Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4 по курсу**

**«Операционные системы»**

Студент: Румынина Екатерина Александровна

Группа: М8О-201Б-21

Вариант: 7

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/KetRum0/mai_os_labs>

**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 аналогичен лабораторной работе 2.

**Общие сведения о программе**

main.c - основная программа, которая считывает ввод и перенаправляет его в родительский процесс

parent.c – программа для реализации родительского процесса

child.c – программа для реализации дочернего процесса

parent.h – заголовочный файл с объявлением ParentRoutine

**Общий метод и алгоритм решения**

Аналогичен лабораторной работе 2, но вместо передачи через pipe используется передача через отображаемый файл и семафор для синхронизации процессов.

**Исходный код**

**main.c**

#include "parent.h"

#include <stdio.h>

#include <stdlib.h>

int main(void) {

char name[256];

scanf("%s",name);

float result;

result = ParentRoutine(name);

printf("%g\n",result);

return 0;

}

**parent.h**

#ifndef OS\_LABS\_PARENT\_H

#define OS\_LABS\_PARENT\_H

#include <stdio.h>

float ParentRoutine(char \*filename);

#endif //OS\_LABS\_PARENT\_H

**parent.c**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <fcntl.h>

#include <semaphore.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

float ParentRoutine(char \*filename){

// char name[256];

// scanf("%s",name);

int file;

if ((file = open(filename, O\_RDONLY))==-1){

perror("open error");

exit(EXIT\_FAILURE);

}

sem\_t \*sem = sem\_open("my\_sem", O\_CREAT,S\_IRUSR | S\_IWUSR, 0);

if (sem == SEM\_FAILED){

perror("sem\_open error");

exit(EXIT\_FAILURE);

}

int id = fork();

if (id == -1){

perror("fork error");

exit(EXIT\_FAILURE);

} else if (id == 0){ //child

dup2(file,STDIN\_FILENO);

close(file);

if (execlp("./child", "./child", NULL) == -1){

perror("execlp error");

exit(EXIT\_FAILURE);

}

} else { //parent

int fd = shm\_open("shared\_file", O\_RDWR | O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH);

if (fd == -1){

perror("shm\_open error (parent)");

exit(EXIT\_FAILURE);

}

sem\_wait(sem);

sem\_close(sem);

struct stat s;

fstat(fd, &s);

int map\_size = s.st\_size;

char \*mapped = (char \*) mmap(NULL, map\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap error (parent)");

exit(EXIT\_FAILURE);

}

close(fd);

float result = atof(mapped);

if (munmap(mapped,map\_size) == -1){

perror("munmap error (parent)");

exit(EXIT\_FAILURE);

}

if (shm\_unlink("shared\_file") == -1){

perror("shm\_unlink error");

exit(EXIT\_FAILURE);

}

return result;

}

}

**child.c**

#include <unistd.h>

#include <stdio.h>

#include <semaphore.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

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

sem\_t \*sem = sem\_open("my\_sem", 1);

if (sem == SEM\_FAILED){

perror("sem\_open error (child)");

exit(EXIT\_FAILURE);

}

float res = 0;

float x;

while (scanf("%f",&x)!=EOF){

res+=x;

}

int maxDigits = 256;

int digits = 0;

char res\_arr[maxDigits];

gcvt(res, maxDigits, res\_arr);

// int flag = 0;

// for (int i = 0; i < maxDigits; i++) {

// if (res\_arr[i]=='.') flag=1;

// if (res\_arr[i]=='0' & flag==1){

// break;

// }

// digits++;

// }

int map\_size = maxDigits;

int fd = shm\_open("shared\_file", O\_RDWR, 0777);

if (ftruncate(fd, map\_size) == -1){

perror("ftruncate (child)");

exit(EXIT\_FAILURE);

}

if (fd == -1){

perror("shm\_open error (child)");

exit(EXIT\_FAILURE);

}

char \*mapped = (char \*)mmap(NULL, map\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

close(fd);

if (mapped == MAP\_FAILED){

perror("mmap error (child)");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < map\_size; i++) {

mapped[i] = res\_arr[i];

}

if (munmap(mapped,map\_size) == -1){

perror("munmap error (child)");

exit(EXIT\_FAILURE);

}

sem\_post(sem);

sem\_close(sem);

}

**Демонстрация работы программы**

ket@ket-laptop:~/Desktop/mai\_os\_labs/lab4$ cat test.txt

1 2 3 4 -1 0.1 0.11

ket@ket-laptop:~/Desktop/mai\_os\_labs/lab4$ ./lab4

test.txt

9.21

**Выводы**

В результате выполнения данной лабораторной работы я освоила принципы работы с файловыми системами и научилась обеспечивать обмен данных между процессами посредством технологии «File mapping»