *SIGBUS* + k + + + обращение по неверному

 *SIGSEGV* + k + + + адресу

 *SIGUSR1*, *USR2* + i - + - пользовательские

 *SIGCLD* + i - + + смерть потомка

1. Сигнал *SIGILL* используется иногда для эмуляции команд с плавающей точкой, что происходит примерно так: при обнаружении "запрещенной" команды для отсутствующего процессора "плавающей" арифметики аппаратура дает прерывание и система посылает процессу сигнал *SIGILL*. По сигналу вызывается функция-эмулятор плавающей арифметики (подключаемая к выполняемому файлу автоматически), которая и обрабатывает требуемую команду. Это может происходить много раз, именно поэтому реакция на этот сигнал **не сбрасывается**.
2. *SIGALRM* посылается в результате его заказа вызовом *alarm*() (см. ниже).
3. Сигнал *SIGCLD* посылается процессу-родителю при выполнении процессом-потомком сисвызова *exit* (или при смерти вследствие получения сигнала). Обычно процессродитель при получении такого сигнала (если он его заказывал) реагирует, выполняя в обработчике сигнала вызов *wait* (см. ниже). По-умолчанию этот сигнал игнорируется.
4. Реакция *SIG***\_***IGN* **не** сбрасывается в *SIG***\_***DFL* при приходе сигнала, т.е. сигнал игнорируется постоянно.
5. Вызов *signal* возвращает старое значение реакции, которое может быть запомнено в переменную вида void (\***f**)(); а потом восстановлено.
6. Синхронное ожидание (сисвызов) может иногда быть прервано асинхронным событием (сигналом), но об этом ниже.

## Деления просесса

Системный вызов *fork*() (вилка) создает **новый** процесс: **копию** процесса, издавшего вызов. Отличие этих процессов состоит только в возвращаемом *fork*-ом значении:

 0 - в новом процессе.

 **pid** нового процесса - в исходном.

Вызов *fork* может завершиться неудачей если таблица процессов переполнена. Простейший способ сделать это:

 main(){

 while(1)

 if( ! *fork*()) *pause*();

 }

Одно гнездо таблицы процессов зарезервировано - его может использовать только суперпользователь (в целях жизнеспособности системы: хотя бы для того, чтобы запустить программу, убивающую все эти процессы-варвары).

## Пайпы и FIFO-файлы.

Процессы могут обмениваться между собой информацией через файлы. Существуют файлы с необычным поведением - так называемые *FIFO*-файлы (**first in**, **first out**), ведущие себя подобно очереди. У них указатели чтения и записи **разделены**. Работа с таким файлом напоминает проталкивание шаров через трубу - с одного конца мы вталкиваем данные, с другого конца - вынимаем их. Операция чтения из **пустой** "трубы" проиостановит вызов *read* (и издавший его процесс) до тех пор, пока кто-нибудь не запишет в FIFOфайл какие-нибудь данные. Операция позиционирования указателя - *lseek*() - **неприме-** **нима** к FIFO-файлам. FIFO-файл создается системным вызовом

 #include <**sys**/**types**.**h**>

 #include <**sys**/**stat**.**h**>

 *mknod*( **имяФайла**, *S***\_***IFIFO* | 0666, 0 );

где 0666 - коды доступа к файлу. При помощи FIFO-файла могут общаться даже неродственные процессы.

Разновидностью FIFO-файла является **безымянный** FIFO-файл, предназначенный для обмена информацией между процессом-отцом и процессом-сыном. Такой файл - канал связи как раз и называется термином "труба" или *pipe*. Он создается вызовом *pipe*:

 int **conn**[2]; *pipe*(**conn**);

Если бы файл-труба имел имя **PIPEFILE**, то вызов *pipe* можно было бы описать как

 *mknod*("**PIPEFILE**", *S***\_***IFIFO* | 0600, 0);

 **conn**[0] = *open*("**PIPEFILE**", *O***\_***RDONLY*);

 **conn**[1] = *open*("**PIPEFILE**", *O***\_***WRONLY*);

 *unlink*("**PIPEFILE**");

При вызове *fork* каждому из двух процессов достанется в наследство пара дескрипторов:

 *pipe*(**conn**);

 *fork*();

 **conn**[0]----<---- ----<-----**conn**[1]

 FIFO

 **conn**[1]---->---- ---->-----**conn**[0]

 процесс A процесс B

Пусть процесс *A* будет посылать информацию в процесс *B*. Тогда процесс *A* сделает:

 *close*(**conn**[0]);

 // т.к. не собирается ничего читать

 *write*(**conn**[1], ... );

а процесс *B*

 *close*(**conn**[1]);

 // т.к. не собирается ничего писать

 *read* (**conn**[0], ... );

Получаем в итоге:

 **conn**[1]---->----FIFO---->-----**conn**[0]

 процесс A процесс B

Обычно поступают еще более элегантно, перенаправляя стандартный вывод *A* в канал **conn**[1]

 *dup2* (**conn**[1], 1); *close*(**conn**[1]);

 *write*(1, ... ); /\* или printf \*/

а стандартный ввод *B* - из канала **conn**[0]

 *dup2*(**conn**[0], 0); *close*(**conn**[0]);

 *read*(0, ... ); /\* или gets \*/

Это соответствует конструкции

 $ *A* | *B*

записанной на языке СиШелл.

Файл, выделяемый под *pipe*, имеет ограниченный размер (и поэтому обычно целиком оседает в буферах в памяти машины). Как только он заполнен целиком - процесс, пишущий в трубу вызовом *write*, приостанавливается до появления свободного места в трубе. Это может привести к возникновению тупиковой ситуации, если писать программу неаккуратно. Пусть процесс *A* является сыном процесса *B*, и пусть процесс *B* издает вызов *wait*, не закрыв канал **conn**[0]. Процесс же *A* очень много пишет в трубу **conn**[1]. Мы получаем ситуацию, когда оба процесса спят:

*A* потому что труба переполнена, а процесс *B* ничего из нее не читает, так как ждет окончания *A*;

*B* потому что процесс-сын *A* не окончился, а он не может окончиться пока не допишет свое сообщение.

Решением служит запрет процессу *B* делать вызов *wait* до тех пор, пока он не прочитает ВСЮ информацию из трубы (не получит EOF). Только сделав после этого *close*(**conn**[0]); процесс *B* имеет право сделать *wait*.

Если процесс *B* закроет свою сторону трубы *close*(**conn**[0]) **прежде**, чем процесс *A* закончит запись в нее, то при вызове *write* в процессе *A*, система пришлет процессу *A* сигнал *SIGPIPE* - "запись в канал, из которого никто не читает".

## Нелокальный переход.

Теперь поговорим про **нелокальный переход**. Стандартная функция *setjmp* позволяет установить в программе "контрольную точку"[\*](#b9), а функция *longjmp* осуществляет прыжок в эту точку, выполняя за один раз выход **сразу из нескольких** вызванных функций (если надо)[\*](#b10). Эти функции не являются системными вызовами, но поскольку они реализуются машинно-зависимым образом, а используются чаще всего как реакция на некоторый сигнал, речь о них идет в этом разделе. Вот как, например, выглядит рестарт программы по прерыванию с клавиатуры:

 #include <**signal**.**h**>

 #include <**setjmp**.**h**>

 *jmp***\_***buf* **jmp**; /\* контрольная точка \*/

 /\* прыгнуть в контрольную точку \*/

 void *onintr*(**nsig**){ *longjmp*(**jmp**, **nsig**); }

 main(){

 int **n**;

 **n** = *setjmp*(**jmp**); /\* установить контрольную точку \*/

 if( **n** ) printf( "Рестарт после сигнала %d\n", **n**);

 *signal* (*SIGINT*, **onintr**); /\* реакция на сигнал \*/

 printf("Начали\n");

 ...

 }

*setjmp* возвращает 0 при **запоминании** контрольной точки. При прыжке в контрольную точку при помощи *longjmp*, мы оказываемся снова в функции *setjmp*, и эта функция возвращает нам значение второго аргумента *longjmp*, в этом примере - **nsig**.

Прыжок в контрольную точку очень удобно использовать в алгоритмах перебора с возвратом (**backtracking**): либо - если ответ найден - прыжок на печать ответа, либо если ветвь перебора зашла в тупик - прыжок в точку ветвления и выбор другой альтернативы. При этом можно делать прыжки и в рекурсивных вызовах одной и той же функции: с более высокого уровня рекурсии в вызов более низкого уровня (в этом случае *jmp***\_***buf* лучше делать автоматической переменной - своей для каждого уровня вызова функции).

## Разделяемая память

shmget создает новый сегмент разделяемой памяти или находит существующий сегмент с тем же ключом shmat подключает сегмент с указанным дескриптором к виртуальной памяти обращающегося процесса shmdt отключает от виртуальной памяти ранее подключенный к ней сегмент с указанным виртуальным адресом начала shmctl служит для управления параметрами, связанными с существующим сегментом После подключения сегмента разделяемой памяти к виртуальной памяти процесса, он может обращаться к соответствующим элементам памяти с использованием обычных машинных команд чтения и записи

shmid = shmget(key, size, flag);

1. size определяет желаемый размер сегмента в байтах
2. если в таблице разделяемой памяти находится элемент, содержащий заданный ключ, и права доступа не противоречат текущим характеристикам процесса, то значением системного вызова является дескриптор существующего сегмента
3. реальный размер сегмента можно узнать с помощью системного вызова shmctl
4. иначе создается новый сегмент с размером не меньше установленного в системе минимального размера сегмента разделяемой памяти и не больше установленного максимального размера
5. создание сегмента не означает немедленного выделения под него основной памяти
6. откладывается до выполнения первого системного вызова подключения сегмента к виртуальной памяти некоторого процесса
7. при выполнении последнего системного вызова отключения сегмента от виртуальной памяти соответствующая основная память освобождается

virtaddr = shmat(id, addr, flags);

1. id - это ранее полученный дескриптор сегмента
2. addr - желаемый процессом виртуальный адрес, который должен соответствовать началу сегмента в виртуальной памяти
3. virtaddr - реальный виртуальный адрес начала сегмента не обязательно совпадает со значением прямого параметра addr
4. если addr == 0, ядро выбирает наиболее удобный виртуальный адрес начала сегмента

shmdt(addr);

1. addr - виртуальный адрес начала сегмента в виртуальной памяти, ранее полученный от системного вызова shmat

shmctl(id, cmd, shsstatbuf);

1. cmd идентифицирует требуемое конкретное действие
2. важна функция уничтожения сегмента разделяемой памяти

## Семафоры

Обобщение классического механизма семафоров общего вида Диекстры

Целесообразность обобщения сомнительна

Обычно использовался облегченный вариант двоичных семафоров

Известен алгоритм реализации семафоров общего вида на основе двоичных

***Семафор в ОС UNIX:***

1. значение семафора
2. идентификатор процесса, который хронологически последним работал с семафором
3. число процессов, ожидающих увеличения значения семафора
4. число процессов, ожидающих нулевого значения семафора

***Три системных вызова:***

1. semget для создания и получения доступа к набору семафоров
2. semop для манипулирования значениями семафоров
3. semctl для выполнения управляющих операций над набором семафоров

id = semget(key, count, flag);

1. key, flag и id - обычный смысл
2. count - число семафоров в наборе семафоров, обладающих одним и тем же ключом
3. индивидуальный семафор идентифицируется дескриптором набора семафоров и номером семафора в наборе
4. если набор семафоров с указанным ключом уже существует, то число семафоров в группе можно узнать с помощью системного вызова semctl

oldval = semop(id, oplist, count);

1. id - дескриптор группы семафоров
2. oplist - массив описателей операций над семафорами группы
3. count - размер этого массива
4. возвращается значение последнего обработанного семафора

***Элемент массива oplist:***

1. номер семафора в указанном наборе семафоров
2. операция
3. флаги

***Если проверка прав доступа проходит нормально***

1. указанные в массиве oplist номера семафоров не выходят за пределы общего размера набора семафоров
2. для каждого элемента массива oplist значение семафора изменяется в соответствии со значением поля "операция"

***Значение поля операции положительно***

1. значение семафора увеличивается на единицу
2. все процессы, ожидающие увеличения значения семафора, активизируются (пробуждаются)

***Значение поля операции равно нулю***

1. если значение семафора равно нулю, выбирается следующий элемент массива oplist
2. иначе число процессов, ожидающих нулевого значения семафора, увеличивается на единицу
3. обратившийся процесс переводится в состояние ожидания (усыпляется)

***Значение поля операции отрицательно***

(1) его абсолютное значение меньше или равно значению семафора

1. это отрицательное значение прибавляется к значению семафора
2. если значение семафора стало нулевым, то ядро активизирует все процессы, ожидающие нулевого значения этого семафора

(2) значение семафора меньше абсолютной величины поля операции

1. число процессов, ожидающих увеличения значения семафора увеличивается на единицу
2. текущий процесс откладывается

Стремление добиться возможности избегать тупиковых ситуаций

Системный вызов semop выполняется как атомарная операция

Флаг IPC\_NOWAIT заставляет ядро ОС UNIX не блокировать текущий процесс

1. лишь сообщать в ответных параметрах о возникновении ситуации, приведшей бы к блокированию процесса

semctl(id, number, cmd, arg);

1. id - это дескриптор группы семафоров
2. number - номер семафора в группе
3. cmd - код операции
4. arg - указатель на структуру, содержимое которой интерпретируется в зависимости от операции

Можно уничтожить индивидуальный семафор в указанной группе

## Очереди сообщений

***Четыре системных вызова:***

1. msgget для образования новой очереди сообщений или получения дескриптора существующей очереди
2. msgsnd для посылки сообщения (его постановки в очередь сообщений)
3. msgrcv для приема сообщения (выборки сообщения из очереди)
4. msgctl для выполнения управляющих действий

msgqid = msgget(key, flag);

Сообщения хранятся в виде связного списка

***Декскриптор очереди сообщений*** - индекс в массиве заголовков очередей сообщений

***В заголовке очереди хранятся:***

1. указатели на первое и последнее сообщение в данной очереди
2. число сообщений
3. общий размер в байтах сообщений, находящихся в очереди
4. идентификаторы процессов, которые последними послали или приняли сообщение через данную очередь
5. временные метки последних выполненных операций msgsnd, msgrsv и msgctl

msgsnd(msgqid, msg, count, flag);

1. msg - это указатель на структуру, содержащую целочисленный тип сообщения и символьный массив
* count - задает размер сообщения в байтах
1. flag определяет действия ядра при выходе за пределы допустимых размеров внутренней буферной памяти

***Условия успешной постановки сообщения в очередь:***

1. процесс должен иметь право на запись в очередь
2. длина сообщения не должна превосходить верхний предел
3. общая длина сообщений не должна превосходить установленного предела
4. тип сообщения должен быть положительным целым числом

Процесс продолжает свое выполнение

Ядро активизирует (пробуждает) все процессы, ожидающие поступления сообщений из очереди

***Превышается верхний предел суммарной длины сообщений***

1. обратившийся процесс откладывается до разгрузки очереди
2. но есть флаг IPC\_NOWAIT (как для семафоров)

count = msgrcv(id, msg, maxcount, type, flag);

1. msg - указатель на структуру данных в адресном пространстве пользователя для размещения принятого сообщения
2. maxcount - размер области данных (массива байтов) в структуре msg
3. type специфицирует тип сообщения, которое желательно принять
4. flag указывает ядру, что следует предпринять, если в указанной очереди сообщений отсутствует сообщение с указанным типом
5. count - реальное число байтов, переданных пользователю

***Значением параметра type является нуль***

1. выбирается первое сообщение
2. копируется в заданную пользовательскую структуру данных
3. процессы, отложенные по причине переполнения очереди сообщений, активизируются
4. если значение параметра maxcount оказывается меньше реального размера сообщения, ядро не удаляет сообщение из очереди и возвращает код ошибки
5. если задан флаг MSG\_NOERROR, то выборка сообщения производится, и в буфер пользователя переписываются первые maxcount байтов сообщения

***Значение type есть положительное целое число***

1. выбирается первое сообщение с таким же типом

***Значение type есть отрицательное целое число***

1. выбирается первое сообщение, значение типа которого меньше или равно абсолютному значению type

***В очереди отсутствуют сообщения, соответствующие спецификации type***

1. процесс откладывается до появления в очереди требуемого сообщения
2. но есть флаг IPC\_NOWAIT

msgctl(id, cmd, mstatbuf);

1. опрос состояния описателя очереди сообщений
2. изменение его состояния
3. уничтожение очереди сообщений
	1. **Старт системы**
	2. **run levels**
	3. **Остановка системы**
	4. **Конфигурирование ядра системы**
	5. **Инсталирование периферии на примере ленточного накопителя.**
	6. **Инсталирование софта**
	7. **Управление процессами**

# Старт системы.

В самом начале, после включения питания выполяется последовательность команд записанная в Boot ROM машины. Boot ROM выполняет общую диагностику и проводит инициализацию устройств необходимую для дальнейшей загрузки операционной системы. В задачи Boot ROM кода входит:

* Определение типа процессора
* Инициализация и тест таймеров
* Нахождение и инициализация видео консоли
* Загрузка конфигурации с EEPROM
* Инициализации системы ввода-вывода включая пользовательский интерфейс и аудио
* Распечатка на консоли copyright и других баннеров, типа процессора EEPROM статуса, количества памяти
* Тестирование памяти и распечатка общего количества памяти и найденных в результате теста ошибок
* Тест и инициализация системы прямого доступа к памяти (DMA)
* Поиск и распечатка информации о встроенных интерфейсных платах
* Тест и инициализация SCSI интерфеса и интерфейса локольной сети
* Предложение о выборе вариантов загрузки

Boot ROM program

HP-UX файловая система

LIF раздел

LAN интерфейс

Диск

При этом возможен вариант запгрузки как с SCSI устройства (диск, CDROM, лента, …) так и через локальную сеть. Загрузочный диск должен быть предварительно сконфигурирован. Так как обьем Boot ROM не может быть большим, в его задачи входит загрузка вторичного загрузчика операционной системы. Для этого загрузочный диск должен быть инициализирован определенным образом. Помимо стандартной файловой системы он еще должен содержать так называемы LIF (Logical Interchange Fomat) раздел в котором записан вторичный загрузчик и ряд необходимых утилит. Посмотреть состав LIF блока можно с помощью команды lifls принимающей в качестве аргумента имя блочного устройства диска:

 lifls /dev/dsk/c0t5d0

 ISL AUTO HPUX LABEL

Для создания LIF области используется команда lifinit (инициализируються только диски которые не являются подмонтированными файловыми системами). Для записи в или копирования из LIF области используется утилита lifcp .Например, команда:

 lifcp /dev/dsk/c0t5d0:ISL a

копирует файл ISL из LIF области в файл с именем а. Во время инсталяции LIF область создается автоматически и необходимости работы с ней практически не появляется, за исключением случаев сбоев.

 После завершения всех тестов и выполнения поиска возможных устройств загрузки в и в случае если параметр SECURE записанный в EEPROM равен OFF возможен вход в меню загрузки boot ROM при нажатии на клавишу ESC. Если SECURE=OFF и процесс загрузки не прерывался нажатием ESC а также EEPROM параметр AUTOBOOT=ON, boot ROM попытается загрузить из LIF области вторичный загрузчик ISL. Устройство загрузки при этом выбирается из EEPROM параметра PRIMARY BOOT PATH. В случае неудачной загрузки, boot ROM будет грузить ISL из устройства имя которого записано в EEPROM параметре ALTERNATE BOOT PATH. Если не удается загрузиться и от туда, система выйдет в boot ROM меню. Для поиска всех возможных устройств загрузки boot ROM имеет команду SEARCH. Для загрузки с какогото конкретного устройства найденого командой SEARCH используется команда BOOT:

 boot [device\_path] [isl]

запущеная без аргументов она приводит к загрузке системы из устройства адрес которого содержится в PRIMARY BOOT PATH. Если указан аргумент isl то система загрузит вторичный загрузчик ISL в интерактивный режим. Основные случаи когда необходима загрузка не с основного устройства перечислены ниже:

* На основном диске нет загрузочного ядра
* LIF область диска повреждена
* Корневая файловая система ОЧЕНЬ сильно запорчена

*Примечание*: В том случае если SECURE=ON (безопасный режим) нет никакой возможности попасть в boot ROM меню за исключением как физически отключить устройства первичной и вторичной загрузки.

 Если был выбран интерактивный режим загрузки ISL то последний после загрузки, не станет автоматически загружать ядро системы а перейдет в диалоговый режим. В этом режиме есть ряд команд влияющих на загрузку системы. Например по команде 700SUPPORT возможна загрузка с CDROM специальной версии ядра системы предназначеной для восстановления системы в том случае если ядро основной системы не загружается. Список утилит которые доступны для запуска ISL можно увидеть по команде LS. Основная утилита – HPUX, предназначенная для загрузки ядра системы. Для того чтоб посмоьреть содержимое директории /stand на устройстве загрузки по умолчанию нужно воспользоваться командой:

 ISL> HPUX ll disk (;0) /stand/

При загрузке ядра возможно указание файла ядра отличного от того что используется по умолчанию (/stand/vmunix) для загрузки а также запустить ядро с определенными параметрами. Например команда:

 ISL> hpux /stand/vmunix.prev

загружает ядро с именем /stand/vmunix.prev (эту команду используют в тех случаях когда вновь собранное ядро не хочет по какимто причинам запускаться и нужно загрузить старое ядро).

А команда:

 ISL> hpux –is /stand/vmunix

загружает ядро с именем /stand/vmunix в однопользовательский режим. Ситуации прикоторых необходима загрузка в однопользовательский режим:

* забыт пароль администратора и его нужно изменить
* поврежден файл /etc/inittab
* какой то из загрузочных скриптов по каким то причинам зависает

Сразу же после получения управления ядро системы выполняет две задачи:

* Находит и монтирует корневую файловую систему
* Запускает процесс init и если ядру не было указано дополнительных аргументов относительно run-level то init переводит систему на default run-level (обычно это многопорльзовательский режим работы)

# Run-levels.

После успешного монтирования корневой файловой системы ядро запускает процесс init. Отличительной особенностью этого процесса является то что его создает непосредственно ядро,он имеет PID=1 и не имеет родительского процесса, в отличии от остальных процессов получающихся в следствие системного вызова fork(). Конфигурациооный файл программы init называется /etc/inittab. Приведем его формат:

 Id:run-levels:action:process

где

 id От одно до четырехбуквенный индекс который идентифицирует

строку файла inittab.

 run-level определяет run-level.в одной строке может быть несколько run-levels.

run-levels определяются как цифры от 0 до 6. Когда boot init пытается измнить run-level, все процессы которые не имеют run-level поля равному изменяемому run-level получают предупреждающий сигнал (SIGTERM) и те которые не завершили работу по истечению 20-ти секундного интервала получат сигнал (SIGKILL).Если run level не определен, то это подразумевает все run levels, с 0 до 6.

 Это поле также может принимать три других значения “a”, “b” и “c”.

Строки имеющие эти значения в поле run-level выполняются только

когда пользовательский init процесс запрашивает их. (независимо от

текущего run level системы).

Они в корне отличается от run levels в которые boot init никогда не

входит a, b, or c. Также выполнение процессов из этих run-levels

никогда не меняет текущий run level системы.

Более того, процессы запущенные с rul-level a, b, или c не терминируются когда boot init изменяет run-level системы. Процессы терминируются лишь когда соответствующая строка inittab помечена как off в поле action или полностью удалена из inittab или система загружается в однопользовательский режим.

action определяет действия этой строки файла, которые могут принимать следующие значения:

boot выполнять процесс только во время чтения inittab исключительно при загрузке системы. Boot init стартует процесс не дожидаясь его окончания и по его завершении не рестартует его заново.

bootwait выполнять процесс только во время чтения inittab исключительно при загрузке системы. Boot init стартует процесс дожидается его окончания и по его завершении не рестартует его заново.

initdefault процесс выполняется только во время начальной

загрузки. Boot init использует эту строку если она существует для того чтобы определить в какой run-level входить в самом начале. Если в этой строке указано несколько run-levels то запускается с наибольшим номером. Если run-level не указан то стартует по умолчанию run-level c номером 6. Если строка initdefaul не найдена в /etc/inittab то при старте системы будет запрошено на какой run-level запускать систему.

off если процесс асоцированный с этой строкой в данный момент запущен то послать ему предупреждающий сигнал (SIGTERM) и подождать 20 секунд его завершения, после чего принудительно завершить его сигналом SIGKILL. Если процесс не запущен – игнорировать эту строку.

once Когда boot init стартует run level который совпадает с указанным в этой строке он не дожидается его окончания и после окончания не запускает его вновь. Если boot init запускает новый run level но процесс все еще в запущеном состоянии от предыдущего run-level то процесс не перестартовывается.

ondemand Эта инструкция есть синоним инструкции respawn за исключением того что она используется только с “a”, “b”, или “c” значениями run-level.

powerfail Запустить процесс асоциированный с этой строкой только в том случае если boot init получит сигнал power-fail signal (SIGPWR).

powerwait Запустить процесс асоциированный с этой строкой

только в случае если boot init получит power-fail signal (SIGPWR) и ждать пока процесс завершит работу перед запуском любых других процессов из inittab.

respawn Если процесс не запущен, то запустить его не дожидаясь окончания (прподолжив сканирование inittab). После завершения процесса запустить его заново. Если процесс запущен – то ничего не делать продолжив сканирование inittab.

 sysinit процессы содержащиеся в строках этого типа будут запускаться перед тем как boot init попытается получить доступ к системной консоли. Это подразумевает что процессы будут запускаться только для инициализации устройств на которых boot init может получать run level информацию. Boot init ожидает завершение процессов запущенных с этим параметром.

wait Когда boot init запускает run-level с этим параметром, он ждет завершения процесса. Любые сканирования файла inittab пока boot init находится на томже run level являються причиной игнорирования этой строки в файле inittab.

process это шелл скрипт который запускается из шела созданного системным вызовом fork() как "sh -c 'exec command'.

Запуск init может сопровождаться следующими аргументами:

/sbin/init [0|1|2|3|4|5|6|S|s|Q|q|a|b|c]

агрументы означают следующее:

0-6 перевод системы на уровень от 0 до 6

a|b|c выполнение действий из файла из строк inittab eкоторые помечены как

специальный run-level a, b, или c без изменения значения текущего run-level.

Q|q реинициализация файла inittab без изменения значения текущего run-level

S|s перевод системы в однопользовательский режим, при этом логическая

системная консоль /dev/syscon изменяется на тот терминал с которого была

запущена команда.

## Остановка системы

Для остановки системы Вы должны иметь права администратора (пользователь с UID=0). Различают два вида остановки системы, первый это перевод системы в однопользовательский режим, при котором все пользовательские и системные процессы работающие в многопользовательском режиме завершаются, и доступ к машине остается лишь через логическую системную консоль (тот терминал с которого была запущена команда). Такой режим часто бывает необходим во время бэкапа или восстановления данных, при установке нового оборудования или програмного обеспечения. После этого для возврата назад в многопользовательский режим нужно воспользоваться командой init. Второй вид остановки системы – это полная остановка системы с последующим выключением питания. Остановка системы может быть произведена как с применением команд hpux, так и с использованием SAM. При использовании SAM в разделе Routine Tasks выбрать пункт System shutdown а затем тип шатдауна:

* Halt system - полная остановка системы
* Reboot - перезагрузка системы
* Go to single user state - перевод системы в однопользовательский режим

При использовании hpux комманд необходимо выполнить переход в корневой каталог (т.к. нельзя размонтировать файловые системы которые используются просцессами), а затем выполнить команду shutdown с одним из параметров:

* cd /
* shutdown –h now - немедленная остановка системы
* shutdown –r now - немедленная перезагрузка системы
* shutdown - немедленный перевод системы в

 однопользовательский режим

* shutdown –h 300 - остановка системы через пять минут.

 При этом раз в всем залогиненым пользователям

будет посылаться уведомление о предстоящей

остановке системы.

Команда shutdown переключает логическую системную консоль /dev/syscon изменяется на тот терминал с которого была запущена команда. shutdown использует программу /usr/sbin/wall для посылки сообщения о остановке или перезагрузки системы на все терминалы на которых есть асктивные пользователи. По умолчанию лишь администратор системы обладает правами на остановку системы, однако существует файл /etc/shutdown.allow который позволяет выполнять остановку системы (но не перевод в однопользовательский режим) пользователям не имеющих администраторских прав. В этом файле указывается имя пользователя и имя системы (для случая кластеров) которую пользователь может остановить. Симвод # исполдьзуется для комментариев, символ + обозначает любое имя. Например:

# пользователь user1 может останавливать систему systemA и systemB

systemA user1

systemB user1

# администратор может останавливать все системы

+ root

# любой пользователь может остановить систему systemC

systemC +

Отсутствие файла /etc/shutdown.allow или отсутствие в нем администратора (root) не может помешать администратору остановить систему.

После запуска, shutdown выполняет:

* сброс на диск всех суперблоков файловых систем находящихся в памяти
* установку real UID в 0
* широковещательную посылку сообщения всем пользователям
* запуск /sbin/rc для выполнения корректного завершения всех основных системных и пользовательских программ
* выполнение пргораммы reboot для реальной остановки или перезагрузки системы.

