+ С we're ready for typing...

Туттуру!

03.12.20

Интерпроцес коммьюникейшн System V...

Сигналы

Сигналы были описаны в методе ко второй лабе по юникс (5 программа).

Внешнее по отношению к процессу асинхронное событие.

Сигналы в юникс  соответствуют каким-либо событиям. Механизм сигналов позволяет процессам реагировать на события, причем эти события могут происходить как вне процесса, так и внутри; как правило, получение процессом сигнала указывает ему на необходимость завершиться.

Важнейшим событием в системе является завершение процесса. Несмотря на то, что в системе определены реакции на определённые события процессов, процесс может сам определить собственную реакцию на получаемый сигнал. Например, получив сигнал, процесс может его проигнорировать, выполнить стандартные действия, связанные с данным сигналом или может вызвать собственный обработчик сигнала, то есть по-своему отреагировать на полученный сигнал.

Сигналы описаны в библиотеке <signal.h>

В классическом юникс определено не более 20 сигналов.

#define NSIG 20

Я перечислю несколько наиболее важных сигналов.

В общем-то те из вас, кто внимательно читал, видели, что они описаны,

но я хочу обратить внимание на такие важные  сигналы, как:

1) # define  SIGHUP 1

Разрыв связи с терминалом

В юникс сигналы обозначаются большими буквами - это принятый способ обозначения предописанных макросов.

Так же есть числовой ID.

2) #define SIGINT 2

ctrl + C

3) #define SIGKILL 9 (я не буду комментировать :^))

4) #define  SIGSEGV 11 - нарушение сегментации

Нужно обратить внимание, почему тут есть такая ошибка, когда мы обсуждали, собственно выполняется страничное преобразование.

Умеем размечать сегмент кода, данных, стека

Эта ошибка означаетс, что в системе выполняется преобразование страницами(?) по запросу

Мы с вами говорил, что системы для защиты адрес пространств

проверяет выход процесса за адрессное пространство

#define  SIGPIPE 13

Запись в канал есть, чтения нет

#define SIGALARM 14

Сигнал побудки ??? (будильник)

#define SIGTERM 15

программное прерывание

Когда используется kill в терминале - возникает програмное прерывание(посылается сигнал номер 15).

Есть два SIGUSR1, SIGUSR2 сигнала,

которые предназначены непосредственно для пользователя.

#define SIGCLD 18 завершился процесс потомок

#define SIGPWR 19 авария питания

#define SIG\_DFL(int((\*)())) 0 все установки будут использоваться по умолчанию

#define SIG\_IGN(int((\*)()))  1 приписывает игнорирование сигнала.

В методе дан более широкий список сигналов. Соотвественно надо понимать, что не везде существет обратнаясовместимость. Может быть свой расширенный набор сигналов.

Средствами посылки и обработки сигналав юникс служат два системных вызова: kill, signal

int kill (int pid, int sig);

два формальных параметра: идентификатор процесса и сигнал.

int pid не обязательно должен быть равен идентификатору процесса. Например, если параметр pid <= 1, то сигнал sig будет послан к группе процессов.

Если pid == 0, то sig будет послан всем процессам с ид группы совпадающими с ид группы процесса, который осуществил системный вызов kill, кроме процессов сpid 0 и 1.

Показательный пример вызова kill:

    kill (getpid(), SIGALARM)

    означает, что сигнал побудки будет послан самому процессу, вызвавшему kill

 Ситемный вызов signal

 void (\*signal(int sig, void (\*handler()(int)))(int)

 Для того, чтобы замаскировать сигнал, необходимо в программе вызвать signal(SIGINT, SIG\_IGN)

 нажатие ctrl+C означает завершение процесса. Вернуть нормальную реакцию можно следующим образом

 signal(SIGNINT, SIG\_DFL)

Сиcтемный вызов signal не входит в стандарт POSIX .1.

Ситемный вызов сигнал определен в ANSI C. Следовательно имеется во всех юникс и линукс системах, но не рекомендуется использовать в кросс-платформенных приложениях, тк его действия будут отличаться в разных системах.

