ПРОЦЕССЫ В ОС UNIX.

Цель работы.

Изучение системных команд ОС UNIX, обеспечивающих идентификацию

процессов, их синхронизацию и обмен информацией между ними в процессе

работы. Разработка взаимодействующих между собой программ, обеспечиваю-

щих синхронизированное решение общей задачи в рамках иерархической

системы "родитель-потомки" или системы равноправных взаимодействующих

процессов.

Взаимодействие процессов в ОС UNIX.

Само назначение ОС UNIX в качестве системы массового обслуживания

обусловило установку в ней мощного механизма для обеспечения взаимо-

действия между различными проходящими в ней процессами. В самом деле,

первоначально цель разработки состояла в том, чтобы обеспечить в машине

одновременную работу многих пользователей, ведущих диалог с ЭВМ или

между собой посредством терминалов. Это требовало:

-с одной стороны, обеспечить надежный обмен информацией между ма-

шиной и пользователем так, чтобы вся информация точно находила своего

адресата;

-с другой стороны, обеспечить надежную, быструю и бесперебойную

передачу информации между процессами пользователя и системными про-

цессами, причем последние должны одновременно обслуживать многих поль-

зователей.

В ОС UNIX решение основано на том, что при запуске системы созда-

ется главный порождающий процесс (ПРОЦЕСС 1), который инициализирует

работу управляющей программы ОБОЛОЧКИ, через которую пользователи и

создают свои процессы. Созданные ОБОЛОЧНОЙ пользовательские процессы

могут, в свою очередь, также порождать другие процессы. Таким образом

создается иерархия выполняемых процессов. Причем каждый из порожденных

процессов при своем создании наследует все свойства порождающего про-

цесса (за исключением тех, которые последний "сознательно" не желает

ему отдавать). Вместе с тем, порожденный процесс в ходе работы может

приобрести некоторые новые свойства (приоритетность, ресурсы, информа-

ционные блоки и т.д.), которые будут характеризовать его локальные

свойства. Как правило, удаление процесса-предка ведет к удалению свя-

занных с ним потомков, хотя это свойство не является всеобщим, т.к. в

ОС UNIX в этом случае предусмотрен механизм переподчинения про-

цессов-потомков, потерявших своего предка, его выше стоящим предком.

Для идентификации любой порождаемый в ОС UNIX процесс получает при

создании уникальный номер - ИДЕНТИФИКАТОР ПРОЦЕССА (PID). При запуске

PID передается предку. Свой PID прцесс может узнать командой

GETPID().

Для определения PID порождащего процесса служит команда

GETPPID().

Процессы, запускаемые одним предком, объединяютя в одну группу, иденти-

фикатором которой будет идентификатор процесса-предка PPID. Идентифика-

торы процессов можно переустанавливать специальными командами ОС UNIX,

с которыми следует познакомиться самостоятельно по ранее рекомендованн-

ой литературе.

Если процесс завершил работу или был удален досточно, ОС очистит

принадлежащие ему зоны памяти, закроет все связанные с ним файлы и иск-

лючит его из очереди процессов. Норьальное завершение процесса выполня-

ется либо оператором RETURN, входящим в функцию MAIN программы, либо

системным вызовом

EXIT (STATUS),

который может появиться в любом месте программы. При этом про-

цессу-потомку будет возвращено значение, которое устанавливается в млад-

шем байте целой переменной STATUS. Процесс-потомок может ожидать завер-

шения своего предка, если в нем установлен системный вызов

PID = WAIT(STATUS),

где PID - переменная целого типа, соответствующая PID завершающегося

процесса.

STATUS - в старшем байте содеожит код, указываемый в младшем байте

операнда EXIT, а в старшем байте устанавливается системный

код завершения, устанавливаемый ядром ОС.

