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: Головенко Анатолий Валерьевич

Группа: М8О–212Б–22

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**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 вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

6 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. 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
3. Происходит fork, после чего идет разделение родительского и дочернего процесса
4. Далее запускается через exec файл child.cpp, который делает маппинг общего файла, для передачи информации и открывает файл, из которого будет считаться информация
5. Так же вводится в child семафор, для того чтобы родительский процесс ждал исполнение дочернего
6. Происходит чтение информации из файла и её обработка
7. Получив ответ, число записывается в файл-буффер, файл закрывается, семафор открывается
8. Дальше родительский файл свыводит ответ из memory\_map и программа завершается

**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>

#include <string>

#include <iostream>

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

{

// считываем и открываем файл

int common\_memory = open("com\_mem.txt", O\_RDWR | O\_CREAT, 0666);

if (common\_memory == -1)

{

return -1;

}

// берём из файла первые 1024 байта

ftruncate(common\_memory, 1024);

// создаём memory\_map - общую память процессов

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

// NULL - выбранное ядро(не выбрано)

// 1024 байта - размер

// PROT\_READ | PROT\_WRITE - доступ к чтению и записи

// MAP\_SHARED - используется всеми процессами

// memory\_map создался на основе file и начинает чтение с 0 байта (смещение = 0)

close(common\_memory);

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

if (sem == SEM\_FAILED)

{

perror("Could not open semaphore");

return -1;

}

// Forking the process

int id = fork();

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

{

return 2;

}

if (id == 0)

{ // child process

execlp("./child", "./child", "mmap\_sem", NULL);

return 3;

}

else

{ // parent process

sem\_wait(sem);

std::cout << buffer << std::endl;

munmap(buffer, 1024);

sem\_close(sem);

sem\_unlink("mmap\_sem");

}

return 0;

}

**child.cpp:**

#include <fcntl.h>

#include <semaphore.h>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

#include <string>

#include <iostream>

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

{

FILE \*file = fopen("output.txt", "r");

int common\_memory = open("com\_mem.txt", O\_RDWR | O\_CREAT, 0666);

if (common\_memory == -1)

{

return -1;

}

// берём из файла первые 1024 байта

ftruncate(common\_memory, 1024);

// создаём memory\_map - общую память процессов

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

// NULL - выбранное ядро(не выбрано)

// 1024 байта - размер

// PROT\_READ | PROT\_WRITE - доступ к чтению и записи

// MAP\_SHARED - используется всеми процессами

// memory\_map создался на основе file и начинает чтение с 0 байта (смещение = 0)

close(common\_memory);

sem\_t \*sem = sem\_open(argv[1], O\_CREAT, 0777, 0); // открываем семафор

if (sem == SEM\_FAILED)

{

perror("Could not open semaphore");

return -1;

}

int num = 0, sum = 0, i = 0;

char symb;

while ((symb = getc(file)) != '\n')

{

if (symb != ' ')

{

num \*= 10;

num += symb - '0';

}

else if (symb == ' ')

{

sum += num;

num = 0;

}

++i;

}

fclose(file);

sum += num;

num = 0;

i = 0;

std::string k = std::to\_string(sum);

for (char elem : k)

{

buffer[i++] = elem;

}

buffer[i] = '\0';

sem\_post(sem);

sem\_close(sem);

munmap(buffer, 1024);

return 0;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(Parallel\_processes)

set(CMAKE\_CXX\_STANDARD 20)

add\_executable(parent parent.cpp)

add\_executable(child child.cpp)

**Пример работы**

1. **Файл:**

123243 8762 1267

**Терминал:**

133272

**2) Файл:**

**Терминал:**

**3) Файл:**

765 98712 8913 6732 6 867 987 16732 9

**Терминал:**

133723

**4)** **Файл:**

1 2

**Терминал:**

3

**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.