Полное название POSIX - данный стандарт разработан национальным институтом стандартов и технологий США (NIST). Последний вариант был опубликован в 1988 году.

POSIX.1 FIPS

Federal

Information

Processing

Standart

Документ был написан для федеральных ведомств, которые покупают компьютерную технику. В любой стране основным покупателем является государство. Рыночные отношения позволяют нам выбирать, игнорировать этот документ или следовать этому документу и получить в клиенты само государство....

Возникновение POSIX привело к появлению других стандартов. POSIX появлися вследствие сильного расхождения ОС Unix  - в этих ОС стали использовать разные наборы системных вызово - возникла ситуация, когда ПО на System5 нельзя было использовать на Unix BSD  и наоборот.

POSIX - чтобы создавать переносимое ПО.

X/open создан группой европейских кмпаний для создания переносимого ПО

X/Open Portability Guide (xPG3) - 1994.

Стандарт X/Open основан на ANSI C, POSIX.1, POSIX. 2, содержит дополнительные конструкции, разработанные в рамках X/Open

В методе ко второй лабе был пример собработчиком сигнала catch\_sig

Если системный вызов signal не входит в POSIX, значит в posix должен быть свой системный вызов

int sigaction(int signal\_num, struct sigaction \*action,  struct sigaction \*old\_action);

struct sigaction определена в библиотеке signal.h

В системном вызове можно определть дополнительные сигналы, которые будут,например, блокироваться при обработке signal\_num.

В лабе подчеркивалось, что с помощью сигналов, которые сопровождают события в системе (асинхронные по отношению к процессу) можно менять ход выполнения программы.

В posix есть две функции для этих целей:

    1) sigsetjmp предназначен для того чтобы отметить одну или несколько позиций в программе - для перехда в одну из этих позиций(?)

    2) siglongjmp

Как мы видим, это гибкий подход к изменению поведения (?) программы.

43:12

Семафоры в System5

Типы семафоров:  бинарный, считающий, множественный(?)

Наборы считающих семафоров - этот тип поддерживают все современные ОС.

Доступ к отдльному семафору из наборавыполняется по индексу. Важнейшим св-вом набора семафоров является то, что одной неделимой операцией можно изменить все или часть набора семафоров.

Освободить семафор может любой процесс, не только тот, который захватил семафор.

Проверить безопасность использования семафоров в распараллеленных приложениях или при взаимодействии параллельных процессов очень непросто.

Если в программе используется большое количество семафоров, то в таких системах вероятность возникновения тупика является очень высокой. Именно поэтому было введена возможность реализованная в системе,  что одной неделимой операцией можно изменить весь или часть набора семафоров.

В адресном пространстве ядра имеется таблица семафоров - системная таблица.

В этой системной таблице отслеживаются все создаваемые в системе семафоры. В каждом элементе такой таблицы находятся следующие данные об одном наборе семафров.

Дескриптор набора семафоров:

1) Имя (идентификатор) целое число, присваивается процессом, который создал набор семафоров. Другие процессы по этому имени могут открыть набор и получить дескриптор набора для доступа (?)

2) UID идентификатор создателя набора семафоров - процесс, эффективный UID которого совпадает с UID создателя, может удалять набор и изменять управляющие параметры набора

3) Права доступа

4) Количество семафоров в наборе

5) время изменения одного или нескольких значений семафоров последним процессом

6)  Время последнего изменения управляющих параметров набора

7)  указатель на массив семафоров

Семантически наборы семафоров представляются как массивы, первый семафор имеет индекс 0.

Каждый отдельный семафор имеет набор параметров.

    Дескриптор отдельного семафора:

    1)  Значение семафора

    2)  Идентификатор процесса, который оперировал семафором в последний раз

    3)  Число заблокированных процессов (на семафоре)

рисунок 61:30

Таблица семафоров

Каждый элемент этой таблицы описывает структуры struct semid\_ds

Каждая строка таблицы описывает отдельный набор семафоров

Каждый семафор описан структурой struct sem.