Для обеспечения синхронизации выполняемых в ОС UNIX процессов слу-

жат СИГНАЛЫ, которые процессы могут передавать друг другу вне зависи-

мости от родственных связей и обрабатывать. Сигналы позволяют опреде-

лить в процессе посылку в них управуляющих символов с терминалов

(SIGINT, SIGOUT, SIGHUP), наличие некоторых аварийных ситуаций

(SIGKILL, SIGTRAP, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGPIPE), возникно-

вение некоторого заранее ожидаемого события (SIGALARM), появление неп-

редусмотренного события (SIGTERM, SIGGLD, SIGPWR) или вызова со стороны

другого пользователя (SIGUSR1, SIGUSR2). Подробнее с сигналами познако-

миться самостоятельно по рекомендованной литературе.

Послать сигнал к другому процессу можно системным вызовом

KILL (PID, SIG),

где SIG - посылаемый сигнал,

PID - идентификатор процесса. Если PID = 0, то сигнал будет послан

всем процессам, имеющим общий с пославшим процессом идентифи-

катор группы.

Чтобы установить реакцию процесса на определенный сигнал использу-

ется системный вызов:

#include <signal.h>

int (\*signal (sig, func())()

int sig; - сигнал, на который следует реагировать после вы-

зова данной функции;

int (\*func)() - функция, которая будет вызвана после получе-

ния указанного сигнала.

ПРИМЕЧАНИЕ: вызовы WAIT, KILL, SIGNAL выполняются только один раз, т.о.

если необходимо их повторное выполнение, то это необходимо

определить в программе явно.

В данном системном вызове имеется две специальные функции

signal (sig, SIG\_IGN) - ранее определенный сигнал sig будет игно-

рирован.

signal (sig, SIG\_DEL) - реакция на sig как ви выполненной ранее

функции или принятое по умолчанию.

Сигналы выполняют роль системы управления процессом передачи ин-

формации. Сама же передача информации осуществляется поименованным или

непоименованным каналами.

Системный вызов

PIPE (FD)

лишь декларирует непоименованный канал. FD - ьассив из двух элементов,

первый из них открывает канал на ввод, а второй на вывод.

В качестве примера преведены две программы для обеспечения ввода

информации в один файл с нескольких терминалов. ВЕДУЩАЯ программа для

каждого канала, с которого будет вводится информация, порождает про-

цесс, описанный ВЕДОМОЙ программой. После этого процесс "ведущей прог-

раммы" перейдет в ожидание сигнала о передаче информации от "ведомой"

программы. "Ведомая" программа выполняет ввод информации со своего тер-

минала и ее передачу по каналу, уведомив "ведущую" программу сигналом

SIGFPE.

/\*

\* ПРОЦЕСС-ПОТОМОК

\*/

#include <signal.h>

#include <sys/types.h>

#include <sys/tty.h>

main (argc,argv)

int argc;

char \*argv[];

{

int n;

char buffer[TTYHOG];

for(;;)

{n = read(0,buffer,TTYHOG);

if(n == 0)

{ /\* сброс из основной задачи \*/

kill (atoi(argv[1]), SIGFPE);

close(1);

exit(0);

}

kill(atoi(argv[1]),SIGTERM);

write(1, buffer, n);

}

}

/\*

\* ПРОЦЕСС-ПРЕДОК

\*/

#include <signal.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/tty.h>

int fd[2];

char pids[TTYHOG];

int slave\_count;

/\*

\* программа обработки сигнала SIGTERM

\*/

service()

{

int n;

char buffer[TTYHOG];

signal(SIGTERM,service);

n = read(fd[0], buffer, TTYHOG);

write(1, buffer, n);

}

/\*

\* программа завершения по сигналу SIGFPE

\*/

sdrop()

{

signal(SIGFPE, sdrop);

if(--slave\_count == 0)

exit(0);

}

/\*

\* программа инициализации потомка

\*/

slave(tty)

char \*tty;

{

if(fork() == 0)

