**Федеральное агентство по образованию Российской Федерации
Белгородский Государственный Университет
Факультет компьютерных наук и телекоммуникаций
Кафедра математического обеспечения и администрирования
 информационных систем**

КУРСОВАЯ РАБОТА

**по дисциплине:** «Операционные системы и оболочки»
**на тему:** «Многопроцессорные системы (Операционные системы UNIX)»

**010503.140702.1407011**

**Студент:** Ю.В.Семенко
**Руководитель курсовой работы:** В.Н. Павлов

 **Оценка курсовой работы:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
**Принял:**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Дата** \_\_\_\_\_\_\_\_\_\_\_\_\_

**Белгород 2009**

ПЛАН

КУРСОВОЙ РАБОТЫ

по дисциплине: «*Операционные системы и оболочки*»

на тему: «*Многопроцессорные системы (Операционные системы UNIX)*».

Введение

1. Структура и специфика многопроцессорных систем
	1. История появления и развития многопроцессорных систем
	2. Общая структура многопроцессорной системы
	3. Спецификация аппаратных средств многопроцессорной системы
2. Операционные системы для работы с многопроцессорными системами
	1. Первые операционные системы для работы в многопроцессорной конфигурации
	2. Виды многопроцессорных систем и операционные системы для работы с ними
3. Операционная система UNIX в многопроцессорной конфигурации
	1. Общий обзор особенностей системы UNIX
		1. Преимущества операционная система UNIX
		2. Недостатки операционная система UNIX
	2. Cистемы UNIX в многопроцессорной конфигурации
	3. Проблемы, связанные с многопроцессорными системами и решение их в операционной системе UNIX

Заключение

Список используемой литературы

Исполнитель:\_\_\_\_\_\_\_\_\_\_/ Семенко Ю.В. /

Руководитель:\_\_\_\_\_\_\_\_\_\_/ Павлов В.Н. /

Дата:\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

ВВЕДЕНИЕ **5**

1. Структура и специфика многопроцессорных систем **6**

1.1 История появления и развития многопроцессорных систем **7**

1.2 Общая структура многопроцессорной системы **10**

1.3 Спецификация аппаратных средств многопроцессорной системы **12**

2. Операционные системы для работы с многопроцессорными системами **14**

2.1 Первые операционные системы для работы в многопроцессорной конфигурации **15**

2.2 Виды многопроцессорных систем и операционные системы для работа с ними **17**

3. Операционная система UNIX в многопроцессорной конфигурации........................**.19**

3.1 Общий обзор особенностей системы UNIX **21**

3.1.1 Преимущества операционная система UNIX **23**

3.1.2 Недостатки операционная система UNIX **25**

3.2 Cистемы UNIX в многопроцессорной конфигурации **27**

3.3 Проблемы, связанные с многопроцессорными системами и решение их в операционной системе UNIX **30**

ЗАКЛЮЧЕНИЕ **37**

Список литературы **38**

**ВВЕДЕНИЕ**

Прошло немногим более 50 лет с момента появления первых электронных вычислительных машин - компьютеров. За это время сфера их применения охватила практически все области человеческой деятельности. Сегодня невозможно представить себе эффективную организацию работы без применения компьютеров в таких областях, как планирование и управление производством, проектирование и разработка сложных технических устройств, издательская деятельность, образование - словом, во всех областях, где возникает необходимость в обработке больших объемов информации. Однако наиболее важным по-прежнему остается использование их в том направлении, для которого они собственно и создавались, а именно, для решения больших задач, требующих выполнения громадных объемов вычислений. Такие задачи возникли в середине прошлого века в связи с развитием атомной энергетики, авиастроения, ракетно-космических технологий и ряда других областей науки и техники.

Потребность решения сложных прикладных задач с большим объемом вычислений и принципиальная ограниченность максимального быстродействия "классических" - по схеме фон Неймана - ЭВМ привели к появлению многопроцессорных вычислительных систем (МВС). Использование таких средств вычислительной техники позволяет существенно увеличивать производительность ЭВМ при любом существующем уровне развития компьютерного оборудования. При этом, однако, необходимо "параллельное" обобщение традиционной - последовательной - технологии решения задач на ЭВМ. Так, численные методы в случае многопроцессорных систем должны проектироваться как системы параллельных и взаимодействующих между собой процессов, допускающих исполнение на независимых процессорах.

Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами, следовательно должно создаваться специальное программное обеспечение для решения этих проблем, а следовательно операционные системы для работы в многопроцессорной конфигурации.

Курсовая работа содержит:

38 страниц;

2 рисунка;

7 литературных источников.

1. **Структура и специфика многопроцессорных систем**

Многопроцессорная архитектура, включает в себя два и более центральных процессоров (ЦП), совместно использующих общую память и периферийные устройства (Рис.1), располагая большими возможностями в увеличении производительности системы, связанными с одновременным исполнением процессов на разных ЦП. Каждый ЦП функционирует независимо от других, но все они работают с одним и тем же ядром операционной системы. Поведение процессов в такой системе ничем не отличается от поведения в однопроцессорной системе - с сохранением семантики обращения к каждой системной функции - но при этом они могут открыто перемещаться с одного процессора на другой. Хотя, это не приводит к снижению затрат процессорного времени, связанного с выполнением процесса. Отдельные многопроцессорные системы называются системами с присоединенными процессорами, поскольку в них периферийные устройства доступны не для всех процессоров.

Параллельная работа нескольких процессоров в режиме ядра по выполнению различных процессов создает ряд проблем, связанных с сохранением целостности данных и решаемых благодаря использованию соответствующих механизмов защиты.

Рис 1. Многопроцессорная конфигурация

* 1. **История появления и развития многопроцессорных систем**

Первым было изобретение процессоров, способных выполнять операции не над содержимым одного или нескольких регистров, а над более крупными фрагментами данных. Их появление связывают с проектом Solomon (1962 год).

История развития многопроцессорных вычислительных систем берёт начало с появления в 70-е годы первого суперкомпьютера семейства CRAY-1.

Аналогичный принцип работы с данными был реализован в самой производительной машине своего времени ILLIAC IV (1972). Если процессор, используя одну команду, может выполнять действия над векторами, то его называют векторным процессором. Впоследствии Сеймур Крей использовал векторные принципы при разработке своих суперкомпьютеров, начиная с Cray-1.

После введения нового класса векторных процессоров обычные процессоры стали «скалярными»; именно поэтому на следующем шаге появились решения с названием «суперскалярные процессоры». Идея суперскалярности заключается в том, что процессор просматривает приближающиеся к исполнению команды и выбирает из них те последовательности, которые могут быть выполнены параллельно. Родоначальником этой идеи был тот же Крей; он реализовал ее в 1965 году в компьютере CDC 6600, затем в процессорах Intel i960 (1988 год) и AMD 29050 (1990) это идея была воспроизведена.

Усовершенствованием фон-неймановской архитектуры является распараллеливание потоков. Данная технология существует в нескольких версиях; среди них — одновременная многопоточность (SMT) и многопоточность на уровне кристалла (CMT). Эти два подхода в основном различаются представлением о том, что есть «поток». Хронологически первым процессором, поддерживающим многопоточность, был процессор DEC Alpha EV4 21064. Драматическая судьба этого процессорного семейства была и остается предметом серьезных обсуждений.

Процессор поддерживает деление на два потока команд, выбираемых из одной задачи в режиме SMT, что обеспечивает суммарное повышение производительности примерно на 30%.