Так же как и при старте, во время остановки системы используется скрипт /etc/rc . Если стартовый скрипт линк (например /sbin/rcN.d/S123test) в последовательности N имеет стоп действие, соответствующий остановочный скрипт должен быть помещен в последовательность

N-1 (/sbin/rcN-1.d/K200test). Действия запущенные на уровне N должны быть остановлены на уровне N-1. Поэтому остановка системы (т.е., переход с уровня 3 напрямую в уровень 0) приведет к корректному завершению всех подсистем путем вызова соответствующих остановочных скриптов.

Помомо команды shutdown существует команда rebooot которая выполняет похожие действия. Например reboot –h вызывает остановку системы а reboot без параметров перезагрузку. Между командами reboot и shutdown есть принципиальная разница. reboot всем процессам в системе (кроме самой себя ☺) посылает 9-й сигнал, который процесс не может перехватить или обработать и после этого вызывает остановку или перезагрузку системы. Поэтому не рекомендуется использовать эту команду в системах где есть критически важные приложения (например сервера баз данных) которые требуют корректной остановки.

## Конфигурирование ядра системы

Для большинства систем вполне хватает стандартной конфигурации ядра, однако в ряде случаев оговоренных ниже возникает необходимость переконфигурирования ядра. К этим случаям можно отнести:

* Добавление или удаление периферии (драйверов устройств) а также псевдодрайверов. В случае удаления периферии удалять драйвер устройства из ядра совсем не обязательно, но крайне желательно, т.к. в этом случае ядро будет меньше по обьему и будет работать более эффектино. Прежде чем удалять драйвер устройства убедитесь не зависят ли от него драйвера других устройств проверив файлы в директории /usr/conf/master.d в таблицах зависимости в секции DRIVER\_DEPENDENCY. Особое внимание следует обратить на файл core-hpux.
* Изменение системных параметров (tunable parameters). В тех случаях когда система работает с большим количеством пользователей часто возникает необходимость изменения стандартных системных параметров. Эти параметры определяются в секции TUNABLE в файлах /usr/conf/master.d. Большинство из них находится в файле core-hpux.
* Инсталирование специализированного програмного обеспечения HP (подсистем). Если вы добавляете в систему специализированное програмное обеспечение например поддержку LAN, ATM, FDDI и.т.п в этом случае также требуется модификация ядра.
* Добавление файловых систем отлдичных от HFS.
* Добавление, удаление или модификация swap и (или) dump областей, устройства консоли или корневой файловой системы.

Драйвера по своим возможностям а также по методу доступа и управлению ими можно разделить на три основных типа:

* Символьные драйвера. Работа с этими драйверами происходит в побайтном режиме без использования буферного кэша. К таким драйверам можно отнести драйвера таких устройств как драйвера последовательных портов, терминалов, магнитных лент …
* Блочные драйвера. Этот тип драйверов позволяет проводить обмен блоками данных. Так например обмен с диском происходит фиксироваными блоками (секторами), даже в том случае если идет операция с данными количество которых меньше размера блока всеравно физически будет прочитан или записан один блок. При работе эти драйвера используют системный буферный кэш.
* Драйвера низкого уровня (raw drivers) Этот тип драйверов производит обмен с блочными устройствами напрямую минуя буферный кэш с ситемы.

Кроме драйверов устройств существует множество других драйверов не имеющих непосредственного отношения к периферии компьютера. Такие драйвера называются псевдодрайверами. Вот примеры некоторых из них:

 /dev/kmem обеспечивает доступ к физической памяти компьютера

 /dev/mem обеспечивает доступ к виртуальной памяти ядра

/dev/null нулевое устройство. Призаписи в него данные удаляются, а при чтении считывается 0 байт

lvm (Logical Volume Manager) обеспечивает построение и доступ к логическим дискам

Драйвера адресуются старшим номером устройства (major number). Помимо него также существует младший номер (minor number) для адресации одного из клонов драйвера. Например в случае с драйвером диска младший номер может означать номер диска.

$ ls -l /dev/dsk/

total 0

brw-r--r-- 1 root sys 31 0x002000 Jun 10 1996 c0t2d0

brw-r--r-- 1 root sys 31 0x005000 Jun 10 1996 c0t5d0

brw-r--r-- 1 root sys 31 0x006000 Jun 10 1996 c0t6d0

$ ls -l /dev/rdsk/

total 0

crw-r----- 1 root sys 188 0x002000 Jun 10 1996 c0t2d0

crw-r----- 1 root sys 188 0x005000 Jan 3 16:47 c0t5d0

crw-r----- 1 root sys 188 0x006000 Jun 10 1996 c0t6d0

Первая команда выводит файлы блочных дисковых устройств, вторая файлы raw дисковых устройств. Как создаются файлы устройств будет показано чуть позже. Доступ к драйверу осуществляется через специальную структуру данных называемою коммутатором устройств каждый элемент которой содержит указатели на соответствующие функции драйвера (d\_open(), d\_close(), d\_strategy(), d\_read(), d\_write(), d\_ioctl, d\_xpoll(), d\_intr(), …), так называемые точки входа. Старший номер является указателем на элемент коммутатора устройств. Блочные и символьные устройства имеют свои собственные коммутаторы. Список драйверов можно посмотреть воспользовавшись программой lsdev.

# Изменение системных параметров

К одним из основных системных параметров относится параметр MAXUSERS который является макросом на основании которого вычисляются множество других параметров (например nproc вычисляется как 20+8\*MAXUSERS). Этот параметр не указывает, как может показаться на первый взгляд на максимальное число пользователей в системе. На системах с небольшим количеством пользователей (рабочие станции) он обычно равен 32, на больших системах (T500) его значение может превышать 200. Перечислим краткий список основных системных параметров:

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Описание** |
| dbc\_max\_pct | 50 | Максимальный размер буферного кэша в процентах от обьема RAM  |
| maxdsiz | 67108864 | Максимальный размер сегмента данных |
| maxssiz | 8388608 | Максимальный размер стека |
| maxtsiz | 67108864 | Максимальный размер сегмента кода |
| maxfiles | 60 | Максимальное количество открытых файлов на процесс |
| maxuprc | 75 | Максимальное число процессов пользователя |
| maxusers | 32 | Макрос через который определяются большинство других параметров |
| nfile | 2172 | Максимальное число открытых файлов в системе |
| nflock | 200 | Максимальное количество заблокированных файлов |
| npty | 60 | Максимальное количество псевдотерминалов в системе |
| semmns | 128 | Максимальное количество семафоров |
| shmmax | 67108864 | Максимальный обьем разделяемой памяти |
| shmmni | 200 | Максимальное количество идентификаторов разделяемой памяти |
| shmseg | 120 | Максимальное количество сегментов разделяемой памяти на процесс |

Для изменения системных параметров можно воспользоваться утилитой SAM либо выполнить изменения конфигурационных файлов и пересобрать ядро вручную. При использовании SAM после запуска sam необходимо:

* Открыть меню “SAM Kernel Configuration”
* Выбрать “Configurable parameters”
* Выбрать нужный параметр из списка
* Активировать пункт меню “Actions->Modify Configurable Parameter”
* Ввести новое значения параметра

После выхода из раздела “Configurable parameters” SAM предложит создать новое ядро и перезагрузить систему.

**Конфигурирование ядра с использованием команд HP-UX.**

Для этого необходимо выполнить следующую последовательность действий:

* cd /stand/buil
* /usr/lbin/sysadm/system\_prep –v –s system

Этот скрипт сосздает файл system являющийся шаблоном конфигурационного файла ядра

* vi system

Редактируем файл ядра

* mk\_kernel –s system

Компилируем ядро. Новое ядро будет содержаться в файле /stand/build/vmunix\_test

* mv /stand/vmunix /stand/vmunix.prev

mv /stand/system /stand/system.prev

Создаем резервные копии старого ядра и конфигурационного файла

* mv /stand/build/vmunix\_test /stand/vmunix

mv /stand/build/system /stand

Перемещаем ядро в директорию из которой идет загрузка

* shutdown –r now

выполняем перезагрузку системы

## Инсталирование периферии

**Файлы устройств**

Обычно файлы устройств располагаются в каталоге /dev и cуществует специальное соглашение по поводу их имен. Имена файлов устройств как правило маркируются в соответствии со следующим соглашением: c#t#d#[s#] **:**

 c# представляет класс интерфейса или интерфейсной карты.

 t# адрес устройства на шине. Обычно он выставляется физически

переключателями расположеными на устройстве.

 d# номер устройства. (для SCSI устройств это логический номер устройства LUN)

 s# необязательный параметр показывающий номер секции устройства. Например для дисков он показывает номер портиции, 0 – указывает на целый диск.

Утилиты lssf и ioscan могут помочь в определении интерфейса к которому подключено то или иное устройство. Например:

bash-2.04$ lssf /dev/dsk/c0t6d0

sdisk card instance 0 SCSI target 6 SCSI LUN 0 section 0 at address 2/0/1.6.0 /dev/dsk/c0t6d0

bash-2.04$ lssf /dev/null

pseudo driver mm minor 0x000002 /dev/null

## Системная конфигурация

Утилита ioscan является одной из наиболее полезных утилит для просмотра системной информации. Ее можно использовать для построения аппаратного адреса устройства. В простейшем виде ioscan показывает аппартный путь (адрес), класс устройства и описание. Опции –u (используемые устройства) или –k (структуры ядра) дают быстрый результат без сканирования оборудования.

# /usr/sbin/ioscan

H/W Path Class Description

=============================================

 bc

1 graphics Graphics

2 ba Core I/O Adapter

2/0/1 ext\_bus Built-in SCSI

2/0/1.2 target

2/0/1.2.0 disk TOSHIBA CD-ROM XM-5401TA

2/0/1.5 target

2/0/1.5.0 disk SEAGATE ST32151N

2/0/1.6 target

2/0/1.6.0 disk SEAGATE ST32151N

2/0/1.7 target

2/0/1.7.0 ctl Initiator

2/0/2 lan Built-in LAN

2/0/4 tty Built-in RS-232C

2/0/6 ext\_bus Built-in Parallel Interface

2/0/8 audio Built-in Audio

2/0/10 pc Built-in Floppy Drive

2/0/11 ps2 Built-in Keyboard

4 ba EISA Adapter

5 ba Core I/O Adapter

5/0/1 hil Built-in HIL

5/0/2 tty Built-in RS-232C

8 processor Processor

9 memory Memory

#

Использование ключа –f приводит к выдаче полной информации включая номер интерфейса или интерфейсной карты.

Class I H/W Path Driver S/W State H/W Type Description

================================================================

bc 0 root CLAIMED BUS\_NEXUS

graphics 0 1 graph3 CLAIMED INTERFACE Graphics

ba 0 2 bus\_adapter CLAIMED BUS\_NEXUS Core I/O Adapter

ext\_bus 0 2/0/1 c720 CLAIMED INTERFACE Built-in SCSI

target 0 2/0/1.2 tgt CLAIMED DEVICE

disk 0 2/0/1.2.0 sdisk CLAIMED DEVICE TOSHIBA CD-ROM XM-5401TA

target 1 2/0/1.5 tgt CLAIMED DEVICE

disk 1 2/0/1.5.0 sdisk CLAIMED DEVICE SEAGATE ST32151N

target 2 2/0/1.6 tgt CLAIMED DEVICE

disk 2 2/0/1.6.0 sdisk CLAIMED DEVICE SEAGATE ST32151N

target 3 2/0/1.7 tgt CLAIMED DEVICE

ctl 0 2/0/1.7.0 sctl CLAIMED DEVICE Initiator

lan 0 2/0/2 lan2 CLAIMED INTERFACE Built-in LAN

tty 0 2/0/4 asio0 CLAIMED INTERFACE Built-in RS-232C

ext\_bus 1 2/0/6 CentIf CLAIMED INTERFACE Built-in Parallel Interface

audio 0 2/0/8 audio CLAIMED INTERFACE Built-in Audio

pc 0 2/0/10 fdc CLAIMED INTERFACE Built-in Floppy Drive

ps2 0 2/0/11 ps2 CLAIMED INTERFACE Built-in Keyboard

ba 2 4 eisa CLAIMED BUS\_NEXUS EISA Adapter

ba 1 5 bus\_adapter CLAIMED BUS\_NEXUS Core I/O Adapter

hil 0 5/0/1 hil CLAIMED INTERFACE Built-in HIL

tty 1 5/0/2 asio0 CLAIMED INTERFACE Built-in RS-232C

processor 0 8 processor CLAIMED PROCESSOR Processor

memory 0 9 memory CLAIMED MEMORY Memory

Использование ключа –n приводит к тому что ioscan дополнительно выдает информацию о файле устройства:

target 0 2/0/1.2 tgt CLAIMED DEVICE

disk 0 2/0/1.2.0 sdisk CLAIMED DEVICE TOSHIBA CD-ROM XM-5401TA

 /dev/dsk/c0t2d0 /dev/rdsk/c0t2d0

target 1 2/0/1.5 tgt CLAIMED DEVICE

disk 1 2/0/1.5.0 sdisk CLAIMED DEVICE SEAGATE ST32151N

 /dev/dsk/c0t5d0 /dev/rdsk/c0t5d0

В том случае когда драйвер устройства не может быть автоматически сконфигурирован и соответствующий файл устройства оказывается несозданным приходится создавать его вручную с помощью команд mkfs или mknod. Ядро взаимодействует с аппаратным обеспечением ассоциируя имя драйвера и аппаратный адрес. Стандартный интерфейс HP-UX к драйверам поставляется вместе с библиотекой /usr/conf/lib/libhp-ux.a. Ядро распознает интерфейсные драйвера и драйвера устройств через младшие и старшие нгомера “прошитые“ в файлах устройств.

Minor number

(24 bits)

Major number

(8bits)

Где находится устройство ? В каком режиме с ним работать ? (например плотность записи на ленту …)

С каким драйвером работать ?

**Старший номер (major number)**

Старший номер, как уже было сказано раньше является индексом в таблице переключателя устройств ядра. Для нахождения правильного старшего номера можно воспользоваться программой lsdev. Она считывает заголовки и список драйверов сконфигурированных в ядре с ихними блочными и символьными старшими номерами. Номера выводятся в десятичной форме, -1 означает либо то что устройство является модулем, драйвер несконфигурирован либо драйвер не поддерживает какогото из режимов (блочного или символьного).

**Младший номер (minor number)**

Младший номер определяет собой: расположение устройства и его драйвер-зависимые характеристики. Некоторые примеры младших номеров файлов устройств приведено ниже. Более полную информацию можно почерпнуть из руководства “Configuring HP-UX for Peripherals”.

*SCSI Disk device*. Рассмотрим что означает младший номер 0x023000 у SCSI устройства.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **bits** | **8-11** | **12-15** | **16-19** | **20-23** | **24-27** | **28-31** |
| **Binary** | 0000 | 0010 | 0011 | 0000 | 0000 | 0000 |
| **hex** | 0 | 2 | 3 | 0 | 0 | 0 |

0000 0010 Первые восемь бит идентифицируют интерфейс или интерфейсную карту

 0011 SCSI адрес диска

оставшиеся биты нулевые.

*SCSI ленточный накопитель*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Bits 16-19** | **20-23** | **24-27** | **28-31** |
| **Binary** | SCSI номер | SCSI LUN | 24 – поведение как у BSD систем при закрытии 25 – без перемотки26 – конфигурационный метод (если 1 то биты от 27 до 31 означают индекс, если 0 то плотность записи)27-31 Индекс/плотность записи |

Создавать файлы устройств можно с помощью команды mknod. Она имеет следующий синтаксис:

 mknod *file\_name* [c|b] *major minor*

например

 mknod /dev/null c 3 0x000002

* 1. **Инсталирование периферии на примере ленточного накопителя.**

Прежде чем перейти к включению ленточного накопителя нужно убедится в том что его SCSI id не совпадает ни с одним из уже установленных устройств (диски, CD-ROM, …). После физического подключения накопителя к SCSI шине необходимо в ядро. В случае использования SAM для этого необходимо войти в раздел Kernel Configuration -> Drivers, в списке драйверов найти stape, и активировать его выбрав опцию Actions -> Add driver to Kernel. После выхода из окна конфигурации ядра SAM предложит перестроить я дро и перегрузить компьютер, также создаст все необходимые файлы устройств в каталоге /dev/rmt. Все тоже самое можно сделать используя командыв hpux как это было указано выше. В качестве имя драйвера нужно использовать stape.

 # cd /stand/build

 # /usr/lbin/sysadm/system\_prep –v –s system

 # vi system

 # /usr/sbin/mk\_kernel –s system

 # mv /stand/system /stand/system.prev

 # mv /stand/vmunix /stand/vmunix.prev

 # mv /stand/build/system /stand/system.prev

 # mv /stand/build/vmunix\_test /stand/vmunix

 # shutdown –r now

 # ioscan –f –C tape

Class I H/W Path Driver S/W State H/W Type Description

================================================================

tape 0 2/0/1.3.0 stape CLAIMED DEVICE HP HP35480A

# lsdev | grep tape

 205 -1 stape tape

# mkdir /dev/rmt

# cd /dev/rmt

# mknod 0m c 205 0x003000

# mknod 0mn c 205 0x003040

# mknod 0mnb c 205 0x0030c0

*Примечание*: младшие номера для файла устройств можно узнать из файла /usr/include/sys/mtio.h (раздел Masks for minor number bits )

* 1. **Инсталирование софта**

Для работы с программными пакетами существует целая серия команд. Эти команды носят название SD-UX команды. Приведем список основных из них:

 swintsall инсталирование программных пакетов

 swremove удаление программных пакетов

 swlist список инсталированого програмного обеспечения

 swacl просмотр и модификация прав для защиты программных компонент

##  Управление процессами

**Что такое процесс ?**

Процесс это запущенная программа обслуживаемая такими компонентами ядра как планровщик задач и подсистемой управления памятью. Процесс состоит из сегмента кода, данных и стека. С процессом ассоциировано два стека – пользовательский и системный. В дополнение к этому роцесс идентифицируется

* програмными данными (переменные, массивы, записи …)
* номером процесса PID, номером процесса родителя PPID, и номером группы процессов PGID
* идентификатором пользователя и группы PID, GID
* информацией о открытых файлах
* текущей рабочей директорией

**Взаимоотношения процессов**

Процессы в системе постороены по иерархическому принципу родитель-потомок. Каждый процесс (за исключением init) имеет одного родителя, но каждый родитель может иметь несколько потомков. Процесс потомок наследует окружение родителя (переменные окружения, открытые файлы, рабочую директорию). Все процессы за исключением init, pagedaemon, и

swapper) принадлежат к группам процессов.

**Процесс ID и родительский процесс ID.**

Во время создания процесса HP-UX назначает ему уникальный номер известный как процесс ID (PID), именно по этому номеру ядро идентифицирует процесс при выполнении системных вызовов. Помомо PID процесс имеет параметр как PPID (PID родителя). Используя программу ps можно посмотреть эти параметры:

 $ ps -f

 UID PID PPID C STIME TTY TIME COMMAND

 torry 3865 3699 2 13:35:43 ttyp3 0:00 ps -f

 torry 3699 3698 0 12:58:21 ttyp3 0:00 ksh

**Идентификаторы пользователя и группы. (реальные и эффективные)**

Помимо PID и PPID процесс имеет еще ряд идентификационных номеров:

 \* реальный идентификатор пользователя (a real user ID)

 \* реальный идентификатор группы (a real group ID)

 \* эффективный идентификатор пользователя (effective user ID)

 \* эффективный идентификатор группы effective group ID.

Реальный идентификатор пользователя это целое число показывающее владельца процесса. Реальный идентификатор группы это целое число показывающее группу к которой принадлежал пользователь создатель процесса. Комманда id показывает оба этих значения.

 %id

 uid=513(torry) gid=20(users)

 % grep 513 /etc/passwd

 torry:EqqHevH:513:20:Torry Ho,[44MY],474-1969 ,:/home/torry:/usr/bin/csh

Эффективный идентификатор пользователя и группы процесса позволяет процессу получать доступ к файлам или выполнять программы как пользователь имеющим ID равным эффективному. Обычно реальный и эффективные идентификаторы процессов совпадают, но не всегда. Когда эффективный идентификатор равен нулю, процесс начинает выполнять системные вызовы как администратор системы.

Эффективный идентификатор пользователя и группы остаються установленными до:

 \* окончания процесса.

\* пока они не заменяться при выполнении системного вызова exec() программы c

 установленными битами setuid или setgid.

\* пока эффективный, реальный или сохраненные идентификаторы группы и пользователя не будут установлены системными вызовами setuid(), setgid(), setresuid().

**Группы процессов**

Каждый процесс за исключением системных процессов таких как init и swapper принадлежат к группе процессов. Когда созхдается задание, шелл присваивает всем процессам в задании одну и туже группу процессов. Сигналы при этом могут распостраняться на все процессы в группе, в этом и заключается преимущество управления заданиями. Каждая группа процессов идентифицируется целым числом которое называется Process Group ID (PGID). PGID у группы процессов равен PID лидера группы – создателя группы. Все процессы в группе имеют одинаковый GID. PGID не может быть использован системой пока живет группа процессов. Время жизни группы процессов определяется как период времени между созданием группы и когда процесс покидает группу. Процесс покидает группу если:

 \* когда другой процесс вызывает wait() или waitpid() функции на неактивный процесс.

 \* при вызове setsid или setpgid системных вызовов.

**Списки доступа группы**

Каждый процесс имеет до NGROUPS\_MAX списков групп к которым он может принадлежать.

NGROUPS\_MAX определено в /usr/include/limits.h, и обычно равняется 20. Процессу разрешается получать доступ к файлам с групповымим правами любой из списка групп. Списки доступа. Групповые права доступа контролируются командой chgrp.

**Сессии**

Каждый процесс является членом сессии. Все процессы запущенные после логина принадлежат к одной сессии. Процесс принадлежит к той же сессии что и его родитель. Процесс может изменить сессию используя системный вызов setsid(), при этом этот процесс будет являться лидером сессии. Временем жизни сессии будет время с момента ее создания до момента завершения последнего процесса.

**Процессы и терминальное взаимодействие.**

Каждая сессия имеет управляющий терминал. Лидер сессии подключенный к управляющему терминалу называется еще контрольным процессом. Исключением являются процессы- демоны (cron, inetd, …) которые не имеют управляющего терминала. Все процессы принадлежащих к одной сессии используют управляющий терминал как стандартное устройство ввода, вывода и ошибок. В любой момент времени лишь одна группа процессов в сессии может находится не в фоновом выполнении и она имеет исключительные права на работу с управляющим терминалом.

**Попытки чтения фоновой группой процессов**

Если процесс из фоновой группы пытается читать из управляющего терминала, этой группе посылается сигнал SIGTTIN, который по умолчанию приостанавливает процесс. В любом случае системный вызов read() возвращает –1.

**Попытки чтения фоновой группой процессов**

Если процесс в фоновой группе пытается записать в управляющий терминал, группа процессов получает при этом сигнал SIGTTOU, который по умолчанию останавливает процесс.

**Создание процессов**

Один процесс может создать другой через:

 \* паралельное выполнение другой программы

\* выполнение другой программы с ожиданием ее окончания

На системном уровне процесс создается во время вызова системного вызова fork() или vfork().

**Системный вызов fork()**

Этот системный вызов создает новый процесс путем клонирования существующего. В старых реализациях HP-UX, система копировала полностью сегмент данных процесса, что негативно сказывалось на скорости и эффективности работы системы. Сейчас реализуется механизм известный как copy-on-write (на самом деле HP-UX реализует механизм copy-on-access), который позволяет использовать общие страницы памяти до момента записи.

**Системный вызов vfork()**

Приложения которым нужно создать независимый процесс могут делать это более эффективно если вместо fork() будут использовать vfork().Использование vfork оправдано только когда процесс потомок сразуже выполняет exec() или \_exit() системные вызовы. При использовании vfork, потомок использует виртуальное адресное пространство родителя, поэтому оба процесса не могут работать одновременно. Процесс родитель при этом засыпает.

**Системный вызов exec()**

Очень часто после вызова fork() процесс запускает exec() на выполнение другой программы, при этом происходит перезапись сегмента кода и данных новым процессом.

**Открытые файлы**

При системных вызовах fork() и vfork() происходит наследование процессом потомков всех открытых файловых дискрипторов. Для системных параметра определяют ограничение на количество открытых файлов на процесс: maxfiles и maxfiles\_lim. Параметр maxfiles определяет мягкий лимит как много открытых файлов может иметь процесс. Мягкий лимит наследуется после вызовов fork() vfork(). Параметр maxfiles\_lim опреджеляет жесткий лимит на количество открытых файлов на один процесс. maxfiles должен быть меньше или равен maxfiles\_lim. Мягкий лимит процесс наследует от своего родителя, который может быть уменьшен или увеличен до жесткого лимита (такое может сделать только процесс с правами администратора) с использованием системного вызова setrlimit().

**Завершение процессов**

Процесс завершается если:

 \* Он успешно окончил свое выполнение

 \* Процесс завершил.себя вызвав системный вызов exit()

 \* Процесс получил сигнал на с фатальным действием

Когда процесс завершается все открытые файлы завершаются и все занятые ресурсы освобождаются, после чего процесс умирает.

**Команды управления процессами.**

Управлять процессами можно либо с использованием команд HP-UX либо с использованием утилиты SAM. Наиболее часто используемые команды будут описаны ниже.

**Команда ps (process status)**

Команда ps показывает следующие параметры процессов:

 \* идентификатор пользователя User ID

 \* идентификатор процесса Process ID

 \* идентификатор родительского процесса Parent process ID

 \* командную строку породившую процесс

 \* терминал с которого была запущена комманда

 \* время (rela time CPU) которое было затрачено процессором на выполнение процесса.

Запущенная без опций она показывает process ID, terminal ID (tty), real CPU time usage, имя команды которую запустил на выполнение пользователь. С ключем –f ps также показывает имя пользователя, PPID, и время с момента когда процесс был fork-нут.

$ ps -f

 UID PID PPID C STIME TTY TIME COMMAND

 torry 3286 2016 9 16:19:03 ttyp1 0:00 ps -f

 torry 25705 25649 0 08:47:58 ttyp1 0:02 -ksh /home/torry [ksh]

 torry 2016 25705 0 15:13:02 ttyp1 0:24 vi processes.tag

Опция –e приводит к выдаче информации о всех активных процессах в системе. Опция -l (long) дополнительно показывает состояние процесса (S), параметр nice (NI), адрес процесса в памяти (ADDR), приоритет (PRI), и размер в блоках (SZ) образа процесса.

$ ps -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME COMD

1 R 513 11009 7793 5 179 20 d6e200 16 ttyu4 0:00 ps

1 S 0 7792 133 15 154 20 e06100 13 214fb0 ttyu4 0:00 rlogind

1 S 513 7793 7792 16 168 20 df5a80 52 7ffe6000 ttyu4 0:00 csh

**Относительные приоритеты процессов -- nice и renice**

Все процессы имеют приоритет устанавливаемый на основе таких факторов как прльзователь запустивший процесс и в каком окружении запущен процесс: с разделением времени или в реального времени. Комманда nice может быть использована для запуска процесса с более низким чем по умолчанию приоритетом. nice не уменьшает приоритетов уже запущеных процессов.

Команда renice позволяет изменять приоритеты уже запущеных процессов.

**Программы для мониторинга производительности системы**

**top**

Периодически показывает информацию о процессах в системе в порядке убывания использования ресурсов. Суммирует состояние системы (среднюю загрузку), использование виртуальной памяти. В отличии от psкоторый дает лишь снимок текущего состояния системы top выдает информацию с периодичностью. На мультипроцессорных системах, top выдает состояние каждого процессора раздельно.

**sar**

Выдает суммированую статистику о системной активности включая загрузку CPU, активность буферов, количество опреций ввода-вывода, терминальную активность, число системных, активность свопинга и п.р.

**vmstat**

Выдает информацию об использовании виртуальной памяти и активность CPU.

**iostat**

Выдает информацию об дисковой активности ввода-вывода, терминальной активности статистику CPU.

## Управление процессами и ядро системы

Процесс может иметь доступ на чтение и запись к своему сегменту данных и стеку но не к другим процессам (за исключением сегментов разделяемой памяти). Процесс обменивается данными с другими процессами либо через системные вызовы либо через разделяемую память.

**Приоритеты процесса**

Все процессы в HP-UX могут находится в двух режимах: пользовательский режим и режим ядра. Когда процесс переключается в режим ядра (например при системном вызове) он исполняет код ядра и пользуется стеком ядра. В зависимости от приоритета и наличия других процессов процесса планировцик процессов может выделять ему больше или меньше квантов процессорного времени.

Приоритеты можно разделить на две части: диапазон POSIX standard приоритетов и диапазхон HP-UXприоритетов. POSIX standard приоритеты всегда выше чем все другие HP-UX приоритеты. Процесс который имеет меньшее числовое значение приоритета имеет преимущество при выполнении над процессом с большим цифровым значением приоритета. Следующий список выводит по категориям диапазоны приоритетов от большого к маленькому:

 1. POSIX standard приоритеты (системный параметр)

POSIX standard приоритеты, известные как RTSCHED приоритеты, являются самыми

высокими приоритетами. Количество RTSCHED приоритетов есть системный параметр

 (rtsched\_numpri), устанавливаемый между 32 и 512 (default 32).

