**Механизмы межпроцессных взаимодействий в операционной системе Unix**

Сергей Кузнецов

**Возникшие проблемы**

Избыточный набор системных средств, предназначенных для обеспечения возможности взаимодействия и синхронизации процессов, которые не обязательно связаны отношением родства

IPC - Inter-Process Communication Facilities

с появлением UNIX System V Release 4.0 все эти средства были узаконены и вошли в фактический стандарт ОС UNIX современного образца

в разных вариантах системы средства IPC реализуются по-разному

эффективность реализации различается

усложняется разработка мобильных асинхронных программных комплексов

Пакет средств IPC

UNIX System V Release 3.0

средства, обеспечивающие возможность наличия общей памяти между процессами (сегменты разделяемой памяти - shared memory segments)

средства, обеспечивающие возможность синхронизации процессов при доступе с совместно используемым ресурсам, например, к разделяемой памяти (семафоры - semaphores)

средства, обеспечивающие возможность посылки процессом сообщений другому произвольному процессу (очереди сообщений - message queues)

Общие свойства всех трех механизмов:

для каждого механизма поддерживается общесистемная таблица, элементы которой описывают всех существующих в данный момент представителей механизма (конкретные сегменты, семафоры или очереди сообщений)

элемент таблицы содержит некоторый числовой ключ, который является выбранным пользователем именем представителя соответствующего механизма

процесс, желающий начать пользоваться одним из механизмов, обращается к системе с системным вызовом из семейства "get", ответным параметром является числовой дескриптор

ключ IPC\_PRIVATE

ключ IPC\_CREAT

защита доступа основывается на тех же принципах, что и защита доступа к файлам

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);

size определяет желаемый размер сегмента в байтах

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

реальный размер сегмента можно узнать с помощью системного вызова shmctl

иначе создается новый сегмент с размером не меньше установленного в системе минимального размера сегмента разделяемой памяти и не больше установленного максимального размера

создание сегмента не означает немедленного выделения под него основной памяти

откладывается до выполнения первого системного вызова подключения сегмента к виртуальной памяти некоторого процесса

при выполнении последнего системного вызова отключения сегмента от виртуальной памяти соответствующая основная память освобождается

virtaddr = shmat(id, addr, flags);

id - это ранее полученный дескриптор сегмента

addr - желаемый процессом виртуальный адрес, который должен соответствовать началу сегмента в виртуальной памяти

virtaddr - реальный виртуальный адрес начала сегмента

не обязательно совпадает со значением прямого параметра addr

если addr == 0, ядро выбирает наиболее удобный виртуальный адрес начала сегмента

shmdt(addr);

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

shmctl(id, cmd, shsstatbuf);

cmd идентифицирует требуемое конкретное действие

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:

значение семафора

идентификатор процесса, который хронологически последним работал с семафором

число процессов, ожидающих увеличения значения семафора

число процессов, ожидающих нулевого значения семафора

Три системных вызова:

semget для создания и получения доступа к набору семафоров

semop для манипулирования значениями семафоров

semctl для выполнения управляющих операций над набором семафоров

id = semget(key, count, flag);

key, flag и id - обычный смысл

count - число семафоров в наборе семафоров, обладающих одним и тем же ключом

индивидуальный семафор идентифицируется дескриптором набора семафоров и номером семафора в наборе

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);

id - дескриптор группы семафоров

oplist - массив описателей операций над семафорами группы

count - размер этого массива

возвращается значение последнего обработанного семафора

Элемент массива oplist:

номер семафора в указанном наборе семафоров

операция

флаги

Если проверка прав доступа проходит нормально

указанные в массиве oplist номера семафоров не выходят за пределы общего размера набора семафоров

для каждого элемента массива oplist значение семафора изменяется в соответствии со значением поля "операция"

Значение поля операции положительно

значение семафора увеличивается на единицу

все процессы, ожидающие увеличения значения семафора, активизируются (пробуждаются)

Значение поля операции равно нулю

если значение семафора равно нулю, выбирается следующий элемент массива oplist

иначе число процессов, ожидающих нулевого значения семафора, увеличивается на единицу

обратившийся процесс переводится в состояние ожидания (усыпляется)

Значение поля операции отрицательно

(1) его абсолютное значение меньше или равно значению семафора

это отрицательное значение прибавляется к значению семафора

если значение семафора стало нулевым, то ядро активизирует все процессы, ожидающие нулевого значения этого семафора

(2) значение семафора меньше абсолютной величины поля операции

число процессов, ожидающих увеличения значения семафора увеличивается на единицу

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

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

semctl(id, number, cmd, arg);

id - это дескриптор группы семафоров

number - номер семафора в группе

cmd - код операции

arg - указатель на структуру, содержимое которой интерпретируется в зависимости от операции

