Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Салихов Тимур Русланович

Группа: М8О–212Б–22

Вариант: 1

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

1 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерниий. Дочерний̆ процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Помимо этого, есть программа calculator.cpp, которая запускается при помощи exec. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **exec** - позволяет запустить исполняемый файл изнутри программы.
3. **open** - открытие файла
4. **сlose** - закрытие файла
5. **mmap -** создать маппинг файла
6. **munmap -** удалить маппинг файла
7. **sem\_open -** открыть семафор
8. **sem\_close -** закрыть семафор

**Общий метод и алгоритм решения**.

1. Программа main открывает файл для передачи данных и делает маппинг
2. Вводится семафор, чтобы заблокировать дочерний процесс
3. Происходит fork, после чего идет разделение родительского и дочернего процесса
4. Родительский процесс читает данные из входного потока и записывает их в memory map.
5. После окончания записи семафор разблокируется и родительский процесс ожидает завершения дочернего.
6. Дочерний процесс запускает calculator при помощи exec
7. Дочерний процесс открывает и маппирует общий файл
8. Как только семафор разблокирован, процесс читает данные из мапа и производит вычисления
9. Вывод записывается в выходной файл

**Основные файлы программы**

**main.cpp:**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <unistd.h>

auto main() -> int {

int memoryd;

memoryd = open("memory.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0666);

ftruncate(memoryd, 1024);

char\* buffer = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, memoryd, 0);

close(memoryd);

sem\_t\* sem = sem\_open("mmap\_sem", O\_CREAT, 0777, 0); // Open semaphore

if (sem == SEM\_FAILED) {

perror("Could not open semaphore");

return -1;

}

// Forking the process

int id = fork();

if (id == -1) { // fork error

return 2;

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

execl("./calculator", "./calculator", "mmap\_sem", NULL);

return 3;

} else { // parent process

char c;

c = getchar();

size\_t i = 0;

while (c != EOF) {

buffer[i++] = c;

c = getchar();

}

buffer[i] = c;

sem\_post(sem); // Unlock semaphore

sem\_close(sem);

munmap(buffer, 1024);

int status;

waitpid(0, &status, 0); // waiting for child process to finish

if (status != 0)

perror("Child process exited with an error");

return status;

}

}

**calculator.cpp:**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

auto main(int argc, char\*\* argv) -> int {

FILE\* f = fopen("output.txt", "w");

fprintf(f, "The sums are: ");

int memoryd;

memoryd = open("memory.txt", O\_RDWR, 0666);

char\* buffer = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, memoryd, 0);

close(memoryd);

sem\_t\* sem = sem\_open(argv[1], O\_CREAT, 0777, 0); // Open semaphore from argument list

if (sem == SEM\_FAILED) {

perror("Could not open semaphore");

return -1;

}

int num = 0, sum = 0;

size\_t i = 0;

sem\_wait(sem); // Wait for semaphore to unlock and then lock it

while (buffer[i] != -1) {

if (buffer[i] != ' ' && buffer[i] != '\n') {

num \*= 10;

num += buffer[i] - '0';

} else if (buffer[i] == ' ') {

sum += num;

num = 0;

} else if (buffer[i] == '\n') {

if (num != 0)

sum += num;

fprintf(f, "%d ", sum);

sum = 0;

num = 0;

}

++i;

}

sem\_close(sem);

munmap(buffer, 1024);

return 0;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(Memory\_Map)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(CMAKE\_CXX\_STANDARD 14)

add\_executable(main main.cpp)

add\_executable(calculator calculator.cpp)

**Пример работы**

**1) Терминал:**

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)> /Users/timursalihov/OperatingSystems/LW3/build/src/main

1 2 3

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)>

**Файл:**

The sums are: 6

**2) Терминал:**

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)> /Users/timursalihov/OperatingSystems/LW3/build/src/main

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)>

**Файл:**

The sums are: 0

**3) Терминал:**

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)> /Users/timursalihov/OperatingSystems/LW3/build/src/main

1 2 3

4 5

19 20

timursalihov@MBP-Timur-2 ~/O/L/b/src (main)>

**Файл:**

The sums are: 6 9 39

**Вывод**

В процессе выполнения работы я познакомился с таким инструментом, как memory map. Memory map создает виртуальную копию файла, что позволяет при работе с фалами не использовать дорогие операции чтения из файла и записи в файл. При использовании memory map ответственность за соответствие файла и его мапа переносится на операционную систему. В процессе выполнения работы я столкнулся с определенными трудностями. Первоначально я не использовал семафор, поэтому пока происходил ввод данных, дочерний процесс уже пытался их прочитать, что приводило к ошибке. Также после добавления семафора, я добавил лишнее увеличение счетчика, из-за чего семафор работал только при первом запуске программы.