Первыми представителями архитектуры CMP (ядра на кристалле) стали процессоры, предназначенные для использования в серверах, это был простой тандем, в таких приборах на одной подложке размещаются два, по сути, независимых ядра. Помимо экономии места подобное решение дает ощутимую экономию энергии, поскольку часть системотехнических компонентов является общей для обоих ядер.

О параллелизме как о светлом будущем компьютерных систем ученые упорно говорили на протяжении всех последних 30-35 лет. Однако до тех пор, пока преодоление связанных с этим сложностей, например с необходимостью разделения приложений на потоки, компенсировалось непрерывно возрастающей производительностью процессоров, никто из производителей по понятным причинам инвестировать в эту перспективу не хотел. Поэтому до появления многоядерных процессоров параллельные вычисления оставались атрибутом суперкомпьютеров.

Идея перехода на многоядерные процессоры появилась недавно, примерно начиная с 2005 года, но она не нова. Еще в далекие 60-е годы преимущества нескольких процессорных ядер перед одним обосновал Сеймур Крей, затем он реализовал свой замысел в суперкомпьютере CDC 6600. Но в силу консервативности взглядов проектировщиков дальнейшего развития этот подход к проектированию центрального процессорного устройства не получил.

Возрождение многоядерности произошло благодаря инженерам корпорации Digital Equipment. Во второй половине 90-х задумались о многоядерности в микропроцессорах; это произошло при переходе процессора от Alpha 21164 (EV5) к Alpha 21264 (EV6). Тогда исследователям удалось установить две важные закономерности, распространяющиеся на процессоры. Во-первых, оказалось, что для линейного роста производительности монолитных (одноядерных) процессоров требуется обеспечить квадратичный рост числа транзисторов. Во-вторых, как следствие, также нелинейно возрастет сложность проектирования. Элементарное рассуждение подсказывает, что суммирование производительности нескольких ядер даст ту же совокупную производительность, что и одно ядро при меньшем числе транзисторов. Вопрос в том, как объединить мощности отдельных ядер, в этом состоит коренная проблема многоядерности. Реакцией на обнаруженные проблемы стал проект Piranha, предполагавший создание 8-ядерного процессора, где каждое ядро должно было иметь отдельную кэш-память для команд и данных, для совместной работы процессоры объединялись коммутатором. Однако после перехода в ведение Compaq проект Piranha был похоронен, процессор так и не был реализован.

Параллельно с проектом Piranha группа исследователей из Стэндфордского университета во главе с Кунле Олокотуном работала над «многоголовым процессором» Hydra. В качестве ядра использовался процессор MIPFS 4600. Работа дала хорошие результаты. Впоследствии Sun Mirosystems купила эту компанию и, заменив ядро MIPFS 4600 на UltraSPARC II и увеличив число ядер вдвое, выпустила экспериментальный процессор Niagara, получивший в серии наименование UltraSPARC T1.

Успеху Niagara способствовало то, что в Sun велась собственная разработка двухъядерного процессора MAJC 5200, где каждое ядро способно обрабатывать четыре потока команд. Процессор UltraSPARC T1 является конвергенцией Hydra и MAJC 5200.

У IBM разработки многоядерных процессоров заметно связаны с производством игр. Так, процессор Xenon предназначен для игровой консоли. Другой проект, Cell, включает двухпотоковое ядро SMT Power и восемь ядер, которые называют синергетическими процессорными элементами (Synergistic Processing Element, SPE), работающими по принципу SIMD. Основное ядро Power выполняет команды из системы команд PowerPC, поддерживая специализированную систему команд SPE. С очевидным опозданием свои предложения сделали корпорации AMD и Intel: борьба на многоядерном поле стала и для производителей процессоров архитектуры x86 одним из самых актуальных вопросов сегодняшнего дня.

* 1. **Общая структура многопроцессорной системы**

При построении многопроцессорной архитектуры может использоваться одна из нескольких концептуальных моделей соединения вычислительных элементов.

На рис.2 показана общая структура МП-системы: связанная архитектура с общей памятью с распределенной обработкой данных и прерываний ввода-вывода. Она полностью симметрична; т. е. все процессоры функционально идентичны и имеют одинаковый статус, и каждый процессор может обмениваться с каждым другим процессором. Симметричность имеет два важных аспекта: симметричность памяти и ввода-вывода.

Память симметрична, если все процессоры совместно используют общее пространство памяти и имеют в этом пространстве доступ с одними и теми же адресами. Симметричность памяти предполагает, что все процессоры могут исполнять единственную копию ОС.

Требование симметричности ввода-вывода выполняется, если все процессоры имеют возможность доступа к одним и тем же подсистемам ввода-вывода, причем любой процессор может получить прерывание от любого источника.

Рис.2 Архитектура многопроцессорной системы.

Основные компоненты

Системные процессоры. В целях обеспечения совместимости с существующими программными средствами, спецификация основывается на процессорах семейства Intel 486 или Pentium. Хотя все процессоры в МП-системе функционально идентичны, спецификация выделяет два их типа: загрузочный процессор(BSP) и прикладные процессоры(AP). Какой процессор играет роль загрузочного, определяется аппаратными средствами или совместно аппаратурой и BIOS. Это сделано для удобства и имеет значение только во время инициализации и выключения. BSP-процессор отвечает за инициализацию системы и за загрузку ОС. AP-процессор активизируется после загрузки ОС.

Контроллеры APIC. Данные контроллеры обладают распределенной архитектурой, в которой функции управления прерываниями распределены между двумя функциональными блоками: локальным и ввода-вывода. Эти блоки обмениваются информацией через шину, называемую шиной коммуникаций контроллера прерываний. Блоки APIC совместно отвечают за доставку прерывания от источника прерываний до получателей по всей МП-системе.

Системная память. В системах, совместимых с МП-спецификацией, используется архитектура памяти стандарта AT: вся память используется как системная за исключением адресов, зарезервированных под устройства ввода-вывода и BIOS.

МП-системы нуждаются в высокой пропускной способности по сравнению с однопроцессорными. Требования возрастают пропорционально числу процессоров на шине памяти.

Шина расширения ввода-вывода. Спецификация обеспечивает построение МП-систем на основе платформ PC/AT, отвечающих промышленным стандартам. В проектах могут быть использованы стандартные шины ISA, EISA, MCA, VL и PCI. BIOS выполняет функции слоя, изолирующего особенности аппаратных средств от ОС и программных приложений. В многопроцессорных системах BIOS выполняет следующие функции: проверяет системные компоненты; строит таблицы конфигурации, используемые ОС; инициализирует процессор и всю остальную систему; дополнительно: передает информацию о конфигурации в ОС, которая идентифицирует все процессоры и другие компоненты МП-систем; переводит все процессоры и другие компоненты многопроцессорной системы в заданное состояние. Одна из главных целей этой спецификации состоит в том, чтобы обеспечить возможность построения микроядерных ОС для многопроцессорных систем.

* 1. **Спецификация аппаратных средств многопроцессорной системы**

Для того чтобы ОС могла работать на многопроцессорных платформах, аппаратные средства должны обладать определенным набором свойств.

Конфигурация системной памяти. Спецификация МП-памяти основывается на стандартной карте памяти РС/АТ размером до 4 ГБайт.

Кэшируемость и доступность физической памяти для процессоров. Кэшируется вся память за исключением области, отведенной для описания регистров локального блока APIC. Все процессоры имеют доступ к главной памяти и участкам памяти, отведенным под ROM BIOS.