 2. приоритеты реального времени (0-127)

 Зарезервированы для процессов SCHED\_RTPRIO стартующих с системного вызова

rtprio()

 3. Системные приоритеты (128-177)

 Используются системными процессами.

 4. Пользовательские приоритеты (178-255)

 Устанавливаются пользовательским процессам.

Ядро может изменять приоритеты процессов разделения времени (128-255) но не процессов реального времени (0-127).

**Состояние процесса**

В процессе своей жизни процесс может менять несколько раз свое состояние.Процессы размиещаются в очередях выполнения планировщика процесса в соответствии с их состоянием как определено в файле /usr/include/sys/proc.h . События, такие как получение сигнала могут переводить процесс из одного состояния в другое.

Процесс может находится в одном из следующих состояний:

 \* idle – процесс простаивает в очереди выполнения планировщика.

\* run - процесс выполняется в очереди планировщика либо в пользовательском режиме либо в режиме ядра.

 \* stopped – процесс остановлен сигналом либо процессом родителем.

\* sleep – процесс не выполняется пока он находится в спящем состоянии в очереди

(например он ждет окончания операции ввода-вывода)

\* zombie – процесс уже фактически не существует, но так как он оставил для родителя

некоторые непрочитанные данные о своем выполнении он не может осуществить завершение.

Когда программа запускает прооцесс, ядро выделяет для него сроку из своей таблицы процессов и процесс находится в idle состоянии. Затем после получения ресурсови попадания в очередь он находится в состоянии run. Если процесс получкает сигнал SIGSTOP или его переводлят в режи трассировки, он переходит в stop состояние. После получения сигнала SIGCONT signal, процесс возвращается в состояние run. Если запущеный процесс ждет освобождение какихто ресурсов (семафор) или завершения операции вво-да вывода – он переходит в спящее состояние. Спящий процесс может быть отсваплен в первую очередь.Как только процесс заканчивает свою работу он переходит в состояние zombie.

**1. Файловая система HFS.**

**2. Менеджер логических дисков LVM**

**3. Особенности файловой системы VxFS**

**4. Операции над файловой системой**

# 1. Файловая система HFS.

Одной из испольуемых HP-UX файловых систем является High Performance File System (HPFS, HFS) известная также как MCKusic или BSD файловая система. Файловая система это структура данных существующая на дисковых устройствах позволяющая сохранять и получать доступ к информации хранящейся в ней в иерархическом виде. Существует ряд причин исходя из которых информацию хранящуюся на дисковых носителях приходится организовывать в виде нескольких файловых систем, к ним можнно отнестиЖ

* Если суммарный обьем данных превышает обьем поддерживаемый одной файловой системой
* Для увеличения производительности системы, разнося данные имеющие разную структуру (средний размер файла, наличия ограничения на дисковое пространство для пользователей) на разные файловые системы или данные к которым идет частый доступ разносятся на разные физические устройства.
* В целях системной безопасности (запрет запуска программ с файловых систем, ограничение дискового места для файловой системы, и.т.п)

В общих чертах файловую систему HP-UX состоит из дерева каталогов (директорий) расположеных иерархически с корневым каталогом в вершине. Все файловые системы имеют свойство “монтируемости” заключающееся в том что любая файловая система может быть смонтирована (прикреплена) к существующей директории. Любую смонтированную файловую систему кроме корневой можно размонтировать. Монтирование корневой файловой системы осуществляет ядро, сразу же после загрузки.

## Структура файловой системы HFS

Все HFS файловые системы имеют в своем составе следующие структуры:

* Главный суперблок
* Набор групп цилиндров

Структуры данных используемые в файловой системе содержатся в файле /usr/include/sys/fs.h. Первичный суперблок это непрерывный блок данных размеров 8К размещающийся перед началом файловой системы который содержит статическую информацию о файловой системы в момент ее создания (или последнего расширения):

* Размер файловой системы
* Количество inodes которые может хранить файловая система
* Позиция свободного места на диске
* Количество групп цилинров
* Позиции суперблоков, крупп цилиндро, блоков inodes и блоков данных
* Размер блока и фрагмента

## Главный суперблок

Помимо этого главный суперблок содержит вспомагательную информацию о том когда последний раз монтировалась, модифицировалась и проверялась файловая система. Потому что главный суперблок содержит исключительно важную информацию о файловой системе, HP-UX всегда хранит его копии в каждой группе цилиндров. Одна из копий загружается в память после загрузки. Главный суперблок размещается в самом начале файловой системы и каждая группа цилиндпров имеет имеет свою копию суперблока. Таким образом достигается резервирование критически важной информации. Суперблоки групп цилиндров на диске обновляются каждый раз когда выполняется команда sync или когда файловая система размонтируется. В файле /etc/sbtab присутствует запись положения всех суперблоков файловых систем.

## Группы цилиндров

Группа цилиндров – это группа дисковых цилиндров идущих подряд. Цилиндр представляет собой совокупность треков каждый из которых размещен на одинаковом расстоянии от центра поверхности дискового носителя. Все треки в цилиндрк доступны за одну операцию чтения/записи дисковой головки. В целях повышения производительности, цилиндры группируются (по умолчанию 16 цилиндров) в группы цилиндров. Каждая группа цилиндров имеет свой набор inodes и свою карту свободного пространства в группе. Такая организация хранения позволяеет минимизировать время поиска данных файла в группе цилиндров. Приведем структуру группы цилиндров:

|  |  |
| --- | --- |
| **Структура** | Размер |
| Главный суперблок 1 | 8 Кб |
| Запасной суперблок | 8 Кб |
| Информация группы цилиндров | 1 блок (4 Кб или 8 Кб) |
| Таблица inodes | Переменный 2 |
| Блоки данных | 0 или более блоков 3 |

 1 только для первой группы цилиндров

 2  см. раздел inodes

 3  см. раздел блоки данных

Раздел информации о группе цилиндров хранит динамические параметры группы цилиндров, такие как:

* Количество inodes и блоков данных в группе цилиндров
* Указатели на последний использованый блок, фрагмент и inode
* Количество свободных фрагментов
* Карту использованных inodes
* Карту свободных блоков

Информация о группе цилиндров занимет один блок (размер блока определяется присоздании файловой системы и обычно равен либо четырем либо восьми килобайтам).

Inodes

Кроме хранения информации о состоянии файловой системы, группы цилиндров хранят ключевую информацию о inodes файловой системы – индексам файлов данных (и директорий). Дисковая inode содержит следующую информацию о файле:

* Тип файла и атрибуты доступа
* Количество ссылок на файл
* Владельца и группу файла
* Размер файла в байтах
* Временные метки (время последнего обращения, последней модификации)
* Указатели на блоки файловой системы содержащих данные

Когда файл открыт процессом, информация о его inode находится в памяти ядра (in-core inode) в сочетании с доаполнительными атрибутами, такими как:

* Статус inode, включая факт блокировки inode, отличается ли in-core inode от дисковой inode в следствие модификации файла, является ли файл точкой монтирования файловой системы …
* Цифровой адрес файловой системы содержащий файл
* Указатель на другие in-core inodes выстроенные в виде списка.

Если inode указывает на специальный (не регулярный) файл, то с ним ассоциируются дополнительные параметры, такие как является ли файл FIFO или pipe, символьным или блочным устройством или же директорией. Когда создается файловая система (команда newfs), создаются inodes. Количество inodes ограничивает количество файлов в файловой системе. При создании файловой системы по умолчанию система подразумевает что в среднем на одну inode припадает 2048 байт данных, что в большинстве случаев является более чем достаточным количеством. Иногда, возможна ситуация когда ядро сигнализирует об ошибке переполнения таблицы in-core inodes (*inode: table is full*). В этом случае необходимо изменить размер этой таблицы увеличив системный параметр ядра ninode.

## Блоки данных

После суперблока, данных о группе цилиндров и таблице inodes идет место зарезервированное под блоки данных. HP-UX поддерживает блоки размером 4,8,16,32 и 64Кб. Размер блока задается при создании файловой системы (команда newfs). Большой размер блока дает выиграш в скорости передачи данных при работе с большими файламино при этом является причиной неэффективного использования дискового пространства приработе с маленькими (которых большинство в HP-UX) файлами. Поэтому в целях экономии дискового пространства блок может быть разделен на несколько фрагментов (1,2 или 4Кб). Размер фрагмента также определяется при создании файловой системы и не может иметь размер меньше чем одна восьмая размера блока.

# Доступ к блокам данных

**1**

**2**

.

.

12

Третий уровень ссылок

Второй уровень ссылок

Первый уровень ссылок

**Данные файла**

Блоки адресуемые 1 напрямую 2

 .

 .

 .

 12

Временные метки

Размер файла в байтах

Владелец / группа

Ссылки на файл

Тип файла и атрибуты

1

2

.

.

12

1

2

.

.

12

1

2

.

.

12

Как было указано выше, inode содержит указатель на блоки данных. В зависимости от размера файла данные содержащиеся в файлах могут быть доступны напрямую через указатели содержащиеся в inode, либо через двойную или тройную ссылку. Первый уровень ссылок позволяет адресовать непосредственно из inode 12 блоков данных, если этого не достаточно для адресации файла соответствующего размера то 12-й блок используется для адресации второго уровня. Размер ссылки составляет 4 байта, поэтому при размере блока в 4096 байт он может адресовать 1024 блока данных. Аналогичным образом осуществляется адресация третьего уровня. При этом ограничение на максимальный размер файла практически снимается. Помимо указателя на блок, inode хранит указатель на фрагмент. Этот указатель может быть интерпретирован как ссылка на целый блок или оддин или несколько его фрагментов. Если обьем данных файла такой что последний блок остается не полностью заполненным то при этом используются фрагмент(ы). Рассмотрим этот случай на примере 20К файла хранимого в 8К блоках. Файл будет хранится в 2-х полных блоках и 4-х фрагментах. Этот случай изображен на рисунке:

20К

 ………

8

24

43

0

………

0

………..

 Размер файла

 8 15 24 31 40 43 46

 1

 2

 блоки адреceсуемые 3

 напрямую с inode

 4

 12

Когда для записи файла требуется блок или фрагмент, система начинает искать свободныйе блоки на диске. Когда файловая система заполненная, выполняется очень долгие линейные поиски для нахождения свободных блоков, и обычно находится блок соседний с тем котрый использовался при записи предыдущего файла. В конце концов это приводит к сильному падению производительности файловой системы. Поэтому для более быстрого поиска свободных блоков на файловой системе резервируется некоторая часть свободного места (minfree). Этот параметр задается при создании файловой системы и может быть изменен в дальнейшем. Как правило это 10% от всего места отведенного под файловую систему.

**Распределение дискового места.**

Свободное место на диске определяется через битовую карту ассоциированную с каждой группой цилиндров. Битовая карта содержит один бит для каждого фрагмента. Для определения свободен ли блок, система проверяет смежные фрагменты. Пример куска битовой карты для файловой системы использующей 1024-х байтные фрагменты и 8192-х байтные блоки показан ниже:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Битовая карта | 00000000 | 00000011 | 11111100 | 11111111 |
| Номера фрагментов | 0-7 | 8-15 | 16-23 | 24-31 |
| Номера блоков | 0 | 1 | 2 | 3 |

Фрагменты с номерами 14-21 в этом примере сободны (отмечены 1), а фрагменты 0-13 и 22-23 уже заняты. Любые восемь подряд идущих фрагментов не могут составлять блок, только восемь фрагментов выровненных по границе блока могут составить блок. HP-UX пытается положить все все файлы находящиеся в одной директории в одну и туже группу цилиндров. Новосозданные директории помещаются в те группы цилиндров где наибольшее количество свободных inodes и наименьшее количество директорий. Если размер файла превішает порог определяемій параметром maxbpg (определяется при создании файловой системі и может меняться в дальнейшем) то HP-UX начинает выделять свободные блоки из другой группы цилиндров. Это позволяет более тесно группировать в одну группу цилиндров файлы находящиеся в одной директории путем размазывания больших файлов по нескольким группам цилинров.

## Модификация файлов в HP-UX

Каждій раз когда происходит запись в файл, данные сначала записываются в буферный кэш находящийся в памяти. Физический диск обновляется ассинхронно по отношению к кэшу. Изменение данных на диске принадлежащие к определенной inode происходит позже, за исключерием если файл біл откріт в синхронном режиме (параметр O\_SYNC O\_SYNCIO в системных вызовах open() и fcntl()). Если система останавливается без сброса буферов на диск то файловая система приходит в сосотояние с нарушеной целостностью. В єтом случае необходимо ее восстановление утилитой fsck. Команда sync может быть использована для принудительного сброса буферов на диск в любой момент времени. Системній демон syncer выполняет периодический сброс буферов на диск. Приведем список изменений происходящих в фаловой системе при выполнении некоторых основных операций над ней:

 Главный суперблок сбрасывается на диск при выполнениии команды

umount или команды sync при условии что файловая

система была модифицирована

 Inodes в зависимости от параметра ядра fs\_async информация

 Обновляется либо синхронно либо ассинхронно по

Отношению к буферному кєшу

 Блоки данных In-core блоки (директории, файлы, пайпы, симлинки,

FIFO) записываются на диск после модификации. Блоки данных файлов буферизируются. Физически запись на диск происходит когда выполняется команда sync или системный вызов fsync() или непосредственно после модификации если на файл при открытии установлен флаг O\_SYNC.

Информация о группе эта информация сбрасывается на диск после цилиндров выполнения sync (fsync).

*Замечание:* команда reboot –n перегружает систему без сброса буферов на диск. Эту команду нужно использовать после выполнения проверки и устранения сбоев корневой файлой системы. Остальные файловые системы необходимо проверять только в отмонтированном состоянии.

## Менджер логических дисков LVM

Перед появлением HP-UX 10.\* управление дисками в сриях HP 800 и HP 700 осуществлялось различным образом. В серии 800 была возможность разбивки диска на жестко определенные партиции, а также управление через LVM. В серии 700 таких возможностей не было, и единственное что можно было использовать – так это использование целого диска для создания файловой системы. С появлением HP-UX 10.\* ситуация радикально изменилась, LVM стал доступен на обеих сериях и является рекомендуемым инструментом для работы с файловыми системами. Он представляет собой псевдодрайвер ядра системы эмулирующий логические диски.

**Что такое Logical Volumes и в каких случаях их следует использовать ?**

Logical Volumes (LV) это набор дисковых участков с одного или более дисков организованных в таком виде, что операционная система видит их как один логический диск. Как и физические дискиони могут быть использованы для поддержки файловых систем, raw областей данных, swap или dump областей. Использование LV оправдано в случаях больших файловых систем которые не умещаются на одном диске и (или) нуждаются в последующем расширении а также в случаях когда необходимо организовать резервирование (зеркалирование) данных или когда к файловой системе предъявляются жесткие требования по производительности.

 Physical Volumes

 (диски)

/dev/dsk/c0t7d0

/dev/dsk/c0t6d0

/dev/dsk/c0t5d0

/dev/vg01

 Volume Groups

 (пул дисков)

./dev/vg01/lvol1

/dev/vg01/lvol2

/dev/vg01/.lvol3

Для использования LVM диски должны быть инициализированны как *physical volumes*. *Physical volumes* идентифицируются именами ссответствующих файлов-устройств дисков /dev/dsk/c*n*t*n*d*n* и/dev/rdsk/c*n*t*n*d*n.* Затем из одного или нескольких дисков собирается *volume group*. Один физический диск может принадлежать только к одной *volume group*. Максимальное число *volume group* которое может быть в системе определяется параметром maxvgs. Одна *volume group* может содержать не более 255 физических дисков. Дисковое пространство из *volume group* распределяется между одной или несколькими *logical volumes*. Дисковое пространство из *logical volumes* может быть использовано для создания файловой системы, под swap или dump области. LVM разбивает каждый диск на набор адресуемых блоков называемых *physical extents*. Их размер определяется во время создания *volume group* и одинаков для всех дисков входящих в *volumes group*. Размер *physical extents* варьируется от 1 до 256 Мб, по умолчанию он равен 4Мб. Базовым блоком для адресации *logical volumes* является *logical extent*, он напрямую отображается в *physical extents*. В HP-UX команды показывающие эти отображения имеют названия *pvdisplay* и *lvdisplay*:

# pvdisplay /dev/dsk/c0t5d0

--- Physical volumes ---

PV Name /dev/dsk/c0t5d0

VG Name /dev/vg00

PV Status available

Allocatable yes

VGDA 2

Cur LV 9

PE Size (Mbytes) 4

Total PE 511

Free PE 81

Allocated PE 430

Stale PE 0

IO Timeout (Seconds) default

# lvdisplay /dev/vg00/lvol1

--- Logical volumes ---

LV Name /dev/vg00/lvol1

VG Name /dev/vg00

LV Permission read/write

LV Status available/syncd

Mirror copies 0

Consistency Recovery MWC

Schedule parallel

LV Size (Mbytes) 48

Current LE 12

Allocated PE 12

Stripes 0

Stripe Size (Kbytes) 0

Bad block off

Allocation strict/contiguous

Если *logical volumes* используется для корневой (root) файловой системы, первичной swap области или dump области, *physical extents* должны распределяться методом *contiguous*. Это означает что между они должны следовать непрерывно на одном физическом диске и между ними не должно возникать прпомежутков. Другие *logical volumes* использующиеся для некорневых файловых систем могут не следовать этому ограничению.

Для определения требуемого объема *logical volume* необходимого для создания файловой системы можно использовать следующую диаграмму:

Место под служебную информацию файловой системы (overhead) ~5%

Место резервируемое файловой системой (minfree) ~10%

Данные хранимые файловой системой

Место требуемое под файловую систему на logical volume

### Управление Logical Volumes (LV)

Системная утилита SAM позволяет выполнять большинство но не все операции над LV. К тем задачам с которыми она справляется можно отнести:

* Создание и удаление volume groups.
* Добавление и удаление дисков из volume groups.
* Создание, удаление и модификация logical volumes.
* Создание и увеличение обьема файловых систем находящихся на logical volumes.
* Cоздание и модификация swap и dump logical volumes.

Для этого, после запуска SAM нужно войти в раздел “Disks and file systems” а затем в один из нужных подразделов. Все дальнейшие действия выполняются с использованием графической оболочки и являются интуитивно понятными. Все тоже самое можно выполнить используя команды HP-UX.

**Создание physical volume (PV).**

 pvcreate /dev/rdsk/c0t6d0

Все данные имеющиеся на этом диске будут потерены, в качестве аргумента программы pvcreate необходимо использовать символьный (raw) файл-устройство диска. Послк инициализации, данный диск можно рассматривать как physical volume.

**Помещение PV в одну из volume groups (VG)**

Если необходимо создать новую VG, то в самом начале нужно сделать директорию для файлов-устройств отвечающих за данную VG:

 mkdir /dev/vg*nn*

 cd /dev/vg*nn*

Затем нужно создать необходимые файлы устройств:

 mknod /dev/vg*nn*/group c 64 0x*NN*0000

В качестве старшего номера устройства всегда нужно использовать 64, 0x*NN*0000 является младшим номером устройств и *NN* представляет собой уникальный среди всех VG номер. Теперь можно приступать к созданию VG:

 vgcreate /dev/vg*nn* /dev/dsk/c0t6d0 …

Вторым (третьим, четвертым …) аргументом этой команды должен быть файл-устройство блочного типа соответствующего physical volume который не является членом какойто из существующих VG.

**Создание Logical Volume.**

lvcreate /dev/vg*NN*

После чего появится блочные и символьные файлы устройств /dev/vg*NN*/lvol*n* и /dev/vg*NN*/rlvol*n.* LVM сам выберет подходящий номер *n*. Для создания LV с именем отличным от того что создается по умолчанию нужно воспользоваться опцией –n. Данный LV будет иметь нулевой размер, в дальнейшем его можно увеличить. Также указав опцию –L можно создать LV заранее необходимого размера (в Мб), при этом реальный размер LV будет округлен в большую сторону и кратен целому количеству physical extents.

**Задачи котоые можно выполнить только с использованием комманд HP-UX**

К ним можно отнести:

* Расширение LV за счет определенного диска
* Создание корневой (root) VG и корневого LV
* Резервное копирование и восстановление конфигурации VG
* Перемещение данных с одного LVM диска на другой
* Уменьшение размера LV

**Расширение LV за счет определенного диска**

Допустим имеется необходимость создать LV на 120Мб, причем первые 60 Мб необходимо взять с одного диска а оставшиеся 60 с другого. Так поступают часто в случаях когда необходимо повысить производительность файловой системы за счет паралельного использования нескольких дисков. Вначале создаем LV нулевого размера:

 lvcreate –n lvol11 /dev/vg00

затем выполняем необходимые расширение ее обьема за счет определенных дисков:

 lvextend –L 60 /dev/vg00/lvol11 /dev/dsk/c0t5d0

 lvextend –L 60 /dev/vg00/lvol11 /dev/dsk/c0t6d0

## Создание корневой VG и корневого LV

Корневой VG это VG который используется системой при загрузке. На нем размещается LV содержащий корневую файловую систему, первичный swap и dump области. Ниже приводятся этапы последовательного создания корневого VG. Во-первых создается PV на котором размещается LIF раздел в котором находятся загрузочные утилиты. Для этого в команде pvcreate используется опция -B:

 pvcreate -B /dev/rdsk/c0t6d0

Создаем корневой LV:

 vgcreate /dev/vgroot /dev/dsk/c0t6d0

Помещаем загрузочные утилиты в LIF область VG:

 mkboot /dev/rdsk/c0t6d0

Записываем в LIF область AUTO файл:

 mkboot –a “hpux (;0)/stand/vmunix” /dev/rdsk/c0t6d0

После выполнения всех этих действий корневая VG готова к созданию на ней LV. Корневой LV должен быть самым первым в этой VG, и следовать сразу за boot областью. Это значит что он должен начинаться с нулевого physical extent. Теперь можно переходить к созданию корневого LV, при его создании нужно включить опцию “смежный LV” (-C) и запретить перемещение bad блоков (-r):

 lvcreate –C y –r n –n root /dev/vgroot

 lvextend –L 160 /dev/vgroot/root /dev/dsk/c0t6d0

В конеце необходимо пометить сосзданный LV как корневой:

 lvlnboot –r /dev/vgroot/root

## Резервное копирование и свосстановление конфигурации Volume Groups

Для создания резервной копии конфигурации VG нужно воспользоваться командой vgcfgbackup. Пежде чем выполнять эту команду нужно убедится что все LV в данной VG находятся в состоянии available/syncd (для этого можно воспользоваться командой vgdisplay –v). По умолчанию команда vgcfgbackup сохраняет конфигурационый файл VG под именем /etc/lvmconf/*volume\_group\_name*.conf. Это имя можно переопределить задав опцию –f.

 Восстановление конфигурации выполняется командой vgcfgrestore. Перед этим необходимо предварительно деактивировать данную VG командой vgchange. Например:

 vgchange –a n /dev/vg01

 vgcfgrestore –n /dev/vg01 /dev/rdsk/c0t6d0

Выполнение этих команд приведет к восстновлению информации о VG vg01 из файла /etc/lvmconf/vg01.conf . Затем необходимо активировать данную VG:

 vgchange –a y /dev/vg01

## Перемещение и переконфигурирование дисков

В жизни могут возникнуть ситуации при которых необходимо:

* Переместить диск входящий в состав VG на другое положение в пределах системы.
* Переместить целую VG с одной системы на другую.

Файл /etc/lvmtab содержит информацию о отображении LVM дисков на соответствующие VG. При любых изменениях связанных с дисками и VG в системе этот файл изменяется, однако это не текстовый файл и напрямую его редактировать нельзя. Вместо этого нужно пользоваться программами vgexport и vgimport.

**Перемещение диска в системе.**

Для перемещения диска в системе на новое место необходимо выполнить следующее:

* Создать резервную копию конфигурации VG в которую входит диск и данных хранящихся на диске
* Деактивировать VG в состав которой входит диск:

vgchange –a n /dev/vgxxx

* Удалить запись ассоциированную с этим диском из /etc/lmvtab а также файлы устройств из каталога /dev/vgxxx

vgexport /dev/vgxxx

* Создать заново VG, и добавить запись в /etc/lvmtab

mkdir */dev/vgxxx*

mknod /*dev/vgxxx/lvolN* c 64 0x010000

vgimport */dev/vgxxx /dev/dsk/cntndn*

* Активировать вньовь ипортированную VG

vgchange –a y /dev/vgxxx

* Создать резервную копию конфигурации VG

vgcfgbackup /dev/vgxxx

### Перемещение диска между системами

Для перемещения диска с одного места на другое между двумя системами необходимо выполнить следующие действия:

* Деактивировать VG

vgchange –a n /dev/vgxxx

* Удалить информацио о VG из /etc/lvmtab указав имя map файла который будет содержать удаляемую информацию (он содержит также имена удаляемых LV из VG)

vgexport –p –v –m plan\_map vgxxx

просмотрев map\_file и убедившись в правильности можно выполнить реальное удаление

 vgexport –v –m plan\_map vgxxx

* Переносим map\_file на новую систему, останавливаем старую систему, переносим диски на новую.
* На новой системе создаем VG

mkdir /dev/vgxxx

cd /dev/vgxxx

mknod /dev/vgxxx c 64 0x080000

* Импортируем новую конфигурацию. Для просмотра используется опция –p, для выполнения реального импорта она должна отстутствовать: (предполагается что в новую систему переносились диски которые распозхнались как /dev/dskc0t2d0 /dev/c0t3d0)

vgimport –p –v –m plan\_map /dev/vgxxx /dev/dskc0t2d0 /dev/c0t3d0

* Активируем VG

vgchange –a y /dev/vgxxx

**Перенос данных на другой physical volume.**

Для переноса данных содержащихся в LV с одного диска на другой необходимо воспользоваться командой pvmove. Например для переноса данных из LV /dev/vg01/lvo1 с диска /dev/dsk/c0t0d0 на /dev/c0t1d0:

 pvmove –n /dev/vg01/lvol1 /dev/dsk/c0t0d0 /dev/c0t1d0

Для полного переноса данных с одного диска на другой можно воспользоваться:

 pvmove /dev/dsk/c0t0d0 /dev/c0t1d0

**Уменьшение обьема Logical Volumes**

Для уменьшения обьема LV используется команда lvreduce. Однако в этом случае данные хранящиеся в файловой системе на этом LV будут потеряны, поэтому необходимо предварительное резервное копирование. Также можно воспользоваться другой техникой – вначале удалить LV командой lvremove, а затем создать LV требуемого размера командой lvcreate.

## 3. Особенности файловой системы VxFS

VxFS является HP-UX реализацией журнальной файловой системы известной как JFS на базе версии корпорации VERITAS. Прежние версии HP-UX (меньше 10.\*) имели в своем арсенале лишь HFS, начиная с 10.01 появилась дополнительная опция на использование VxFS в качестве файловой системы. Однако VxFS не может быть использована как файловая система для раздела /stand из которого происходит зщагрузка ядра. В ранних версиях 10-го релиза HP-UX она так же не могла быть использована для корневой файловой системы. По сравнению с HFS имеет меньшее время восстановление при сбоях и имеет повышеную производительность на больших объемах данных, т.к. блок используемый ею может состоять из множества физических блоков. Т.к. это журнальная файловая система то она позволяет вести online backup. Однако в отличии от HFS она потребляет больше памяти.

В стандартной поставке HP-UX включены лишь базовые элементы VxFS. Дополнительные функциональные возможности возможны только при использовании отдельного пакета под названием HP OnlineJFS.

**4.** **Операции над файловой системой**

Все операции над файловой системой могут быть произведены с использованием системной утилиты SAM или напрямую с использованием команд HP-UX. Файловые системы могут создаваться на LVM и не LVM дисках. Использование LVM дисков является более предпочтительным.

### Создание файловой системы

Для создания файловых систем используют команду newfs. Она имеет следующий синтаксис:

newfs [-F *FStype*] [-o *specific\_options*] [-V] special

где

 *special* имя символьного файла-стройства

 *FSType* тип файловой системы (HFS, VxFS)

 *specific\_options* опции специфические для данного типа файловой системы

Если не указывать опцию –F то тип файловой системы берется из файла /etc/fstab, если в нем нет соответствующей special строки то используется тиип по умолчанию из файла /etc/defaults/fs. При создании HFS возможно использование опций –L (длинные имекна файлов 255 знаков) или –S (короткие имена файлов 14 знаков). В VxFS все имена файлов длинные. После создания файловой системы ее можно монтировать.

## Монтирование и демонтирование файловых систем

Для монтирования файловых систем или получения информации о смонтированных файловых системах используется команда mount. Монтирование заключается в присоединении файловой системы к директории. Примеры использования команды mount:

 mount –a монтирование всех файловых систем указанных в

файле /etc/fstab

mount /tmp монтирование на каталог /tmp соответствующей из /etc/fstab файловой системы.

mount /dev/vg00/lvol1 /tmp монтирование файловой системы находящейся в LV /dev/vg00/lvol1 в каталог /tmp

mount выдача информации о смонтированных файловых системах

После загрузки системы все файловые системы перечисленные в файле /etc/fstab автоматически монтируются. Список всех смонтированных в данный момент файловых систем находится в файле /etc/mnttab.

