ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 4 по курсу

по курсу «Операционные системы»

Группа: М8о-207Б-18

Студент:

Токарев Никита Станиславович

Преподаватель:

Миронов Евгений Сергеевич

Оценка:

Дата:

Москва,2019

Оглавление

1.Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант № 14:

Родительский процесс считывает числа со стандартного входного ввода. Дочерний процесс вычисляет квадратный корень этих чисел и передает результаты на печать родительскому процессу.

2.Структура программы

В данной работе в моя программа состоит из двух файлов:

1. main.c

2. Makefile

3.Описание программы

В данной программе Создается временный файл благодаря системному вызову mkstemp(char *template): Функция mkstemp() создает временный файл с именем, соответствующим строке из template. Последние шесть символов template должны быть XXXXXX, и именно они заменяются строкой, которая делает имя файла уникальным. Возвращает mkstemp -1, при ошибке. Также в данной работе все процессы завершал я функцией unlink(),пока все процессы связанные с закрываемым файлом не завершатся , файл будет доступен.

Затем с помощью функции mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset) — я отражаю данный файл. Затем с помощью именованных семафоров я организовываю взаимодействие дочернего и родительского процесса. В родительском процессе я блокирую один семафор и открываю другой. В дочернем процессе я делаю тоже самое, но противоположно. Вообщем работа происходит таким образом: sem_wait(sem); - приостановление выполнения функции main() до тех пор, пока не будет вызвана функция sem_post(sem);. В моей программе именованный семафор output отвечает за родительский процесс, а qet_sqrt отвечает за дочерний процесс.

Также после отображения файла у меня есть адрес для полей временного файла куда я записываю данные. Так как файл представляет из собой массив из указателей типа char*, то в разные ячейки я записываю результат вычислений и сами входные данные.

4.Листинг программы

//main.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <errno.h>
#include <math.h>
#include <stdbool.h>
#define mmap_error 1
#define fork_error  2
```

```
#define semaphores_error 3
#define mkstemp_failed 4
#define write_failed 5
#define munmap_failed 6


int get_file()
{
	char *t_f = strdup("/tmp/t_f.XXXXXX");//временная папка tmp
	int fd = mkstemp(t_f);//создает временный файл
	if(fd == -1) {
		perror("mkstemp_failed");
		exit(mkstemp_failed);
	}
	unlink(t_f);
	free(t_f);
	if(write(fd, "_gg_", 4) < 0) {
		perror("write_failed");
		exit(write_failed);
	}
	return fd;
}


int main(void) {
   int fd = get_file();
	unsigned char* memory = (unsigned char*)mmap(NULL, 4, PROT_WRITE | PROT_READ, MAP_SHARED, fd, 0);//разделен с другими процессами
	if (memory == MAP_FAILED) {
		perror("mmap_failed");
```

```
            exit(mmap_error);
        }
        sem_t* get_sqrt = sem_open("get_sqrt", O_CREAT, 0777, 0);
        sem_t* output = sem_open("output", O_CREAT, 0777, 0);
        if (get_sqrt == SEM_FAILED || output == SEM_FAILED) {
            perror("Semaphores doesn't create");
            exit(semaphores_error);
        }
        sem_unlink("get_sqrt");//реальное удаление семафора не будет
```
осуществлено до тех пор, пока он не будет окончательно закрыт.
```
        sem_unlink("output");
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork_failed");
        exit(fork_error);
    } else if (pid > 0) {
        double begin_str,result;
        while(scanf("%lf", &begin_str) > 0) {
            double* set = (double*)((unsigned char *)(memory) + 0);
            *set = begin_str;
            sem_post(get_sqrt);
                    sem_wait(output);//ожидание доступа для output
            double* get = (double*)((unsigned char *)(memory) + 1);
            result = *get;
            printf("%.3lf^(1/2) = %.3lf\n",begin_str, result);

            }
            sem_close(get_sqrt);
            sem_close(output);
            close(fd);
```

```
    } else {//child prc
        double new_str;
        while (true) {
                        sem_wait(get_sqrt);//ожидание доступа для get_sqrt
            double* get = (double*)((unsigned char *)(memory) + 0);
            new_str = *get;
            new_str = pow(new_str,0.5);
            double *set = (double*)((unsigned char *)(memory) + 1);
            *set = new_str;
            sem_post(output);//разблокировка output
                }
                sem_close(get_sqrt);
                sem_close(output);
                close(fd);
    }
        if(munmap(memory, 4) == -1) {
                perror("munmap_failed");
                exit(munmap_failed);
        }
    return 0;
}
//Makefile
CC = gcc

CFLAGS = -g  -Wextra  -pedantic -Werror

OBJ = main.o

main: main.o
        $(CC) $(CCFLAGS) -o main main.o -lpthread -lm
```

```
main.o: main.c


clean:
	@rm -r *.o main
```

5.Результат работы

```
nikita@nikita-HP:~/oc/lr4v14$ ./main
23
23.000^(1/2) = 4.796
123
123.000^(1/2) = 11.091
s
nikita@nikita-HP:~/oc/lr4v14$ ./main
5s
5.000^(1/2) = 2.236
nikita@nikita-HP:~/oc/lr4v14$ ./main
4 s
4.000^(1/2) = 2.000
nikita@nikita-HP:~/oc/lr4v14$ ./main
121
121.000^(1/2) = 11.000
45
45.000^(1/2) = 6.708
```

6.Вывод

Отображение файла в память (на память) - это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти (диапазон адресов оперативной памяти). При этом чтение данных из этих адресов фактически приводит к чтению данных из отображенного файла, а запись данных по этим адресам приводит к записи этих данных в файл. Также данный файл может использоваться несколькими процессами, для записи или чтения. Ключевым является системный вызов mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset) с помощью которого можно отобразить файл с необходимыми параметрами.