{ /\* переназначение стандартного ввода на терминал \*/

int tfd;

close(0);

tfd = open(tty,0);

if(tfd != 0)

{

fprintf(stderr,"bad tty %S\n",tty);

exit(1);

}

/\* переназначение стандартного вывода на канал \*/

close(1);

dup(fd[1]);

close(fd[0]);

close(fd[1]);

/\* вызов потомка \*/

execl("./slave","slave",pids,0);

fprintf(stderr,"can't exec slave on %S\n",tty);

exit(1);

}

}

/\*

\* основная программа

\*/

main(argc, argv)

int argc;

char \*argv[];

{

int i;

signal(SIGTERM,service);

signal(SIGFPE,sdrop);

sprintf(pids,"%06d",getpid());

pipe(fd);

slave\_count = argc - 1;

for(i=1;i<argc;i++)

slave(argv[i]);

close(fd[1]);

for(;;)

pause();

}

Непоименованные каналы, в основном, обеспечивают связь между про-

цессами-родственниками. Кроме того, в непоименованных каналах затрудни-

тельно организовать двусторонний обмен информацией. Поэтому в OС UNIX

были введены поименованные каналы. Если между двумя любыми процессами

создать два таких канала, то между ними можно осуществить двусторонний

обмен информацией, который также синхронизируется сигналами.

Именованный канал можно создать системным вызовом

mknod (namefile, IFIFO|0666, 0)

где namefile - имя канала;

0666 - к каналу разрешен доступ на запись и на чтение любому зап-

росившему процессу.

Именованный канал может создать администратор системы командой

$/etc/mknod namefile p

$chmod 666 namefile

В качестве примера использованы две программы, одна из которых уп-

равляет доступом к базе данных и выводит в канал контекст запроса

(PROGRAM1), а другая (PROGRAM2) реализует управление доступом к файлу,

получает контекст запроса и его обрабатывает, а в первую программу

посылает подтверждение о получении.

/\*

\* ДАННЫЕ, ОБЩИЕ ДЛЯ ВСЕХ ПРОГРАММ

\* PACKET.H

\*/

struct packet

{

int pk\_pid; /\* идентификатор процесса \*/

int pk\_blk; /\* номер блока файла \*/

int pk\_code;/\* код запроса \*/

}

/\* коды запросов \*/

#define RQ\_READ 0 /\* запрос на чтение \*/

#define CONNECT 1 /\* запрос на обслуживание \*/

#define SENDPID 2 /\* ответ на запрос о подсоединении \*/

/\* имена каналов \*/

#define DNAME "datapipe"

#define CNAME "ctrlpipe"

/\* размер блока файла \*/

#define PBUFSIZE 512

/\*

\* ПРОГРАММА ОБРАБОТКИ БАЗЫ ДАННЫХ

\*/

#include <stdio.h>

#include <signal.h>

#include <fcntl.h>

#include "packet.h"

int datapipe,ctrlpipe,datafile,got\_sig;

/\* маршрут базы данных \*/

char \*dataname = "usr/lib/tmac/tmac.c";

handler()

{

signal(SIGUSR1,handler);

got\_sig++;

}

process(pkp,spkp)

struct packet \*pkp,spkp;

{

char pbuf[PBUFSIZE];

/\*

\* запись в файл FIFO пройдет ТОЛЬКО если он уже открыт на чтение

\*/

datapipe = open(DNAME,O\_WRONLY|ONDELAY);

switch(pkp->pk\_code)

{

case CJNNECT:

write(datapipe,spkp,sizeof(struct packet));

break;

case RQ\_READ:

lseek(datafile,pkp->pk\_blk\*PBUFSIZE,0);

read(datafile,pbuf,PBUFSIZE);

write(datapipe,pbuf,PBUFSIZE);

break;

default:

fprintf(stderr,"unknown packet code\n");

exit(1);

}

got\_sig = 0;

kill(pkp->pk\_pid,SIGUSR1);

while(!got\_sig);

close(datapipe);