Для демонтирования файловых систем используется команда umount. Синтаксис ее очень схож с командой mount.

 umount –a монтирование всех файловых систем указанных в

файле /etc/mnttab

umount /tmp размонтирование файловой системы от каталога /tmp

umount /dev/vg00/lvol1 размонтирование файловой системы

Файловая система может быть размонтирована только в том случае если нет процессов использующих ее в данный момент. Для определения какие процессы используют файловую систему применяют команду fuser, например:

 fuser –u /dev/vg00/lvol1 выводит ID всех процессов использующих данную FS

fuser –ku /dev/vg00/lvol1 посылает сигнал завершения всем процессам использующим данную FS

## Проверка файловых систем

Если система некорректно завершила свою работу (пропадание питания, system panic) то все смонтированные на тот момент файловые системы с огромной вероятностью будут содержать ошибки. Для их устранения применяется утилита fsck. Она вызываетс ясистемой после перезагрузки из стартового скрипта /etc/bcheckrc и проверяет все файловые системы из файла /etc/fstab.

**Файловая система HFS**

При проверке файловых систем нужно выполнить следующие шаги:

1. Проверить наличие каталога lost+found на корню проверяемой файловой системы. Если его нет , то создать его.
2. Терминировать все процессы использующие фанную файловую систему.
3. Размонтировать файловую систему командой umount.
4. Запустить fsck –p . Опция –p позволяет устранять большинство проблем без вмешательства со стороны администратора (неинтерактивно). Если fsck находит проблему на решение которой она не может принять решение то fsck заканчивает свою работу с сообщением об ошибке. Если fsck завершила свою работу удачно, значит все повреждения файловой системы были исправлены. В противном случае:
5. Запустить fsck без аргументов –p или –P (интерактивно). При этом возможна потеря данных или имени файлов. После окончания работы fsck перегрузите систему киспользуя команду reboot -n. Если не выполнить такую перезагрузку, то можно заново повредить файловую систему.
6. Проверить содержимое каталога lost+found. В него fsck сбрасывает все данные которые не подлежат восстановлению.

**Файловая система VxFS**

В том случае когда нужно проверить VxFS (разумеется не в случае физического сбоя) нет необходимости проверять всю файловую систему а джостаточно проверить так называемый iintent log содержащий записи всех изменений ожидаемых в структуре файловой системы, который представляет собой не что иное как лог транзакций которые система хотела осуществить перед сбоем. Проверка intent log происходит очень быстро и время восстановления VxFS существенно меньше чем HFS. В случае дискового сбоя сканирование intent log-а неоправдано и в такой ситуации необходимо сканирование всей файловой системы. Для этого нужно запускать fsck –o full.

**Сокращение обьема файловой системы**

Для уменьшения обьема файловой системы нужно выполнить следующие действия:

* Сделать резервную копию файловой системы
* Размонтировать файловую систему
* Уменьшить обьем LV командой lvreduse.
* Создать заново файловую систему командой newfs.
* Смонтировать ее
* Восстановить данные из резервной копии

### Увеличение обьема файловой системы

Для увеличения обьема файловой системы необходимо выполнить следующие действия:

* Размонтировать файловую систему

umount /dev/vg01/lvol01

* Увеличить размер LV (например до суммарного обьема 1.5 Гб)

lvextend –L 1500 /dev/vg01/lvol1

* Увеличить обьем файловой системы:

extendfs /dev/vg01/lvol01

* Смонтировать файловую систему:

mount /dev/vg01/lvol1 /xxx

# Мониторинг использования дискового пространства

Команды df и bdf позволяют вести мониторинг свободного и занятого дискового пространства и числа inodes. Команда df выводит значения в 512 байтных блоках, bdf в килобайтах. Например:

bash-2.04# bdf -i

Filesystem kbytes used avail %used iused ifree %iuse Mounted on

/dev/vg00/lvol3 86016 77263 7875 91% 4566 2078 69% /

/dev/vg00/lvol7 163840 137219 24231 85% 8164 6304 56% /var

/dev/vg00/lvol6 339968 296156 41079 88% 17857 10947 62% /usr

/dev/vg00/lvol4 360448 318130 39684 89% 8867 10573 46% /opt

/dev/vg00/home 204800 186722 8178 96% 13797 2163 86% /home

/dev/vg00/lvol5 32768 2594 28244 8% 204 7524 3% /tmp

/dev/dsk/c0t2d0 381402 381402 0 100% -1 0 100% /cdrom

/dev/vg00/lvol1 47829 21052 21994 49% 25 7655 0% /stand

 Для контроля места занимаемого какойто директорией можно использовать команду du. Наример:

bash-2.04# du -s /usr/contrib/\*

2594 /usr/contrib/X11R6

352 /usr/contrib/X11drivers

5346 /usr/contrib/Xm1.2

11118 /usr/contrib/bin

374 /usr/contrib/doc

0 /usr/contrib/etc

0 /usr/contrib/games

0 /usr/contrib/include

262 /usr/contrib/lib

432 /usr/contrib/man

514 /usr/contrib/src

#  Организация веб-сайта

## Выбор операционной системы.

Выбор операционной системы, прежде всего, определяется тем, что имеется в наличии. В настоящее время www-сервера строятся на основе Windows NT и Unix-системах преимущественно, поскольку платформа OS/2 фактически не используется из-за неудачной маркетинговой политики, а MacOS не предназначена для использования как серверная технология.

Предпочтительнее всего – иметь Unix-подобную систему. Она не намного сложнее в администрировании «навороченного» NT-сервера, несмотря на бытующее мнение, и значительно снижает затраты на ПО, администрирование и поддержку www-сервера, как, например, за счет того, что существует огромное количество бесплатного ПО и ПО с открытым исходным кодом в рамках лицензии GPL (1,2) за которое не придется платить деньги. Кроме того, бесплатность не является основным преимуществом, в первую очередь ПО с открытым исходным кодом получает все большее распространение и признание благодаря именно открытости программного кода и технологий. Получая такое ПО, вы получаете десятки тысяч добровольных разработчиков и тестеров во всем мире. Что это означает. Такое ПО гораздо быстрее обновляется, ошибки и бреши в системе безопасности исправляются значительно быстрее, чем в коммерческих продуктах, и, как это не удивительно, оно является более стабильным. Кроме этого некоторые специфические задачи для веб-сайта можно решить только на основе Unix-систем и ПО с открытым кодом в силу того, что необходимо создавать собственные модули, парсеры и программы и серверные приложения для интерактивной работы с сервером; понятно, что сделать это на основе коммерческого ПО сложно в связи с тем, что вы имеете дело с готовыми бинарными файлами и не имеете возможности их менять. Отрицательные стороны: необходимость постоянно следить за новостями и обновлениями (коммерческие организации рассылают соответствующие бюллетени и CD, здесь – необходимо подписаться на рассылку по электронной почте и самому загружать обновления); второе, необходимы некоторые базовые знания программирования и особенностей операционной системы для установки новой версии и внесения исправлений в ПО. Причем на платформе Windows делать это будет значительно труднее.

## Выбор программного обеспечения сервера.

Под программным обеспечением сервера понимается программа-демон, которая обрабатывает http-запросы, а так же все сопровождающее программное обеспечение: клиенты баз данных, специфические компиляторы, парсеры, анализаторы логов, генераторы статистики, счетчики и т.п.

Итак, ПО веб-сервера в первую очередь – это именно процесс (демон) который обрабатывает запросы и обычно, собственно, и носит название «веб-сервер». Существует достаточно много различных технологий и веб-серверов от разных производителей, но, как и везде, есть бесспорные лидеры. Прежде всего, это Microsoft IIS, Apache, Netscape FastTrack Server, а так же менее известные Stronghold и FHTTPD. Немного статистики: наиболее распространенным сервером в странах СНГ является Apache – около 80% веб-сайтов построены на нем, порядка 15% принадлежит Microsoft IIS, оставшиеся 5% с переменным успехом разделены между другими оставшимися веб-серверами. Для платформы HP-UX прежде всего следует рассмотреть Apache, Netscape FastTrack Server, как совместимые с платформой и возможные к использованию. Недостатком упоминавшихся ранее Stronghold и FHTTPD является следующее, первый – коммерческий с ограниченной лицензией и ограниченной поддержкой новых технологий, второй сервер – является так же и сервером обслуживания FTP-запросов, что значительно снижает безопасность и устойчивость сервера.

Сопутствующее ПО. В самом простейшем случае это различные интерпретаторы, парсеры и/или модули, которые могут понадобиться для создания интерактивного сайта, обработки форм и исполнения приложения. В минимальный комплект как минимум будут входить: компилятор С, интерпретатор языка Perl, которые понадобятся для создания cgi-скриптов, обработки форм и запросов, а так же оживления сайта; кроме этого на сервере можно установить компилятор Java, если в перспективе планируется использовать сервелеты, а так же некоторые другие средства разработки – Tcl/Tk, PHP, если предполагается использование баз данных, то для работы необходимо будет установить, как минимум, клиенты к базам данных и соответствующие библиотеки, как, например, MySQL или PostgresSQL. Кроме того, для оживления сайта, мониторинга посещаемости можно установить программы подсчета посещений и анализа логов (WWWCount, Webalizer), если вы не собираетесь использовать счетчики внешних рейтинговых систем, которые значительно замедляют загрузку страниц сайта, а так же имеют обыкновение не работать в течение длительных периодов времени или работать некорректно.

## Анализ веб-серверов.

Netscape FastTrack Server.

По заверениям компании Netscape это веб-сервер начального уровня пригодный для малых и средних серверов с устойчивым базовым движком, пригодным для создания статических и динамических сайтов, а так же предлагающий простейшие инструменты для работы с базами данных на веб-сайте. Сервер поддерживает HTTP 1.1 и SSL 3 технологии, сервелеты на основе JavaScript 1.1, а так же доступ к ODBC базам данных. Для корпоративных сайтов и крупных серверов предлагается использовать усовершенствование версии до Netscape Enterprise Server, что значительно дешевле, чем изначально покупать лицензию Enterprise Server

Apache. (Apache Software Foundation)

Веб-сервер разрабатывается на основе движка NCSA httpd 1.3, который был переработан Apache Software Foundation, лицензирование – GPL2, с открытым исходным кодом. Этот достаточно мощный сервер, который пригоден как для небольших и средних сайтов, так и для крупных корпоративных проектов в настоящее время используется на 60% веб-сайтов всего мира. Сервер поддерживает HTTP 1.1 и SSL 3 технологии, сервелеты на основе JavaScript 1.1, технологии SSI, так же содержит более 50 модулей, позволяющих реализацию проекта практически любой сложности. Кроме этого, к серверу можно подключить не меньшее количество модулей от сторонних производителей, как, например, FastCGI от OpenMarket Group (разработчиков FHTTPD), что значительно расширяет его возможности. Сервер доступен в вариантах под платформы Windows NT и все Unix-системы.

## Инсталляция веб-сервера.

Инсталляция Netscape FastTrack Server.

Инсталляция Netscape FastTrack Server достаточно проста, поскольку это сугубо коммерческий продукт, поставляемый в бинарном виде, и устанавливается он стандартными средствами, как, например, для HP-UX это с помощью swinstall.

Далее следует вручную исправить в файле /opt/ns-fasttrack/admserv/admpw пароль доступа. Проще всего убрать всякий пароль для пользователя root, тем более, что по умолчанию заход с административными правами разрешен только с localhost. Дальнейшая конфигурация и настройка осуществляется из браузера после запуска ns-admin и не представляет сложностей. Принципиально необходимо настроить практически такие же параметры как и для apache только все это делается через веб-интерфейс.

Инсталляция Apache.

Архитектуру сервера можно описать так:

Apache core

DSO1

DSO2

DSOn

Для инсталляции веб-сервера Apache, вполне достаточно стандартной поставки в Unix-системе, т.е. компилятора СС. Однако, рекомендуется все-таки поставить GCC (Gnu-CC) компилятор, особенно в том случае, если вы собираетесь использовать модули от сторонних производителей, обычно для их инсталляции требуется именно GCC. Для того чтобы сконфигурировать и запустить Apache под Windows NT, можно воспользоваться доступной готовой инсталляцией в традиционном стиле Windows-приложений, если же собирать сервер из исходных кодов, то потребуется инсталляция MS Developer Studio или MS Visual C как минимум.

Рассмотрим процесс инсталляции в Unix-среде. Для того чтобы успешно скомпилировать сервер, необходимо знать основы работы с шеллом и некоторое стандартное программное обеспечение под Unix.

Процесс инсталляции:

1. Развернем дистрибутив apache-x.x.x в какую либо директорию (можно в home)
2. Для сборки сервера без подгружаемых модулей достаточно запустить конфигурационный скрипт с такими ключами:

./configure --with-layout=Apache --prefix=/usr/local/httpd/ --enable-shared=max --enable-suexec --suexec-caller=#65533 --suexec-docroot=/usr/local/httpd/ --suexec-logfile=/usr/local/httpd/logs/suexec\_log --suexec-safepath=/usr/local/httpd/cgi-bin --sbindir=/usr/local/httpd/bin

1. После того как отработает конфигурационный скрипт, достаточно набрать make, потом make install и можно приступать к конфигурированию сервера.

Для проверки правильности компиляции следует запустить

/usr/local/httpd/bin/httpd -l

Результат будет приблизительно таким:

Compiled-in modules:

 mod\_env.c

 http\_core.c

 mod\_so.c

suexec: enabled; valid wrapper /usr/local/apache/bin/suexec

2а. Для сборки сервера в идеальном варианте, когда сам сервер содержит только suexec wrapper, http\_core и статически вкомпилированную библиотеку поддержки подгружаемых модулей mod\_so, необходимо запустить скрипт автоконфигурации configure с такими ключами (для удобства приведен пример скрипта, который выполняется из шелла, поскольку более полусотни ключей неудобно набирать, особенно при возникновении опечаток)

./configure \

"--with-layout=Apache" \

"--enable-module=so" \

"--prefix=/usr/local/httpd/" \

"--enable-module=env" \

"--enable-shared=env" \

"--enable-module=setenvif" \

"--enable-shared=setenvif" \

"--enable-module=mime" \

"--enable-shared=mime" \

"--enable-module=negotiation" \

"--enable-shared=negotiation" \

"--enable-module=alias" \

"--enable-shared=alias" \

"--enable-module=rewrite" \

"--enable-shared=rewrite" \

"--enable-module=userdir" \

"--enable-shared=userdir" \

"--enable-module=dir" \

"--enable-shared=dir" \

"--enable-module=autoindex" \

"--enable-shared=autoindex" \

"--enable-module=access" \

"--enable-shared=access" \

"--enable-module=auth" \

"--enable-shared=auth" \

"--enable-module=auth\_anon" \

"--enable-shared=auth\_anon" \

"--enable-module=auth\_dbm" \

"--enable-shared=auth\_dbm" \

"--enable-module=auth\_db" \

"--enable-shared=auth\_db" \

"--enable-module=digest" \

"--enable-shared=digest" \

"--enable-module=headers" \

"--enable-shared=headers" \

"--enable-module=cern\_meta" \

"--enable-shared=cern\_meta" \

"--enable-module=expires" \

"--enable-shared=expires" \

"--enable-module=asis" \

"--enable-shared=asis" \

"--enable-module=include" \

"--enable-shared=include" \

"--enable-module=cgi" \

"--enable-shared=cgi" \

"--enable-module=actions" \

"--enable-shared=actions" \

"--enable-module=status" \

"--enable-shared=status" \

"--enable-module=info" \

"--enable-shared=info" \

"--enable-module=log\_config" \

"--enable-shared=log\_config" \

"--enable-module=log\_agent" \

"--enable-shared=log\_agent" \

"--enable-module=log\_referer" \

"--enable-shared=log\_referer" \

"--enable-module=usertrack" \

"--enable-shared=usertrack" \

"--enable-module=imap" \

"--enable-shared=imap" \

"--enable-module=proxy" \

"--enable-shared=proxy" \

"--enable-module=mmap\_static" \

"--enable-shared=mmap\_static" \

"--enable-suexec" \

"--suexec-caller=#65533" \

"--suexec-docroot=/usr/local/httpd/" \

"--suexec-logfile=/usr/local/httpd/logs/suexec\_log" \

"--suexec-safepath=/usr/local/httpd/cgi-bin" \

"--sbindir=/usr/local/httpd/bin" \

3а. После того как отработает конфигурационный скрипт, достаточно набрать make, потом make install и можно приступать к конфигурированию сервера.

Для проверки правильно ли скомпилирован сервер запустим его с ключом –l:

/usr/local/httpd/bin/httpd -l

Соответствующий правильному результат будет следующий:

Compiled-in modules:

 http\_core.c

 mod\_so.c

suexec: enabled; valid wrapper /usr/local/apache/bin/suexec

## Стратегическое планирование. Определение объема работы.

Концепция:

1. Опрос клиента: вопросники, профили посетителей
2. Стратегический план: задачи маркетинга, обзор конкурентов,
3. требования к рекламе, потребности пользователей ресурса.
4. Функциональный план: технические задачи, задачи функциональности.

Административные задачи:

1. Выбор команды: продюсер, арт-директор/дизайнер, автор/эдитор, программист.
2. Расписание и бюджет проекта.

Многие приступают к работе над новым сайтом с выбора цветов, шрифтов, технологий и т.п. Тогда как логичнее начать с анализа проекта, планирования предстоящей работы и поиска ответов на ряд важных вопросов, касающихся будущего сайта. Время, потраченное на сбор и обработку нужной информации, окупится с лихвой, помогая избежать потери денег и энергии в дальнейшем.

1. Вы получили деловое предложение. В первую очередь постарайтесь узнать как можно больше о вашем заказчике, о его фирме; сферу бизнеса, в котором он работает, и занимаемую в ней нишу.

2. Получите у него ответы на следующие вопросы:

* Зачем ему нужен сайт? Почему он его создает и какие цели и задачи преследует?
* - Желает ли он моментальной отдачи или подготавливает почву для более агрессивного бизнеса в Сети в дальнейшем?
* - Что входит в его цели: продать продукцию или сервис, увеличить покупательский рынок, обеспечить поддержку уже проданному товару?
* - Возможно, он надеется подняться на уровень конкурентов, которые давно имеют свои ресурсы в Интернет?
* - Если бы у него была возможность донести только одну идею до своих пользователей, какой бы она была?
* - Какие дальнейшие действия ожидает он от посетителей его Веб-страниц? Обращение за более полной информацией? Визит в магазин или офис, покупка напрямую с сервера?
* - Как заказчик измерит успех или неудачу сайта? Как узнать, что сайт получился удачным - понравилось начальнику? Выиграл Сетевые награды? Популярен в среде пользователей?
* - Кого он хочет видеть своим посетителем? (Дизайн и содержание сайта должны оправдывать ожидания и удовлетворять потребности пользователя)
* - Почему к нему будут приходить? За какой информацией?
* - А какую информацию он может предоставить? Одна из распространенных ошибок профессионалов - это убежденность в том, что пользователи достаточно просвещены в вопросах, касающихся деятельности фирмы. Вероятнее всего, это не так. Маркетинг заказчика только выиграет от результатов обучения возможных клиентов. Ваша с ним задача - выбрать из моря информации ту, в которой нуждается посетитель.
* - Интересуется ли он привлечением одноразовых посетителей (для счетчика) или надеется превратить случайных прохожих в регулярных пользователей? Лояльность требует частого обновления материалов.

3. Узнайте, кто является конкурентом заказчика. Как выглядят их Интернет ресурсы. Очень важно, чтобы сайт клиента не подражал другим, а представлял уникальный и последовательный в себе имидж. Составьте документ с подробным описанием того, что вы знаете о данной индустрии в Интернете. Сделайте выводы - что работает и что не работает для схожей группы пользователей на других сайтах. Что бы вы могли использовать на сайте заказчика, и что использовать не стоит.

4. Когда вы утвердите профиль будущего посетителя (см. вопросы выше), составьте список возможных требований (например, основной группой пользователей будет молодежь, которая дозванивается в Интернет через слабый модем и терпеть не может ждать загрузки страницы дольше, чем 30 сек.) и пожеланий (они хотели бы иметь возможность обмениваться мнениями на доске или болтать в чате) различных посетительских групп.

1. Следующие, решите вопросы по функциональности ресурса.
* Что заказчик находит необходимым для его сайта? К примеру, динамические страницы, генерируемые базой данных, веб-коммерция, каталоги, программы? Сделать-то можно все, что он пожелает, дело в стоимости и сроках исполнения.
* - Спросите, кто будет мониторить и обновлять подобные функции?
* Имеет ли заказчик неординарные требования к безопасности?
* - Как и где будет хоститься сайт?
* - Существуют ли какие-либо исключения?
* - Кто будет обновлять и поддерживать ресурс?
* - Какие у заказчика планы на будущее, касательно Веб-сайта?

6. Технические спецификации будут служить основой всех скриптов, css, графики, html, java и других технических элементов сайта. Определите, какое программное обеспечение потребуется пользователям для просмотра ресурса, а также общий технический подход к разработке и поддержке сайта. Хотелось бы знать резолюцию монитора посетителя, скорость их модема, сколько памяти в его системе, количество воспроизводимых цветов, какие плагинсы (программные модули) имеют и т.д. Подобную информацию можно найти в уже существующей у заказчика базе данных пользователей, из интервью, опросов клиентов заказчика, или обычных логических предположений.

7. Вы должны договориться с заказчиком о бюджете проекта, сроках исполнения (если важно) и расписании. Одновременно стоит подумать над тем, какие специалисты вам понадобятся. Возможно, вас не затруднит создать небольшой сайт своими силами, но для серьезного проекта лучше воспользоваться знаниями и опытом професcионалов.

Успех Веб-ресурса покоится на равновесии между достижением желаемых результатов и затраченных в процессе средств. Не стоит пытаться прыгнуть выше головы; потеря времени и денег - результат суперожиданий и нереальных запросов. При планировании, в первую очередь, стоит принимать во внимание нужды будущих посетителей, а не последние достижения технологии и эгоцентричный энтузиазм разработчиков. Мы не создаем сайты для себя, мы создаем их для наших посетителей.

## Тактическое планирование сайта. Разработка структуры.

1. Карта сервера
2. Подготовка контекста, создание и эдитирование
3. Обозрение материалов
4. Технические спецификации (тактическая сторона)
5. Навигация и разметка страниц

Имея на руках результаты опросов заказчика и профили будущих посетителей, пришло время шевелить мозгами в поисках идей, которые помогут веб-сайту достичь поставленных перед ним задач.

Первый ваш шаг - набросок "карты сайта". Это графическая диаграмма, показывающая путь продвижения посетителя по сайту. Ваша карта должна включать в себя каждую его страницу и то, в какой зависимости они находится относительно других страниц и материалов. Очень важно получить "добро" заказчика на "карту", и отметить любые изменения в структуре, такие как удаление или добавление страниц, что может повлиять на бюджет проекта.

Планирование контекста происходит в тесном сотрудничестве с заказчиком. Возможно, у него уже есть готовые материалы, другие должны быть модифицированы, переписаны или созданы с нуля. Необходимо четко разграничить обязанности между вами. Кто будет ответственен за что? Если заказчик поставляет весь материал, составьте ему расписание. Передача текстов вечно занимает больше времени, чем мы рассчитываем и часто именно заказчик виновен в растягивании сроков проекта. Определитесь с текстом для каждой страницы до начала работы над дизайном. Навигация сайта будет напрямую зависеть от количества и глубины содержания, а его стиль и тон окажут влияние на визуальный язык композиции.

Тактическую сторону технических спецификаций можно отразить в коротком документе, определяющим подход и используемые технологии при кодировании страниц и визуальной разметке.

* Будут ли страницы генерироваться "на лету" из датабазы?
* - Будут ли использованы Cascading Style Sheets (CSS)?
* - Потребуется ли наличие модуля (plug-in) или специального контроля на машине пользователя?
* - Под какой минимум необходимо оптимизировать цветовую палитру?
* - Вы будете писать код сами или использовать WYSIWYG "Что вижу, то и получаю" эдитор (Dreamweaver, к примеру)?
* - Будут ли использованы DHTML, Channels, Push технологии?

Затем вы должны определиться с навигацией. Как будет посетитель продвигаться по сайту? Какая связь существует между страницами, куда можно попасть от куда и т.д. Для большинства проектов нет нужды придумывать новые навигационные схемы. Учитесь на сайтах, которые вам нравятся больше всего: проще - лучше. Подгоните существующую удачную модель под нужды своего проекта и используйте ее. Людям необходима возможность легко передвигаться по сайту, только и всего.

## Дизайн интерфейса.

1. Творческие поиски и разработка стиля
2. Создание прототипов
3. Утверждение заказчиком
4. Графическое производство: дизайн датабаз и форм, анимации, графика в GIF/Jpeg, обработка в html.

Вы изучили своего заказчика, представляете содержание каждой будущей страницы и вовсю идет работа над текстами. Самое время заняться интерфейсом. Начните с самой важной страницы, работайте над ней до тех пор, пока у вас не будет как минимум 15 набросков различных вариантов, из которых 5 можно развить дальше. Когда страничка начнет дымиться, поработайте над другими. Продолжайте искать идеи, рисовать и думать. Попробуйте поэксперементировать с различными эффектами: real audio/vidio, динамическое html, необычное использование тегов и т.д.

Пусть ваш альбом наполнится зарисовками для входного экрана, навигационных элементов, ключевой графики и второстепенных, украшающих элементов. Используйте коллаж. Проанимируйте частицу большого изображения. Поиграйте со шрифтами, слоями, текстурами, эмоциями. Как только вы найдете удачную тему, проработайте ее как можно глубже, потом отложите в сторону и примитесь за новую.

Вообразите себя пользователем и попробуйте смотреть на окружающий мир, Интернет его глазами. Походите по сайтам, где он может бывать, полистайте его журналы, пообщайтесь с его возможным окружением...

Используйте графический редактор. Очень удобно скопировать окно браузера (F13 или Print Scrn, далее Ctrl+double click в окне графического редактора, затем Ctrl+V в новый файл) в Фотошоп или Иллюстратор и накладывать слои композиции прямо на него.

Выберите 3 законченных варианта: ваш любимый, несколько экстравагантный и консервативный, который, вы уверены, понравится заказчику. Создайте для них прототипы, которые он сможет посмотреть и одобрить. Прототип - черновой вариант слинкованных между собой основных страниц сайта. Не обязательно писать код для этих страниц, вы прекрасно можете обойтись image-maps. Пореже страницы на части и пролинкуйте графику между собой. Не тратьте время на ее оптимизацию, и объясните заказчику, что она немного потеряет в качестве при дальнейшей обработке. Задача прототипов - хорошо смотреться в окне браузера и понравится заказчику.

## Программирование.

1. Создание: CGI, Javascript, Java, формы, датабазы
2. HTML: кодирование
3. Валидация: содержания, грамматических ошибок, работы линков
4. Тестирование в различных браузерах.

После того как вы разработали дизайн и сверстали модели основных страниц, можно приниматься за техническую работу - HTML кодирование, CGI формы, датабазы и прочее программирование.

Несколько советов:

1. Когда над сайтом работают несколько человек, обязательно стоит договорится об общей логической системе наименований. Если каждый будет скидывать свои творения куда захочется, это будет существенно затруднять поиск коллегам, и в конце все равно приведет к переименованию.

2. Рассортируйте ваши графические файлы по классам в зависимости от их размера. Старайтесь держать количество таких классов минимальным. Например: заголовки, подзаголовки, "ноготки", полная графика и т.д. Графические размеры от 1 до 5.

3. Подберите оптимальную палитру для графики в одном отдельно взятом классе и используйте ее на всех файлах данного класса. Работая над большим сайтом удобно использoвать DeBabelizer (http://www.equilibrium.com), сценарии которого позволяют очень быстро и точно процессить большое кол-во файлов. Так же, он, в отличие от Photoshop (http://www.adobe.com), не сдвигает цвета "безопасной веб-палитры" в сторону при оптимизации.

4. Используйте <!-- комментарии --> в своем коде, оставляя пояснения для программистов и создателей текстов.

5. Используйте абсолютно схожий код на различных страницах, для облегчения глобального поиска и изменений.

6. Если на странице очень много текста и других материалов, удалите все "возвраты каретки" и коменты из кода, что существенно снизит размер файла.

7. Потратьте время на изучение тех программ, с которыми работаете, ваша продуктивность резко повысится. Следите за выходом новых версий и обновляйте свои инструменты регулярно.

8. Делайте копии (Back up?) того, что сделали каждый день по окончанию работы.

9. Проверяйте свою работу на разных браузерах и основных ОС. Поменяйте настройки в браузерах выставив большие или маленькие размеры шрифтов, изменив цветовые настройки, откройте все меню и бары которые имеются.

10. Проверяйте:

* Грамматику
* - Правильное название файлов
* - Работу линков
* - Название страниц в поле Титул
* - Фоновые цвета
* - Целостность и тегов
* - Визуальное выравнивание
* - Что получится при изменении размеров окна браузера, шрифтов, цветов?
* - Различия резолюций. А как оно будет смотреться в монохроме?

11. Тестируйте, тестируйте и еще раз тестируйте. Подключайте как можно больше людей к этому процессу. Для заказчика это отличная возможность "поучаствовать" в проекте. Если есть возможность проверить на будущих посетителях, не упускайте ее. Внимательно следите как люди со свежим взглядом на сайт реагируют на вашу навигацию и систему наименований. Не пытайтесь помочь им "найти" что-либо, но попросите комментировать каждый шаг. Например, куда они ожидают попасть при клике на ту или иную кнопку? Часто бывает очень удобно распечатать весь сайт на бумаге и проверять стилистические и грамматические ошибки таким образом.