На семафорах определены следующие системные вызовы:

    semget(); создаёт набор семафоров

    semctl(); позволяет изменять управляющие параметры набора семафоров

    semop(); изменяет значение семафора - отдельного, набора или части набора семафоров

    на семафоре определена структура struct sembuf

    1) ushort sem\_num - индекс семафора

    2) short sem\_op - операция на семафоре

    3) short sem\_flg - флаги, определенные на семафоре

    В отличие от классических семафров дейкстры (у него две операции), на семафоре юникс определны три операции

    1)  sem\_op > 0 освобождление процесса, который выполнил системный вызов semop() - увеличение значения семафора, которое приведет к разблокировке заблокированных на данном семафоре процессов

    2) sem\_op == 0 процесс, который выполнил такой системный вызов будет переведен в состояние ожидания до момента освобождения ресурса, но ресурс не захватывается

    3)  sem\_op < 0 захват ресурса - уменьшение значения семафора.

    Декрементировать семафор можно только если семафор имеет положительное значение.

    На семафорах определны специальные флаги.

    IPC\_NOWAIT - информирует ядро системы о нежелании процесса переходить в состояние ожидания

    Объясняется стремлением избежать блокировки всех процессов, которыенаходятся в очереди к семафору. В том случае, если захвативший ресурс процесс завершится аварийно или получит сигнал kill

    Это важно в силу того, что сигнал kill невозможно перехватить. Убиваемый процесс не может выполнить освобождение семафора и открыть другим процессам доступ к ресурсу, которым он может управлять.

    SEM\_UNDO - указывает ядру, что оно должно отслеживать изменение значения семафора в результате системного вызова semop() и при завершении процесса, вызвавшего semop(), ядро ликвидирует сделанные изменения для того, чтобы процессы не были заблокированы на семафоре навечно.

    75:06

    Рассмотри пример создания набора семафоров и определения операций на данном наборе.

    #include <sys/types.h>

    #include <sys/tpc.h>

    #include <sys/sem.h>

// объявляем набор из двух семафоров и задаем значения каждого из семафоров: первое значение - это индекс, второе - операция, третье - флаги (см объявление структуры)

struct sem\_buf sbuf[2] = {{0, -1, SEM\_UNDO |

IDC\_NOWAIT}, {1, 0, 1}};

int main()

{

    // определим права доступа к каждому из семафоров

   int perms = S\_IRWXU | S\_IRNXG | S\_IRWXO;

   // дескриптор набора семафоров

   int fd = semget(100, 2, IPC\_CREATE | perms);

* // если semget выполнен удачно - нам удалось создать набор семафоров
* if (fd == -1) { perror("semget");  exit(1); }
* if (semop(fd,  sbuf, 2) == -1) { perror("semop");  }
* return 0;

}

Что делает данная программа:

    В программе открывается набор из двух семафоров с ид 100. Этот набор будет создан, если такого набора еще не было. Для всех категорий польз установлены полные права доступа. Если вызов semget успешен, то выполняется semop, который декрементирует значение первого семафора, если это возможно, и выполняет проверку на 0 второго семафора.

    86:06

    Сегменты разделяемой памяти

Программыне каналы - потоки(?), которые создаются ядром системы.

88:48

Если процессы имеют защ адресное пространство - другой процесс не может обратить в такое адресное пространство. Для взаимодействия процессов существует одно единственное адресное пространсто - область памяти ядра системы. В ядре создаются средства взаимодействия процессов (например, программыне каналы).

Особенность разделяемых сегментов памяти: в отличие от других средств взаимодействия, разделяемые сегменты подключаются к адр пространству процессов. Это искючает необходимость копирования данных из адр простр процесса в адр пространство ядра и наоборот. Сегменты разделяемой памяти были задуманы как средство повышения производительности при передаче сообщения от одного процесса другому.

рисунок 92:41

Существует таблица разделяемых сегментов.