Можно уничтожить индивидуальный семафор в указанной группе

Очереди сообщений

Четыре системных вызова:

msgget для образования новой очереди сообщений или получения дескриптора существующей очереди

msgsnd для посылки сообщения (его постановки в очередь сообщений)

msgrcv для приема сообщения (выборки сообщения из очереди)

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:

указатели на первое и последнее сообщение в данной очереди

число сообщений

общий размер в байтах сообщений, находящихся в очереди

идентификаторы процессов, которые последними послали или приняли сообщение через данную очередь

временные метки последних выполненных операций msgsnd, msgrsv и msgctl

Структуры данных, используемые для организации очередей сообщений

msgsnd(msgqid, msg, count, flag);

msg - это указатель на структуру, содержащую целочисленный тип сообщения и символьный массив

count - задает размер сообщения в байтах

flag определяет действия ядра при выходе за пределы допустимых размеров внутренней буферной памяти

Условия успешной постановки сообщения в очередь:

процесс должен иметь право на запись в очередь

длина сообщения не должна превосходить верхний предел

общая длина сообщений не должна превосходить установленного предела

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

обратившийся процесс откладывается до разгрузки очереди

но есть флаг IPC\_NOWAIT (как для семафоров)

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

msg - указатель на структуру данных в адресном пространстве пользователя для размещения принятого сообщения

maxcount - размер области данных (массива байтов) в структуре msg

type специфицирует тип сообщения, которое желательно принять

flag указывает ядру, что следует предпринять, если в указанной очереди сообщений отсутствует сообщение с указанным типом

count - реальное число байтов, переданных пользователю

Значением параметра type является нуль

выбирается первое сообщение

копируется в заданную пользовательскую структуру данных

процессы, отложенные по причине переполнения очереди сообщений, активизируются

если значение параметра maxcount оказывается меньше реального размера сообщения, ядро не удаляет сообщение из очереди и возвращает код ошибки

если задан флаг MSG\_NOERROR, то выборка сообщения производится, и в буфер пользователя переписываются первые maxcount байтов сообщения

Значение type есть положительное целое число

выбирается первое сообщение с таким же типом

Значение type есть отрицательное целое число

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

В очереди отсутствуют сообщения, соответствующие спецификации type

процесс откладывается до появления в очереди требуемого сообщения

но есть флаг IPC\_NOWAIT

msgctl(id, cmd, mstatbuf);

опрос состояния описателя очереди сообщений

изменение его состояния

уничтожение очереди сообщений

Программные каналы

Создание неименованного программного канала

pipe(fdptr);

fdptr - это указатель массива из двух целых чисел для размещения дескриптора для чтения из программного канала (с помощью read) и записи в программный канал (с помощью write)

обычные дескрипторы файлов

два элемента таблицы открытых файлов процесса

Создание именованных программных каналов (или получение доступа к существующим)

Обычный системный вызов open

если канал открывается на запись, и ни один процесс не открыл его для чтения, то процесс блокируется до тех пор, пока некоторый процесс не откроет этот канал для чтения

аналогично обрабатывается открытие для чтения

имеется флаг NO\_DELAY

Запись и чтение: read и write

при записи данные помещаются в начало канала

при чтении выбираются из конца канала

возможны откладывания процессов

Окончание работы процесса: close

при выполнении последнего закрытия канала по записи все процессы, ожидающие чтения из программного канала, активизируются с возвратом кода ошибки из системного вызова

Программные гнезда (sockets)

Поддерживаемый ядром механизм, скрывающий особенности сетевой среды и позволяющий единообразно взаимодействовать процессам

выполняющимся на одном компьютере

в пределах одной локальной сети

разнесенным на разные компьютеры территориально распределенной сети

Первое решение:

UNIX BSD 4.1 в 1982 г.

Три составляющих:

компонент уровня программных гнезд (независящий от сетевого протокола и среды передачи данных)

компонентом протокольного уровня (независящий от среды передачи данных)

компонентом уровня управления сетевым устройством

Одна из возможных конфигураций программных гнезд

Допустимые комбинации протоколов и драйверов задаются при конфигурации системы

во время работы системы менять нельзя

По духу организация программных гнезд близка к идее потоков

Но менее гибкая схема

не допускает изменения конфигурации "на ходу"

Взаимодействие процессов основано на модели "клиент-сервер"

процесс-сервер "слушает (listens)" свое программное гнездо

процесс-клиент пытается общаться с процессом-сервером через другое программное гнездо

ядро поддерживает внутренние соединения и маршрутизацию данных от клиента к серверу

Программные гнезда с общими коммуникационными свойствами, такими как способ именования и протокольный формат адреса, группируются в домены

"домен системы UNIX" для процессов, которые взаимодействуют через программные гнезда в пределах одного компьютера