Сравнительная таблица популярности браузеров (по данным StatMarket за Январь 2001)

 Microsoft 68.94% Netscape 29.09% Other 1.96


## Публикация и Маркетинг.

Публикация:

1. Выбор хост-провайдера, покупка или регистрация имени домейна
2. FTP веб-сайта на сервер: создание дерикторий, установка permissions
3. Последние проверки и валидации

Маркетинг:

1. Регистрация сервера в основных поисковиках
2. Раскрутка
3. Поддержка: страница новостей, автоматические обновления
4. Презентация сервера и Празднование столь знаменательного события

Последние работы над сайтом могут включать следующее:

* Создание банеров/заголовков про запас
* - Добaвление последнего контекста
* - Запуск CGI сценариев
* - обновление линков (старайтесь всегда давать условные пути к документам, а не полный URL. К примеру, ../images/dot.gif вместо http://www.yoursite.ru/images/dot.gif Таким образом вам не придется менять данные во всех ссылках при переносе сайта на сервер нового провайдера, если в будущем возникнет подобная необходимость.)
* - написание мета-тегов
* - добавление авторских прав и предупреждений
* - убедитесь, что графика везде имеет атрибуты width и height, соответствует основной цветовой палитре, имеет "Alt" пояснения.

При выборе имени домейна, старайтесь выбирать простые и легко запоминающиеся слова. Следуйте следующим правилам:

* *Краткость и Простота.* Короткие адреса легче запоминаются, в них сложнее сделать ошибку: Gazeta.ru, Pupkin.com
* - *Описание.* Хорошо, когда по адресу можно догадаться о содержании сайта. Имя компании не всегда является достаточным пояснением: beerclub.com тумана не напустит

- *Запоминаемость.* Лучшие адреса добавляют к простоте элемент интереса - частенько игру слов, как в www.anekdotov.net - который поможет посетителям помнить это название через часы, дни и месяцы. Идеально, когда адрес можно запомнить, услышав в разговоре или по радио. Скопируйте структуру сайта со своей машины на сервер заказчика и еще раз протестируйте каждую страничку в брaузере. Конфигурация чужого сервера может потребовать изменений в сценариях или расширениях файлов (htm вместо html). Убедитесь все работает так, как вы предполагали.

Проведите хорошую рекламную компанию своему сайту.

# Веб-страницы и веб-приложения.

## Классификация веб-объектов.


## Спецификация DTD. Понятие ортогональности и методы ее реализации.

Согласно спецификации W3C минимальный html-документ обязательно должен содержать всего два тега, без которых вообще не является никаким документом. Это теги !DOCTYPE (DTD) и TITLE. Почему настолько важен тег содержащий DTD? Именно тег !DOCTYPE определяет то, что документ является HTML, а не, например, XML-файлом.

Пример:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"

 "http://www.w3.org/TR/REC-html40/strict.dtd">

DTD – Document Type Definition – определяет тип и принадлежность документа, а так же его версия и некоторые другие параметры, как например, в спецификации DTD полностью описаны все теги HTML 4.2 менее чем на 20 страницах.

Ортогональность содержания и представления информации подразумевает собой независимость содержания информации от способа ее представления. Говоря упрощенным языком, это ситуация, когда информативность или информативная часть документа не зависит от способа ее отображения, дизайна, размещения, стилей, шрифтов и т.п. К сожалению, практически ни один из существующих на данный момент языков разметки не обладает в чистом виде таким свойством, за исключением XML, который наиболее близок к этому, но все равно еще в значительной мере остается на стадии разработки. Например, в том же языке HTML таблица может использоваться для предоставления информации именно как таблица, но может быть использована так же для размещения элементов дизайна в необходимом порядке и относительном расположении на странице.

## Веб-страницы. Языки разметки. (HTML, XML)

1. Основы HTML

В HTML 4.2 включены механизмы поддержки таблиц стилей, скриптов, внедрения объектов, улучшена поддержка направления текста справа налево и смешанного текста, а также внесены изменения в формы с целью обеспечения удобства работы пользователей с физическими недостатками.

HTML-документ имеет достаточно строго структурирован:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>

<head>

 <title>Untitled</title>

</head>

<body>

…

</body>

</html>

 СПИСОК ЭЛЕМЕНТОВ HTML

1. [**базисные элементы**](#general) (все документы на HTML должны содержать их)
2. [**определение структуры**](#structure) (вид задается параметрами программы-просмотрщика)
3. [**внешний вид**](#presentation) (автор определяет внешний вид текста)
4. [**ссылки и графика**](#links)
5. [**разделители**](#dividers)
6. [**списки**](#lists)
7. [**фон и цвета**](#color)
8. [**специальные символы**](#special)
9. [**формы**](#forms)
10. [**таблицы**](#tables)
11. [**фреймы (frames)**](#frames)
12. [**язык Java**](#java)
13. [**разное**](#misc)

**Внимание:** Если вам не ясна разница между [HTML 2.0](file:///C%3A%5Ccontent%5Cmarkup_specs%5Carticle-8.html), HTML 3.0, [HTML 3.2](file:///C%3A%5Ccontent%5Cmarkup_specs%5Carticle-7.html), и дополнениями Netscape, я советую прочитать [комментарии W3C](http://www.w3.org/pub/WWW/MarkUp/Activity) о развитии языка HTML.

УПОТРЕБЛЯЕМЫЕ СИМВОЛЫ

URL URL (адрес) внешнего файла (или просто имя файла в том же каталоге)

? Любое число (т.е. <H?> значит <H1>, <H2>, <H3>, и т.д.)

% Любой процент (т.е. <hr align=left width="%>" значит <hr align=left width="50%>", и т.д.)

\*\*\* Любой текст (т.е. ALT="\*\*\*" значит заполните текст)

$$$$$$ Любое 16ричное число (т.е. BGCOLOR="#$$$$$$" значит BGCOLOR="#00FF1C", и т.д.)

,,, Разделенное запятыми (т.е. COORDS=",,," значит COORDS="0,0,50,50", и т.д.)

| Альтернативы (т.е. ALIGN=LEFT|RIGHT|CENTER значит один из них)

СОВМЕСТИМОСТЬ

*(помните, HTML эволюционирует и просмотрщики бывают разные)*

 (без отметки) Включено в HTML 3.2;

 должно работать с последними версиями просмотрщиков

N1.0 дополнение Netscape, введенное в Navigator'е версии 1.0

N1.1 дополнение Netscape, введенное в Navigator'е версии 1.1

N2.0 дополнение Netscape, введенное в Navigator'е версии 2.0

N3.0b дополнение Netscape, введенное в Navigator'е версии 3.0 бета

|  |
| --- |
| **БАЗИСНЫЕ ЭЛЕМЕНТЫ** |
|  | **Тип документа** | <HTML></HTML> | (начало и конец файла) |
|  | **Имя документа** | <TITLE></TITLE> | (должно быть в заголовке) |
|  | **Заголовок** | <HEAD></HEAD> | (описание документа, например его имя) |
|  | **Тело** | <BODY></BODY> | (содержимое страницы) |

|  |
| --- |
| **ОПРЕДЕЛЕНИЕ СТРУКТУРЫ** |
|  | **Заглавие** | <H?></H?> | (стандарт определяет 6 уровней) |
|  | **с выравниванием** | <H? ALIGN=LEFT|CENTER|RIGHT></H?> |  |
|  | **Секция** | <DIV></DIV> |  |
|  | **с выравниванием** | <DIV ALIGN=LEFT|RIGHT|CENTER></DIV> |  |
|  | **Цитата** | <BLOCKQUOTE></BLOCKQUOTE>  | (обычно выделяется отступом) |
|  | **Выделение** | <EM></EM> | (обычно изображается курсивом) |
|  | **Дополнительное выделение** | <STRONG></STRONG> | (обычно изображается жирным шрифтом) |
|  | **Отсылка, цитата** | <CITE></CITE> | (обычно курсив) |
|  | **Код** | <CODE></CODE> | (для листингов кода) |
|  | **Пример вывода** | <SAMP></SAMP> |  |
|  | **Ввод с клавиатуры** | <KBD></KBD> |  |
|  | **Переменная** | <VAR></VAR> |  |
|  | **Определение** | <DFN></DFN> | (часто не поддерживается) |
|  | **Адрес автора** | <ADDRESS></ADDRESS> |  |
|  | **Большой шрифт** | <BIG></BIG> |  |
|  | **Маленький шрифт** | <SMALL></SMALL> |  |

|  |
| --- |
| **ВНЕШНИЙ ВИД** |
|  | **Жирный** | <B></B> |  |
|  | **Курсив** | <I></I> |  |
| N3.0b | **Подчеркнутый** | <U></U> | (часто не поддерживается) |
|  | **Перечеркнутый** | <STRIKE></STRIKE> | (часто не поддерживается) |
| N3.0b | **Перечеркнутый** | <S></S> | (часто не поддерживается) |
|  | **Верхний индекс** | <SUB></SUB> |  |
|  | **Нижний индекс** | <SUP></SUP> |  |
|  | **Печатная машинка** | <TT></TT> | (изображается как шрифт фиксированной ширины) |
|  | **Форматированый** | <PRE></PRE> | (сохранить формат текста как есть) |
|  | **Ширина** | <PRE WIDTH=?></PRE> | (в символах) |
|  | **Центрировать** | <CENTER></CENTER> > | (как текст, так и графика) |
| N1.0 | **Мигающий** | <BLINK></BLINK> | (наиболее осмеянный элемент) |
|  | **Размер шрифта** | <FONT SIZE=?></FONT> | (от 1 до 7) |
|  | **Изменить размер шрифта** | <FONT SIZE="+|-?"></FONT> |  |
| N1.0 | **Базовый размер шрифта** | <BASEFONT SIZE=?> | (от 1 до 7; по умолчанию 3) |
|  | **Цвет шрифта** | <FONT COLOR="#$$$$$$"></FONT> |  |
| N3.0b | **Выбор шрифта** | <FONT FACE="\*\*\*"></FONT> |  |
| N3.0b | **Многоколоночный текст** | <MULTICOL COLS=?></MULTICOL> |  |
| N3.0b | **Пробел между колонками** | <MULTICOL GUTTER=?></MULTICOL> | (по умолчанию 10 точек) |
| N3.0b | **Ширина колонки** | <MULTICOL WIDTH=?></MULTICOL> |  |
| N3.0b | **Пустой блок** | <SPACER> |  |
| N3.0b | **Тип пустого блока** | <SPACER TYPE=horizontal| vertical|block> |  |
| N3.0b | **Величина пустого блока** | <SPACER SIZE=?> |  |
| N3.0b | **Размеры пустого блока** | <SPACER WIDTH=? HEIGHT=?> |  |
| N3.0b | **Выравнивание** | <SPACER ALIGN=left|right|center> |  |

|  |
| --- |
| **ССЫЛКИ И ГРАФИКА** |
|  | **Ссылка** | <A HREF="URL"></A> |  |
|  | **Ссылка на закладку** | <A HREF="URL#\*\*\*"></A> | (в другом документе) |
| <A HREF="#\*\*\*"></A> | (в том же документе) |
| N2.0 | **На другое окно** | <A HREF="URL" TARGET="\*\*\*| |\_blank|\_self|\_parent|\_top"></A> |  |
|  | **Определить закладку** | <A NAME="\*\*\*"></A> |  |
|  | **Отношение** | <A REL="\*\*\*"></A> | (часто не поддерживается) |
|  | **Обратное отношение** | <A REV="\*\*\*"></A> | (часто не поддерживается) |
|  | **Графика** | <IMG SRC="URL"> |  |
|  | **Выравнивание** | <IMG SRC="URL" ALIGN=TOP|BOTTOM|MIDDLE|LEFT|RIGHT> |  |
| N1.0 | **Выравнивание** | <IMG SRC="URL" ALIGN=TEXTTOP| ABSMIDDLE|BASELINE|ABSBOTTOM> |  |
|  | **Альтернатива** | <IMG SRC="URL" ALT="\*\*\*"> | (выводится если картинка не изображается) |
|  | **Карта** | <IMG SRC="URL" ISMAP> | (нужна также программа) |
|  | **Локальная карта** | <IMG SRC="URL" USEMAP="URL"> |  |
|  | **Определение карты** | <MAP NAME="\*\*\*"></MAP> |  |
|  | **Области карты** | <AREA SHAPE="RECT" COORDS=",,," HREF="URL"|NOHREF> |  |
|  | **Размеры** | <IMG SRC="URL" WIDTH=? HEIGHT=?> | (в точках) |
|  | **Окантовка** | <IMG SRC="URL" BORDER=?> | (в точках) |
|  | **Отступ** | <IMG SRC="URL" HSPACE=? VSPACE=?> | (в точках) |
| N1.0 | **Заменитель в низком разрешении** | <IMG SRC="URL" LOWSRC="URL"> |  |
| N1.1 | **Обновить** | <META HTTP-EQUIV="Refresh" CONTENT="?; URL=URL"> |  |
| N2.0 | **Включить объект** | <EMBED SRC="URL"> | (вставить объект в страницу) |
| N2.0 | **Размер объекта** | <EMBED SRC="URL" WIDTH=? HEIGHT=?> |  |

|  |
| --- |
| **РАЗДЕЛИТЕЛИ** |
|  | **Параграф** | <P></P> | (закрывать элемент часто не обязательно) |
|  | **Выравнивание** | <P ALIGN=LEFT|CENTER|RIGHT></P>  |  |
|  | **Новая строка** | <BR> | (одиночный перевод строки) |
|  | **Убрать выравнивание** | <BR CLEAR=LEFT|RIGHT|ALL> |  |
|  | **Горизонтальный разделитель** | <HR> |  |
|  | **Выравнивание** | <HR ALIGN=LEFT|RIGHT|CENTER> |  |
|  | **Толщина** | <HR SIZE=?> | (в точках) |
|  | **Ширина** | <HR WIDTH=?> | (в точках) |
| N1.0 | **Ширина в процентах** | <HR WIDTH="%"> | (в процентах от ширины страницы) |
|  | **Сплошная линия** | <HR NOSHADE> | (без трехмерных эффектов) |
| N1.0 | **Нет разбивки** | <NOBR></NOBR> | (запрещает перевод строки) |
| N1.0 | **Перенос** | <WBR> | (где разбивать строку для переноса при необходимости) |

|  |  |
| --- | --- |
| **СПИСКИ** |  |
|  | **Неупорядоченный** | <UL><LI></UL> | (<LI> перед каждым элементом) |  |
|  | **Компактный** | <UL COMPACT></UL> |  |  |
|  | **Тип метки** | <UL TYPE=DISC|CIRCLE|SQUARE> | (для всего списка) |  |
| <LI TYPE=DISC|CIRCLE|SQUARE> | (этот и последующие) |  |
|  | **Нумерованный** | <OL><LI></OL> | (<LI> перед каждым элементом) |  |
|  | **Компактный** | <OL COMPACT></OL> |  |  |
|  | **Тип нумерации** | <OL TYPE=A|a|I|i|1> | (для всего списка) |  |
| <LI TYPE=A|a|I|i|1> | (этот и следующие) |  |
|  | **Первый номер** | <OL START=?> | (для всего списка) |  |
| <LI VALUE=?> | (этот и следующие) |  |
|  | **Список определений** | <DL><DT><DD></DL> | (<DT>=термин, <DD>=определение) |  |
|  | **Компактный** | <DL COMPACT></DL> |  |  |
|  | **Меню** | <MENU><LI></MENU> | (<LI> перед каждым элементом) |  |
|  | **Компактное** | <MENU COMPACT></MENU> |  |  |
|  | **Каталог** | <DIR><LI></DIR> | (<LI>перед каждым элементом) |  |
|  | **Компактный** | <DIR COMPACT></DIR> |  |  |

|  |
| --- |
| **ФОН И ЦВЕТА** |
|  | **Фоновая картинка** | <BODY BACKGROUND="URL"> |  |
|  | **Цвет фона** | <BODY BGCOLOR="#$$$$$$"> | (порядок: красный/зеленый/синий) |
|  | **Цвет текста** | <BODY TEXT="#$$$$$$"> |  |
|  | **Цвет ссылки** | <BODY LINK="#$$$$$$"> |  |
|  | **Пройденная ссылка** | <BODY VLINK="#$$$$$$"> |  |
|  | **Активная ссылка** | <BODY ALINK="#$$$$$$"> |  |
|  |

|  |
| --- |
| **СПЕЦИАЛЬНЫЕ СИМВОЛЫ*(обязаны быть в нижнем регистре)*** |
|  | **Специальный символ** | &#?; | (где ? это код ISO 8859-1) |
|  | **<** | &lt; |  |
|  | **>** | &gt; |  |
|  | **&** | &amp; |  |
|  | **"** | &quot; |  |
|  | **Торговая марка ТМ** | &reg; |  |
|  | **Copyright** | &copy; |  |
|  | **Неразрывный пробел** | &nbsp; |  |
|  |

|  |
| --- |
| **ФОРМЫ**  |
|  | **Определить форму** | <FORM ACTION="URL" METHOD=GET|POST></FORM> |  |
| N2.0 | **Посылка файла** | <FORM ENCTYPE="multipart/form-data"></FORM> |  |
|  | **Поле ввода** | <INPUT TYPE="TEXT|PASSWORD|CHECKBOX|RADIO| IMAGE|HIDDEN|SUBMIT|RESET"> |  |
|  | **Имя поля** | <INPUT NAME="\*\*\*"> |  |
|  | **Значение поля** | <INPUT VALUE="\*\*\*"> |  |
|  | **Отмечен?** | <INPUT CHECKED> | (checkboxes и radio boxes) |
|  | **Размер поля** | <INPUT SIZE=?> | (в символах) |
|  | **Максимальная длина** | <INPUT MAXLENGTH=?> | (в символах) |
|  | **Список вариантов** | <SELECT></SELECT> |  |
|  | **Имя списка** | <SELECT NAME="\*\*\*"></SELECT> |  |
|  | **Число вариантов** | <SELECT SIZE=?></SELECT> |  |
|  | **Множественний выбор** | <SELECT MULTIPLE> | (можно выбрать больше одного) |
|  | **Опция** | <OPTION> | (элемент который может быть выбран) |
|  | **Опция по умолчанию** | <OPTION SELECTED> |  |
|  | **Ввод текста, размер** | <TEXTAREA ROWS=? COLS=?></TEXTAREA> |  |
|  | **Имя текста** | <TEXTAREA NAME="\*\*\*"></TEXTAREA> |  |
| N2.0 | **Разбивка на строки** | <TEXTAREA WRAP=OFF|VIRTUAL|PHYSICAL></TEXTAREA> |  |

|  |
| --- |
| **Таблицы** |
|  | **Определить таблицу** | <TABLE></TABLE> |  |
|  | **Окантовка таблицы** | <table border=?></TABLE> |  |
|  | **Расстояние между ячейками** | <TABLE CELLSPACING=?> |  |
|  | **Дополнение ячеек** | <TABLE CELLPADDING=?> |  |
|  | **Желаемая ширина** | <TABLE WIDTH=?> | (в точках) |
|  | **Ширина в процентах** | <TABLE WIDTH="%"> | (проценты от ширины страницы) |
|  | **Строка таблицы** | <TR></TR> |  |
|  | **Выравнивание** | <TR ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM> |  |
|  | **Ячейка таблицы** | <TD></TD> | (должна быть внутри строки) |
|  | **Выравнивание** | <TD ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM> |  |
|  | **Без перевода строки** | <TD NOWRAP> |  |
|  | **Растягивание по колонке** | <TD COLSPAN=?> |  |
|  | **Растягивание по строке** | <TD ROWSPAN=?> |  |
| N1.1 | **Желаемая ширина** | <TD WIDTH=?> | (в точках) |
| N1.1 | **Ширина в процентах** | <TD WIDTH="%"> | (проценты от ширины страницы) |
| N3.0b | **Цвет ячейки** | <TD BGCOLOR="#$$$$$$"> |  |
|  | **Заголовок таблицы** | <TH></TH> | (как данные, но жирный шрифт и центровка) |
|  | **Выравнивание** | <TH ALIGN=LEFT|RIGHT| CENTER|MIDDLE|BOTTOM> |  |
|  | **Без перевода строки** | <TH NOWRAP> |  |
|  | **Растягивание по колонке** | <TH COLSPAN=?> |  |
|  | **Растягивание по строке** | <TH ROWSPAN=?> |  |
| N1.1 | **Желаемая ширина** | <TH WIDTH=?> | (в точках) |
| N1.1 | **Ширина в процентах** | <TH WIDTH="%"> | (проценты ширины таблицы) |
| N3.0b | **Цвет ячейки** | <TH BGCOLOR="#$$$$$$"> |  |
|  | **Заглавие таблицы** | <CAPTION></CAPTION> |  |
|  | **Выравнивание** | <CAPTION ALIGN=TOP|BOTTOM> | (сверху/снизу таблицы) |

|  |
| --- |
| **ФРЕЙМЫ**  |
| N2.0 | **Документ с фреймами** | <FRAMESET></FRAMESET> | (вместо <BODY>) |
| N2.0 | **Высота строк** | <FRAMESET ROWS=,,,></FRAMESET> | (точки или %) |
| N2.0 | **Высота строк** | <FRAMESET ROWS=\*></FRAMESET> | (\* = относительный размер) |
| N2.0 | **Ширина колонок** | <FRAMESET COLS=,,,></FRAMESET> | (точки или %) |
| N2.0 | **Ширина колонок** | <FRAMESET COLS=\*></FRAMESET> | (\* = относительный размер) |
| N3.0b | **Ширина окантовки** | <FRAMESET BORDER=?> |  |
| N3.0b | **Окантовка** | <FRAMESET FRAMEBORDER="yes|no"> |  |
| N3.0b | **Цвет окантовки** | <FRAMESET BORDERCOLOR="#$$$$$$"> |  |
| N2.0 | **Определить фрейм** | <FRAME> | (содержание отдельного фрейма) |
| N2.0 | **Документ** | <FRAME SRC="URL"> |  |
| N2.0 | **Имя фрейма** | <FRAME NAME="\*\*\*"|\_blank|\_self| \_parent|\_top> |  |
| N2.0 | **Ширина границы** | <FRAME MARGINWIDTH=?> | (правая и левая границы) |
| N2.0 | **Высота границы** | <FRAME MARGINHEIGHT=?> | (верхняя и нижняя границы) |
| N2.0 | **Скроллинг?** | <FRAME SCROLLING="YES|NO|AUTO"> |  |
| N2.0 | **Постоянный размер** | <FRAME NORESIZE> |  |
| N3.0b | **Окантовка** | <FRAME FRAMEBORDER="yes|no"> |  |
| N3.0b | **Цвет окантовки** | <FRAME BORDERCOLOR="#$$$$$$"> |  |
| N2.0 | **Содержание без фреймов** | <NOFRAMES></NOFRAMES> | (для просмотрщиков не поддерживающих фреймы) |

|  |
| --- |
| **ЯЗЫК JAVA** |
|  | **Applet** | <APPLET></APPLET> |  |
|  | **Applet - имя файла** | <APPLET CODE="\*\*\*"> |  |
|  | **Параметры** | <APPLET PARAM NAME="\*\*\*"> |  |
|  | **Applet - адрес** | <APPLET CODEBASE="URL"> |  |
|  | **Applet - имя** | <APPLET NAME="\*\*\*"> | (для ссылок из других частей страницы) |
|  | **Альтернативный текст** | <APPLET ALT="\*\*\*"> | (для программ не поддерживающих Java) |
|  | **Выравнивание** | <APPLET ALIGN="LEFT|RIGHT|CENTER"> |  |
|  | **Размеры** | <APPLET WIDTH=? HEIGHT=?> | (в точках) |
|  | **Отступ** | <APPLET HSPACE=? VSPACE=?> | (в точках) |

|  |
| --- |
| **РАЗНОЕ** |
|  | **Комментарий** | <!-- \*\*\* --> | (игнорируется просмотрщиком) |
|  | **Пролог HTML 3.2** | <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN"> |  |
|  | **Поиск** | <ISINDEX> | (означает начальную точку поиска) |
|  | **Приглашение** | <ISINDEX PROMPT="\*\*\*"> | (текст приглашения для поля ввода) |
|  | **Запустить поиск** | <A HREF="URL?\*\*\*"></a> | (используйте действительно знак вопроса) |
|  | **URL этого файла** | <BASE HREF="URL"> | (должно быть в заголовке) |
| N2.0 | **Имя базового окна** | <BASE TARGET="\*\*\*"> | (должно быть в заголовке) |
|  | **Отношение** | <LINK REV="\*\*\*" REL="\*\*\*" HREF="URL"> | (должно быть в заголовке) |
|  | **Метаинформация** | <META> | (должно быть в заголовке) |
|  | **Стили** | <STYLE></STYLE> | (часто не поддерживается) |
|  | **Программа** | <SCRIPT></SCRIPT> | (часто не поддерживается) |

1. Основы XML и объектная модель представления данных

Бурное развитие Интернет технологий вовлекло в международную паутину миллионы пользователей. Требования к электронному обмену возросли, и уже существующий протокол HTML многие группы пользователей перестал удовлетворять.

В начале февраля 1998 г международная организация W3C утвердила спецификацию "Extensible Markup Language(XML) 1.0". Уже сегодня появляются новые языки, созданные на основе XML. Возникают многочисленные Web-сервера, использующие и технологию XML для организации хранящейся на них информации.

Современные приложения требуют не только более гибкий протокол представления данных, но и механизм, позволяющий определить структуру документа и описывать содержащие в нем элементы.

Язык XML предназначен для создания новых языков разметки. С его помощью можно описать целый класс объектов данных, называемых XML - документами, ориентированными на конкретную предметную область. XML позволяет определить допустимый набор тэгов, их атрибуты и внутреннюю структуру документа. Тэги (подобно тэгам в HTML) представляют специальные инструкции, предназначенные для формирования в документах определенной структуры и четких отношений между различными элементами этой структуры.

Можно выделить следующий круг задач, связанных с созданием и обработкой структурированной информации, для решения которых может использоваться XML:

1. Разработка сложных информационных систем, с большим количеством приложений, связанных потоками информации самой различной структуры. XML - документы выполняют роль универсального формата для обмена информацией между отдельными компонентами большой программы.
2. XML является базовым стандартом для нового языка описания ресурсов, RDF, позволяющего упростить многие проблемы в Web, связанные с поиском нужной информации, обеспечением контроля за содержимым сетевых ресурсов, создания электронных библиотек и т.д.
3. XML может использоваться в обычных приложениях для хранения и обработки структурированных данных в едином формате.
4. XML позволяет описывать данные произвольного типа и используется для представления специализированной информации.
5. XML может служить мощным дополнением к HTML для распространения в Web "нестандартной" структурированной информации
6. XML-документы могут использоваться в качестве промежуточного формата данных в трехзвенных системах при поиске информации в удаленных базах данных. Сегодня на рассмотрение W3C предложена спецификация нового языка запросов к базам данных XQL.
7. Информация, содержащаяся в XML-документах, может изменяться, передаваться на машину клиента и обновляться по частям. Разрабатываемые спецификации XLink и Xpointer позволяют ссылаться на отдельные элементы документа, c учетом их вложенности и значений атрибутов.
8. Использование стилевых таблиц (XSL) позволяет обеспечить независимое от конкретного устройства вывода отображение XML- документов и фильтрацию данных.

Тэги языка кодируются и выделяются относительно основного содержимого документа и служат в качестве инструкций для программы, производящей действия над содержимым документа на стороне клиента.

Исторически сложилось таким образом, что в системах для обозначения этих команд использовались символы "<" и ">", внутри которых помещались названия инструкций и их параметры. Сейчас такой способ обозначения тэгов является стандартным.

Например, для создания элемента Ivanov в имени заказчика используется тэг <CustumerName>. В программе это выглядит следующим образом:

<CustumerName> Ivanov </CustumerName>

Определения тэгов может легко расширяться. Так для указания более полных реквизитов заказчика можно определить тег <Custumer>, в который включено не только имя, телефон заказчика, но и адрес компании.

<Custumer>

<CustumerName> Ivanov </CustumerName>

<phone>312-12-13<phone>

<Company>Bussines Trade Consulti</Company>

</Custumer>

Можно создать массив заказчиков, определив тег <Custumers>:

<Custumers>

<Custumer>

<CustumerName> Ivanov </CustumerName>

<phone>312-12-13<phone>

<Company>Bussines Trade Consulti</Company>

</Custumer>

<Custumer>

<CustumerName> Petrov </CustumerName>

<phone>315-15-16<phone>

<Company> Trade Forest Company</Company>

</Custumer>

<Custumer>

......

</Custumer>

</Custumers>

Из приведенного примера видно, что XML - документы подлежат четкой структуризации и имеют четкую иерархическую структуру следования элементов. Элементы имеют своих родителей - корневые элементы и наследников - дочерние элементы.

Документ XML состоит из элементов. Элемент - это структурная единица XML- документа. Заключая данные об имени заказчика в тэги <CustumerName> </CustumerName>, XML-процессор определит как элемент. Содержимым элемента CustumerName является значение. В нашем примере имеется два значения (Ivanov и Petrov) элемента CustumerName.

