ЛАБОРАТОРНАЯ РАБОТА №4 ИСПОЛЬЗОВАНИЕ СИГНАЛОВ В ОС LINUX

Цель работы – изучение механизма взаимодействия процессов с использованием сигналов.

Теоретическая часть

Сигналы не могут непосредственно переносить информацию, что ограничивает их применимость в качестве общего механизма межпроцессного взаимодействия. Тем не менее, каждому типу сигналов присвоено мнемоническое имя (например, ***SIGINT***), которое указывает, для чего обычно используется сигнал этого типа. Имена сигналов определены в стандартном заголовочном файле ***<signal.h>*** при помощи директивы препроцессора ***#define***. Как и следовало ожидать, эти имена соответствуют небольшим положительным целым числам. С точки зрения пользователя получение процессом сигнала выглядит как возникновение прерывания. Процесс прерывает исполнение, и управление передается функции-обработчику сигнала. По окончании обработки сигнала процесс может возобновить регулярное исполнение. Типы сигналов принято задавать специальными символьными константами. Системный вызов ***kill()*** предназначен для передачи сигнала одному или нескольким специфицированным процессам в рамках полномочий пользователя.

#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int signal);

Послать сигнал (не имея полномочий суперпользователя) можно только процессу, у которого эффективный идентификатор пользователя совпадает с эффективным идентификатором пользователя для процесса, посылающего сигнал. Аргумент ***pid*** указывает процесс, которому посылается сигнал, а аргумент ***sig*** – какой сигнал посылается. В зависимости от значения аргументов:

pid > 0 сигнал посылается процессу с идентификатором ***pid***;

pid=0 сигнал посылается всем процессам в группе, к которой принадлежит посылающий процесс;

pid=-1 и посылающий процесс не является процессом суперпользователя, то сигнал посылается всем процессам в системе, для которых идентификатор пользователя совпадает с эффективным идентификатором пользователя процесса, посылающего сигнал.

pid = -1 и посылающий процесс является процессом суперпользователя, то сигнал посылается всем процессам в системе, за исключением системных процессов (обычно всем, кроме процессов ***с pid = 0*** и ***pid = 1***).

pid < 0, но не ***–1***, то сигнал посылается всем процессам из группы, идентификатор которой равен абсолютному значению аргумента ***pid*** (если позволяют привилегии).

если ***sig = 0***, то производится проверка на ошибку, а сигнал не посылается. Это можно использовать для проверки правильности аргумента ***pid*** (есть ли в системе процесс или группа процессов с соответствующим идентификатором).

Системные вызовы для установки собственного обработчика сигналов:

#include <signal.h>
void (*signal (int sig, void (*handler) (int)))(int);
int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);

Структура ***sigaction*** имеет следующий формат:

```
struct sigaction {
   void (*sa_handler)(int);
   void (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t sa_mask;
   int sa_flags;
   void (*sa_restorer)(void);
```

Системный вызов ***signal*** служит для изменения реакции процесса на какой-либо сигнал. Параметр ***sig*** – это номер сигнала, обработку которого предстоит изменить. Параметр handler описывает новый способ обработки сигнала – это может быть указатель на пользовательскую функцию-обработчик сигнала, специальное значение ***SIG_DFL*** (восстановить реакцию процесса на сигнал ***sig*** по умолчанию) или специальное значение ***SIG_IGN*** (игнорировать поступивший сигнал ***sig***). Системный вызов возвращает указатель на старый способ обработки сигнала, значение которого можно использовать для восстановления старого способа в случае необходимости.

Пример пользовательской обработки сигнала ***SIGUSR1***.

```
void *my_handler(int nsig) {    код  функции-обработчика сигнала }
int main() {
   (void) signal(SIGUSR1, my_handler); }
```

Системный вызов ***sigaction*** используется для изменения действий процесса при получении соответствующего сигнала. Параметр ***sig*** задает номер сигнала и может быть равен любому номеру. Если параметр ***act*** не равен нулю, то новое действие, связянное с сигналом ***sig***, устанавливается соответственно ***act***. Если ***oldact*** не равен нулю, то предыдущее действие записывается в ***oldact***.

