Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №5

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Студенты: Суслов А.В.

Эльвейн Н.Л.  
 Олимпиади М.Б.

Преподаватели: Хайленко Е.А.  
 Филиппова Е.В.

Вариант: 1

1. **Цель работы**

Практическое освоение механизма синхронизации процессов и их

взаимодействия посредством программных каналов.

1. **Задание**

Вариант №1:

Исходный процесс создает два программных канала К1 и К2 и порождает новый процесс Р1, а тот, в свою очередь, еще один процесс Р2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс Р1 помещает в канал К1, а процесс Р2 в канал К2, откуда они процессом Р1 копируются в канал К1 и дополняются новой порцией данных.

Обработка данных основным процессом заключается в чтении информации из программного канала К1 и печати её. Кроме того, посредством выдачи сообщений необходимо информировать обо всех этапах работы программы (создание процесса, завершение посылки данных в канал и т.д.).

1. **Описание использованных системных вызовов**
   1. **Функции работы с каналами**

**int pipe(int \_\_pipedes[2])**

Возвращает два дескриптора файла: один для записи данных в канал, другой – для чтения. Операции передачи данных выполняются с помощью системных вызовов read и write.

**ssize\_t read(int \_\_fd, void \*\_\_buf, size\_t \_\_n)**

Выполняет чтение из файла по дескриптору \_\_fd не более \_\_n байт в память \_\_buf. Возвращает количество фактически прочитанных байт. Тип ssize\_t эквивалентен short.

**ssize\_t write(int \_\_fd, const void \*\_\_buf, size\_t \_\_n)**

Записывает в файл по дескриптору \_\_fd не более \_\_n байт из памяти \_\_buf. Возвращает количество фактически записанных байт.

**int close(int \_\_fd)**

Закрывает дескриптор файла. Фактически это может быть как файл, так и канал.

* 1. **Функции работы с сигналами**

**int sigemptyset(sigset\_t \*\_\_set)**

Очищает набор сигналов (sigset\_t). Наборы сигналов используются для их массовой обработки.

**int sigaddset(sigset\_t \*\_\_set, int \_\_signo)**

Добавляет сигнал в набор по идентификатору сигнала.

**int sigprocmask(int \_\_how, sigset\_t \*\_\_set, sigset\_t \*\_\_oset)**

Применяет действие \_\_how к набору сигналов. Варианты: блокировать (класть в очередь до разблокировки или принудительной обработки), игнорировать.

**int kill(pid\_t \_\_pid, int \_\_sig)**

Отправляет сигнал \_\_sig процессу с указанным pid, если pid положителен.

* 1. **Функции работы с процессами**

**pid\_t getpid()**

Возвращает идентификатор текущего процесса. Тип pid\_t эквивалентен int.

**pid\_t wait(void \*\_\_stat\_loc)**

Ожидает поступления сигнала SIGCHLD от дочернего процесса о его завершении. Возвращает pid завершившегося процесса, а в аргумент по ссылке кладётся код, с которым потомок завершился. Если на момент вызова дочерних процессов нет, возвращает -1 и 0 соответственно.

1. **Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <sys/wait.h>

#pragma region Global vars

#define RD 0 // Read end of pipe

#define WR 1 // Write end of pipe

#define DATA\_LEN 40 // Data length by task

#define FORK\_ERROR 1 // Exit status for failed fork()

#define DATA\_ERROR 2 // Exit status for failed write()

#pragma endregion

typedef unsigned char byte; // this is missing in standard libs for whatever reason.

struct data\_t { // struct for data by task

pid\_t pid;

char data[DATA\_LEN];

data\_t() { // default constructor

pid = 0;

memset(data, 0, DATA\_LEN); // zero memory

}

data\_t(pid\_t \_pid, const char \*\_data) { // constructor with data filling

pid = \_pid;

strncpy(data, \_data, DATA\_LEN - 1); // memcpy for DATA\_LEN is unsafe

data[DATA\_LEN] = 0; // add string-terminator manually

}

};

void send\_message(const int \_fd, const char \*msg, pid\_t pid = 0) {

if(pid == 0) // if no sender is present

pid = getpid(); // get current process ID

data\_t \*data = new data\_t(pid, msg); // how to replace it to C style?

ssize\_t c = write(\_fd, data, sizeof(data\_t)); // write data to pipe

if(c == -1) { // write() returns -1 in case of failure

fprintf(stderr, "[!] Data from %d couldn't be written to %d\n", pid, \_fd);

exit(DATA\_ERROR); // finish current process

}

free(data); // C style

}

int main(int argc, char \*\*argv) {

// here we go, P0 process

// piping by http://stackoverflow.com/a/26384903/1543625

int K1[2]; // Child-to-parent pipe K1

pipe(K1); // create one-way K1 pipe from P0 to P1

// create K2

int K2[2];

pipe(K2); // create one-way K2 pipe from P1 to P2

// common usage vars

data\_t buf;

ssize\_t c;

pid\_t wait\_pid;

int wait\_ret;