}

main()

{

struct packet pk;

struct packet sendpk;

ctrlpipe = open(CNAME,O\_RDONLY|O\_NDELAY);

datafile = open(dataname,0);

handler();

for(;;)

{

int n;

while(n = read(ctrlpipe, &pk, sizeof(pk)))

{

process(&pk, &sendpk);

}

}

}

/\*

\* ПРОГРАММА ТРАНСЛЯЦИИ ЗАПРОСА ПОЛЬЗОВАТЕЛЯ

\*/

#include <stdio.h>

#include <signal.h>

#include "packet.h"

int datapipe, ctrlpipe;

handler()

{ /\* перехват сигналов\*/

signal(SIGUSR1,handler);

got\_sig++;

}

/\*

\* программа для связи с процессом, обслуживающим базу данных

\*/

#include <stdio.h>

#include <signal.h>

#include <fcntl.h>

#include "packet.h"

int datapipe, ctrlpipe;

extern int got\_sig;

connect()

{

struct packet pk;

datapipe = open(DNAME,O\_RDONLY|O\_NDELAY);

ctrlpipe = open(CNAME,O\_WRONLY|O\_NDELAY);

if(datapipe < 0 || ctrlpipe < 0)

{

fprintf(stderr,"cannot open pipe\n");

exit(1);

}

pk.pk\_pid = getpid();

pk.pk\_code = CONNECT;

got\_sig = 0;

write(ctrlpipe, &pk, sizeof(pk));

while( !got\_sig);

read(datapipe, &pk, sizeof(pk));

kill(pk.pk\_pid, SIGUSR1);

return(pk.pk\_pid);

}

/\*

\* функция для считывания одного блока из базы данных

\*/

#include <stdio.h>

#include <signal.h>

#include "packet.h"

extern int datapipe, ctrlpipe, got\_sig;

request (ptr, blk, spid)

char \*ptr;

int blk;

int spid;

{

struct packet pk;

pk.pk\_pid = getpid();

pk.pk\_blk = blk;

pk.pk\_code = RQ\_READ;

got\_sig = 0;

write(ctrlpipe, &pk, sizeof(pk));

while (!got\_sig);

read (datapipe, ptr, PBUFSIZE);

kill(spid, SIGUSR1);

}

/\*

\* главная программа

\*/

main (argc,argv)

int argc;

char \*argv[];

{

int spid; /\* идентификатор сервера \*/

int blk;

char buffer[PBUFSIZE];

blk = atoi(argv[1]);

/\* перехват сигнала \*/

handler();

spid = connect();

for(;;)

{

request (buffer, blk, spid);

fwrite (buffer, PBUFSIZE, 1, stdout);

}

}

Создать канал можно и между процессами, запускаемыми в SHELL. Как

это сделать разберите самостоятельно по рекомендованной ранее книге Ду-

наева.

Контрольные вопросы для самоконтроля.

1. Что такое процесс и чем он отличается от программы?

2. Как порождаются процессы в ОC UNIX?

3. Что такое програмный идентификатор процесса? Опишите системные

вызовы для работы с ним.

4. Приведите примеры, где необходимо использовать програмный иден-

тификатор процесса.

5. Как запустить канал непосредственно из SHELL?

6. Опишите формат системного вызова PIPE и особенности его работы.

7. Опишите формат системного вызова MKNOD и особенности его работы.

8. Что такое сигналы и как они используются в ОС UNIX?

9. Опишите работу системного вызова KILL.

10. Опишите работу системного вызова SIGNAL>

11. Какие ограничения существуют на длину файла канала?

12. В чем особенности и отличия поименованных и непоименованных

каналов?

13. Сколько сигналов необходимо использовать для синхронизации

процессов использующих каналы?

14. Каков формат и какие методы использования системного вызова

WAIT?

15. Опишите работу системного вызова FORK>

16. Опишите системный вызов EXEC и его модификации.