Большинство типов сигналов ***UNIX*** предназначены для использования ядром, хотя есть несколько сигналов, которые посылаются от процесса к процессу:

SIGALRM – сигнал таймера (***alarm clock***). Посылается процессу ядром при срабатывании таймера. Каждый процесс может устанавливать не менее трех таймеров. Первый из них измеряет прошедшее реальное время. Этот таймер устанавливается самим процессом при помощи системного вызова ***alarm()***;

SIGCHLD – сигнал останова или завершения дочернего процесса (***child process terminated or stopped***). Если дочерний процесс останавливается или завершается, то ядро сообщит об этом родительскому процессу, послав ему

данный сигнал. По умолчанию родительский процесс игнорирует этот сигнал, поэтому, если в родительском процессе необходимо получать сведения о завершении дочерних процессов, то нужно перехватывать этот сигнал;

SIGHUP – сигнал освобождения линии (***hangup signal***). Посылается ядром всем процессам, подключенным к управляющему терминалу (***control terminal***) при отключении терминала. Он также посылается всем членам сеанса, если завершает работу лидер сеанса (обычно процесс командного интерпретатора), связанного с управляющим терминалом;

SIGINT – сигнал прерывания программы (***interrupt***). Посылается ядром всем процессам сеанса, связанного с терминалом, когда пользователь нажимает клавишу прерывания. Это также обычный способ остановки выполняющейся программы;

SIGKILL – сигнал уничтожения процесса (***kill***). Это довольно специфический сигнал, который посылается от одного процесса к другому и приводит к немедленному прекращению работы получающего сигнал процесса;

SIGPIPE – сигнал о попытке записи в канал или сокет, для которых принимающий процесс уже завершил;

SIGPOLL – сигнал о возникновении одного из опрашиваемых событий (***pollable event***). Этот сигнал генерируется ядром, когда некоторый открытый дескриптор файла становится готовым для ввода или вывода;

SIGPROF – сигнал профилирующего таймера (***profiling time expired***). Как было упомянуто для сигнала ***SIGALRM***, любой процесс может установить не менее трех таймеров. Второй из этих таймеров может использоваться для измерения времени выполнения процесса в пользовательском и системном режимах. Этот сигнал генерируется, когда истекает время, установленное в этом таймере, и поэтому может быть использован средством профилирования программы;

SIGQUIT – сигнал о выходе (***quit***). Очень похожий на сигнал ***SIGINT***, этот сигнал посылается ядром, когда пользователь нажимает клавишу выхода используемого терминала. В отличие от ***SIGINT***, этот сигнал приводит к аварийному завершению и сбросу образа памяти;

SIGSTOP – сигнал останова (***stop executing***). Это сигнал управления заданиями, который останавливает процесс. Его, как и сигнал ***SIGKILL***, нельзя проигнорировать или перехватить;

SIGTERM – программный сигнал завершения (***software termination signal***). Программист может использовать этот сигнал для того, чтобы дать процессу время для «наведения порядка», прежде чем посылать ему сигнал ***SIGKILL***;

SIGTRAP – сигнал трассировочного прерывания (***trace trap***). Это особый сигнал, который в сочетании с системным вызовом ptrace используется отладчиками, такими как ***sdb, adb, gdb***;

SIGTSTP – терминальный сигнал остановки (***terminal stop signal***). Он формируется при нажатии специальной клавиши останова;

SIGTTIN – сигнал о попытке ввода с терминала фоновым процессом (***background process attempting read***). Если процесс выполняется в фоновом режиме и пытается выполнить чтение с управляющего терминала, то ему посылается этот сигнал. Действие сигнала по умолчанию – остановка процесса;

SIGTTOU – сигнал о попытке вывода на терминал фоновым процессом (***background process attempting write***). Аналогичен сигналу ***SIGTTIN***, но генерируется, если фоновый процесс пытается выполнить запись в управляющий терминал. Действие сигнала по умолчанию – остановка процесса;

SIGURG – сигнал о поступлении в буфер сокета срочных данных (***high bandwidth data is available at a socket***). Он сообщает процессу, что по сетевому соединению получены срочные внеочередные данные;

SIGUSR1 и ***SIGUSR2*** – пользовательские сигналы (***user defined signals 1 and 2***). Так же, как и сигнал ***SIGTERM***, эти сигналы никогда не посылаются ядром и могут использоваться для любых целей по выбору пользователя;

SIGVTALRM – сигнал виртуального таймера (***virtual timer expired***). Третий таймер можно установить так, чтобы он измерял время, которое процесс выполняет в пользовательском режиме.

Наборы сигналов определяются при помощи типа ***sigset_t***, который определен в заголовочном файле ***<signal.h>***. Выбрать определенные сигналы можно, начав либо с полного набора сигналов и удалив ненужные сигналы, либо с пустого набора, включив в него нужные. Инициализация пустого и полного набора сигналов выполняется при помощи процедур ***sigemptyset*** и ***sigfillset*** соответственно. После инициализации с наборами сигналов можно оперировать при помощи процедур ***sigaddset*** и ***sigdelset***, соответственно добавляющих и удаляющих указанные вами сигналы.