Контроль за правильностью использования порядка использования элементов осуществляется при помощи специального набора правил, называемых DTD (Document Type Definition)- описаниями, которые используются программой клиента при анализе документа.

Производя в последствии поиск в XML документе, программа клиента будет опираться на информацию, заложенную в его структуру - используя элементы документа, определенные в DTD.

В общем случае XML- документы должны удовлетворять следующим синктатическим правилам:

1. В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация;
2. Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь парный закрывающий тэг;
3. XML учитывает регистр символов;
4. Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки;
5. Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов;
6. Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования ( пробелы, переводы строк, табуляции не игнорируются)

В случае, если элемент не содержит данных, т.е. является пустым, то начальный и конечные тэги такого элемента можно объединить в один. При этом не обязательно ставить косую черту перед закрывающей угловой скобкой (например, в вышеприведенном примере отсутствие факса в компании пару тэгов <fax></fax> можно заменить на <fax/>;)

При необходимости, каждому элементу можно задать параметры, уточняющие его характеристики. При этом используются атрибуты элемента. Атрибут - это пара "название" = "значение", которую необходимо задавать при определении элемента в начальном тэге. Пример:

<container Type="20f">123456</container > двадцати футовый контейнер

<container Type ="30f ">654321</container> тридцати футовый контейнер

Просмотр XML документов осуществляется специальной программой анализатором. На сегодняшний день разработано около десятка подобных анализаторов. В своем новом браузере Internet Explorer 5 фирма Microsoft уже предусмотрела анализ XML документов.

Анализ документа в Internet Explorer 5 осуществляется тремя вариантами: просмотр аналогично HTML документу, форматирование документа с использованием специальных стилевых таблиц - XSL и анализ с помощью сценариев, написанных на Java Script ил VBScript.

Поиск нужного элемента или поддерева осуществляется при помощи XQL запроса. XQL является частью XML и переводится как язык запросов для XML (XML Query Language). Идет дискуссия об утверждении языка запросов в качестве общепринятого стандарта, который может заменить SQL.

Синтаксис языка запросов очень гибок и позволяет осуществлять поиск элемента как по названию, значению атрибутов, содержанию, так и учитывать вложенность и положение в дереве элементов. При помощи запросов мы можем выделять из общего дерева необходимые нам элементы и применять к ним необходимые инструкции. Запрос возможно применять как к самому XML документу, так и к ссылкам URL.

Язык запросов напоминает обычный способ определения пути к ресурсу - список узлов дерева, разделенных символом "/". Для указания на текущий элемент используется символ "." , на родительский - "..", для выделения всех дочерних элементов - символ "\*", для выделения элемента, расположенного просто "ниже" по дереву(не важно на каком уровне вложенности) - "//". Условие на значение в запросе должно заключаться в символы "[" и "]". Для выбора значения атрибута в условии указывается символ @.

Примеры простых XQL шаблонов:

|  |  |
| --- | --- |
| "/Customer " | корневой элемент |
| "Customers/" | возвращает дочерние элементы для элемента Customers |
| "Customers //" | список всех элементов, вложенных в Customers |
| "container[@Type]" | список элементов container, в котором определен атрибут Type |
| "container[@Type =20f]" | поиск всех двадцатифутовых контейнеров, т.е. элементов container, в котором значение атрибута Type равно "20f" |
| "Customers[address]" | список элементов Customers, которые содержат хотя бы один элемент address, выражение в квадратных скобках может быть составным. |

Как мы видим, XML документ в отличие от EDIFACT сообщения позволяет более наглядно представить объектную модель данных. Использование языка описания XML запросов - XQL позволяет адекватно формализовать любой из существующих "бизнес" запросов (оформленных в виде стандартных документов) для информационных систем.

Разбор XML документов в отличие от EDI-систем возможен стандартными анализаторами, что значительно удешевляет разработку новых информационных систем. Использование встроенных транспортных протоколов делает эти системы полностью совместимыми с существующими программными средствами и WEB технологиями.

## Веб-страницы. Программирование (JavaScript, CSS, SSI, CGI, PHP)

1. JavaScript

Что такое JavaScript

JavaScript - новый язык для составления скриптов, разработанный фирмой [Netscape](http://www.netscape.com/). С помощью JavaScript Вы можете легко создавать интерактивные Web-страницы. В данном руководстве Вы увидите, что можно сделать с помощью JavaScript, и даже более того - увидите, *как* это сделано.

JavaScript - это не Java!

Многие люди считают, что JavaScript - это то же самое, что и Java, лишь потому, что эти языки имеют схожие названия. На самом деле это не так. Я считаю, что сейчас будет излишне показывать Вам все различия между этими языками - так что запомните лишь то, что JavaScript - это не Java.
Чтобы получить дополнительную информацию по затронутой теме, обратитесь пожалуйста к введению, опубликованному на сайте [Netscape](http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html)

Размещение JavaScript на HTML-странице

Код скрипта JavaScript размещается непосредственно на HTML-странице. Чтобы увидеть, как это делается, давайте рассмотрим следующий простой пример:

<html>

<body>

<br>

Это обычный HTML документ.

<br>

 <script language="JavaScript">

 document.write("А это JavaScript!")

 </script>

<br>

Вновь документ HTML.

</body>

</html>

С первого взгляда пример напоминает обычный файл HTML. Единственное новшество здесь - конструкция:

 <script language="JavaScript">

 document.write("А это JavaScript!")

 </script>

Это действительно код JavaScript. Чтобы видеть, как этот скрипт работает, запишите данный пример как обычный файл HTML и загрузите его в браузер, имеющий поддержку языка JavaScript.

А это результат выполнения этого файла (если Вы используете браузер, имеющий поддержку JavaScript, то у Вас будет 3 строки):

*Это обычный HTML документ.*

*А это JavaScript!*

*Вновь документ HTML.*

События

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя. Если он щелкает по некоторой кнопке, происходит событие *"Click"*. Если указатель мыши пересекает какую-либо ссылку гипертекста - происходит событие *MouseOver*. Существует несколько различных типов событий.
Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелка - *Click*. Программа - обработчик событий, которую мы должны использовать в данном случае, называется *onClick*. И она сообщает компьютеру, что нужно делать, если произойдет данное событие. Приведенный ниже код представляет простой пример программы обработки события *onClick*:

<form>

<input type="button" value="Click me" onClick="alert('Yo')">

</form>

1. CSS

**Немного о CSS**

Так что это такое - СSS? Cascading Style Sheets (Таблицы Каскадных Стилей) - это язык, содержащий набор свойств для описания внешнего вида любых HTML документов. С его помощью дизайнер имеет полный контроль над стилем и расположением каждого элемента веб страницы, что проще и гораздо функциональнее использования обычного набора HTML тегов. Приведу пример: Вам нужно создать жирный красный подчеркнутый текст.

ПРИМЕР HTML:

<font color="red"><strong><u> Какой-то текст </u></strong></font>

А если подобный стиль нужно использовать несколько раз? Хорошо если раз 5, а если 10-20? Вот тут нам и поможет СSS. Существует три вида таблиц стилей: Внутренние таблицы стилей, Глобальные таблицы стилей и Связанные таблицы стилей. Внутренние таблицы стилей (Inline Style Sheets) при помощи специального атрибута помещаются прямо в HTML теги. Глобальные (Global Style Sheets) определяют стиль элементов во всем документе. Связанные (Linked Style Sheets) могут быть использованы для нескольких документов сразу и хранятся во внешнем файле. Подробнее обо всем этом написано ниже.

**Структура и правила**

***Селекторы (Selectors):***

Синтаксис:
селектор {свойства}

Любой элемент HTML - это возможный CSS селектор. Свойства селектора определяют стиль элемента, для которого он определен.

ПРИМЕР:

H1 {color:red; size:20pt;}

Все элементы H1 в документе будут красного цвета, размером в 20 точек (pt, point).

***Классовые селекторы (Class Selectors):***

Синтаксис:
селектор.класс {cвойства}

**CLASS** - атрибут элемента в HTML, определяющий его класс. В CSS можно описать собственные стили для различных классов одних и тех же элементов.

ПРИМЕР:

H1.blue {color:blue; size:20pt;}

Все элементы H1 с атрибутом CLASS="blue" станут синими.

Классы могут так же быть описаны без явного привязывания их к определенным элементам.

Синтаксис:
.класс {свойства}

ПРИМЕР:

.green {color:green;}

В данном случае все элементы с атрибутом CLASS="green" станут зелеными.

***ID селекторы (ID Selectors):***

Cинтаксис:
#id {свойства}

**ID** - индивидуально именованный стиль. C его помощью можно создавать стилистические исключения cреди элементов одного класса.

Идентификаторы используются в основном для придания одному или нескольким элементам одного класса индивидуальных свойств. Скажем, Вы создали класс blue - синий курсив. Но Вам понадобился жирный подчеркнутый текст синим курсивом. Конечно, можно создать новый класс, но зачем? Проще описать ID. Например "boldunderline". И все элементы класса blue с значением ID "boldunderline" станут жирным подчеркнутым синим курсивом. Произойдет как бы синтез свойств класса blue и идентификатора boldunderline.

ПРИМЕР:

<html>

<head>

<title> Пример CSS </title>

</head>

<style>

.blue {color:blue; font-style:italic}

#boldunderline {text-decoration:underline; font-weight:bold}

</style>

<body>

<p class="blue"> Здравствуйте, это моя домашняя страница. </p>

<p class="blue" id="boldunderline"> Пока еще в стадии разработки ... </p>

<p id="boldunderline">... Но скоро откроется </p>

</body>

</html>

Как видно из примера, атрибут ID может быть использован без указания класса (последний параграф примера. Тогда параграф будет обладать только свойствами ID "boldunderline" (в примере - жирный, подчеркнутый текст).

***Контекстуальные селекторы (Contextual Selectors):***

Контекстные селекторы - это сочетания нескольких обыкновенных селекторов. Стиль задается только элементам в заданной последовательности в зависимости от каскадного порядка.

ПРИМЕР:

P EM {color:silver;}

В данном примере все элементы EM внутри элементов P будут иметь заданный стиль.

***Придание нескольким элементам одинаковых свойств:***

Скажем Вам нужно придать нескольким элементам Вашей веб страницы одинаковых свойств. В этом случае при определении селекторы перечисляются через запятую перед блоком свойств.

ПРИМЕР:

h1,h2,h3,p,strong {color:green; font-style:italic;}

Все элементы h1, h2, h3, p и strong будут зелеными.

***Псевдоклассы и псевдоэлементы :***

Синтаксис:
селектор:псевдокласс { свойства }
селектор.класс:псевдокласс { свойства }
селектор:псевдоэлемент { свойства }
селектор.класс:псевдоэлемент { свойства }

Псевдоклассы и псевдоэлементы - это особые классы и элементы, присущие CSS и автоматически определяемые поддерживеющими CSS браузерами. Псевдоклассы различают разные типы одного элемента, создавая при определении собственные стили для каждого из них. Псевдоэлементы являются частями других элементов, задавая этим частям отличный от элемента в целом стиль.

***Список псевдоклассов и псевдоэлементов :***

**Anchor Pseudo Classes** - эти псевдоклассы элемента <a href=" ">, обозначающего ссылку. Псевдоклассы этого элемента: (ссылка), active (активная ссылка), visited (посещенный ранее URL), hover (псевдокласс, возникающий при поднесении курсора к ссылке, не работает в Нетскейпе).

**First Line Pseudo-element** - first-line. Этот псевдоэлемент может быть использован с block-level элементами (p, h1 и т.д.). Он изменяет стиль первой строки этих элементов.

**First Letter Pseudo-element** - first-letter. Похож на first-line, но влияет не на всю строку, а только на первый символ.

ПРИМЕР :

a:link,a:visited {color:blue}

a:active {color:red}

a:hover {text-decoration:none}

В данном примере все элементы Anchor (ссылки) будут синими. При нажатии (в активном состоянии) поменяют цвет на красный. И при подведении курсора мышки исчезнет подчеркивание.

**Примечание** : описания нескольких свойств для одного селектора, контекстуального селектора, класса, индивидуально именованного стиля или группы объединенных селекторов отделяются друг от друга точкой с запятой ";".

**Внутренние Таблицы Стилей**

Как уже говорилось, использование Внутренних стилей мало чем отличается от использования обычных HTML тегов. Они задают стиль только одному элементу документа при помощи атрибута STYLE в HTML теге.

ПРИМЕР HTML:

<font color="blue" size="3" face="Arial"> Вперед в будущее </font>

ПРИМЕР INLINE STYLE SHEET:

<font style="color:blue; font-size:12pt; font-family:Arial"> Вперед в будущее </font>

Как можно заметить, код Inline Style Sheet получился больше чем HTML. Поэтому ISS следует использовать только если необходимо задать определенному элементу свой индивидуальный стиль, существующий в классификации CSS и нереализованный в HTML. Или же при необходимости абсолютно позиционировть данный элемент.

**Глобальные Таблицы Стилей**

Глобальные стили задают вид элементов всего документа. Для этого используется тег <STYLE type="text/css">. Он размещается в заголовке документа.

ПРИМЕР:

<html>

<head> <title> Пример Глобальных Таблиц Стилей </title>

</head>

<STYLE type="text/css">

h1{color:red; font-style:italic; font-size:32px}

.blue{color:blue}

#bold{font-weight:bold}

</STYLE>

<body>

<h1> Этот заголовок написан крупным красным курсивом </h1>

Вот <font class="blue"> это </font> слово - синие, a <font id="bold"> это</font> - жирное.

</body>

</html>

В данном примере все элементы H1 будут написаны крупным красным курсивом, все элементы с указанным классом BLUE будут синими , а все элементы с идентификатором ID="Bold" станут жирными. Для простоты вместо <STYLE type="text/css"> можно использовать просто тег <STYLE>, что менее грамотно.

**Связанные Таблицы Стилей**

Связанные таблицы стилей используются для придания нескольким документам одного стиля и xранятся в отдельном файле. Это очень привлекательно, когда нужно выдержать сайт в одном стиле, не утруждая себя составлением таблиц для каждого документа.

ПРИМЕР:

*Файл styles.css*

<STYLE type="text/css">

body {background:black; font-size:9pt; color:red; font-family:Arial Black}

.base{color:blue; font-style:italic}

h1 {color:white}

#bold {font-weight:bold}

</STYLE>

В самих же HTML документах делается ссылка на этот файл при помощи тега <LINK>. Выглядит это так: <LINK rel="STYLESHEET" TYPE="text/css" HREF="путь до файла">

ПРИМЕР:

*Файл Index.html*

<html>

<head>

<title> Просто еще один пример </title>

</head>

<LINK rel="stylesheet" type="text/css" href="styles.css">

<body>

Содержание Документа

</body>

</html>

1. **SSI**

**SSI** расшифровывается как *S*erver *S*ide *I*nclude. SSI - это набор команд, позволяющий включить в страницу информацию, недоступную средствами HTML, такую как вывод программ, значения переменных окружения и статистику по файлам (размер, дата создания и др.). Все это и более позволяют делать CGI-скрипты, но зачем писать целый скрипт для получения размера файла, если можно воспользоваться SSI?

SSI работает очень просто: при получении запроса на документ, содержащий SSI-директивы, http-сервер обрабатывает документ, выполняет все SSI-директивы, а получившийся в результате документ возвращает клиенту. Http-сервер не проверяет автоматически все документы на наличие SSI-директив, а только те, которые относятся к типу, указанному в настройках сервера.

Для указания http-серверу, какие документы считать содержащими SSI-директивы, в *srm.conf*, находящемся в */etc/httpd/conf* надо раскомментировать следующие строки:

AddType text/html .shtml

AddHandler server-parsed .shtml

После этого все документы с расширением **.shtml** будут проверяться на SSI-директивы. Но удобнее будет определить чтобы все **.html**-файлы могли содержать SSI. Это делается добавление в *srm.conf* следующих строчек:

AddType text/html .html # не обязательно, т.к. он и так описан.

AddHandler server-parsed .html

Теперь надо разрешить выполнение SSI в каталоге, в котором лежат твои файла. Найди в *access.conf* описание каталога, в котором лежат твои html'ы и добавь в его *Options* директиву *Includes*:

#было так

Options Indexes

#стало так

Options Indexes Includes

Если твой каталог не прописан, его надо прописать. Почитай доку по [Apache](http://www.apache.org), чтоб узнать как это делается. А если у тебя нет возможности изменять *srm.conf* или *access.conf*, т.е. ты не **root**, тогда за пивом, а потом к админу.

SSI-директивы включаются в html-код в виде комментариев, но определенного формата:

<!--#SSI\_директива="параметры" -->

Использовать можно следующие SSI-директивы:

**echo var**

Используется для вывода значения переменной окружения.

Твой IP - <!--#echo var="REMOTE\_ADDR"--> <BR>

Твой браузер - <!--#echo var="HTTP\_USER\_AGENT"--> <BR>

Мой сервер - <!--#echo var="SERVER\_NAME"--> <BR>

Ты пришел со страницы - <!--#echo var="HTTP\_REFERER"--> <BR>

А вот вывод:

Твой IP - 193.125.78.114

Твой браузер - Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)

Мой сервер - [www.citforum.ru](http://www.citforum.ru)

Ты пришел со страницы - (none)

**include file**

**include virtual**

Включает в html-файл в месте, где идет вызов SSI, содержимое указанного файла. Отличие **include file** от **include virtual** в том, что в первом случает используется файл, находящийся в том же каталоге, что и html-файл, во втором случает указывается путь к файлу относительно текущего каталога. Если с помощью **include file** или **include virtual** включается cgi-скрипт, то в содержимое страницы попадет вывод cgi-скрипта, а не его содержание.

**fsize file**

Вставляет размер указанного файла.

Размер файла этой страницы:

<!--#fsize file="ssi.shtml"--> <BR>

Вывод:

Размер файла этой страницы: 12k

**flastmod file**

Вставляет время последней модификации указанного файла.

Время последней модификации файла этой страницы:

<!--#flastmod file="ssi.shtml"--> <BR>

Вывод:

Время последней модификации файла этой страницы: Wednesday, 21-Jun-2000 16:31:33 MSD

**exec cmd**

**exec cgi**

Запускает внешнюю программу (**exec cmd**) или cgi-скрипт (**exec cgi**) и вставляет в содержимое страницы вывод.

**config errmsg**

**config sizefmt**

**config timefmt**

Изменяет различный параметры конфигурации SSI. **config errmsg** изменяет стандартное сообщение об ошибке на введенное пользователем. Сообщение об ошибке возникает при неправильном выполнении SSI-директивы, например при отсутствии cgi-скрипта, который ты пытаешься запустить.

Это стандартная ошибка при запуске скрипта, которого нет: <BR>

<!--#exec cgi="/cgi-bin/nonexistence.pl"--> <BR>

А теперь заменим сообщение об ошибке и повторим: <BR>

<!--#config errmsg="Ошибка, пишите <a href=mailto:dh@null.ru>автору ошибки</a>"-->

<!--#exec cgi="/cgi-bin/nonexistence.pl"--> <BR>

Вывод:

Это стандартная ошибка, возникающая при запуске скрипта, которого нет:
[an error occurred while processing this directive]

А теперь заменим сообщение об ошибке и повторим:

Ошибка, пишите автору ошибки

Директива **config sizefmt** изменяет формат вывода размера файла.

Размер файла этой страницы в килобайтах:

<!--#config sizefmt="abbrev"-->

<!--#fsize file="ssi.shtml"--> <BR>

Размер файла этой страницы в байтах:

<!--#config sizefmt="bytes"-->

<!--#fsize file="ssi.shtml"--> <BR>

Вывод:

Размер файла этой страницы в килобайтах: 12k

Размер файла этой страницы в байтах: 12,463

Директива **config timefmt** меняет формат вывода даты и времени.

Время модификации файла этой страницы в секундах с 01.01.1970:

<!--#config timefmt="%s"-->

<!--#flastmod file="ssi.shtml"--> <BR>

Время модификации файла этой страницы в читабельном виде:

<!--#config timefmt="%d.%m.%Y %H:%M:%S"-->

<!--#flastmod file="ssi.shtml"--> <BR>

Вывод:

Время модификации файла этой страницы в секундах с 01.01.1970: 961590693

Время модификации файла этой страницы в читабельном виде: 21.06.2000 16:31:33

**printenv**

Выводит все переменные окружения. Параметров не имеет.

Список переменных окружения:<BR>

<!--#printenv -->

Вывод:

Список переменных окружения:

DOCUMENT\_ROOT=/webdata/citforum/htdocs HTTP\_ACCEPT=\*/\* HTTP\_ACCEPT\_ENCODING=gzip, deflate HTTP\_ACCEPT\_LANGUAGE=uk HTTP\_CACHE\_CONTROL=max-age=432000 HTTP\_HOST=www.citforum.ru HTTP\_USER\_AGENT=Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) HTTP\_VIA=1.0 proxy.univ.kiev.ua:3128 (Squid/2.3.STABLE2) HTTP\_X\_FORWARDED\_FOR=10.25.0.7 PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/home/andy/bin:/usr/local/pgsql/bin REMOTE\_ADDR=193.125.78.114 REMOTE\_HOST=red-gw.univ.kiev.ua REMOTE\_PORT=2205 SCRIPT\_FILENAME=/webdata/citforum/htdocs/internet/html/ssi.shtml SERVER\_ADDR=195.230.75.98 SERVER\_ADMIN=info@citforum.ru SERVER\_NAME=www.citforum.ru SERVER\_PORT=80 SERVER\_SIGNATURE= SERVER\_SOFTWARE=Apache/1.3.12 (Unix) UNIQUE\_ID=OlXJbMPmS2IAAI-qcYQ GATEWAY\_INTERFACE=CGI/1.1 SERVER\_PROTOCOL=HTTP/1.0 REQUEST\_METHOD=GET QUERY\_STRING= REQUEST\_URI=/internet/html/ssi.shtml SCRIPT\_NAME=/internet/html/ssi.shtml DATE\_LOCAL=05.01.2001 16:17:32 DATE\_GMT=05.01.2001 13:17:32 LAST\_MODIFIED=21.06.2000 16:31:33 DOCUMENT\_URI=/internet/html/ssi.shtml DOCUMENT\_PATH\_INFO= USER\_NAME=andreya DOCUMENT\_NAME=ssi.shtml

**set var**

Устанавливает значение переменной окружения. Формат такой set var="ИМЯ\_ПЕРЕМЕННОЙ" value="ЗНАЧЕНИЕ".

Значение переменной SERVER\_NAME:

<!--#echo var="SERVER\_NAME" --><BR>

Сменим значение:

<!--#set var="SERVER\_NAME" value="www.dhls.ru" --><BR>

Значение переменной SERVER\_NAME:

<!--#echo var="SERVER\_NAME" --><BR>

Вывод:

Значение переменной SERVER\_NAME:www.citforum.ru

Сменим значение:

Значение переменной SERVER\_NAME: www.dhls.ru

**if/else**

Наличие такой команды не может не радовать... Применяется для управления выводом страницы по условию. Синтаксис такой:

<!--#if expr="УСЛОВИЕ1" -->

HTML-код, который будет выводиться, если УСЛОВИЕ1 истинно

<--#elif expr="УСЛОВИЕ2" -->

HTML-код, который будет выводиться, если УСЛОВИЕ1 ложно, а УСЛОВИЕ2 истинно

<--#else -->

HTML-код, который будет выводиться, если все условия ложны

<--#endif -->

Условие - это либо строка, которая является истинной, если непустая, или набор операторов сравнения строк. Операторы могут быть **=**,**!=**,**<**,**<=**,**>** и **>**. Если вторая строка заключена в "**/**"(слэши), то условие истинно, если в первой строке встречается хоть одно вхождение второй строки. Можно объединять несколько операторов сравнения с помощью операторов **&&**(И) и **||**(ИЛИ). Для группирования условий используются "**()**"(скобки).

Броузер:<b>

<!--#if expr="$HTTP\_USER\_AGENT=/Nav/ || $HTTP\_USER\_AGENT=/Mozilla/" -->

Netscape Navigator

<!--#elif expr="$HTTP\_USER\_AGENT=/MSIE/" -->

Internet Explorer

<!--#else -->

Неизвестный (<!--#echo var="HTTP\_USER\_AGENT" -->)

<!--#endif -->

</b>

Вывод:

Броузер: **Netscape Navigator**

1. **CGI**

**CGI - Common Gateway Interface** является стандартом интерфейса (связи) внешней прикладной программы с информационным сервером типа HTTP, Web сервер.
Обычно гипертекстовые документы, извлекаемые из WWW серверов, содержат статические данные. С помощью CGI можно создавать CGI-программы, называемые ***шлюзами***, которые во взаимодействии с такими прикладными системами, как система управления базой данных, электронная таблица, деловая графика и др., смогут выдать на экран пользователя динамическую информацию.
Программа-шлюз запускается WWW сервером в реальном масштабе времени. WWW сервер обеспечивает передачу запроса пользователя шлюзу, а она в свою очередь, используя средства прикладной системы, возвращает результат обработки запроса на экран пользователя. Программа-шлюз может быть закодирована на языках C/C++, Fortran, Perl, TCL, Unix Schell, Visual Basic, Apple Script. Как выполнимый модуль, она записывается в поддиректорий с именем cgi-bin WWW сервера.
Передача данных шлюзам

Для передачи данных об информационном запросе от сервера к шлюзу, сервер использует командную строку и переменные окружения. Эти переменные окружения устанавливаются в тот момент, когда сервер выполняет программу шлюза.

**Запросы для различных методов**

Информация шлюзам передается в следующей форме:

**имя=значение&имя1=значение1&..,**

где имя - имя переменной (из оператора [FORM](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Cform.shtml), например), и значение - ее реальное значение. В зависимости от метода, который используется для запроса, эта строка появляется или как часть URL (в случае метода [GET](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Crqst.shtml#GET)), или как содержимое [HTTP](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Chttp.shtml) запроса (метод [POST](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Crqst.shtml#POST)). В последнем случае, эта информация будет послана шлюзу в стандартный поток ввода.

На файловый дескриптор стандартного потока ввода посылается [CONTENT\_LENGTH](#content_length) байт. Так же сервер передает шлюзу [CONTENT\_TYPE](#content_type) (тип передаваемых данных). Сервер не обязан посылать символ конца файла после отсылки CONTENT\_LENGTH байт данных и после того, как шлюз их прочитает.

*Пример*

Возьмем результат работы формы с методом POST (METHOD="POST") в качестве примера. Пусть получено 7 байт, закодированных примерно так:
**a=b&b=c.**

В этом случае, сервер установит значение CONTENT\_LENGTH равным 7 и CONTENT\_TYPE в application/x-www-form-urlencoded. Первым символом в стандартном потоке ввода для шлюза будет "a", за которым будет следовать остаток закодированной строки.

**Аргументы командной строки**

Шлюз в командной строке от сервера получает:

1. остаток URL после имени шлюза в качестве первого параметра (первый параметр будет пуст, если присутствовало только имя шлюза), и
2. список ключевых слов в качестве остатка командной строки для скрипта поиска, или
3. чередующиеся имена полей формы с добавленным знаком равенства (на четных позициях) и соответствующих значений переменных (на нечетных позициях).

Ключевые слова, имена полей формы и значения передаются раскодированными (из HTTP URL формата кодирования) и перекодированными в соответствии с правилами кодирования Bourne shell, так что шлюз в командной строке получит информацию в том виде, как она есть, без необходимости осуществлять дополнительные преобразования.

*Запросы оператора FORM*

Запросы оператора FORM обрабатываются таким образом, что каждый параметр, отвечающий за имя поля, оканчивается знаком равенства, а остаток представляет собой значение этого параметра. Если присутствует что либо после имени скрипта (шлюза), то эта информация передается в качестве первого параметра. Иначе первый параметр будет пуст.

*Примеры:*

 /htbin/foo/x/y/z?name1=value1&name2=value2

вызывается как:

 /.../foo /x/y/z name1= value1 name2= value2

а

 /htbin/foo?name1=value1&name2=value2

вызывается как:

 /.../foo '' name1= value1 name2= value2

**CGI переменные окружения**

Следующие переменные окружения не являются специфичными по типу запросов и устанавливаются для всех запросов.

SERVER\_SOFTWARE

Название и версия информационного сервера, который отвечает на запрос (и запускает шлюз). Формат: имя/версия

SERVER\_NAME

Имя хоста, на котором запущен сервер, DNS имя, или IP адрес в том виде, в котором он представлен в URL.

GATEWAY\_INTERFACE

Версия CGI спецификации на тот момент, когда компилировался сервер. Формат: CGI/версия

Следующие переменные окружения являются специфичными для разных запросов, и заполняются перед вызовом шлюза:

SERVER\_PROTOCOL

Имя и версия информационного протокола, в котором пришел запрос. Формат: протокол/версия

SERVER\_PORT

Номер порта, на который был послан запрос

REQUEST\_METHOD

Метод, который был использован для запроса. Для HTTP, это "[GET](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Crqst.shtml#GET)", "[HEAD](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Crqst.shtml#HEAD)", "[POST](file:///C%3A%5Cwww%5Cdoc2html%5Cwork%5Cbestreferat-222451-13961197855568%5Cinput%5Crqst.shtml#POST)", и т.д.

PATH\_INFO

Дополнительная информация о пути, которую передал клиент. Другими словами, доступ к шлюзу может быть осуществлен по виртуальному пути, за которым следует некоторая дополнительная информация. Эта информация передается в PATH\_INFO.