Управление памятью (блокировка). Для защиты целостности некоторых критических операций с памятью Intel-совместимые процессоры используют специальный сигнал. Разработчики системных программных средств должны использовать этот сигнал для управления доступом процессоров к памяти.

Для гарантии AT-совместимости блокировка некорректных операций с памятью в AT-совместимых шинах в согласованной системе должна реализовываться строго в соответствии со спецификациями на шины.

Упорядочение записей в памяти. Применяется при управлении устройствами ввода-вывода, чтобы операции с памятью и вводом-выводом выполнялись строго в запрограммированном порядке. Строгое упорядочивание операций ввода-вывода поддерживается процессорами.

Управление прерываниями. В МП-совместимой системе прерывания управляются контроллерами APIC используя по одному локальному блоку на процессор. Число блоков ввода-вывода должно быть не менее одного.

Чтобы обеспечить расширение функций и внесение изменений в будущем, архитектура APIC определяет только программный интерфейс блоков APIC. Режимы прерывания. В спецификации определены три режима прерывания:

1. Режим PIC эффективно обходит все компоненты APIC и заставляет систему функционировать в Однопроцессорном режиме.

2. Режим виртуальной линии - использует APIC как виртуальную линию, в остальном совпадает с режимом PIC.

3. Режим симметричного ввода-вывода - позволяет работать с многими процессорами.

Отображение памяти APIC. В согласованной МП-системе все контроллеры APIC должны быть реализованы как описанные в памяти устройства ввода-вывода. Базовые адреса APIC находятся в верхней части адресного пространства памяти. Все локальные блоки отображаются в одних и тех же адресах, которые не подлежат коллективному использованию, а используются каждым процессором индивидуально. Контроллеры ввода-вывода отображаются так, чтобы обеспечить их совместное использование всеми процессорами(полную симметричность доступа).

Таймеры интервалов. Локальные блоки содержат 32-битный программируемый таймер с двумя независимыми входами. Таймеры блока ввода-вывода имеют один вход.

Поддержка перезагрузки. Для приведения всех систем компьютера в начальное состояние требуется возможность перезагрузки системы. В системе может выполняться "жесткая" перезагрузка, которая устанавливает все компоненты системы в начальное состояние (при включении питания)

"Мягкая" загрузка только частично инициализирует процессор.

Таблицы конфигурации МП-систем ОС должна иметь доступ к информации о конфигурации МП-системы. Предусмотрено два метода передачи этой информации в ОС: минимальный, позволяющий задать конфигурацию посредством выбора одного из нескольких подразумеваемых наборов значений параметров аппаратуры; и максимальный, обеспечиващий высокую гибкость при проектировании аппаратных средств за счет произвольных установок.

Функции BIOS в МП-системе. В зависимости от многопроцессорных компонентов в МП-системе BIOS может иметь следующие функции:

1. Перевод АР в "спящий" режим.

2. Инициализация контроллеров APIC и других МП-компонент.

3. Создание таблицы конфигурации МП-системы. **2. Операционные системы для работы с многопроцессорными системами**

Операцио́нная систе́ма, ОС (англ. operating system) — базовый комплекс компьютерных программ, обеспечивающий интерфейс с пользователем, управление аппаратными средствами компьютера, работу с файлами, ввод и вывод данных, а также выполнение прикладных программ и утилит. Важное свойство ОС является отсутствие или наличие в ней средств поддержки мно-гопроцессорной обработки - мультипроцессирование. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами. В наши дни становится общепринятым введение в ОС функций поддержки многопроцессор-ной обработки данных. Такие функции имеются в операционных системах Solaris 2.x фирмы Sun, Open Server 3.x компании Santa Crus Operations, OS/2 фирмы IBM, Windows NT и NetWare 4.1, Windows 7. Многопроцессорные ОС могут клас-сифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: асимметричные ОС и симметричные ОС.

Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам.

Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

Это характеристики, связанные с управлением только одним типом ресурсов - процессором. Важное влияние на облик операционной системы в целом, на воз-можности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами - подсистем управления памятью, файлами, устройствами ввода-вывода. Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенапра-вление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов. При реализации сетевых функций возникают задачи, связанные с распределенным характером хранения и обработки данных в сети: ведение справочной информации о всех доступных в сети ресурсах и серве-рах, адресация взаимодействующих процессов, обеспечение прозрачности досту-па, тиражирование данных, согласование копий, поддержка безопасности данных.

 **2.1 Первые операционные системы для работы в многопроцессорной конфигурации**

Первой операционной системой для многопроцессорных систем была

операционная система COS (Cray Operating System 1976 год) предназначеная для режима пакетной обработки и дистанционной пакетной обработки заданий, полученных с удаленных терминалов. Операционная система рассчитывалась на мультипрограммную обработку одновременно до 63 активных задач. Для обеспечения работы в режиме дистанционного доступа в качестве машины-сателлита вычислительной системы Cray-1 используется минимашина "Eclipse". С этой машиной связаны внешние каналы центрального вычислителя с помощью специальных сопрягающих устройств. На минимашину возлагаются функции управления приемом-передачей информации, управления линиями связи, т. е. функции процессора передачи данных.

Позднее была разработа Unicos — название нескольких вариантов операционной системы Unix, созданных компанией Cray для своих суперкомпьютеров. Unicos является следующей операционной системой компании после Cray Operating System (COS). Она обеспечивала работу сетевых кластеров и совместимость на уровне исходного кода с некоторыми другими разновидностями Unix. Unicos впервые была представлена в 1985 году, в качестве операционной системы суперкомпьютера Cray-2, а позднее была портирована и на другие модели Cray

Наиболее развитые ОС, такие как OS/360 (IBM), SCOPE (CDC) и завершённый уже в 1970-х годах MULTICS (МТИ и Bell Labs), предусматривали возможность исполнения на многопроцессорных компьютерах.

Корпорация Be Inc. была создана в 1990 году бывшим исполнительным директором Apple Жаном-Луи Гассе с целью создать компьютерную систему нового поколения (заметим, что Гассе пошёл по пути предыдущего директора Apple Стива Джобса и его платформы NeXT). На подъёме энтузиазма Гассе и его соратники объединились, чтобы сделать простой в использовании, надёжный компьютер, свободный от пережитков прошлого. Планировалось создать именно компьютер, операционная система была только побочной целью. Например, файловая система не должна была содержать каталогов вообще, все файлы лежали на диске без распределения по директориям, а принадлежность к той или иной программе определялась на основе атрибутов — информации о файле, не записанной в самом файле.

Первые версии BeOS работали только на специально созданных компьютерах BeBox, которые тоже производились компанией Be, Inc.. Отличительными особенностями этих компьютеров были многопроцессорность (в одной из версий компьютера стояло 7 процессоров, в другой — 2), в двухпроцессорной версии — индикаторы загрузки процессоров на передней панели системного блока (известные также под названием Blinky Lights), тёмно-синий цвет системного блока и 52-битный порт на задней панели, известный под названием «Geek Port», с простым и удобным управлением для программиста — чтобы поощрять создание внешних устройств под BeOS. BeBox назывались «Silicon Graphics для бедняка» из-за их мощности, позволяющей творить чудеса с графикой, и относительно невысокой стоимости. Первые BeBox прототипы построены на AT&T Hobbit процессорах, но их производство было остановлено. Тогда Ве решила использовать PowerPC процессоры. С 1995 до 1997 всего было продано около двух тысяч компьютеров BeBox, большинство из которых работают и по сей день. Один из них можно увидеть в компьютерном музее в Бостоне. 30 января 1997 Be, Inc. объявила, что уходит из hardware бизнеса и BeBox производство было остановлено.