<sys/shm.h>

дескриптор описывается структурой struct shmid\_ds

особенностью разделяемых сегментов является то, что процесс получает указатель на разделяемый сегмент

Для разделяемой памяти определены системные вызовы:

    shmget() ; создает разделяемый сегмент

    shmctl() изменяет упр параметры сегмента

    shmat() attach получение указателя на разделяемый сегмент

    shmdt() detach отделение сегмента от адресного пространства процесса

    пример 97:10

    int main()

    {

* int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;
* // id = 100, размер 1024 байта
* int fd = shmget(100, 1024, IPC\_CREATE | perms);
* if (fd == -1) {perror("shmget"); exit(1) }
* // необходимо подключить, созданную разд память к адр пространству процесса
* char \*adr = (char\*)shmat(fd, 0, 0);

          if (addr == (char \*) -1) { perror("shmat"); exit(1);}

          // если сегмент подключен, то в него записывается сообщние

          strcpy(addr, "hello");

          // после этого разд сегмент отключается

          if (shmdt(addr) == -1) perror("shmdt");

          return 0;

    }

Двух приведенных примеров - создание и изменение значений семафоров и создание разделяемого сегмента  - достаточно для лабы 3 юникс.

107:26

Системное ограничение | значение

**SHMMNI**- make число разд сегментов

**SHMMIN**- минимально возможный размер разд сегмента в байтах

**SHMMAX**- максимально возможный размер разд сегмента в байтах

Если процесс попытается создать новый разд сегмент а их число превысит максимальное (SHMMNI ), то процесс будет заблокирован до тех пор, пока другой процесс не освободить какой-то разделяемый сегмент

Если процесс попытается создать разд сегмент, размер которого превышает макс значения (SHMMAX ), то системный вызов не будет выполнен - возвращена ошибка

110:49

Программные каналы

В современных ОС юникс существуют программные каналы, которые принято называть pipe - именованные и неименованные.

Почему pipe (труба) : (рисунок трубы112:00)

В один конец втекает, из другого вытекает - передача данных в одном направлении (потоковая передача данных).

Если надо передавать данные в обе стороны, то нужна вторая труба, в которойдвижение информации будет противоположно первому.

Именованные программные каналы и неименованные - два типа программных каналов, которые поддерживаются современными ОС.

115:06

Именованные программные каналы

mknode - создание именованного программного канала (см лаба 2 по юникс)

При создании именованного программного канала, это будет труба типа FIFO (тип потоковой передачи данных)

Являются в системе специальными файлами и видны в файловой системе, более того - они имеют идентификатор в файловой системе. Любой процесс, который знает идентификатор именованного программного канала, может работать с этим программным каналом.

117:03

Неименованные программные каналы

Не имеют идентификатора, но поддерживаются средствами файловой системы, имеют дескриптор.

В силу особенностей системного вызова fork

(системный вызов форк создает новый процесс, процесс потомок, который является копией процесса предка - наследует код предка, дескрипторы открытых файлов, сигнальную маску и тд) неименованными программными каналами могут пользоваться процессы-родственники, тк процессы потомки наследуют от предка дескрипторы открытых файлов.

Программыне каналы имеют средства управления доступом к программному каналу

int fd[2]; // создается массив дескрипторов

pipe(fd); // передается в пайп чтобы управлять доступом

В канал нельзя писать, если из него читают,

Из канала нельзя читать, если в него пишут

(Канал закрывается на чтение при записи и закрывается на запись при чтении)

Важно

------------------

Труба буферизуется на трех уровнях

Программные каналы буферизуются в системной памяти (в области данных ядра системы)при переполнении системной памяти, буфера, имеющие наибольшее время сущствования переписываются на диск. Используются стандартные функции работы с памятью. Если процесс записывает в пайп больше 4096 байт, то труба будет буферизоваться по времени, приостанавливая процесс, который записывает в друбу до тех пор, пока данные не будут прочитаны. Ограничение значений канала (4096) - повысить эффективность операции обмена. При операции обмена ОС выделяет системные буферы. Если его размер не превышает 4096, то такой канал может целиком разместиться в системном буфере. (Это важная мысль)

Чтение из памяти одиночной переменной только на 30 % быстрее, чем передача одной страницы. Это связано с тем, что в системе оптимизируется передача страниц. (буфера соответсвуют размеру страницы)

 --примечание к лабе 2 по UNIX--

 Прежде чем читать из канала, его надо закрыть для записи, прежде чем писать в канал, его надо закрыть для чтения