PATH\_TRANSLATED

Сервер передает преобразованную версию PATH\_INFO, которая включает в себя путь, преобразованный из виртуального в физический.

SCRIPT\_NAME

Виртуальный путь к шлюзу, который должен выполняться, используемый для получения URL.

QUERY\_STRING

Информация, следующая за ? в URL, к которому относится данный шлюз. Это информация представляет собой строку запроса. Она не должна быть декодирована ни коим образом. Вне зависимости от командной строки эта переменная всегда должна быть установлена при наличии такой информации, .

REMOTE\_HOST

Имя хоста, производящего запрос. Если сервер не имеет такой информации, он должен установить REMOTE\_ADDR, а это поле оставить не установленным.

REMOTE\_ADDR

IP адрес хоста, производящего запрос.

AUTH\_TYPE

Если сервер поддерживает идентификацию пользователя, и шлюз является защищенным от постороннего доступа, этот специфичный для протокола метод идентификации используется для проверки пользователя.

REMOTE\_USER

Используется в ситуациях, аналогичных предыдущему случаю, для хранения имени пользователя.

REMOTE\_IDENT

Если HTTP сервер поддерживает идентификацию пользователя согласно RFC 931, то эта переменная будет содержать имя пользователя, полученное от сервера.

CONTENT\_TYPE

Для запросов, которые содержат дополнительную добавочную информацию, такие как HTTP POST и PUT, здесь содержится тип данных этой информации.

CONTENT\_LENGTH

Длина данных, которую передает клиент.

В дополнение к этим, если запрос содержит дополнительные поля заголовка запроса, они помещаются в переменные окружения с префиксом HTTP\_, за которым следует имя заголовка. Любые символы '-' в заголовке меняются на символы подчеркивания '\_'. Сервер может исключить любые заголовки, которые он уже обработал, такие как Authorization, Content-type, и Content- length. Если необходимо, сервер может исключить любые (или вообще все) дополнительные поля заголовка в случае, когда их включение может привести к превышению предела размера переменных окружения. Примером такой переменной может служить переменная HTTP\_ACCEPT, которая была определена в спецификации CGI/1.0. Другим примером может служить заголовок User-Agent.

HTTP\_ACCEPT

Список MIME типов, которые клиент может обработать, как задано в HTTP заголовках. Другие протоколы должны получить эту информацию из других мест (если она им необходима). Каждый тип в этом списке должен быть отделен запятой согласно HTTP спецификации. Формат: тип/подтип, тип/подтип

HTTP\_USER\_AGENT

Просмотрщик, который использует клиент для посылки запроса. Общий формат: программа/версия библиотека/версия.

Вывод информации шлюзом

**Основные концепции**

Шлюз осуществляет свой вывод в стандартный поток вывода. Этот вывод может представлять собой или документ, сгенерированный шлюзом, или инструкции серверу, где получить необходимый документ.

Как правило, шлюз производит свой вывод, который интерпретируется и посылается обратно клиенту. Преимущество этого подхода состоит в том, что шлюз не должен посылать полный HTTP/1.0 заголовок на каждый запрос.

**Заголовок выходного потока**

Для некоторых шлюзов может быть необходимо избегать обработки сервером их вывода, и общаться с клиентом непосредственно. Для того, чтобы отличить такие шлюзы от остальных, CGI требует, чтобы их имена начинались с префикса nph-. В этом случае, на шлюзе лежит ответственность за возвращение клиенту синтаксически правильного ответа.

**Заголовки с синтаксическим разбором**

Вывод шлюза начинается с маленького заголовка. Он содержит текстовые строки, в том же формате, как и в HTTP заголовке и завершается пустой строкой (содержащей только символ перевода строки или CR/LF).

Любые строки заголовка, не являющиеся директивами сервера, посылаются непосредственно клиенту. В настоящий момент, CGI спецификация определяет три директивы сервера:

Content-type

MIME тип возвращаемого документа.

Location

Это поле используется в случае, когда необходимо указать серверу, что возвращается не сам документ, а ссылка на него.

Если аргументом является URL, то сервер передаст клиенту указание на перенаправление запроса. Если аргумент представляет собой виртуальный путь, сервер вернет клиенту заданный этим путем документ, как если бы клиент запрашивал его непосредственно.

1. Status

Эта директива используется для задания серверу HTTP/1.0 строки-статус, которая будет послана клиенту. Формат: nnn xxxxx, где nnn - 3-х цифровой статус-код, и xxxxx строка причины, такая, как "Forbidden" (Запрещено).

Примеры

Предположим, имеется некоторый текстовый конвертер в HTML. Когда он оканчивает свою работу, он должен произвести следующий вывод в стандартный выходной поток:

--- начало вывода ---

Content-type: text/html

--- конец вывода ---

Теперь рассмотрим шлюз, который, в некоторых случаях, должен выдать документ /path/doc.txt с данного сервера, как если бы он был непосредственно востребован клиентом через http://server:port/path/doc.txt. В это случае вывод шлюза будет таков:

--- начало вывода ---

Location: /path/doc.txt

--- конец вывода ---

Наконец, предположим, что шлюз возвращает ссылки на gopher сервер, например на gopher://gopher.ncsa.uiuc.edu/. Вывод шлюза будет следующий:

--- начало вывода ---

Location: gopher://gopher.ncsa.uiuc.edu/

--- конец вывода ---

**Non-parsed headers**

Допустим теперь, что у нас имеется шлюз, который общается с клиентом непосредственно. Как уже отмечалось, его имя должно начинаться с префикса nph- и он должен возвращать допустимый HTTP заголовок. В этом случае, если доступ к шлюзу был осуществлен со значением SERVER\_PROTOCOL равным HTTP/1.0, его вывод должен удовлетворять HTTP/1.0:

--- начало вывода ---

HTTP/1.0 200 OK

Server: NCSA/1.0a6

Content-type: text/plain

--- конец вывода ---

***Примечание!*** Материал подготовлен на основе документа ["CGI Specification at NCSA"](http://hoohoo.ncsa.uiuc.edu/cgi/)

1. **PHP**

PHP - язык написания сценариев, внедренный в HTML. Многое из синтаксиса заимствовано из C, Java и Perl, с добавлением специфичных для PHP возможностей. Задача языка состоит в том, чтобы дать возможность Web-разработчикам легко и быстро создавать динамично изменяемые html-страницы.

Что может делать PHP3?

Возможно самая сильная и значимая возможность в PHP3 - уровень интеграции с базами данных. Написание веб-страницы работающей с базой данных невероятно проста. В настоящее времени поддерживаются следующие базы данных:

 Oracle

 Adabas D

 Sybase

 FilePro

 mSQL

 Velocis

 MySQL

 Informix

 Solid

 dBase

 ODBC

 Unix dbm

 PostgreSQL

Краткая История PHP

PHP был задуман где-то в конце 1994 года Расмусом Ледорфом(Rasmus Lerdorf). Ранние невыпущенные версии использовались на его домашней странице для того, чтобы следить за тем кто просматривал его интерактивное резюме. Первая используемая версия cтала доступна где-то в начале 1995 и была известна как Personal Home Page Tools. Она состояла из очень упрощенного движка синтаксического анализатора, который понимал только несколько специальных макрокоманд и ряд утилит, которые затем были в общем использовании на домашних страницах. Гостевые книги, счетчики и некоторые другие дополнения.

Довольно трудно дать какую-либо жесткую статистику, но отмечено, что к 1996 г. PHP/FI был использован по крайней мере на 15,000 веб-сайтах во всем мире. В середине 1997г. эта цифра выросла до более чем 50,000. В середине 1997г. также наблюдалось изменение в разработке PHP. Из частного любимого проекта Расмуса, которому способствовала горстка людей , это превратилось в намного более организованную рабочую группу. Синтаксический анализатор был заново переписан Зевом Сураски(Zeev Suraski) и Анди Гутмансом(Andi Gutmans), и этот новый синтаксический анализатор стал основой для PHP Версии 3.

Сегодня ( в середине -1998г.) как PHP/FI так и PHP3 поставляется с рядом коммерческих продуктов типа C2's StrongHold web server и RedHat Linux и консервативной оценкой, основанной на экстрaполяции чисел, предоставленных NetCraft было бы то, что PHP используется на 150,000 cайтах во всем мире. В перспективе, их больше чем сайтов, запущенных на Netscape's flagship Enterprise server в Интернете.

## Модульность и ортогональность с использованием существующих технологий.

Ортогональность и модульность кода сайта логично взаимосвязаны. Разработка сайта, изначально предусматривающая модульность, скорее всего будет подразумевать так же его ортогональность.

При создании сайта сразу следует предусмотреть в его структуре возможность модульности, как, например, все JavaScript-сценарии следует выносить во внешние файлы, которые можно разместить в отдельной директории сайта, к которой будет ограничен доступ, что значительно повысит безопасность сайта, сделает его структуру более простой у удобочитаемой. Далее, все общие элементы дизайна и стилевого оформления лучше всего описать в виде CSS и так же вынести во внешний файл или файлы. В этом плане CSS наиболее продвинутая технология в плане разграничения контента и его оформления, здесь четко различается иерархическая структура оформления и само содержание, CSS содержания не касается.

При разработке PHP-скриптов и приложений рекомендуется для наиболее часто используемых участков кода создавать функции и выносить их в отдельные библиотеки функций, а так же серверно-зависимые настройки, хосты баз данных, пароли и имена для аутентификации выносить в конфигурационные файлы, для которых следует определить отдельный определенный класс. Таким образом мы получаем масштабируемый, модульный и легко переносимый код.

Следующая технология, которая позволяет разделить содержание и оформление сайта – SSI. После CSS это наиболее приближенная к идее независимости контента от его представления технология. Во включаемые файлы посредством SSI обычно выносятся заголовки сайтов, элементы оформления нижней части страницы, а так же меню и вообще любые другие общие элементы сайта. Например, если мы выносим меню во включаемые файлы, то это позволит нам быстро добавлять в структуру сайта новые разделы, причем изменения придется делать и синхронизировать не во множестве страниц, а в одном файле описания меню, далее он автоматически включится во все страницы.

## Веб-приложения.

Под веб-приложениями понимается все, что не относится к простому гипертекстовому документу, как то, апплеты, флэш-приложения, презентации, скрипты, ActiveX компоненты и сценарии, а так же обычные программы и модули и плагины браузера. Кроме этого под веб-приложениями так же могут пониматься интерфейсы, интерфейсы к базам данных, интернет-магазины, системы новостей и многое другое. Практически все что требует программирования может быть отнесено к приложениям.

## Общие требования к страницам сайта.

Общие требования к страницам сайта не содержат ничего специфического и подробно нигде не описаны. Следует выделить такие общие требования.

1. размер одной страницы. Средний размер страницы рекомендуется в среднем делать не более 60Кб, для сплэш-страниц и презентационных страниц размер может быть до70-90Кб.
2. Графика. Не рекомендуется перегружать страницу графикой и флэш-презентациями сверх необходимого, а так же излишней графикой.
3. Прозрачная навигация. Навигация на сайте должна быть прозрачной, т.е. интуитивно понятной и достаточно простой. В графическом меню обязательно должен быть прописан альтернативный текст для пользователей текстовых браузеров и для тех кто любит отключать графику на страницах.
4. Разбивка текста. Большие текстовые блоки следует разбивать на страницы, которые содержат 3-4 экрана, опять же из соображений размера – они быстро выкачиваются, а так же удобочитаемости.

4.б.8. Мета-теги.

Метатеги есть эквивалент соответствующих полей HTTP-заголовка и широко используются для переопределения стандартных записей.

META-таги имеют два возможных атрибута

* <META HTTP-EQUIV="имя" CONTENT="содержимое">
* - <META NAME="имя" CONTENT="содержимое">

META-таги должны находиться в заголовке HTML-документа между <HEAD> и </HEAD> (особенно это важно для документов, использующих фреймы).

Наиболее полезные с точки зрения применения метатеги следующие:

<META HTTP-EQUIV="Refresh" Content="3, URL=http://www.name.com/page.html">

что эквивалентно HTTP-заголовку

Refresh: 3; URL=http://www.name.com/page.html

Пример типичного заголовка с мета-тэгами:

<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

<meta name="language" content="en,ru,ua">

<meta name="robots" content="ALL">

<meta name="rating" content="GENERAL">

<meta name="distribution" content="GLOBAL">

<meta name="classification" content="Higher Education">

<meta name="reply-to" content="www@www.univ.kiev.ua">

<meta name="revisit-after" content="10 days">

<meta name="description" content="Taras Shevchenko Kyiv University is an institution of higher education that trains specialists in many fields of knowledge and carries out research. Although this description can be applied to any other institution of similar kind, Kyiv University enjoys a special status among the establishment of higher learning in Ukraine. It is the number-one Ukrainian university, and a major centre of advanced learning and progressive thinking. It consists of more faculties and departments than any other school in Ukraine and provides training of specialists in a greater number of fields than any other comparable Ukrainian educational institution. For many years the University has been generating progressive ideas, shaping Ukrainian intellect, and providing champions of upheld national liberation activity in Ukraine.">

<meta name="KeyWords" content="Kyiv University, University, Ukraine, Higher education, fundamental science, Taras Shevchenko, Science, research">

## Совместимость с различными браузерами.

*Теперь давайте поговорим о браузерах. А что о них говорить, скажете Вы, и так все ясно - это программа, при помощи которой мы смотрим Web-странички. Ну что же, это правильно, основное их предназначение заключается в этом - просмотре Web-страничек. Но почему же в мире тогда так много броузеров? И почему они показывают странички по-разному?*

Давайте вспомним. Что лежит в основе Web-страничек? Верно, язык HTML, и определятся он как текст в формате ASCII со вставкой специальных разметочных кодов (или тэгов). И когда броузер отображает страничку, то он находит эти коды и выполняет соответствующие им действия (например, встречая тэг <b> начинает вывод последующего текста полужирным шрифтом, а встречая </b> - прекращает это делать). Полный набор таких тэгов называется спецификацией и поддерживается WWW-консорциумом, в который входят все основные компании, разрабатывающие программное обеспечение для Internet (в том числе, и Microsoft с Netscape). В идеале, все браузеры должны удовлетворять данным спецификациям, но, к сожалению, реальная жизнь и конкуренция между ведущими производителями броузеров вносят свои коррективы.

Напомню, что первоначально HTML позиционировался как язык логической разметки текста, независимый от устройства вывода. То есть тэги воспринимались как указание к выделению части текста некоторым образом. Отсюда в языке имеется множество т.н. логических тэгов типа <strong>, <em>, : , к сожалению, редко применяющихся. Фраза "независимый от устройства вывода" обозначает, что результат интерпретации HTML-кода может быть выведен не только на экран, но и, к примеру, на звуковую карту (когда компьютер сам будет читать Вам содержимое HTML-странички) или на специальное устройство, воспроизводящее содержимое странички кодом Брайля. Но получилось так, что язык HTML, в конце концов, превратился в язык описания страниц на экране компьютера.

В настоящий момент имеются два браузера, особенности которых должен учитывать web-дизайнер при изготовлении страничек. Это "Netscape Navigator" и "Microsoft Internet Explorer" (а вдруг кто не знает На самом деле браузеров гораздо больше, но они все вместе занимают всего около 1% рынка и, как правило, стремятся к совместимости с одним из двух лидеров. Между собой лидирующие браузеры делят рынок в соотношении примерно 75/25 (на данный момент лидирует Internet Explorer). Так что, в дальнейшем будут рассматриваться только эти программы.

Первое правило профессионального web-дизайна

Для чего же мы должны учитывать то, как смотрятся наши странички в обоих браузерах, если имеется стандарт языка HTML, и все они обязаны правильно просматривать код? А в том-то и дело, что интерпретируют они код по-разному (иногда очень по-разному), причем не только версии разных браузеров, но и разные версии одного и того же браузера. Причиной этого явилась конкуренция компаний, когда они вводили новые тэги для повышения конкурентоспособности своих продуктов. И как следствие любой закрытой программы, они по разному понимают значение тех или иных тэгов.

Отсюда, первое правило - профессиональный web-дизайнер просто обязан знать, как просматриваются его странички во всех популярных браузерах и их различных версиях.

Второе правило профессионального web-дизайна

Кроме этого, приходится также учитывать тот факт, что пользователи имеют компьютеры с различной аппаратной конфигурацией. Я не имею ввиду, что кто-то ходит по Web на 286 машине (а вдруг правда? , но есть много вполне современных компьютеров, которые обладают низкой разрешающей способностью. За примерами далеко ходить не надо - ноутбук. Большое их количество выпускается с черно-белым экраном, а цветные ноутбуки чаще всего ограничены палитрой в 256 цветов. Или, например, палмтопы (становящиеся популярными карманные компьютеры) под управлением Windows CE - они изначально были черно-белыми.

Вот, добрались и до второго правила - профессиональный Web-дизайнер должен просматривать странички при различном количестве цветов. И в случае необходимости делать странички с использованием только безопасной палитры.

Третье правило профессионального web-дизайна

Все странички принадлежат одной большой объединенной сети (WWW называется , и в этой сети находятся совершенно разные компьютеры: IBM PC-совместимые, Apple Macintosh, различные версии UNIX, Amiga, : Да и все что угодно. И все эти платформы, так скажем, немного отличаются при выводе графики на экран. Вряд ли шрифт в Linux будет точно таким же, как в Windows, да и палитра цветов по умолчанию может отличаться, и алгоритм дайзеринга при выводе на экран, и : В общем, если сайт важный и представляет интерес для широкого круга пользователей, то обязательно протестируйте сайт на разных платформах и под разными операционными системами. Это было третье правило профессионального Web-дизайнера

И не забывайте про текстовые браузеры, самым известным представителем которых является Lynx. Сайт может быть сколь угодно красив, но что от этого толку, если его не увидят! Поэтому всегда думайте о том, как сайт будет выглядеть в текстовом варианте. Навигация должна оставаться простой и понятной. И достигается это простыми способами: если у вас меню в виде картинки, то не поленитесь снабдить все картинки подписями, а если не получается, то продублируйте меню текстовыми ссылками внизу страницы. Владельцы текстовых браузеров будут счастливы И заметьте, что ни один из текстовых браузеров не отображает таблицы, а текст внутри ячеек выводит сплошным потоком (слева направо и сверху вниз). То же самое с фреймами.

В общем, не все ладно в мире браузеров. Придут времена и все они будут показывать странички так, как описано в спецификации, но сейчас : Сейчас нам приходится учитывать все нюансы и особенности браузеров при изготовлении Web-страниц.


# Конфигурация и управление веб-серверами.

## Встроенные средства управления сервером. (apachectl, apxs)

Встроенные средства управления apache содержатся как в самом сервере, так и в отдельных скриптах, которые идут с сервером.

Простейшиие методы отладки и запуска сервера реализованы с помощью отдельных функций, которые вызываются запуском сервера с ключами.

httpd –l – показывает способ компилирования апача и встроенные модули.

httpd –h – онлайн-хелп,

httpd –x – запуск единственного клона сервера, обычно используется для отладки.

Скрипт apachectl – интерфейс контроля сервера, основной скрипт для корректного запуска/перезапуска сервера, особенно необходим/незаменим при использовании SSL. Формат :

Apachectl {start|stop|restart|startssl|configtest|status|fullstatus}

Скрипт apxs - APache eXtenSion tool – наиболее важный скрипт для инсталляции и добавления новых модулей к серверу. Пример использования:

$ apxs -i -a -c mod\_foo.c

 gcc -fpic -DSHARED\_MODULE -I/path/to/apache/include -c mod\_foo.c

 ld -Bshareable -o mod\_foo.so mod\_foo.o

 cp mod\_foo.so /path/to/apache/libexec/mod\_foo.so

 chmod 755 /path/to/apache/libexec/mod\_foo.so

 [activating module `foo' in /path/to/apache/etc/httpd.conf]

 $ apachectl restart

 /path/to/apache/sbin/apachectl restart: httpd not running, trying to st

art

 [Tue Mar 31 11:27:55 1998] [debug] mod\_so.c(303): loaded module foo\_mod

ule

 /path/to/apache/sbin/apachectl restart: httpd started

 $ \_

Logresolve – наиболее полезный модуль с точки построения статистики посещений и мониторинга популярности сервера и попыток его взлома, в простейшем виде из обычного лога он делает лог-файл с разрезолвленными именами, что значительно упрощает его анализ.

## Глобальные разделы конфигурации.

Конфигурация сервера apache разбита на несколько основных разделов, как для удобства, так и для простоты парсинга файла.

1. Директивы, контролирующие работу сервера целиком - 'global environment'.
2. Директивы, контролирующие поведение дефолтного, или основного сервера, далее эти директивы распространяются как используемые по-умолчанию для всех виртуальных серверов.
3. Директивы виртуальных хостов, которые иерархически наследуют общие настройки и могут переопределять их.

Основные директивы глобальной конфигурации.

ServerType standalone (inetd)

Определяет тип сервера.

ServerRoot "/usr/local/httpd"

Путь к корневой директории сервера.

PidFile /usr/local/httpd/logs/httpd.pid

Путь к pid-файлу.

Timeout 300

Количество секунд которое ждет сервер перед отсылкой сообщения «сайт недоступен».

MaxKeepAliveRequests 200

Количество одновременно поддерживаемых запросов, рекомендуется ставить большим, но не ставить 0 – бесконечность.

MinSpareServers 1

MaxSpareServers 15

Параметры количества одновременно запускаемых серверов на каждый хост – первый не рекомендуют ставить 0, и не ставить достаточно большим из-за ограниченности памяти, второй – может быть достаточно большим.

StartServers 3

Минимальное количество запускаемых серверов, когда к сайтам нет никаких обращений.

MaxClients 256

Количество одновременно запросов клиентов, рекомендуется ставить максимум –256.

Listen 80

Порт. Стандартный порт для HTTP – 80, альтернативный – 8080.

Далее идет секция модулей, менять в которой что-либо необходимо только при добавлении новых модулей.

User httpd

Group httpd

Пользователь и группа процесса. (по умолчанию обычно - nobody)

ServerAdmin www@univ.kiev.ua

Далее идут настройки корневого хоста.

ServerName www.univ.kiev.ua

DocumentRoot "/usr/local/httpd/www.univ/www"

<Directory />

 Options FollowSymLinks Indexes IncludesNOEXEC

 AllowOverride All

</Directory>

Далее идут настройки виртуальных хостов.

Примеры

1.

<Directory "/usr/local/httpd/www.univ/www">

 Options FollowSymLinks Indexes IncludesNOEXEC

 AllowOverride All

 Order allow,deny

 Allow from all

</Directory>

2.

<Directory /usr/local/httpd/www.icc/www/squid/>

 Options FollowSymLinks Indexes

 AllowOverride All

 Order allow,deny

 Deny from all

 allow from icc.univ.kiev.ua

 AuthType Digest

 AuthDigestFile /usr/local/httpd/www.icc/www/squid/.htaccess

 AuthName "ICC Secret Area. Users Control"

 require valid-user

 satisfy any

</Directory>

3.

<VirtualHost kiev.philosophy.ru>

 ServerAdmin hostmaster@univ.kiev.ua

 DocumentRoot /usr/local/httpd/www.philosophy/www

 ServerName kiev.philosophy.ru

 ErrorLog /usr/local/httpd/www.philosophy/log/error\_log

 CustomLog /usr/local/httpd/www.philosophy/log/access\_log combined

 AddType "text/html;charset=koi8-r" .html .htm

</VirtualHost>

## Вспомогательные скрипты – просмотр и ротация логов, статистика посещений.

Ротация логов и их резолвинг может быть реализован с помощью стандартных средств сервера, для apache это logresolve и logrotate.

Кроме этого вы можете захотеть написать свои собственные скрипты, на пример на языке bash которые автоматизируют вашу работу. Пример:

### Going to start log cutting

for i in `ls /usr/local/httpd/` ;

 do

if [ -f /usr/local/httpd/$i'/log/access\_log' ] ; then

 /usr/bin/tail -1000 /usr/local/httpd/$i'/log/access\_log' > /usr/local/httpd/$i'/log/ttt'

 cp /usr/local/httpd/$i'/log/ttt' /usr/local/httpd/$i'/log/access\_log'

 rm /usr/local/httpd/$i'/log/ttt'

fi

done

Кроме этого могут быть использованы программы третьих сторон, различные варианты, в том числе и бесплатные и с открытым исходным кодом, доступны в большом количестве в Internet, как например одна из наиболее распространенных программ – webalizer генерирует очень полезную и наглядную статистику, написана на языке С и поэтому достаточно быстро работает. Кроме этого, лог-файлы вам все равно придется удалять (или ротэйтить), поскольку довольно быстро они займут все свободное место на диске, а иметь статистику за год-два всегда полезно для отслеживания динамики развития и популярности сайта.

## Безопасность веб-сервера.

Кpаткое описание пpоблемы:

Публичные веб-сеpвеpа пpодолжают оставаться объектами атак хакеpов, котоpые хотят с помощью этих атак нанести уpон pепутации оpганизации или добиться каких-либо политических целей. Хоpошие меpы защиты могут защитить ваш сайт от тех непpиятностей, котоpые будет иметь ваша оpганизация в случае успешной атаки на него.

Уязвимые опеpационные системы:

Любая веpсия Unix или Windows NT, котоpая используется как веб-сеpвеp.

Ущеpб от атаки:

Возможен pазличный ущеpб - от пpостого блокиpования pаботы сеpвеpа до замены его содеpжимого поpногpафическим матеpиалом, политическими лозунгами или удаления гpупп файлов, а также pазмещения на сеpвеpе пpогpамм-тpоянских коней

Как pешить пpоблему:

Соблюдать все пpавила безопасности, описанные ниже, и опеpативно устанавливать все испpавления пpогpамм, о котоpых вам сообщила ваша гpуппа компьютеpной безопасности или пpоизводитель ваших пpогpамм, используемых на веб-сеpвеpе.

Оценка pиска:

Публичные веб-сеpвеpа взламываются почти ежедневно; угpоза того, что будет совеpшена атака и на ваш веб-сеpвеp, - pеальна.

Пpавила обеспечения безопасности WWW-сеpвеpа:

1. Разместите ваш веб-сеpвеp в демилитаpизованной зоне (DMZ). Сконфигуpиpуйте свой межсетевой экpан (файpволл) таким обpазом, чтобы он блокиpовал входящие соединения с вашим веб-сеpвеpом со всеми поpтами, кpоме http (поpт 80) или https (поpт 443).
2. Удалите все ненужные сеpвисы с вашего веб-сеpвеpа, оставив FTP (но только если он нужен на самом деле) и сpедство безопасного подключения в pежиме удаленного теpминала, такое как SSH. Любой ненужный, но оставленный сеpвис может стать помощником хакеpа пpи оpганизации им атаки.
3. Отключите все сpедства удаленного администpиpования, если они не используют шифpования всех данных сеансов или одноpазовых паpолей.
4. Огpаничьте число людей, имеющих полномочия администpатоpа или супеpпользователя (root).
5. Пpотоколиpуйте все действия пользователей и хpаните системные жуpналы либо в зашифpованной фоpме на веб-сеpвеpе либо на дpугой машине в вашем интpанете.
6. Пpоизводите pегуляpные пpовеpки системных жуpналов на пpедмет выявления подозpительной активности. Установите несколько пpогpамм-ловушек для обнаpужения фактов атак сеpвеpа (напpимеp, ловушку для выявления PHF-атаки). Напишите пpогpаммы, котоpые запускаются каждый час или около того, котоpые пpовеpяют целостность файла паpолей и дpугих кpитических файлов. Если такая пpогpамма обнаpужит изменения в контpолиpуемых файлах, она должна посылать письмо системному администpатоpу.
7. Удалите все ненужные файлы, такие как phf, из диpектоpий, откуда могут запускаться скpипты (напpимеp, из /cgi-bin).
8. Удалите все стандаpтные диpектоpии с документами, котоpые поставляются с веб-сеpвеpами, такими как IIS и ExAir.
9. Устанавливайте все необходимые испpавления пpогpамм на веб-сеpвеpе, касающиеся безопасности, как только о них становится известно.
10. Если вы должны использовать гpафический интеpфейс на консоли администpатоpа веб-сеpвеpа, удалите команды, котоpые автоматически запускают его с помощью инфоpмации в .RC-поддиpектоpиях и вместо этого создайте команду для его pучного запуска. Вы можете затем пpи необходимости использовать гpафический интеpфейс, но закpывать его тотчас же после того, как вы пpоизведете необходимые действия. Не оставляйте гpафический интеpфейс pаботающим пpодолжительный пеpиод вpемени.
11. Если машина должна администpиpоваться удаленно, тpебуйте, чтобы использовалась пpогpамма, устанавливающая защищенное соединение с веб-сеpвеpом (напpимеp, SSH). Не позволяйте устанавливать с веб-сеpвеpом telnet-соединения или неанонимные ftp-соединения (то есть те, котоpые тpебуют ввода имени и паpоля) с недовеpенных машин. Неплохо будет также пpедоставить возможность установления таких соединений лишь небольшому числу защищенных машин, котоpые находятся в вашем интpанете.
12. Запускайте веб-сеpвеp в chroot-pежиме или pежиме изолиpованной диpектоpии (в этом pежиме эта диpектоpия кажется коpневой диpектоpией файловой системы и доступ к диpектоpиям файловой системы вне ее невозможен), чтобы нельзя было получить доступ к системным файлам.
13. Используйте анонимный FTP-сеpвеp (если он конечно вам нужен) в pежиме изолиpованной диpектоpии для диpектоpии, отличной от диpектоpии, являющейся коpнем документов веб-сеpвеpа.
14. Пpоизводите все обновления документов на публичном сеpвеpе из вашего интpанета. Хpаните оpигиналы ваших веб-стpаниц на веб-сеpвеpе в вашем интpанете и сначала обновляйте их на этом внутpеннем сеpвеpе; потом копиpуйте обновленные веб-стpаницы на публичный сеpвеp с помощью SSL-соединения. Если вы будете делать это каждый час, вы избежите того, что испоpченное содеpжимое сеpвеpа будет доступно в Интеpнет долгое вpемя.
15. Пеpиодически сканиpуйте ваш веб-сеpвеp такими сpедствами, как ISS или nmap, для пpовеpки отсутствия на нем известных уязвимых мест.
16. Оpганизуйте наблюдение за соединениями с сеpвеpом с помощью пpогpаммы обнаpужения атак (intrusion detection). Сконфигуpиpуйте эту пpогpамму так, чтобы она подавала сигналы тpевоги пpи обнаpужении попыток пpименить известные атаки или подозpительных действиях с веб-сеpвеpом, а также пpотоколиpовала такие соединения для детального анализа. Эта инфоpмация сможет впоследствии вам помочь устpанить уязвимые места и усилить вашу систему защиты.