17. Можно ли использовать системный вызов PIPE для обмена информа-

цией между процессами, не состоящими в родственных отношения? Если мож-

но, то как это выполнить?

18. Какие действия выполняет системный вызов EXIT и каков его фор-

мат?

Задания лабораторной работы.

1) Перед выполнением работы ознакомьтесь с изложенным выше материалом, проанализируйте какие изменения произошли с появлением стандарта С++11.

2) Выполдните задание в соотвествии с вашим вариантом.

Варианты выполнения лабораторной работы.

N вар || вариант задач (последующих пунктов)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 3 и 4

2 3 и 5

3 5 и 6

4 3 и 7

5 3 и 9

6 5 и 10

7 13 и 15

8 14 и 16

9 7 и 11

10 10 и 12

11 4 и 8

12 5 11

13 6 и 8

14 3 и 8

15 7 и 8

16 4 и 16

17 10 и 15

18 9 и 12

19 11 и 13

20 12 и 15

21 5 и 10

22 5 и 12

23 13 и 16

24 10 и 11

Варианты задач

3) Разработайте два процесса, из которых ПРОЦЕСС-ПОТОМОК порождает ПРОЦЕСС-ПРЕДОК и через непоименованный канал взаимодействует сним, нередавая содержимое некоторого файла поблочно. Если предок не может войти в файл, он должен передать сообщение на вывод STDERR. Потомок получет блоки файла и их выводит на экран, используя STDOUT. По окончании файла предок передает потомку сигнал заиершения и после окончания работы потомка (о чем получает сигнал) прекращает свою работу.

4) Зфдачу 3 решине при условии, что порождаются два процесса-предка. Для идентификации выводимой информации каждый из процессов-предков должен выдать на экран свой PID.

5) Решите задачу 3 при условии, что предок снимает информацию из файла, а предок передает ее на экран.

6) Решите задачу 5 при условии, что два предка снимают информацию из двух разных файлов, а предок передает информацию на экран, снабдив идентификатором процесса-предка, приславшего ее.

7) В рамках задачи 3 организуйте работу с файлом, записи которого переменной длины. Для проведения эксперимента на машине можно использовать, например, описанную в предыдущей лабораторной работе процедуру COPYFILE и создавать файл с клавиатуры.

8) Перекопируйте файл на экран или в другой файл непосредственно через SHELL. Дополнительные условия возьмите из первой задачи, которую вы выполняете.

9) Разработать два процесса ПОТОМОК и ПРЕДОК, связанные непоименованным каналомтак, чтобы потомок поблочно забирал информацию из некоторого файла и каждый блок передавал предку. Последний после передачи блока информации на экран должен уничтожиться, сообщив об этом предку. Т.о. потомок перед передачей каждого блока информации должен заново создать предка.

10) Задачу 9 модернизируйте так, чтобы предок забирал информацию из файла, а предок ее передавал на экран, а после этого запускал нового предка, чтобы получить следующий блок информации.

11) Задачу 9 переработайте для рабиты с блоками переменной длины.

12) Задачу 10 переработайте для работы с блоками переменной длины.

13) Приведенную в виде примера программу для работы с поименованными каналами нередерайте так, чтобы обрабатывать любой файл с записями постоянной длины. Завершите программутак, чтобы по окончании обработки обе программы нормально завершались. Режим обработки файла вы можете выбрать по своему желанию.

14) Переработайте приведенную в виде примера программу работы с поименованными каналами, чтобы с ее помощью создать базу данных, иключающую:

PID - идентификатор обслуживающей программы;

NAMBER - номер записи;

DATA - дата создания.

Окончинием работы служит признак конца набора данных с терминала (например ctrl+Z). По окончании работы сервер должен выдать на экран содержимое полученного файла.

15) В рамках условий задачи 13 обеспечьте возможность работы с записями переменной длины.

16) Модернизируйте задачу 14 для работы с записями переменной длины.