****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М80-207Б-18

Студент: Ильминский Никита Сергеевич

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2019.

***Постановка задачи***Составить программу на языке C, осуществляющую работу с процессами и взаимодействие между ними: программа должна определять, в какой координатной четверти лежит введенная пользователем точка – родительский процесс должен считывать две координаты точки и передавать их через канал дочернему процессу; дочерний процесс должен определять четверть и далее передавать результат родительскому процессу.

***Алгоритм***

Для выполнения данной задачи необходимо создать канал передачи данных, дочерний процесс, который с помощью канала будет принимать точку, введенную из родительского процесса, вычислять ответ и передавать обратно через канал родителю. Родитель должен ждать записи от родительского процесса, считывать из канала ответ и выводить его. Также должна происходить обработка возможных ошибок.

***Описание программы***

Программа состоит из одного файла main.c. В этом файле подключаются необходимые библиотеки для работы с системными вызовами; далее объявляются файловые дескрипторы int pipe1fd[2], int pipe2fd[2] и происходит создание каналов pipe(pipe1fd) для записи и pipe(pipe2fd) для чтения и одновременная проверка на ошибки при выполнении вызова; после этого, используя fork(), родительский процесс создает дочерний процесс, и при помощи идентификатора процесса производится

работа

-с родительским процессом (при pid > 0): вводятся координаты, передаются по каналу, производится блокирование до записи от дочернего процесса и затем получение ответа из канала;

-c дочерним процессом (при pid == 0): из канала считываются координаты и вычисляется координатная четверть, номер передается обратно родительскому процессу через канал;

-с неудачным вызовом fork() (при pid == -1): происходит вывод ошибки и выход из программы.

***Листинг***

*main.c*

/\*Вариант 23: родительский процесс считывает две координаты, передает

их через канал дочернему процессу. Дочерний процесс определяет,

к какокй четверти относится точка, а далее передает результат

родительскому процессу.\*/

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

int point[2] = {0};

char quarter;

int pipe1fd[2];

int pipe2fd[2];

if(pipe(pipe1fd) == -1) {

perror("Unable to create a pipe1!");

exit(1);

}

if(pipe(pipe2fd) == -1) {

perror("Unable to create a pipe2!");

exit(1);

}

pid\_t pid = fork();

if(pid > 0) {

printf("Enter the coordinates: ");

close(pipe1fd[0]);

close(pipe2fd[1]);

while(scanf("%d %d", &point[0], &point[1])) {

if(write(pipe1fd[1], point, sizeof(int) \* 2) == -1) {

perror("Parent process failed to write to pipe!");

}

printf("parent process waiting...\n");

if(read(pipe2fd[0], &quarter, 1) == -1) {

perror("Parent process failed to read from pipe!");

}

printf("The answer is: %c\n", quarter);

printf("\nEnter the coordinates: ");

}

close(pipe1fd[1]);

close(pipe2fd[0]);

} else if(pid == 0) {

close(pipe1fd[1]);

close(pipe2fd[0]);

while(read(pipe1fd[0], point, sizeof(int) \* 2)) {

if(point[0] == 0 && point[1] == 0) {

printf("The point is the origin! Assigning quarter to 0...\n");

quarter = '0';

}

if(point[0] == 0 && point[1] != 0) {

printf("The point lays on the Oy axis! Assigning quarter to 0...\n");

quarter = '0';

}

if(point[0] != 0 && point[1] == 0) {

printf("The point lays on the Ox axis! Assigning quarter to 0...\n");

quarter = '0';

}

if(point[0] > 0 && point[1] > 0) {

quarter = '1';

}

if(point[0] < 0 && point[1] > 0) {

quarter = '2';

}

if(point[0] < 0 && point[1] < 0) {

quarter = '3';

}

if(point[0] > 0 && point[1] < 0) {

quarter = '4';

}

printf("child process writing...\n");

if(write(pipe2fd[1], &quarter, 1) == -1) {

perror("Child procces failed to write to pipe!");

}

}

close(pipe1fd[0]);

close(pipe2fd[1]);

} else if(pid == -1) {

perror("Unable to fork!");

exit(1);

}

return 0;

}

***Демонстрация работы программы***

root@DESKTOP-5B8SU4Q:~/labs/3sem/os# gcc main.c

root@DESKTOP-5B8SU4Q:~/labs/3sem/os# ./a.out

Enter the coordinates: 2 4

parent process waiting...

child process writing...

The answer is: 1

Enter the coordinates: 0 0

parent process waiting...

The point is the origin! Assigning quarter to 0...

child process writing...

The answer is: 0

Enter the coordinates: -0 -0

parent process waiting...

The point is the origin! Assigning quarter to 0...

child process writing...

The answer is: 0

Enter the coordinates: 124 0

parent process waiting...

The point lays on the Ox axis! Assigning quarter to 0...

child process writing...

The answer is: 0

Enter the coordinates: -467823 -2827

parent process waiting...

child process writing...

The answer is: 3

Enter the coordinates: 0 -2342

parent process waiting...

The point lays on the Oy axis! Assigning quarter to 0...

child process writing...

The answer is: 0

Enter the coordinates: quit

root@DESKTOP-5B8SU4Q:~/labs/3sem/os#

***Вывод***

В результате выполнения данной лабораторной работы я получил первый опыт написания программ для ОС, использующих системные вызовы, такие как fork(), pipe(), write(), read(), waitpid(), close(). Принципы работы данных системных вызовов не вызывают больших трудностей при понимании и написании программы:

* при вызове fork() из родительского процесса происходит создание дочернего процесса, при этом происходит копия родительского процесса в дочерний (копируется текст программы, данные, стек, таблица дескрипторов, текущая позиция указателя записи-чтения; при этом используется механизм copy-on-write - при чтении области данных используется общая копия, в случае изменения данных — создается новая копия); процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork(), и благодаря этому можно написать отдельный код для каждого процесса.
* при помощи pipe() осуществляется взаимодействие между процессами, при этом создается канал передачи данных, использующий два файловых дескриптора для записи и чтения данных при помощи write() и read(); pipe также применяется для синхронизации (предотвращения “гонок” между процессами), при этом используется механизм закрытия дескрипторов чтения/записи при помощи close().
* с помощью waitpid() (wait() для одного доч. процесса) родительский процесс может приостанавливать свою работу до завершения работы дочерних процессов, что также используется для синхронизации.

Эти знания понадобятся мне при дальнейшей работе с ОС.