ЕСЛИ ВАШ ВЕБ-САЙТ БЫЛ ВЗЛОМАН:

CIAC pекомендует следующие шаги пpи пpовеpке веб-сеpвеpа:

1. Установить ВСЕ испpавления, связанные с безопасностью, как для самого веб-сеpвеpа, так и для опеpационной системы.
2. Удалить ВСЕ ненужные файлы, такие как phf из диpектоpии со скpиптами. Удалить стандаpтные диpектоpии с документами, поставляемые с веб-сеpвеpом (напpимеp, с IIS и ExAir).
3. Пpовеpить ВСЕ логины пользователей на веб-сеpвеpе и удостовеpиться в том, что они имеют тяжело угадываемые паpоли.
4. Пpовеpить ВСЕ сеpвисы и откpытые поpты на веб-сеpвеpе, чтобы удостовеpиться в том, что вместо них не установлены пpогpаммы-тpоянские кони.
5. Пpовеpить, нет ли подозpительных файлов в диpектоpиях /dev, /etc и /tmp.

## Организация доступа и разграничение прав пользователей.

Обычно пользователи , которые имеют доступ к сайту пользуются FTP для доступа к сайту. Для того чтобы сделать этот процесс безопасным, следует организовать доступ для каждого пользователя только к своей директории, обычно это делается с использованием chroot в FTP. Кроме того, не следует давать ftp-пользователям шелл на сервере, оптимальнее всего делать шеллесс эккаунты или эккаунты с ограниченными правами. Для пользователей можно ввести общую группу, например www, если необходим совместный доступ к нескольким сайтам, при этом группа не должна совпадать с группой демона сервера.

## . Подключение новых модулей и апгрейд программного обеспечения веб-сайта.

Основной вопрос который необходимо решить – когда производить апгрейд сайта и нужно ли подключать новые модули. Тут нам необходимо вернуться несколько назад к инсталляции и в вспомнить, что все-таки лучше собирать серверное ПО в модульном варианте. Таким образом нам не приходится перекомпилировать весь веб-сервер, достаточно скомпилировать сам модуль и добавить его в конфигурацию.

Например, нам необходимо добавить модуль FastCGI к существующей конфигурации. Наши действия:

Развернем mod\_fastcgi\_2.2.4 и запустим такое в папке дистрибутива:

**/usr/local/httpd/bin/apxs -o mod\_fastcgi.so -c \*.c**

**/usr/local/httpd/bin/apxs -i -a -n fastcgi mod\_fastcgi.so**

**В httpd.conf, ежели не добавилось автоматом - надо соответственно добавить**

**AddModule mod\_fastcgi.c**

**И**

**LoadModule fastcgi\_module mod\_fastcgi.so**

Апгрейд. Когда следует апгрейдить сервер до более свежей версии? Прежеде всего это следует делать или устанавливать патч, когда в текущей версии ПО обнаружены security holes, ошибки или участки кода приводящие к слету системы. Далее, поскольку апгрейд достаточно нудная и трудоемкая процедура, следует на сайте проидводителя прочитать CHANGES или WHATSNEW-документы и определиться – есть ли в новой версии что-либо, что вам очень необходимо и есть ли там какие-то новые полезные свойства, ради которых стоит сменить версию? Если таковых не обнаружено, то заниматься сменой версии, соответственно не надо.

# Администрирование веб-сервера.

## Логгирование и поиск ошибок.

Большая часть ошибок может быть обнаружена в error\_log, который ведет сервер, как, например, остутствующие картинки, ссылки, ошибки в скриптах могут быть обнаружены только таким образом, если они размещены на сервере и уже используются. Просмотр логов так же необходим для поиска попыток взлома и нарушения использования сервера.

Веб-сервера

Многие организации сейчас поддерживают внешние WWW-сайты, описывающие их компанию или сервисы. По причинам безопасности эти сервера обычно размещаются за брандмауэром компании. Веб-сайты могут быть как домашними разработками, так и тщательно разработанными средствами продвижения товаров. Организации могут тратить значительное количество денег и времени на разработку веб-сайта, предоставляющего большой объем информации в удобном виде или создающего имидж компании. Другими словами, веб-сайт организации является одной из форм создания имиджа и репутации компании.

Должны быть назначены ответственные за создание, управление и администрирование внешнего веб-сайта компании. В больших компаниях это может входить в обязанности нескольких должностей. Например, коммерческий директор может отвечать за выявление и реализацию новых способов продвижения товаров и услуг, а администратор веб-сайта - за соблюдение на нем общей стратегии, включая координированную подготовку его содержимого и контроль за его бюджетом. Начальник отдела продаж может отвечать за представление отчетов о доходах, связанных с ведением веб-сайта. А вебмастер будет отвечать за технические аспекты веб-сайта, включая разработку, поддержание связи с ним, интранет, электронную почту, и безопасность брандмауэра. Скорее всего программисты будут отвечать за работоспособность веб-сайта, включая его установку, разработку программ для него, их отладку и документирование. Вебартист может заниматься созданием графических образов для него.

В более маленьких организациях программист или вебмастер может выполнять большую часть описанных выше обязанностей и предоставлять доклады пресс-службе или начальнику отдела продаж. Наконец, в очень маленькой организации эти обязанности могут стать дополнительными обязанностями системного аналитика или администратора ЛВС. Независимо от того, как администрируется вебсайт, все люди, исполняющие эти обязанности должны претворять в жизнь политику компании, разработанную ее руководством. Верхнее звено руководства организацией может отвечать за утверждение создания новых веб-сайтов или переработку имеющихся.

Кроме того, внутренние веб-сайты компании, расположенные внутри брандмауэра организации, часто используются для рассылки информации компании сотрудникам. Часто посылаются поздравления с днем рождения, графики мероприятий, телефонные справочники и т.д. Также внутренние веб-сайты используются для распространения внутренней информации о проектах, являясь иногда центром информации для исследовательских групп. Хотя внутренние веб-сайты не являются так же видимыми, как внешние страницы, они должны администрироваться с помощью специально разработанных руководств и директив. Руководители групп должны отвечать за это.

Любой может создать веб-сайт для распространения информации по любым вопросам, не связанным с организацией. Организация должна принять решение о том, стоит ли разрешать сотрудникам делать это на чужих веб-сайтах.

Большинство организаций используют Интернет для распространения информации о себе и своих сервисах. Так как они представляют информацию, а не скрывают ее, они описывают веб-сайт как "публичный", на котором не содержится никакой конфиденциальной информации, и оттуда не может исходить никакой угрозы. Проблема заключается в том, что хотя эта информация может быть публично доступной, веб-сайт является частью организации, и должен быть защищен от вандализма.

Публичные вебсайты в MGM/Universal Studios, Nation of Islam, Министерстве юстиции США и ЦРУ могут подтвердить это утверждение. В них были совершены проникновения в 1996 году. Атакующие использовали уязвимые места в операционной системе, под управлением которой работали веб-сервера. Они изменили ряд страниц веб-сайтов, и в некоторых случаях добавили порнографические изображения, и в одном случае вставили оскорбления.

Хотя единственным следствием таких проникновений была потеря репутации, это может оказаться достаточным, чтобы не захотеть испытать это во второй раз. Если бы атакующие модифицировали описания сервисов организации, фальсифицировали цены, то последствия могли бы быть гораздо серьезнее.

Примеры политик веб-серверов

Низкий риск

*Пользователь*

На веб-сайтах организации не может размещаться оскорбительный или нудный материал.

На веб-сайтах организации не может размещаться персональные рекламные объявления

*Менеджер*

Менеджерам и пользователям разрешено иметь веб-сайт.

Материалы о сотрудниках на веб-сайтах или доступные с их помощью должны быть минимальны.

На веб-сайтах организации не может размещаться оскорбительный или нудный материал.

Конфиденциальная информация ее должна делаться доступной.

*Сотрудник отдела автоматизации*

Должен поддерживаться и быть доступен для внутреннего пользования локальный архив программ веб-серверов и средств публикации информации на них.

Средний риск

*Пользователь*

Пользователям запрещено устанавливать или запускать веб-сервера.

В отношении веб-страниц должен соблюдаться установленный в организации порядок утверждения документов, отчетов, маркетинговой информации и т.д.

*Менеджер*

Менеджерам и пользователям разрешено иметь веб-страницы для участия в проекте или выполнения своих должностных обязанностей

*Сотрудник отдела автоматизации*

Веб-сервер и любые данные, являющиеся публично доступными, должны быть размещены за пределами брандмауэра организации.

Веб-сервера должны сконфигурированы так, чтобы пользователи не могли устанавливать CGI-скрипты

Все сетевые приложения, кроме HTTP, должны быть отключены (например, SMTP, FTP и т.д.)

Информационные сервера должны быть размещены в защищенной подсети для изоляции их от других систем организации. Это уменьшает вероятность того, что информационный сервер будет скомпрометирован и использован для атаки на другие системы организации.

При использования средств администрирования с помощью WWW, ограничьте доступ к нему только авторизованных систем (с помощью IP-адресов, а не имен хостов). Всегда меняйте пароли по умолчанию.

Высокий риск

*Пользователь*

Пользователям запрещено загружать, устанавливать или запускать программы веб-серверов.

Должен производиться контроль сетевого трафика для выявления неавторизованных веб-серверов. Операторы этих серверов будут подвергаться дисциплинарным наказаниям.

*Менеджер*

Руководство организации должно дать в письменном виде разрешение на работу веб-сервера, подключенного к Интернету.

Все содержимое веб-серверов компании, присоединенных к Интернету, должно быть утверждено и установлено веб-мастером.

Конфиденциальная информация не должна быть доступна с помощью веб-сайта.

К информации, размещенной на веб-сервере, применимы все законы о защите информации. Поэтому, перед размещением информации в Интернете, она должна быть просмотрена и утверждена так же, как утверждаются бумажные официальные документы организации. Должны быть защищены авторские права, и получено разрешение о публикации информации на веб-сайте.

Все публично доступные веб-сайты должны регулярно тестироваться на предмет корректности ссылок, и не должны находиться в состоянии "under construction". При реконструкции областей они должны делаться недоступными.

*Сотрудник отдела автоматизации*

Не должно иметься средств удаленного управления веб-сервером (то есть с мест, отличных от консоли). Все действия администратора должны делаться только с консоли. Вход в систему с удаленного терминала с правами суперпользователя должен быть запрещен.

Программы веб-серверов и операционной системы, под управлением которой работает веб-сервер, должны содержать все исправления, рекомендованные производителем для этой версии.

Входящий трафик HTTP должен сканироваться, и о случаях появления неавторизованных веб-серверов должно докладываться

Ограничение доступа к информации пользователями, адрес которых заканчивается на .GOV или .COM, обеспечивает минимальную защиту для информации, не разрешенной для показа всем. Может использоваться отдельный сервер или отдельная часть для информации с ограниченным доступом.

За всеми веб-сайтами должен осуществляться контроль как составная часть администрирования сети. Действия всех пользователей, заподозренных в некорректном использовании Интернете, могут быть запротоколированы для обоснования применения к ним в дальнейшем административных санкций.

На UNIX-системах веб-сервера не должны запускаться с правами суперпользователя.

Разработка и использование CGI-скриптов должно контролироваться. CGI-скрипты не должны обрабатывать входные данные без их проверки . Любые внешние программы, запускаемые с параметрами в командной строке, не должны содержать метасимволов. Разработчики отвечают за использование правильных регулярных выражений для сканирования метасимволов командного процессора и их удаление перед передачей входных данных программа на сервере и операционной системе.

Все WWW-сервера организации, подключенные к Интернету, должны находиться между брандмауэром и внутренней сетью организации. Любые внутренние WWW-сервера организации, обеспечивающие работу критических приложений организации должны быть защищены внутренними брандмауэрами. Критическая, конфиденциальная и персональная информация никогда не должны храниться на внешнем WWW-сервере.

## Обеспечение безопасности .

1. брандмауэр

1. фильтрация пакетов и разделение сетей


## Создание резервных копий. Технологии: Backup, mirroring.

 Необходимость создания резервных копий становится очевидной после первого слета системы, как, например, когда вы теряете практически все данные и на их восстановление уходит несколько суток, стоит задуматься об избежании таких ситуаций.

 Две основные технологии резервного копирования – простой бэкап и мирроринг. Простой бекап – это когда вы копируете всю существенную информацию сайта на какой-либо носитель, винчестер, стример, магнитооптику, PCMCI-диски и т.п. При этом рекомендуется делать так же копию всех установленных позже программ и бинарных файлов. Если позволяют размеры носителя – оптимально сделать полный бекап всей системы, в таком случае при слете сервера время восстановления определяется скоростью чтения из устройства.

 Мирроринг. Технология может быть реализована двумя методами. Первый – резервный винчестер, который может быть реализован в одном корпусе на основе технологии SCSI настройкой адаптера или с использованием технологий RAID. Второй – создание резервного сервера-зеркала, для которого информация синхронизирована с основным. Это достаточно дорогой способ, так как требует дополнительных расходов.

# Система безопасности HP-UX

# Политика и планирование системы безопасности

Не имеется несколько методов для разработки политики защиты. **Вот более общий подход**.

* Идентифицировать то, что Вы должны защитить. Это может быть активы типа даные пользователей, доступ до аппаратных средств, данные, документация и т.д.
* Идентифицировать потенциальные угрозы вашим ресурсам.. Они включают угрозы от природных явлений (наводнения, землетрясения), невежество и недостаток обучения пользователей и намеренных нарушений защиты.
* Оценить вероятность этих угроз, повреждающих ваши ресурсы.
* Прокласифицировать риски уровнем серьезности, и определить стоимость для сокращения того риска (это также известно как оценка риска).
* Осуществляют меры, которые защитят ресурсы.

**Общие действия защиты включают следующее:**

* Ограничить вход в систему доступ к программному обеспечению.

* Пользователи должны выходить или используют команду блокировки при не использовании их терминалов.
* Сохранить резервные ленты (диски) в отдаленных местах
* Стереть устаревшие данные.

**Подержка системы защиты включает**:

* (identification) Идентификация пользователей. Все пользователи должны иметь уникальный идентефикатор(ID) входа в систему, состоящим из названия и пароля.
* (authentication) Установление подлинности пользователей. Когда пользователь войдет, система подтверждает подлинность его пароля, проверяя существование в файле пароля.
* (authorization) Разрешение пользователей. На системном уровне, HP-UX обеспечивает два вида компьютерного использования – обычный и суперпользователь. Индивидуальным пользователям можно предоставлять или ограниченный доступ к системным файлам через традиционные разрешения файла, списки контроля доступа, и запретить SAM
* (audit) Ревизия gользователей. HP-UX дает возможность ревизовать компьютерное использование пользователями и события.

## Установка Trusted Системы

HP-UX предлагает дополнительный инструментарий для безопасности системы.

Для конвертации в trusted систему можно использовать SAM в разделе Auditing and Security. Также можна сделать это вручною редактируя скрипт **/etc/rc.config.d/auditing**.

После «конвертации» стелаються следующие действия

1. Создает новый, защищенная база данных пароля в /tcb/files/auth/.
2. Зашифровка паролей с /etc/passwd файла до защищенной базы данных пароля и заменяют поле пароля в /etc/passwd со звездочкой (\*). Вы должны копировать /etc/passwd файл, чтобы записать на ленту перед преобразованием.
3. Вынуждает всех пользователей использовать пароли
4. Создает audit ID для каждого пользователя.я.
5. Устанавливает audit флажок на для всех существующих пользователей
6. Конвертирует at,batch и crontab файлы, чтобы использовать установлные audit ID

Для аудитинга используют следующие команды:

audsys(1M) установка/отмена фудитинга и показывает ревезионные файли

audusr(1M) выбор ползователя для аудита

audevent(1M) просмотр и изменения событий и системеных вызовов

audomon(1M) устанавливает аудит файл и размер для мониторинга

audisp(1M) показывает аудит установки (записи)

Также все это можно сделать визуально в SAM разделе Auditing and Security

## Управление паролями и системным доступом

Пароль - наиболее важный индивидуальный код (символы) идентификации пользователя. Этим, система подтверждает подлинность пользователя, чтобы позволить доступ к системе. Администратор и обычный пользователь в системе долженй совместно использовать ответственность за защиту пароля. Администратор исполняет следующие задачи защиты:

* Генерирует ID и для системы новым пользователям. Чтобы поддерживать секретность пароля, SAM генерирует Номер Разрешения для каждого нового пользователя. Этот номер должен использоваться для первого входа в систему. Как только этот номер был проверен, новому пользователю будет дано установить свой новый пароль
* устанавливает надлежащий доступ на /etc/passwd и зашифрованом пароле, в /tcb/files/auth/user\_initial/user\_name файлы.
* Устанавливает старение пароляоля.
* Удаление(стирание) паролей у каких вышел срок действия

Каждый пользователь должен выполнять следующие правила:

* Помнить пароль и держать его в секрете
* Изменять переодически пароли
* Следить за изменеием своих данных
* Для каждой машине иметь разные пароли

# Управлением доступом к файлам и каталогам

ВHP-UX системе, Вы используете ls -l команду, чтобы видеть полную распечатку разрешений файла и ls -ld, чтобы перечислить разрешения каталога.

Chmod (1) команда позволяет Вам изменять разрешения каталогов и файлов.

Вы можете дополнительно использовать списки контроля доступа (ACLs), чтобы расширитьтрадиционный механизм разрешения, давая пользователям большeую степень управления доступом. Разрешения доступа и ограничения могут быть определены к степени детализации определенных пользователей и групп.

chacl (1) создает и изменяет ACLs и lsacl (1) показывает списки ACLs файлов.

Команда chacl - подмножество команды chmod. Любые определенные разрешения, которые Вы назначаете с командой chacl, добавлены к большему количеству общих разрешений, назначенных с командой chmod. Например, предположите, что Вы используете команду chmod, чтобы позволить только непосредственно разрешение записи myfile. Вы можете использовать команду chacl, чтобы делать исключение и позволять ваше разрешение записи администратора myfile также.

Используйте chmod с -A опцией при работе с файлами, которые имеют дополнительные назначенные разрешения. Дополнительные разрешения будут удалены.

Вот пример использования команды

**$ chacl 'user.group operator mode' file\_name**

где user и group указывает название в систему пользователя и группу; знак процента (%) означает всех пользователей или группы. Оператор указывает добавление (+) или отрицание (-) разрешения и знаки "=" (=) средства " это разрешение точно. " Режим указывает позволенные разрешения: чтение (r), запись (w), и выполнение /поиск (x). Оператор немедленно предшествует режиму (например, + rw добавляет разрешения записи и чтение; -rw запрещает чтение и разрешения записи)

Вот еще примеры:

**$ chacl 'carolyn.users=rw' myfile**

 **$ ll myfile**

 **-rw-r-----+ 1 nora users 236 Mar 8 14:23 myfile**

 **$ lsacl myfile**

 **(carolyn.users,rw-) (nora.%,rwx) (%.users,r--)(%.%,---) myfile**

Для установки разрешения по умалчанию переменую окружения **umask**. В нем передаються параметры защиты. Напримар umask=022 (2 –w, 1 –x, 4 –r) означает что во всех открытых фыйлах по умолчаню не будет прав зяписи для групы и всех остальных пользователей.

Администратор должен установить

* начальные права для каталога пользователя и дальше пользователь должен следить за защитой своей информации.
* «правыльные» права на утсройства. (/dev)

Ниже привиден список команд для системы контроля доступа файловой системы

chacl(1) - change ACLs of files.

getaccess(1) - list access rights to files.

lsacl(1) - list access control lists of files.

getaccess(2) - get a user's effective access rights to a file.

getacl, fgetacl(2) - get access control list information.

setacl, fsetacl(2) - set access control list information.

acltostr(3C) - convert ACL structure to string form.

chownacl(3C) - change owner/group represented in a file's ACL.

cpacl(3C), fcpacl(3C) - copy ACL and mode bits from one file to another.

setaclentry(3C), fsetaclentry(3C) - add/modify/delete a file's ACL entry.

strtoacl(3C) - parse and convert ACL structure to string form.

strtoaclpatt(3C) - parse and convert ACL pattern strings to arrays.

## Контроль безопасности сети (networks)

Сетевые системы более узяввымие в палне защищености чем без нее (standalone). Сеть увеличивает системны доступ а так же добавляет большой риск в безопасности системы.

Если вы не можете управлять всей защитой сети то вам необходимо защищать каждую станцию отдельно.

Ниже приведены основые механизмы контроля доступом по сети

1. Перечинь экспортиртируемыз файловых систем /etc/exports.

/etc/exports содержит входы, которые состоят из имени пути файловой системы, сопровождаемой списком компьютеров или групп компьютеров, позволенных доступ к файловой системе. Любой вход, состоящий из только имя пути без того, чтобы следоваться компьютерным названием- файловая система, доступная каждому компьютеру на сети. /etc/exports входы могли бы содержать названия групп компьютеров. Вы можете выяснять то, какие индивидуальные машины включены в группу, проверяя /etc/netgroup.

2. Перечислить узлов, которые имеют эквивалентные базы паролей в /etc/hosts. equiv.

3. Проверить, что каждый узел в административном домене не расширяет привилегии на любых невключенных узлов.

Вы должны повторить шаги 1 и 2 для каждого узла в домене(области).

4. Конролируйте root и локальную защиту на каждом узле в вашем административном домене Пользователь с привилегиями суперпользователя на любой машине в домене может приобретать те привилегии на каждой машине в домене. !!!

5. Поддержать последовательность названий пользователей, uid, и gid среди файлов пароля в вашем административном домене.

 6.Поддержать последовательность файлов группы на всех узлах в вашем административном домене.

Режимы, владельцы, и группы на всех системных файлах должы быть установлены тщательно. Все отклонения от этих значений должны быть отмечены и исправлены.

Обратите внимание на файлы которые находяться в **/etc**. Ниже приведен список наиболее употребляемых файлов

networks название сетей и их адреса

hosts название станций а также их адреса

hosts.equiv название и адреса станций в которые эквивалентны даной станции

services база данных сервисов

exports список экспорта файловых систем, экспортируемых в NFS клиенту

protocols база данныз протоколов

inetd.conf файл конфигурации Internet

netgroup Список сетевых групп.

Использують indetd.sec для контроля внешнего доступа. Файл находиться в /var/adm/inetd.sec

По следующему формату : **<service name> <allow/deny> <host/net addresses, host/net names>**

**Мониторинг системы**

Имееться набор команд для мониторинга системы. Ниже приведены краткие возможности и характеристики наиболее часто используемых:

**SAR** – показывает активные ресурсы сиситемы (system activity reporter)

Запуск команды возможен в 2х вариантах:

**sar [-ubdycwaqvmAMS] [-o file] t [n]**

и

**sar [-ubdycwaqvmAMS] [-s time] [-e time] [-i sec] [-f file]**

Первая форма показывает октивность комапьютера n раз с периодом t секунд. Если указана опция –o то информацию скидывает в файл. По умолчанию n = 1.

Другая форма без осуществления выборки указанного интервала, sar извлекает данные от предварительно зарегистрированного файла, или тот, указанный -f опцией или, по умолчанию, стандартные действия операционной системы ежедневный файл данных /var/adm/sa/sadd в течение текущего дня dd. Начальные и конечные времена сообщения могут быть ограничены через -s и -e параметры времени формы hh [:mm [:ss]]. -i опция выбирает отчеты в секундных интервалах. Иначе, все интервалы, найденные в файле данных сообщены.

Расмотрим опции:

-u использование CPU.(значение по умолчанию); часть времени, выполняющегося в одном из нескольких режимов. На многопроцессорной системе, если -M опция используется вместе с -u опцией, для каждого -CPU использование также как среднее использование CPU всех процессоров. Если -M опция не используется,тогда показывает среднее использование CPU всех процессоров:

 cpu номер CPU(только на многопроцессорной системе с -M опцией);

%usr использование пользователем (непgjrfривигильованый режим);

 %sys системный режим;

%wio простой с некоторым процессом, ожидающим Ввод - вывод (только блочных I/O, необработанный ввод /вывод, или VM обозначенные загрузки/выгрузки свопа);

 %idle “простой» проссора.

-b показывает использования буфера

bread/s- количество физический чтений на секунду с буф. на диск.(или устройсва)

-bwrit/s количество физический записей на секунду с буф. на диск.(или устройсва)

 lread/s - lwrit/s количество байт чтения записи на устройства.

 %rcache отношения буфера- bread/lread

 %wcache bwrit/lwrit

 pread/s Количество чтений не с блочного устройства

 pwrit/s Количество записей не с блочного устройства

-d показывает информацию о каждом block устройстве, к которому процесор имел отношение за определенный интервал времени

-y ------- для non-block устройств ---

-с системные вызовы

-w свопинг и переключения системы

-a использование файлов

-q показывает среднюю длину сообщения и процент от занятого времени

-v более детальная информация об использовании файлов,inode, процессов.

-m информация о семафорах

-A показывает всю информацию

-M информация для индивидуального CPU в много-процесорных машинах

**VMSTAT**- показывает статистику виртуальной памяти

Параметры запуска комманды:

vmstat [-dnS] [interval [count]]

vmstat -f | -s | -z

-d сообщает количество обменов между диском

-n выводин информацию в 80 колоночном представлени.

-S количество прцессов которые свопяттся

interval период отображения

count количество повторов

-f количество fork ( деления процесса)

-s количество paging сообщений

-z очищает все sum структуры ядра.

**IOSTAT** мониторинг I|O устройств

Параметры запуска:

 iostat [-t] [interval [count]]

interval период отображения

count количество повторов

-t отображает статистику для терминалов.

Для каждого диска статистика предоставляеться форматом:

 device имя устройства

bps количество передаваемых байт за секунду

sps количество seekов за секунду

msps в мс. средний seek

**TOP** отображает состояние “горячих” процесов.

Параметры запуска:

 top [-s time] [-d count] [-q] [-u] [-n number]

-s timе время между обновлениями

-d count отображает количество и после этого выходит

-n number количество «верхних» процесов

-q запускает с самым большим приоритетом (nice –20)

-u мвесто username показывает userID (для экономии проц. времени)

### LSOF –отображает открытые файлы и процесы кто их открыл. Очень полезно для контроля доступа к устройствам.

**NETSTAT** показывает состояние сети.

Параметры запуска:

netstat [-aAn] [-f address-family] [system [core]]

netstat [-mMnrsv] [-f address-family] [-p protocol] [system [core]]

netstat [-gin] [-I interface] [interval] [system [core]]

netstat отображает статистику для сетевых интерфейсов и протоколов, также как содержания различных связанных сетью структур данных. Выходной формат изменяется согласно отобранным параметрам. Некоторые параметры игнорируются когда используется в комбинации с другими параметрами.

 Команда netstat берет одну из трех форм, показанных выше:

* Первая форма команды отображает список активных сокетов для каждого протокола.
* Вторая форма отображает содержание одной из других сетевых структур данных согласно отобранной опции.
* Третья форма отображает информацию конфигурации для каждого сетевого интерфейса. Это также отображает сетевые данные трафика относительно конфигурированных сетевых интерфейсов, произвольно модифицированных в каждом интервале, измеренном мгновенно.

Опции:

-a Показывает состояние всех сокетов.

-A Адресный блок и протокол

-f address-family Показывает сокеты с определеного семейства: inet,unix (AF\_INET,AF\_UNIX)

-g Показывает информацию о широковещательных интерфейсах

-i Состояние интерфейсов

-I interface Состояние определленого интерфейса

-m Колсичество памяти и общая информация о сокетах

-M Multicast таблица маршрутизации

-n Показывать адрес сетки как номер

-p protocol Показывать всю статистику определьоного протокола

-r Таблица маршрутизации

-s Показать статистику по всех протоколах

-v Дополнительная информация

Существует множество прормам мониторинга системы. Некоторые можно мониторить с помощю команды SAM. Также можно найти дополнительную информацию на сайте:

<http://hpux.cae.qiax.edu>