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: Епифанов Евгений Валерьевич

Группа: М8О–212Б–22

Вариант: 4

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 при необходимости 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.

4 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление parent child(args: fileName) pipe1 pipe2 In/out User File In Out 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 float. Количество чисел может быть произвольным.

**Общие сведения о программе**

Основной файл программы - main.cpp. Также имеется файл child.cpp, который запускается при помощи exec. Для сборки файлов используется система CMake. В программе используются следующие системные вызовы:

1. **fork** – создает дочерний процесс.
2. **exec** – позволяет запустить исполняемый файл внутри программы.
3. **open** – открывает файл.
4. **сlose** – закрывает файл.
5. **mmap –** создает карту памяти.
6. **munmap –** удаляет карту памяти.
7. **sem\_open –** открывает семафор.
8. **sem\_close –** закрывает семафор.
9. **sem\_post –** разблокирует семафор.
10. **sem\_wait –** блокирует семафор.

**Общий метод и алгоритм решения**.

1. В файле main.cpp открывается файл для передачи данных и происходит создание карты памяти.
2. Создается семафор.
3. Fork формирует дочерний процесс.
4. Файл child.cpp запускается с помощью exec.
5. Родительский процесс читает данные из входного потока и записывает их в buffer – карту памяти.
6. Дочерний процесс взаимодействует с картой памяти.
7. После окончания записи семафор разблокируется, родительский процесс будет ждать завершения дочернего процесса.
8. Как только семафор разблокируется, дочерний процесс считает данные из карты памяти и произведет вычисления.
9. Итог вычислений записывается в выходной файл.

**Основные файлы программы**

**main.cpp:**

#include <unistd.h> // for fork() and pipe

#include <string>

#include <semaphore.h>

#include <sys/mman.h>

int main()

{

int mfile = open("memory\_file.txt", O\_RDWR | O\_CREAT, 0666); // file descriptor has been created

if (mfile == -1)

{

perror("Couldn't open the file!");

return -1;

}

ftruncate(mfile, 1024); // file reduction to 1024 bytes

char\* buffer = (char\*)mmap(NULL, 1024, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mfile, 0); // creating a memory map

if (buffer == MAP\_FAILED)

{

perror("Failed to create the buffer!");

return -1;

}

close(mfile); // closing a file

sem\_t\* sem = sem\_open("semaf", O\_CREAT, 0777, 0); // creating a semaphore

if (sem == SEM\_FAILED)

{

perror("Couldn't open the semaphore!");

return -1;

}

int id = fork(); // create a process

if (id == -1) // error

{

return -1;

}

else if (id == 0) // child

{

execlp("./child", "./child", "semaf", NULL); // launching a file for execution

return 0;

}

else // parent (id > 0)

{

char c;

c = getchar();

size\_t ind = 0;

while (c != EOF)

{

buffer[ind++] = c;

c = getchar();

}

buffer[ind] = c;

sem\_post(sem); // unlocking the semaphore

sem\_close(sem); // closing the semaphore

munmap(buffer, 1024); // cleaning a memory map

int status;

waitpid(0, &status, 0); // awaiting process status change

if (status != 0) return -1;

}

return 0;

}

**child.cpp:**

#include <unistd.h> // for read() and write()

#include <string>

#include <semaphore.h>

#include <sys/mman.h>

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

{

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

fprintf(f, "The answer is: ");

int mfile = open("memory\_file.txt", O\_RDWR, 0666);

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

close(mfile);

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

if (sem == SEM\_FAILED)

{

perror("Couldn't open the semaphore!");

return -1;

}

size\_t i = 0;

float divisible = 0;

float number = 0;

std::string snumber = "";

bool notempty = 0;

sem\_wait(sem); // blocking the semaphore

while(buffer[i] != -1)

{

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

{

snumber = snumber + buffer[i];

}

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

{

number = std::stof(snumber);

if (number == 0)

{

fprintf(f, "div\_by\_zero");

exit(-1);

}

divisible /= number;

fprintf(f, "%f ", divisible);

divisible = 0;

number = 0;

snumber = "";

}

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

{

number = std::stof(snumber);

if (number == 0)

{

fprintf(f, "div\_by\_zero");

exit(-1);

}

if (divisible == 0)

divisible = number;

else

divisible /= number;

number = 0;

snumber = "";

}

++i;

}

sem\_close(sem);

munmap(buffer, 1024);

return 0;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(Processes\_with\_semaphore)

set (CMAKE\_CXX\_STANDARD 20)

add\_executable(main main.cpp)

add\_executable(child child.cpp)

**Пример работы**

**1) Терминал:**

1. evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main
2. 80 2 10 4
3. evgeniy@EP-iMac build %

**Файл:**

1. The answer is: 1.000000

**2) Терминал:**

1. evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main
2. 89 0 45.8
3. evgeniy@EP-iMac build %

**Файл:**

1. The answer is: div\_by\_zero

**3) Терминал:**

1. evgeniy@EP-iMac build % /Users/evgeniy/Desktop/OS\_Labs/Lab1/src/build/main
2. 67.87 43.2 9.1
3. evgeniy@EP-iMac build %

**Файл:**

1. The answer is: 0.172644

**Вывод**

В процессе выполнения лабораторной работы я получил навыки работы 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.