Описание данных процедур:

#include <signal.h>

/* Инициализация*/

int sigemptyset (sigset_t *set);

int sigfillset (sigset_t *set);

/*Добавление и удаление сигналов*/

int sigaddset (sigset_t *set, int signo);

int sigdelset (sigset_t *set, int signo);

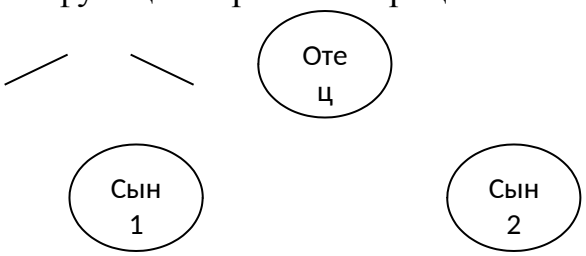
Процедуры ***sigemptyset*** и ***sigfillset*** имеют единственный параметр – указатель на переменную типа ***sigset_t***. Вызов ***sigemptyset*** инициализирует набор ***set***, исключив из него все сигналы. И, наоборот, вызов ***sigfillset*** инициализирует набор, на который указывает ***set***, включив в него все сигналы. Приложения должны вызывать ***sigemptyset*** или ***sigfillset*** хотя бы один раз для каждой переменной типа ***sigset_t***.

Процедуры ***sigaddset*** и ***sigdelset*** принимают в качестве параметров указатель на инициализированный набор сигналов и номер сигнала, который должен быть добавлен или удален. Второй параметр, ***signo***, может быть

символическим именем константы, таким как ***SIGINT***, или настоящим номером сигнала, но в последнем случае программа окажется системно-зависимой.

Порядок выполнения работы

1. Изучить теоретическую часть лабораторной работы.
2. Организовать функционирование процессов следующей структуры:



Процессы определяют свою работу выводом сообщений вида :

N pid ppid текущее время (мсек) (***N*** – текущий номер сообщения) на экран. “Отец” одновременно, посылает сигнал ***SIGUSR1*** “сыновьям”. “Сыновья” получив данный сигнал, посылают в ответ “Отцу” сигнал ***SIGUSR2***. “Отец” получив сигнал ***SIGUSR2***, через время ***t=100*** мсек одновременно, посылает сигнал ***SIGUSR1*** “сыновьям”. И так далее… Написать функции-обработчики сигналов, которые при получении сигнала выводят сообщение о получении сигнала на экран. При получении/посылке сигнала они выводят соответствующее сообщение:

N pid ppid текущее время (мсек) ***сын такой-то get/put SIGUSRm***.

Предусмотреть механизм для определения “Отцом”, от кого из “Сыновей” получен сигнал.

Варианты индивидуальных заданий

Создать дерево процессов согласно варианта индивидуального задания.

Процессы непрерывно обмениваются сигналами согласно табл. 2 . Запись в таблице 1 вида: ***1->(2,3,4,5)*** означает, что исходный процесс ***0*** создаёт дочерний процесс ***1,*** который, в свою очередь, создаёт дочерние процессы ***2,3,4,5.*** Запись в таблице 2 вида: ***1->(2,3,4) SIGUSR1*** означает, что процесс 1 посылает дочерним процессам ***2,3,4*** одновременно (т.е. за один вызов ***kill()***) сигнал ***SIGUSR1***.Каждый процесс при получении или посылке сигнала выводит на консоль информацию в следующем виде:

N pid ppid послал/получил USR1/USR2 текущее время (мксек)

где ***N***-номер сына по табл. 1

Процесс 1, после получения ***101*** –го по счету сигнала ***SIGUSR,*** посылает сыновьям сигнал ***SIGTERM*** и ожидает завершения всех сыновей, после чего завершается сам***.*** Процесс 0 ожидает завершения работы процесса 1 после чего

завершается сам. Сыновья, получив сигнал ***SIGTERM,*** завершают работу с выводом на консоль сообщения вида:

pid ppid завершил работу после X-го сигнала SIGUSR1 и Y-го сигнала SIGUSR2

где ***X, Y*** – количество посланных за все время работы данным сыном сигналов ***SIGUSR1*** и ***SIGUSR2***

Для создания правильной последовательности сигналов в соответствие с таблицей задания необходимо для каждого процесса написать свой обработчик сигналов в котором он (процесс) принимает сигнал от предыдущего (в таблице) процесса и посылает следующему (в таблице) процессу!!