"домен Internet" для процессов, которые взаимодействуют в сети в соответствии с семейством протоколов TCP/IP

Два типа программных гнезд

с виртуальным соединением (stream sockets)

дейтаграммные гнезда (datagram sockets)

Виртуальные соединения:

передача данных от клиента к серверу в виде непрерывного потока байтов с гарантией доставки

до начала передачи данных должно быть установлено соединение

Дейтаграммные программные гнезда:

не гарантируют абсолютной надежной, последовательной доставки сообщений и отсутствия дубликатов дейтаграмм

не требуется предварительное установление соединений

По умолчанию обеспечивается подходящий протокол для каждой допустимой комбинации "домен-гнездо"

TCP для виртуальных соединений

UDP для дейтаграммного способа коммуникаций

Создание нового программного гнезда:

sd = socket(domain, type, protocol);

domain - домен гнезда

type - тип (с виртуальным соединением или дейтаграммное)

protocol - желаемый сетевой протокол

sd - дескриптор программного гнезда

Закрытие (уничтожение) гнезда

close(sd)

Связывание ранее созданного программного гнезда с именем:

bind(sd, socknm, socknlen);

sd - дескриптор ранее созданного программного гнезда

socknm - адрес структуры, которая содержит имя (идентификатор) гнезда, соответствующее требованиям домена данного гнезда и используемого протокола

для домена системы UNIX имя является именем объекта в файловой системе

при создании программного гнезда создается файл

socknlen - длина в байтах структуры socknm

Запрос связи с существующим программным гнездом со стороны процесса-клиента:

connect(sd, socknm, socknlen);

смысл параметров, как у функции bind

имя программного гнезда на другой стороне коммуникационного канала

у гнезда с дескриптором sd и у гнезда с именем socknm должны быть одинаковые домен и протокол

если тип гнезда с дескриптором sd - дейтаграммный, то connect служит для информирования системы об адресе назначения пакетов, которые в дальнейшем будут посылаться с помощью функции send

Информирования о том, что процесс-сервер планирует установление виртуальных соединений через указанное гнездо:

listen(sd, qlength);

qlength - максимальная длина очереди запросов на установление соединения, которые должны буферизоваться системой, пока их не выберет процесс-сервер

Выборка процессом-сервером очередного запроса на установление соединения с указанным программным гнездом служит функция accept:

nsd = accept(sd, address, addrlen);

sd - дескриптор существующего программного гнезда, для которого ранее была выполнена функция listen

address - массив данных, в который должна быть помещена информация, характеризующая имя программного гнезда клиента

addrlen - адрес, по которому находится длина массива address

выполнение функции приводит к установлению виртуального соединения

nsd - новый дескриптор программного гнезда, который должен использоваться при работе через данное соединение

по адресу addrlen помещается реальный размер массива данных, которые записаны по адресу address

Передача и прием данных через программные гнезда с установленным виртуальным соединением:

count = send(sd, msg, length, flags);

count = recv(sd, buf, length, flags);

В send:

msg указывает на буфер с данными, которые требуется послать

length - длина этого буфера

flags == MSG\_OOB внеочередная посылка данных

В recv:

buf указывает на буфер, в который следует поместить принимаемые данные

length - максимальная длина этого буфера

flags == MSG\_PEEK перепись сообщения в пользовательский буфер без его удаления из системных буферов

Вместо send и recv можно использовать read и write

выполняются аналогично send и recv

Для посылки и приема сообщений в дейтаграммном режиме:

count = sendto(sd, msg, length, flags, socknm, socknlen);

count = recvfrom(sd, buf, length, flags, socknm, socknlen);

смысл параметров sd, msg, buf и lenght аналогичен смыслу одноименных параметров функций send и recv

socknm и socknlen функции sendto задают имя программного гнезда, в которое посылается сообщение

могут быть опущены, если до этого вызывалась функция connect

параметры socknm и socknlen функции recvfrom позволяют серверу получить имя пославшего сообщение процесса-клиента

Немедленная ликвидация установленного соединения:

shutdown(sd, mode);

немедленно остановить коммуникации

со стороны посылающего процесса

со стороны принимающего процесса

с обеих сторон

в зависимости от значения параметра mode

shutdown отличаются от close:

выполнение close последней "притормаживается" до окончания попыток системы доставить уже отправленные сообщения

shutdown разрывает соединение, но не ликвидирует дескрипторы ранее соединенных гнезд

для ликвидации требуется вызов close

Потоки (streams)

UNIX System V

библиотека TLI (Transport Layer Interface)

транспортный сервис на основе стека протоколов TCP/IP

Позволяют организовывать разнообразные виды коммуникации процессов

Многообразие и сложность набора функций библиотеки TLI

Относится к теме реализаций семиуровневой модели ISO/OSI