BSD/OS - коммерческая версия Berkley Software Distribution (BSD Unix) операционная система разработанная компанией Berkley Software Design, inc. (BSDI). Основное отличие от свободно распространяемых UNIX систем более серьёзная поддержка многопроцессорных архитектур и закрытых драйверов.

В январе 2003 года, была выпущена версия 5.0-RELEASE. Эта версия предоставляла расширенную поддержку многопроцессорности и многопоточности.

На сегодняшний день в связи с переходом на ядерную конфигурацию наиболее популярные операционные системы Windows, а именно: Windows Vista и Windows 7.

**2.2 Виды многопроцессорных систем и операционные системы для работы с ними**

Основной характеристикой при классификации многопроцессорных систем является наличие общей (SMP системы) или распределенной (MPP системы) памяти. Это различие является важнейшим фактором, определяющим способы параллельного программирования и, соответственно, структуру программного обеспечения.
Системы с общей памятью: к системам этого типа относятся компьютеры с SMP архитектурой, различные разновидности NUMA систем и мультипроцессорные векторно-конвейерные компьютеры. Характерным словом для этих компьютеров является "единый": единая оперативная память, единая операционная система, единая подсистема ввода-вывода. Только процессоры образуют множество. Единая UNIX-подобная операционная система, управляющая работой всего компьютера, функционирует в виде множества процессов. Каждая пользовательская программа также запускается как отдельный процесс. Операционная система сама каким-то образом распределяет процессы по процессорам. В принципе, для распараллеливания программ можно использовать механизм порождения процессов. Однако этот механизм не очень удобен, поскольку каждый процесс функционирует в своем адресном пространстве, и основное достоинство этих систем - общая память - не может быть использован простым и естественным образом. Для распараллеливания программ используется механизм порождения нитей - легковесных процессов, для которых не создается отдельного адресного пространства, но которые на многопроцессорных системах также распределяются по процессорам.
Системы с распределенной памятью: в системах этого типа на каждом вычислительном узле функционирует собственные копии операционной системы, под управлением которых выполняются независимые программы. Это могут быть как действительно независимые программы, так и параллельные ветви одной программы. В этом случае единственно возможным механизмом взаимодействия между ними является механизм передачи сообщений. Наиболее популярными операционными системами для данной архитектуры являются операционные системы семейства UNIX, а именно чаще всего используется LINUX.
Кластерные технологии стали логическим продолжением развития идей, заложенных в архитектуре MPP систем. Если процессорный модуль в MPP системе представляет собой законченную вычислительную систему, то следующий шаг напрашивается сам собой: почему бы в качестве таких вычислительных узлов не использовать обычные серийно выпускаемые компьютеры. Привлекательной чертой кластерных технологий является то, что они позволяют для достижения необходимой производительности объединять в единые вычислительные системы компьютеры самого разного типа, начиная от персональных компьютеров и заканчивая мощными суперкомпьютерами. Кластер - это связанный набор полноценных компьютеров, используемый в качестве единого вычислительного ресурса. Преимущества кластерной системы перед набором независимых компьютеров очевидны. Во-первых, разработано множество диспетчерских систем пакетной обработки заданий, позволяющих послать задание на обработку кластеру в целом, а не какому-то отдельному компьютеру. Эти диспетчерские системы автоматически распределяют задания по свободным вычислительным узлам или буферизуют их при отсутствии таковых, что позволяет обеспечить более равномерную и эффективную загрузку компьютеров. Во-вторых, появляется возможность совместного использования вычислительных ресурсов нескольких компьютеров для решения одной задачи. Для создания кластеров обычно используются либо простые однопроцессорные персональные компьютеры, либо двух- или четырех- процессорные SMP-серверы. При этом не накладывается никаких ограничений на состав и архитектуру узлов. Каждый из узлов может функционировать под управлением своей собственной операционной системы. Чаще всего используются стандартные ОС: Linux, FreeBSD, Solaris, Tru64 Unix, Windows NT.

**3. Операционная система UNIX в многопроцессорной конфигурации**

В середине 60-х годов сотрудники компании General Electrics, AT&T Bell Laboratories и Масачусетского института написали операционную систему Multics, в которой было реализовано то, о чем даже и не задумывались другие разработчики программных продуктов. Проект, из-за слабой развитости и отсутствия поддержки других разработчиков, был закрыт в конце 60-х годов.

После Multics был разработан UNIX, который, в отличии от предшественницы, поддерживал не только многозадачность, но и работал на разных платформах, включая предков современных ПК, которых тогда называли как мини-компьютеры. В основе создания операционной системы лежит игра Space Travel, которая была разработана программистом Кеном Томпсоном. Так уж получилось, что игра не смогла заработать на компьютере того времени Honeywell 635, и поэтому Кен Томпсон ожидал выхода более мощного компьютера. В конце концов для того, чтобы игра заработала, пришлось в буквальном смысле этого слова создать для нее операционную систему. Пришлось усовершенствовать файловую систему для ускорения работоспособности системы, добавить функции управления системными устройствами, процессором, памятью. Управление всеми функциональными возможностями занимался мощный командный процессор Bourne Again Shell, в народе просто Bash. UNIX стал одной из первых POSIX-совместимых операционных систем. (POSIX - Portable Opereating System Interface for computer environments, интерфейс переносимой операционной системы.)

За весьма короткий срок операционная система превратилась в мощную многозадачную и многопользовательскую операционную систему, которую некто Браян Керниган предложил назвать Unics, название походило от системы Multics, а уже позже Unics назвали короче - UNIX. Первой официальной версией системы является выпущенная в ноябре 1971 года система UNIX 1.0. Операционная система UNIX была написана на ассемблере. Кена Томпсона Деннис Ритчи предложил внести в исходный код операционной системы некоторые изменения, вплоть до того, что исходный код системы был позже написан с использованием языка программирования C.

Неофициально UNIX стал продаваемым программным продуктом уже с 1972 года. Лицензию на владение системой покупали многие компании, включая IBM, Nowell и даже сама Microsoft. В 1975 году Калифорнийский университет в Беркли купил права на Unix и выпустил свою систему - Berkeley Software Distribution, BSD. позже последовали различные выпуски Unix BSD, BSD 4 и так далее, из которых до сегодняшнего дня сохранилась только FreeBSD - сетевая многозадачная и многопользовательская операционная система, ныне основной конкурент Linux.

В 1980 году компания Microsoft, по велению моды, купила Unix и себе, создав на ее основе операционную систему Xenix. На основе данной операционной системы был выпущен MS DOS. Система была однозадачной и однопользовательской и не поддерживала многозадачность. Многие удивляются, почему команды DOS похожи на команды интерпретатора Unix? Просто потому что DOS - это сильно модифицированный UNIX.

В конце 80-х годов UNIX был полноправным владельцем компьютерного рынка. К этому времени в системе уже был реализован четкий графический интерфейс пользователя (GUI, Graphic User Interface), их в составе системы обычно было два, которые частично сохранились и до наших времен: Motiff и Open Look. Позже эти два стандарта были объеденены в один: CDE - Common Desktop Environment, еще позже - KDE.

За время своего существования проект UNIX стал коммерческим и широкопродаваемым. На его основе была создана операционная система Minix (своего рода мини Unix). Распрастранялась система бесплатно, на основе общедоступной лицензии GPL (General Public Licence). Уже на основе Minix был создан гигант Linux, создатель которого Линус Торвальдс.