Во всех заданиях должен быть контроль ошибок (если к какому-либо каталогу нет доступа, необходимо вывести соответствующее сообщение и продолжить выполнение).

Вывод сообщений об ошибках должен производиться в стандартный поток вывода сообщений об ошибках (***stderr***) в следующем виде:

имя_модуля: текст_сообщения.

Пример: pid: 1.exe***: Error open file: 1.txt***

Имя модуля, имя файла берутся из аргументов командной строки.

Варианты индивидуальных заданий в табл.1, табл.2.

Дерево процессов

Таблица 1

№	Дерево процессов
1	1->2 2->(3,4) 4->5 3->6 6->7 7->8
2	1->(2,3,4) 2->(5,6) 6->7 7->8
3	1->(2,3,4,5) 2->6 3->7 4->8
4	1->(2,3) 2->(4,5) 5->6 6->(7,8)
5	1->(2,3,4,5) 5->(6,7,8)
6	1->(2,3) 3->4 4->(5,6,7) 7->8
7	1->2 2->(3,4) 4->5 3->6 6->7 7->8
8	1->(2,3,4,5,6) 6->(7,8)
9	1->2 2->(3,4,5) 4->6 3->7 5->8
10	1->2 2->3 3->(4,5,6) 6->7 4->8
11	1->(2,3) 3->4 4->(5,6) 6-7 7->8
12	1->2 2->(3,4) 4->5 3->6 6->7 7->8
13	1->(2,3,4,5) 5->(6,7) 7->8

14	**1->2 2->(3,4,5) 4->6 3->7 5->8**
15	**1->2 2->3 3->(4,5,6) 6->7 4->8**
16	**1->(2,3,4,5) 2->(6,7) 7->8**

Последовательность обмена сигналами

Таблица 2

№	Последовательность обмена сигналами
1	***1->2 SIGUSR1 2->(3,4) SIGUSR2 4->5 SIGUSR1 3->6 SIGUSR1 6->7 SIGUSR1 7->8 SIGUSR2 8->1 SIGUSR2***
2	***1->(2,3,4) SIGUSR1 2->(5,6) SIGUSR2 6->7 SIGUSR1 7->8 SIGUSR1 8->1 SIGUSR2***
3	***1->(2,3,4,5) SIGUSR2 2->6 SIGUSR1 3->7 SIGUSR1 4->8 SIGUSR1 8->1 SIGUSR1***
4	***1->(2,3) SIGUSR1 2->(4,5) SIGUSR1 5->6 SIGUSR1 6->(7,8) SIGUSR1 8->1 SIGUSR1***
5	***1->(2,3,4,5) SIGUSR1 5->(6,7,8) SIGUSR1 8->1 SIGUSR1***
6	***1->(2,3) SIGUSR1 3->4 SIGUSR2 4->(5,6,7) SIGUSR1 7->8 SIGUSR1 8->1 SIGUSR2***
7	***1->2 SIGUSR1 2->(3,4) SIGUSR2 4->5 SIGUSR1 3->6 SIGUSR1 6->7 SIGUSR1 7->8 SIGUSR1 8->1 SIGUSR1***
8	***1->(2,3,4,5,6) SIGUSR2 6->(7,8) SIGUSR1 8->1 SIGUSR2***
9	***1->2 SIGUSR2 2->(3,4,5) SIGUSR1 4->6 SIGUSR1 3->7 SIGUSR1 5->8 SIGUSR1 8->1 SIGUSR2***
10	**1->(8,7,6) *SIGUSR1* 8->4 *SIGUSR1* 7->4*SIGUSR2* 6->4 *SIGUSR1* 4->(3,2) *SIGUSR1* 2->1 *SIGUSR2***
11	**1->(8,7) *SIGUSR1* 8->(6,5) *SIGUSR1* 5->(4,3,2) *SIGUSR2* 2->1 *SIGUSR2***
12	**1->(8,7,6,5) *SIGUSR1* 8->3 *SIGUSR1* 7->3 *SIGUSR2* 6->3 *SIGUSR1* 5->3 *SIGUSR1* 3->2*SIGUSR2* 2->1 *SIGUSR2***
13	**1->6 *SIGUSR1* 6->7 *SIGUSR1* 7->(4,5) *SIGUSR2* 4->8 *SIGUSR1* 5->2 *SIGUSR1* 8->2 *SIGUSR2* 2->1 *SIGUSR2***
14	**1->8 *SIGUSR1* 8->7 *SIGUSR1* 7->(4,5,6) *SIGUSR2* 4->2 *SIGUSR1* 2->3 *SIGUSR1* 3->1 *SIGUSR2***