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

Выполнили: Махнёва Е. А.

Соловьёв Б. С

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

Вариант: 9

1. **Цель работы**

Практическое освоение механизма синхронизации процессов и их

взаимодействия посредством программных каналов.

1. **Задание**

Вариант №9:

Исходный процесс создает два программных канала К1 и К2 и порождает два процесса Р1 и Р2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс Р2 помещает в канал К1, затем они оттуда читаются процессом Р1, переписываются в канал К2, дополняются своими данными. Обработка данных основным процессом заключается в чтении информации из программного канала К2 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>

#define RD 0

#define WR 1

#define BUFSIZE 512

typedef struct {

int pid;

char msg[40];

} msg\_t;

const char \*msg1 = "Surprise!";

const char \*msg2 = " Where's my space?";

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

msg\_t M;

int k1[2], k2[2];

printf("Process MAIN has started.\n");

pipe(k1);

printf("MAIN:\tpipe K1 was created.\n");

pipe(k2);

printf("MAIN:\tpipe K2 was created.\n");

int p1, p2;

switch (p1 = fork()) {

case -1: {/\*error\*/} break;

case 0: {//child

printf("P1:\tprocess P1 has started.\n");

while (read(k1[RD], &M, sizeof(M)) <= 0);

printf("P1:\tprocess P2 has finished.\n");

printf("P1:\tmessage was received.\n");

strncat(M.msg, msg2, sizeof(M.msg));

M.pid = getpid();

write(k2[WR], &M, sizeof(M));

printf("P1:\tmessage was resended.\n");

exit(EXIT\_SUCCESS);

} break;

default: {/\*parent\*/} break;

}

switch(p2 = fork())

{

case -1: {/\*error\*/} break;

case 0: { //child

printf("P2:\tprocess P2 has started.\n");

strncpy(M.msg, msg1, sizeof(M.msg));

M.pid = getpid();

write(k1[WR], &M, sizeof(M));

printf("P2:\tmessage was sended.\n");

exit(EXIT\_SUCCESS);

} break;

default: {/\*parent\*/} break;

}

while (read(k2[RD], &M, sizeof(M)) <= 0);

printf("MAIN:\tprocess P1 has finished.\n");

printf("MAIN:\tReceived message [pid=%d]:\n%s\n", M.pid, M.msg);

return 0;

}

1. **Результат работы программы**

Process MAIN has started.

MAIN: pipe K1 was created.

MAIN: pipe K2 was created.

P1: process P1 has started.

P2: process P2 has started.

P2: message was sended.

P1: process P2 has finished.

P1: message was received.

P1: message was resended.

MAIN: process P1 has finished.

MAIN: Received message [pid=7978]:

Surprise! Where's my space?