**3.1 Общий обзор особенностей системы UNIX**

За время, прошедшее с момента ее появления в 1969 году, система UNIX стала довольно популярной и получила распространение на машинах с различной мощностью обработки, от микропроцессоров до больших ЭВМ, обеспечивая на них общие условия выполнения программ. Система делится на две части. Одну часть составляют программы и сервисные функции, то, что делает операционную среду UNIX такой популярной; эта часть легко доступна пользователям, она включает такие программы, как командный процессор, обмен сообщениями, пакеты обработки текстов и системы обработки исходных текстов программ. Другая часть включает в себя собственно операционную систему, поддерживающую эти программы и функции.

Особенности UNIX, отличающие данное семейство от других ОС:

■ Файловая система древовидная, чувствительная к регистру символов в именах, очень слабые ограничения на длину имён.

■ Нет поддержки структурированных файлов ядром ОС, на уровне системных вызовов файл есть поток байт.

■ Командная строка находится в адресном пространстве запускаемого процесса, а не извлекается системным вызовом из процесса интерпретатора.

■ Ввод/вывод только через дескрипторы файлов.

■ Традиционно крайне слабая поддержка асинхронного ввода/вывода, по сравнению с VMS и Windows NT.

■ Интерпретатор команд есть обыкновенное приложение, общающееся с ядром обыкновенными системными вызовами (в RSX-11 и VMS интерпретатор команд выполнялся как специальное приложение, специальным образом размещенное в памяти, пользующееся специальными системными вызовами, поддерживались также системные вызовы, дающие возможность приложению обращаться к своему родительскому интерпретатору команд).

■ Команда командной строки есть не более чем имя файла программы, не требуется специальная регистрация и специальная разработка программ как команд (что являлось обычной практикой в RSX-11, RT-11).

■ Не принят подход с программой, задающей пользователю вопросы о режимах своей работы, вместо этого используются параметры командной строки (в VMS, RSX-11, RT-11 программы работали также с командной строкой, но при её отсутствии выдавали запрос на ввод команд).

■ Пространство имён устройств на диске в каталоге /dev, поддающееся управлению администратором, в отличие от подхода Windows, где это пространство имен размещается в памяти ядра, и администрирование этого пространства (например, задание прав доступа) крайне затруднено из-за отсутствия его постоянного хранения на дисках (строится каждый раз при загрузке).

■ Широкое использование текстовых файлов для хранения настроек, в отличие от двоичной базы данных настроек, как, например, в Windows.

■ Широкое использование утилит обработки текста для выполнения повседневных задач под управлением скриптов.

■ «Раскрутка» ОС после загрузки ядра путём исполнения скриптов стандартным интерпретатором команд.

■ Адресное пространство делится на глобальное для всех процессов ядро и на локальную для процесса части, нет «групповой» части адресного пространства, как в VMS и Windows NT, как и возможности загрузки туда кода и его исполнения там.

■ Использование двух уровней привилегий процессора вместо четырёх в VMS.

■ Отказ от использования оверлеев в пользу деления программы на несколько программ поменьше, общающихся через конвейеры или временные файлы.

■ Отсутствие APC и аналогов, то есть произвольных (а не жестко перечисленных в стандартном множестве) сигналов, не доставляемых до явного пожелания процесса их получить (Windows, VMS).

Концепция сигнала уникальна для UNIX, и крайне сложна в переносе на другие ОС, такие, как Windows.

**3.1.1 Преимущества операционная система UNIX**

UNIX завоевала высокую степень популярности. Одной из главных причин успеха являлся способ распространения системы. Корпорация AT&T, ограниченная законами в своих возможностях, продавала лицензии и исходные коды системы по достаточно низкой цене, поэтому UNIX стала популярной среди многих пользователей по всему миру. Так как в комплект поставки входили и исходные коды, пользователи имели возможность экспериментировать с ними, улучшать их, а также обмениваться друг с другом созданными изменениями. Корпорация AT&T встраивала многие из новшеств в следующие версии системы.

Развитие системы UNIX оказалось очень открытым процессом. Добавления поступали из учебных заведений, коммерческих организаций и от хакеров-энтузиастов из разных континентов и стран мира. Даже после коммерциализации UNIX многие производители ОС поддержали концепцию открытых систем и сделали свои разработки доступными для других, создавая спецификации открытых систем, такие как NFS.

Оригинальная версия UNIX имела хороший дизайн, который являлся базисом последующего успеха более поздних реализаций и вариантов системы. Одну из сильнейших сторон системы можно охарактеризовать выражением «красота в краткости». Ядро небольшого размера имело минимальный набор основных служб. Утилиты небольшого размера производили малый объем манипуляций с данными. Механизм конвейера совместно с программируемой оболочкой позволял пользователям комбинировать эти утилиты различными способами, создавая мощные, производительные инструменты.

Файловая система UNIX является примером вышеописанного подхода. В отличие от других операционных систем, обладающих сложными методами доступа к файлам система UNIX интерпретирует файлы как простую последовательность байтов. Приложения могут записывать в содержимое файлов любую структуру и использовать различные собственные методы доступа, не ставя об этом в известность операционную систему.

Многие системные приложения используют для представления своих данных символьные строки. Возможно, что структурированное хранение информации в двоичном формате представляется более эффективным, однако представление в виде текста позволило пользователям легко просматривать и производить манипуляции с этими файлами без применения специальных инструментов. Текст является общей, универсальной и обладающей высокой степенью переносимости формой данных, которые можно легко обрабатывать множеством различных утилит.

Еще одной особенностью системы UNIX стал простой унифицированный интерфейс с устройствами ввода-вывода. Представляя все устройства как файлы, система позволяет пользователям применять один и тот же набор команд и системных вызовов для доступа и работы с различными устройствами, равно как и для работы с файлами. Разработчики могут создавать программы, производящие ввод-вывод, не заботясь, с чем именно производится обмен, с файлом, терминалом пользователя, принтером или другим устройством. Таким образом, используемый совместно с перенаправлением данных интерфейс ввода-вывода системы UNIX — простой и одновременно мощный.

Причиной успеха и распространения UNIX стала ее высокая степень переносимости. Большая часть ядра системы написана на языке С. Это позволило относительно легко адаптировать UNIX к новым аппаратным платформам. Первая реализация системы появилась на популярной тогда машине PDP-11 и затем была перенесена на VAX-11, имевшую не меньшую популярность. Многие производители «железа» после создания новых компьютеров имеют возможность просто перенести на них уже имеющуюся систему UNIX вместо того, чтобы создавать для своих разработок операционную систему заново.

**3.1.2 Недостатки операционная система UNIX**

Известно, что любая медаль имеет две стороны. Один из наиболее объективных обзоров UNIX был создан не кем иным, как Денисом Ритчи. В январе 1987 года на конференции USENIX в разделе «Ретроспективы UNIX» Ритчи сделал доклад, где провел анализ многих недостатков системы. Хотя UNIX изначально была весьма простой системой, это вскоре закончилось. Например, AT&T добавила в стандартную библиотеку ввода-вывода буферизацию данных, что повысило ее эффективность и сделало программы переносимыми на не-UNIX-системы.

UNIX сама по себе являлась замечательной операционной системой, однако большинству пользователей нужна была не сама система, а, в первую очередь, возможность выполнять определенное приложение. Пользователи хотели работать с определенными программами (например, текстовыми редакторами, финансовыми пакетами или программами для создания изображений), потратив на это минимум расходов и усилий. Недостатки простого унифицированного (обычно графического) пользовательского интерфейса в первых системах UNIX были основной причиной его неприятия массами.

