Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Межпроцессорное взаимодействие через memory-mapped files**

Студент: Никулин Кристиан Ильич

Группа: М8О–208Б–21

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

## Задание

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 3

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа представляет из себя один файл main.c

**Общий метод и алгоритм решения**.

С помощью вызова fork создаются родительский и дочерний процессы, родительский процесс считывает название будущего файла и строку целых чисел, которые передаются в дочерний процесс. Дочерний процесс создаёт файл и записывает в него деление первого числа из этой строки на все последующие. Данные передаются между процессами с помощью mmap. Ключей для запуска программа не имеет.

**Основные файлы программы**

**main.c:**

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/uio.h>

#include <sys/mman.h>

#define BUF\_SIZE 128

int main() {

printf("Print output file name:\n");

// WRITE FILE NAME TO MAP

char \* file = mmap(NULL, sizeof(char) \* BUF\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, 0, 0);

fgets(file, BUF\_SIZE, stdin);

if (file == MAP\_FAILED) {

perror("mmap error");

exit(1);

}

printf("Enter the numbers:\n");

// WRITE NUMBERS TO MAP

char \* arr = mmap(NULL, sizeof(char) \* BUF\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, 0, 0);

fgets(arr, BUF\_SIZE, stdin);

if (arr == MAP\_FAILED) {

perror("mmap error");

exit(2);

}

// CHILD

int child = fork();

if (child == -1) {

perror("Error during creating fork\n");

exit(3);

} else if (child == 0) {

int f = open(file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0777);

if (f == -1) {

perror("Error during creating file\n");

exit(4);

}

int neg = 0;

// FIRST NUMBER

char \* result = malloc(sizeof(char) \* BUF\_SIZE);

int len = 0;

char c = arr[len];

if (c == '-')

{

neg = 1;

}

while (c != ' ') {

result[len] = c;

len++;

c = arr[len];

}

len++;

result[len] = '\0';

c = arr[len];

int res = atoi(result);

if (neg)

{

res \*= -1;

}

// NUMBERS

int index = 0;

while ((c >= 48 && c <= 57) || c == ' ' || c == '-') {

char \* number = malloc(sizeof(char) \* BUF\_SIZE);

index = 0;

while (c != ' ') {

number[index] = c;

index++;

len++;

c = arr[len];

}

len++;

c = arr[len];

index++;

number[index] = '\0';

int num = atoi(number);

if (num < 0)

{

num \*= -1;

if (neg == 1)

{

neg = 0;

}

else

{

neg = 1;

}

}

if (num == 0) {

printf("Zero found\n");

break;

}

res /= num;

char buffer[30];

int len = sprintf(buffer, "%d", res);

if (neg)

{

write(f, "-", 1);

}

write(f, buffer, len);

write(f, "\n", 1);

free(number);

}

if (munmap(arr, BUF\_SIZE) == -1)

{

perror("UNmap error");

exit(5);

}

if (munmap(file, BUF\_SIZE) == -1)

{

perror("UNmap error");

exit(6);

}

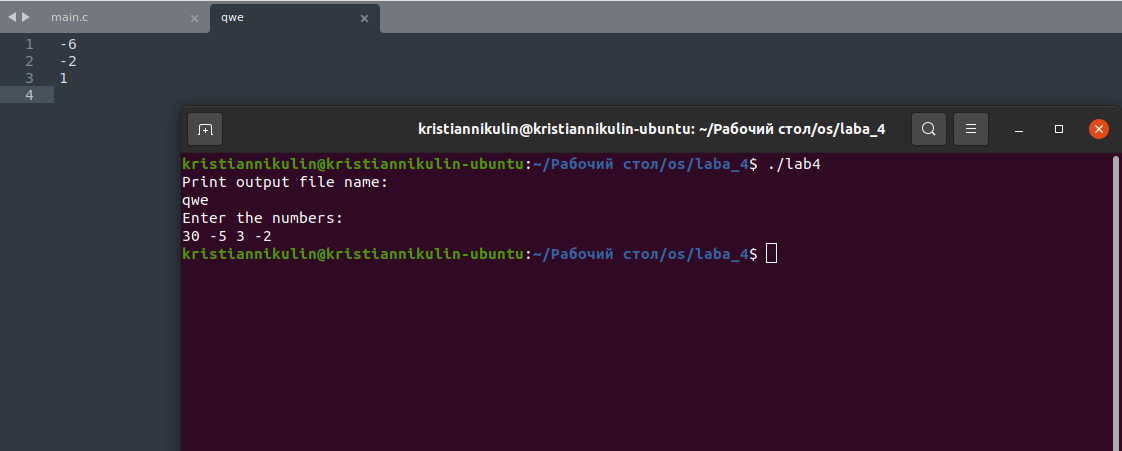
return 0;

}

return 0;

}

**Пример работы**



**Вывод**

Проделав лабораторную работу, я расширил свои навыки в работе с процессами в ОС Unix и освоил технологию обмена данных между процессами с помощью файл маппинга.