// signals by http://citforum.ru/programming/unix/signals/

int sig;

sigset\_t sigset;

sigemptyset(&sigset); // clear set of signals

sigaddset(&sigset, SIGCONT); // add SIGCONT to set

sigprocmask(SIG\_BLOCK, &sigset, 0); // block all signals in set to all processes

pid\_t pid\_p1; // Process ID of child

switch(pid\_p1 = fork()) { // do fork, pid\_p1 = new pid in parent, and = 0 in child

case -1: // error

fprintf(stderr, "[!] Could not fork to P1\n");

exit(FORK\_ERROR);

break;

case 0: { // child P1

close(K1[RD]); // this is recommended

fprintf(stderr, "[i] Process P1 created\n");

send\_message(K1[WR], "Hey, what can I do for you?"); // send custom message

fprintf(stderr, "[i] Data from P1 sent to K1\n");

pid\_t pid\_p2; // identifier of child P2 process

switch(pid\_p2 = fork()) { // do fork, pid\_p2 = new pid in parent, = 0 in chld

case -1: // error in P1

fprintf(stderr, "[!] Could not fork to P2\n");

exit(FORK\_ERROR);

break;

case 0:

close(K2[RD]); // this is recommended

fprintf(stderr, "[i] Process P2 created\n");

while(sigwait(&sigset, &sig) < 0 && sig != SIGCONT); // wait 4 signal

fprintf(stderr, "[i] SIGCONT received in P2\n");

send\_message(K2[WR], "I'm alive!"); // send custom message

fprintf(stderr, "[i] Data from P2 sent to K2\n");

kill(pid\_p1, SIGCONT); // send signal to parent process

fprintf(stderr, "[i] Process P2 exiting\n");

exit(EXIT\_SUCCESS); // exit normally

break;

};

// P1 code

close(K2[WR]); // this is recommended

kill(pid\_p2, SIGCONT); // send signal to continue to child process

fprintf(stderr, "[i] SIGCONT sent to P2\n");

wait\_pid = wait(&wait\_ret); // wait till P2 die

if(wait\_ret != EXIT\_SUCCESS) { // if child process exited with error

fprintf(stderr, "[x] Leaving P1 with %d code\n", wait\_ret);

exit(wait\_ret); // exit self with the same error

}

c = read(K2[RD], &buf, sizeof(data\_t)); // read 1 message from K2

c = write(K1[WR], &buf, sizeof(data\_t)); // resend it to K1

fprintf(stderr, "[i] Data from K2 re-sent to K1\n");

send\_message(K1[WR], "I've done my best, good bye!"); // send custom message

fprintf(stderr, "[i] Data from P1 sent to K1\n");

fprintf(stderr, "[i] Process P1 exiting\n");

exit(EXIT\_SUCCESS); // exit normally

}

break;

};

// parent (P0) code

close(K1[WR]); // this is recommended

wait\_pid = wait(&wait\_ret); // wait till P1 die

if(wait\_ret != EXIT\_SUCCESS) {// if child process exited with error

fprintf(stderr, "[x] Leaving P0 with %d code\n", wait\_ret);

exit(wait\_ret); // exit self with the same error

}

fprintf(stdout, "Messages queued in K1:\n");

while((c = read(K1[RD], &buf, sizeof(data\_t))) > 0) { // read each queued message

fprintf(stdout, "Message from PID = %d: %s\n", buf.pid, buf.data);

}

return 0;

}

1. **Результат работы программы**

**[striker@centos7x64 Debug]$** ./lab5\_rm\_self

[i] Process P1 created

[i] Data from P1 sent to K1

[i] SIGCONT sent to P2

[i] Process P2 created

[i] SIGCONT received in P2

[i] Data from P2 sent to K2

[i] Process P2 exiting

[i] Data from K2 re-sent to K1

[i] Data from P1 sent to K1

[i] Process P1 exiting

Messages queued in K1:

Message from PID = 7453: Hey, what can I do for you?

Message from PID = 7454: I'm alive!

Message from PID = 7453: I've done my best, good bye!

**[striker@centos7x64 Debug]$** ./lab5\_rm\_self 2> /dev/null

Messages queued in K1:

Message from PID = 7409: Hey, what can I do for you?

Message from PID = 7410: I'm alive!

Message from PID = 7409: I've done my best, good bye!