Получилось так, что элегантность и эстетичность, свойственная UNIX, требует от пользователей, желающих эффективно работать в системе, творческого мышления и определенной изобретательности.

В какой-то степени система UNIX явилась жертвой своего собственного успеха. Простота лицензионных условий и переноса на различные аппаратные платформы стала причиной неконтролируемого роста и лавинообразного распространения различных реализаций ОС. Каждый пользователь имел право вносить свои собственные изменения в систему, в результате группы разработчиков часто создавали несовместимые между собой варианты. Изначально существовали две основные ветви развития UNIX, разрабатываемые компаниями AT&T и BSD. Каждая реализация имела оригинальную файловую систему, архитектуру памяти, сигналы и принципы работы с терминалами. Позже другие поставщики предложили новые варианты UNIX, стараясь привести их к некоторой степени совместимости с реализациями AT&T и BSD. Однако чем дальше, тем все менее предсказуемой становилась ситуация, а разработчикам приложений требовалось все больше усилий, чтобы приспособить свои программы ко всем различным вариантам UNIX.

Стандартизация систем стала лишь частичным решением проблемы, так как встретила определенное сопротивление. Поставщики стремились добавить в свои разработки какие-либо уникальные функции, стараясь создать продукт, имеющий отличия от остальных, и тем самых показать его преимущества среди конкурирующих вариантов.

Ричард Рашид, один из разработчиков системы Mach, предложил свою версию причин неудач UNIX. Причиной создания ОС Mach стало наблюдение за эволюционированием системы UNIX, которая имела минимальные возможности для построения инструментов пользователя. Большие сложные инструменты создавались путем комбинирования множества простых функций. Однако такой подход не был перенесен на ядро системы.

Традиционное ядро UNIX было недостаточно гибким и расширяемым, оно имело минимальные возможности для дополнительного использования кода. Позже разработчики стали просто добавлять новые коды в ядро системы, делая его основой для новых функциональных средств. Ядро очень быстро стало раздутым, сложным и абсолютно немодульным. Разработчики Mach попытались решить эти проблемы, переписав систему заново с нуля, взяв за основу небольшое количество основных функций. В современных реализациях UNIX вышеописанная система решается различными способами, например добавляются гибкие структуры к подсистемам.

**3.2 Cистемы UNIX в многопроцессорной конфигурации**

Центральный процессор представляет собой ресурс, который используется всеми процессами системы. Часть ядра, распределяющая процессорное время между процессами, называется планировщиком (scheduler). В традиционных системах UNIX планировщик использует принцип вытесняющего циклического планирования. Процессы, имеющие одинаковые приоритеты, будут выполняться циклически друг за другом, и каждому из них будет отведен для этого определенный период (квант) времени, обычно равный 100 миллисекундам. Если какой-либо процесс, имеющий более высокий приоритет, становится выполняемым, то он вытеснит текущий процесс (конечно, если тот не выполняется в режиме ядра) даже в том случае, если текущий процесс не исчерпал отведенного ему кванта времени.

В традиционных системах UNIX приоритет процесса определяется двумя факторами: фактором «любезности» и фактором утилизации. Пользователи

могут повлиять на приоритет процесса при помощи изменения значения его «любезности», используя системный вызов nice (но только суперпользователь имеет полномочия увеличивать приоритет процесса). Фактор утилизации определяется степенью последней (то есть во время последнего обслуживания процесса процессором) загруженности CPU процессом. Этот фактор позволяет системе динамически изменять приоритет процесса. Ядро системы периодически повышает приоритет процесса, пока тот не выполняется, а после того, как процесс все-таки получит какое-то количество процессорного времени, его приоритет будет понижен. Такая схема защищает процессы от «зависания» так как периодически наступает такой момент, когда ожидающий процесс получает достаточный уровень приоритета для выполнения.

Процесс, выполняющийся в режиме ядра, может освободить процессор в том случае, если произойдет его блокирование по событию или ресурсу. Когда процесс снова станет работоспособным, ему будет назначен приоритет ядра. Приоритеты ядра обычно выше приоритетов любых прикладных задач. В традиционных системах UNIX приоритеты представляют собой целые числа в диапазоне от 0 до 127, причем чем меньше их значение, тем выше приоритет процесса (так как система UNIX почти полностью написана на языке С, в ней используется стандартный подход к началу отсчета от нуля).

Синхронизация

 Ядро системы UNIX является реентерабельным. В любой момент времени в ядре могут быть активны сразу несколько процессов. Конечно, на одно процессорных системах только один из них окажется текущим, в то время как остальные будут блокированы, находясь в режиме ожидания освобождения процессора или иного системного ресурса. Так как все эти процессы используют одну и ту же копию структур данных ядра, необходимо обеспечивать некоторую форму синхронизации для предотвращения порчи ядра.

В системе UNIX применяется несколько различных технологий синхронизации. Система изначально создана невытесняющей. Это означает, что процесс, выполняющийся в режиме ядра, не может быть вытеснен другим процессом, даже если отведенный ему квант времени уже исчерпан. Процесс должен самостоятельно освободить процессор. Это обычно происходит в тот момент, когда процесс приостанавливает свою работу в ожидании необходимого ресурса или какого-то события; когда процесс завершил функционирование в режиме ядра и когда собирается возвращаться в режим задачи. В любом случае, так как процесс освобождает процессор добровольно, он может быть уверен, что ядро системы находится в корректном состоянии. Создание ядра системы не вытесняющим является гибким решением большинства проблем, связанных с синхронизацией. Вернемся к примеру, показанному на рис. 2.6. В данном случае ядро системы может обрабатывать связанный список без его блокировки, не беспокоясь о возможном вытеснении. Существуют три ситуации, при возникновении которых необходима синхронизация:

♦ операции блокировки;

♦ прерывания;

♦ работа многопроцессорных систем.

Операции блокировки

 Операция блокировки — это операция, которая блокирует процесс (то есть переводит процесс в спящий режим до тех пор, пока блокировка не будет снята). Так как ядро системы не является вытесняющим, оно может манипулировать большинством объектов (структурами данных и ресурсами) без возможности причинения им какого-либо вреда, так как заранее известно, что никакой другой процесс не может получить к ним в это время доступ.

Синхронизация

 Ядро системы UNIX является реентерабельным. В любой момент времени в ядре могут быть активны сразу несколько процессов. Конечно, на одно процессорных системах только один из них окажется текущим, в то время как остальные будут блокированы, находясь в режиме ожидания освобождения процессора или иного системного ресурса. Так как все эти процессы используют одну и ту же копию структур данных ядра, необходимо обеспечивать некоторую форму синхронизации для предотвращения порчи ядра.

В системе UNIX применяется несколько различных технологий синхронизации. Система изначально создана невытесняющей. Это означает, что процесс, выполняющийся в режиме ядра, не может быть вытеснен другим процессом, даже если отведенный ему квант времени уже исчерпан. Процесс должен самостоятельно освободить процессор. Это обычно происходит в тот момент, когда процесс приостанавливает свою работу в ожидании необходимого ресурса или какого-то события; когда процесс завершил функционирование в режиме ядра и когда собирается возвращаться в режим задачи. В любом случае, так как процесс освобождает процессор добровольно, он может быть уверен, что ядро системы находится в корректном состоянии. Создание ядра системы не вытесняющим является гибким решением большинства проблем, связанных с синхронизацией. Вернемся к примеру, показанному на рис. 2.6. В данном случае ядро системы может обрабатывать связанный список без его блокировки, не беспокоясь о возможном вытеснении. Существуют три ситуации, при возникновении которых необходима синхронизация:

♦ операции блокировки;

♦ прерывания;

♦ работа многопроцессорных систем.

**3.3 Проблемы, связанные с многопроцессорными системами и решение их в операционной системе UNIX**

 В классической постановке для системы UNIX предполагается использование

однопроцессорной архитектуры, состоящей из одного ЦП, памяти и периферийных устройств. Многопроцессорная архитектура, напротив, включает в себя два и более ЦП, совместно использующих общую память и периферийные, располагая большими возможностями в увеличении производительности системы, связанными с одновременным исполнением процессов на разных ЦП. Каждый ЦП функционирует независимо от других, но все они работают с одним и тем же ядром операционной системы. Поведение процессов в такой системе ничем не отличается от поведения в однопроцессорной системе - с сохранением

семантики обращения к каждой системной функции - но при этом они могут открыто перемещаться с одного процессора на другой. Хотя, к сожалению, это не приводит к снижению затрат процессорного времени, связанного с выполнением процесса. Отдельные многопроцессорные системы называются системами с присоединенными процессорами, поскольку в них периферийные устройства доступны не для всех процессоров. Параллельная работа нескольких процессоров в режиме ядра по выполнению различных процессов создает ряд проблем, связанных с сохранением целостности данных и решаемых благодаря использованию соответствующих механизмов защиты.

Защита целостности структур данных ядра системы UNIX обеспечивается двумя способами: ядро не может выгрузить один процесс и переключиться на контекст другого, если работа производится в режиме ядра, кроме того, если при выполнении критического участка программы обработчик возникающих прерываний может повредить структуры данных ядра, все возникающие прерывания тщательно маскируются. В многопроцессорной системе, однако, если два и более процессов выполняются одновременно в режиме ядра на разных процессорах, нарушение целостности ядра может произойти даже несмотря на принятие защитных мер, с другой стороны, в однопроцессорной системе вполне достаточных. Если вопрос об опасности возникновения нарушения целостности оставить открытым, как бы редко подобные нарушения ни случались, ядро утратит свою неуязвимость и его поведение станет непредсказуемым. Избежать этого можно тремя способами:

 1.Исполнять все критические операции на одном процессоре, опираясь на стандартные методы сохранения целостности данных в однопроцессорной системе;

 2.Регламентировать доступ к критическим участкам программы, используя элементы блокирования ресурсов;

 3.Устранить конкуренцию за использование структур данных путем соответствующей переделки алгоритмов.

 Главный и подчиненный процессы. Систему с двумя процессорами, один из которых - главный (master) - может работать в режиме ядра, а другой - подчиненный (slave) - только в режиме задачи, впервые реализовал на машинах типа VAX 11/780 Гобл. Главный процессор несет ответственность за обработку всех обращений к операционной системе и всех прерываний. Подчиненные процессоры ведают выполнением процессов в режиме задачи и информируют главный процессор о всех производимых обращениях к системным функциям. Выбор процессора, на котором будет выполняться данный процесс, производится в соответствии с алгоритмом диспетчеризации. В соответствующей записи таблицы процессов появляется новое поле, в которое записывается идентификатор выбранного процессора; предположим для простоты, что он показывает, является ли процессор главным или подчиненным. Когда процесс производит обращение к системной функции, выполняясь на подчиненном процессоре, подчиненное ядро переустанавливает значение поля идентификации процессора таким образом, чтобы оно указывало на главный процессор, и переключает контекст на другие процессы. Главное ядро запускает на выполнение процесс с наивысшим приоритетом среди тех процессов, которые должны выполняться на главном процессоре. Когда выполнение системной функции завершается, поле идентификации процессора перенастраивается обратно, и процесс вновь возвращается на подчиненный процессор. Если процессы должны выполняться на главном процессоре, желательно, чтобы главный процессор обрабатывал их как можно скорее и не заставлял их ждать своей очереди чересчур долгоЕсли в тот момент, когда подчиненный процессор делает запрос на исполнение системной функции, главный процесс выполняется в режиме задачи, его выполнение будет продолжаться до следующего переключения контекста. Главный процессор реагировал бы гораздо быстрее, если бы подчиненный процессор устанавливал при этом глобальный флаг; проверяя установку

флага во время обработки очередного прерывания по таймеру, главный процессор произвел бы в итоге переключение контекста максимум через один таймерный тик. С другой стороны, подчиненный процессор мог бы прервать работу главного и заставить его переключить контекст немедленно, но данная возможность требует специальной аппаратной реализации. Программа обработки прерываний по таймеру на подчиненном процессоре следит за периодичностью перезапуска процессов, не допуская монопольного использования процессора одной задачей. Кроме того, каждую секунду эта программа выводит подчиненный процессор из состояния бездействия (простоя). Подчиненный процессор выбирает для выполнения процесс с наивысшим приоритетом среди тех процессов, которые не нуждаются в главном процессоре. Единственным местом, где целостность структур данных ядра еще подвергается опасности, является алгоритм диспетчеризации, поскольку он не предохраняет от выбора процесса на выполнение сразу на двух процессорах. Например, если в конфигурации имеется один главный процессор и два подчиненных, не исключена возможность того, что оба подчиненных процессора выберут для выполнения в режиме задачи один и тот же процесс. Если оба процессора начнут выполнять его параллельно, осуществляя чтение и запись, это неизбежно приведет к искажению содержимого адресного пространства процесса. Избежать возникновения этой проблемы можно двумя способами. Во-первых, главный процессор может явно указать, на каком из подчиненных процессоров следует выполнять данный процесс. Если на каждый процессор направлять несколько процессов, возникает необходимость в сбалансировании нагрузки (на один из процессоров назначается большое количество процессов, в то время как другие процессоры простаивают). Задача распределения нагрузки между процессорами ложится на главное ядро. Во-вторых, ядро может проследить за тем, чтобы в каждый момент времени в алгоритме диспетчеризации принимал участие только один процессор, для этого используются механизмы, подобные семафорам.

 Семафоры. Поддержка системы UNIX в многопроцессорной конфигурации может включать в себя разбиение ядра системы на критические участки, параллельное выполнение которых на нескольких процессорах не допускается. При ближайшем рассмотрении сразу же возникают два вопроса: как использовать семафоры и где определить критические участки. Если при выполнении критического участка программы процесс приостанавливается, для защиты участка от посягательств со стороны других процессов алгоритмы работы ядра однопроцессорной системы UNIX используют блокировку. Механизм установления блокировки:

 выполнять пока (блокировка установлена) /\* операция проверки \*/

приостановиться (до снятия блокировки);

установить блокировку;

механизм снятия блокировки:

снять блокировку;

вывести из состояния приостановния все процессы, приостановленные в результате блокировки;

Блокировки такого рода охватывают некоторые критические участки, но не работают в многопроцессорных система. Предположим, что блокировка снята и что два процесса на разных процессорах одновременно пытаются проверить ее наличие и установить ее. В момент t они обнаруживают снятие блокировки, устанавливают ее вновь, вступают в критический участок и создают опасность нарушения целостности структур данных ядра. В условии одновременности имеется отклонение: механизм не сработает, если перед тем, как процесс выполняет операцию проверки, ни один другой процесс не выполнил операцию установления блокировки. Если, например, после обнаружения снятия блокировки процессор A обрабатывает прерывание и в этот момент процессор B выполняет проверку и устанавливает блокировку, по выходе из прерывания процессор A так же установит блокировку. Чтобы предотвратить возникновение подобной ситуации, нужно сделать так, чтобы процедура блокирования была неделимой: проверку наличия блокировки и ее установку следует объединить в одну операцию, чтобы в каждый момент времени с блокировкой имел дело только один процесс.

 Алгоритм выделения буфера иллюстрирует сложную схему блокирования, на примере алгоритма wait показана синхронизация выполнения процессов, схема блокирования драйверов реализует изящный подход к решению данной проблемы, и наконец, метод решения проблемы холостой работы процессора показывает, что нужно сделать, чтобы избежать конкуренции между процессами.

 Выделение буфера: алгоритм работает с тремя структурами данных: заголовком буфера, хеш-очередью буферов и списком свободных буферов. Ядро связывает семафор со всеми экземплярами каждой структуры. Другими словами, если у ядра имеются в распоряжении 200 буферов, заголовок каждого из них включает в себя семафор, используемый для захвата буфера; когда процесс выполняет над семафором операцию P, другие процессы, тоже пожелавшие захватить буфер, приостанавливаются до тех пор, пока первый процесс не исполнит операцию V. У каждой хеш-очереди буферов также имеется семафор, блокирующий доступ к очереди. В однопроцессорной системе блокировка хеш-очереди не нужна, ибо процесс никогда не переходит в состояние приостанова, оставляя очередь в несогласованном (неупорядоченном) виде. В многопроцессорной системе, тем не менее, возможны ситуации, когда с одной и той же хеш-очередью работают два процесса; в каждый момент времени семафор открывает доступ свободных буферов нуждается в семафоре для защиты содержащейся в нем информации

от искажения. Первая часть алгоритма , реализованная в многопроцессорной системе с использованием семафоров. Просматривая буферный кеш в поисках указанного блока, ядро с помощью операции P захватывает семафор, принадлежащий хеш-очереди. Если над семафором уже кем-то произведена операция данного типа, текущий процесс приостанавливается до тех пор, пока процесс, захвативший семафор, не освободит его, выполнив операцию V. Когда текущий процесс получает право исключительного контроля над хеш-очередью, он приступает к поиску подходящего буфера. Предположим, что буфер находится в хеш-очереди. Ядро (процесс A) пытается захватить буфер, но если оно использует операцию P и если буфер уже захвачен, ядру придется приостановить свою работу, оставив хеш-очередь заблокированной и не допуская таким образом обращений к ней со стороны других процессов, даже если последние ведут поиск незахваченных буферов. Пусть вместо этого процесс A захватывает буфер, используя операцию CP; если операция завершается успешно, буфер становится открытым для процесса. Процесс A захватывает семафор, принадлежащий списку свободных буферов, выполняя операцию CP, поскольку семафор захватывается на непродолжительное время и, следовательно, приостанавливать свою работу, выполняя операцию P, процесс просто не имеет возможности. Ядро убирает буфер из списка свободных буферов, снимает блокировку со списка и с хеш-очереди и возвращает захваченный буфер.

Предположим, что операция CP над буфером завершилась неудачно из-за того, что семафор, принадлежащий буферу, оказался захваченным. Процесс A освобождает семафор, связанный с хеш-очередью, и приостанавливается, пытаясь выполнить операцию P над семафором буфера. Операция P над семафором будет выполняться, несмотря на то, что операция CP уже потерпела неудачу. По завершении выполнения операции процесс A получает власть над буфером. Так как в оставшейся части алгоритма предполагается, что буфер и хеш-очередь захвачены, процесс A теперь пытается захватить хеш-очередь (\*). Поскольку очередность захвата здесь (сначала семафор буфера, потом семафор очереди) обратна вышеуказанной очередности, над семафором выполняется операция CP. Если попытка захвата заканчивается неудачей, имеет место обычная обработка, требующаяся по ходу задачи. Но если захват удается, ядро не может быть уверено в том, что захвачен корректный буфер, поскольку содержимое буфера могло быть ранее изменено другим процессом, обнаружившим буфер в списке свободных буферов и захватившим на время его семафор. Процесс A, ожидая освобождения семафора, не имеет ни малейшего представления о том, является ли интересующий его буфер тем буфером, который ему нужен, и поэтому прежде всего он должен убедиться в правильности содержимого буфера; если проверка дает отрицательный результат, алгоритм запускается сначала. Если содержимое буфера корректно, процесс A завершает выполнение алгоритма.

Фиктивные процессы: когда ядро выполняет переключение контекста в однопроцессорной системе, оно функционирует в контексте процесса, уступающего управление. Если в системе нет процессов, готовых к запуску, ядро переходит в состояние простоя в контексте процесса, выполнявшегося последним. Получив прерывание от таймера или других периферийных устройств, оно обрабатывает его в контексте того же процесса.

 В многопроцессорной системе ядро не может простаивать в контексте процесса, выполнявшегося последним. Посмотрим, что произойдет после того, как процесс, приостановивший свою работу на процессоре A, выйдет из состояния приостанова. Процесс в целом готов к запуску, но он запускается не сразу же по выходе из состояния приостанова, даже несмотря на то, что его контекст уже находится в распоряжении процессора A. Если этот процесс выбирается для запуска процессором B, последний переключается на его контекст и возобновляет его выполнение. Когда в результате прерывания процессор A выйдет из простоя, он будет продолжать свою работу в контексте процесса A до тех пор, пока не произведет переключение контекста. Таким образом, в течение короткого промежутка времени с одним и тем же адресным пространством (в частности, со стеком ядра) будут вести работу (и, что весьма вероятно, производить запись) сразу два процессора.

 Решение этой проблемы состоит в создании некоторого фиктивного процесса;

когда процессор находится в состоянии простоя, ядро переключается на кон-

текст фиктивного процесса, делая этот контекст текущим для бездействующего

процессора. Контекст фиктивного процесса состоит только из стека ядра; этот

процесс не является выполнимым и не выбирается для запуска. Поскольку каждый процессор простаивает в контексте своего собственного фиктивного процесса, навредить друг другу процессоры уже не могут.

**ЗАКЛЮЧЕНИЕ**

В современном мире многопроцессорные системы развиваются стремительно, но изменение аппаратной архитектуры влечет за собой изменение программного обеспечения, а также грозит появлением новых проблем по совмещению и распараллеливанию процессов. В данной курсовой работе я постаралась раскрыть понятие "многопроцессорные системы", а также рассмотрела решение проблем, связанных с их конфигурации в такой операционной системе, как UNIX.

Несмотря на то, что большинство возникающих нюансов разрешимо**,** все же остаются огромные пробелы в данной области. Поэтому я считаю данный вопрос очень важным для рассмотрения, так как в наше время очень популярен переход на многопроцессорные системы (основанные на ядрах) и возникает много проблем при использовании старого программного обеспечения в том числе и операционных систем, которые необходимо разрешать.

**Список используемой литературы**

1. Многопроцессорные ЭВМ и методы их проектирования / Б.А.Бабаян, А.В.Бочаров, В.С.Волин и др. - М.: Высшая школа, 1990.

2. Архитектура UNIX/ Морис Бах Изд.: Prentice-Hall, 1986 г.,

3. http://ru.wikipedia.org/

4.http://rsusu1.rnd.runnet.ru

5. http://window.edu.ru/

6. http://www.parallel.ru/history/

7. http://evm-story.narod.ru/