Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Останина Анна Андреевна

Группа: М8О-208Б-22

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий
2. Цель работы
3. Задание
4. Описание работы программы
5. Исходный код
6. Консоль
7. Запуск тестов
8. Выводы

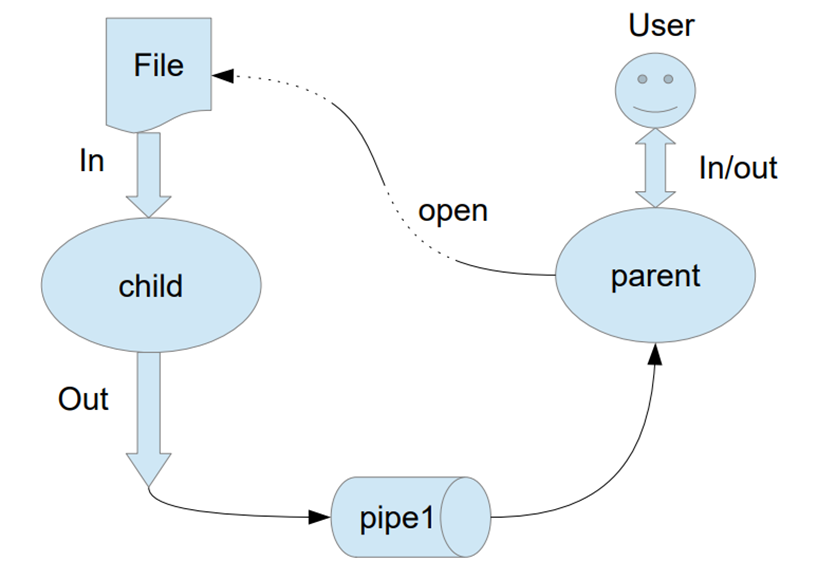
### Репозиторий

<https://github.com/Imariiii/os_labs>

### Цель работы

### Освоение принципов работы с файловыми системами и 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).

### 9 вариант) В файле записаны команды вида:«число число число». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.

### Описание работы программы

В программе работы были использованы следующие системные вызовы:

* sem\_unlink() - удаление именного семафора
* shm\_unlink() - удаление именованного сегмента разделяемой памяти
* shm\_open() - создание или открытие объекта разделяемой памяти
* ftruncate() - установка размера разделяемой памяти
* mmap() - отображение файлов в адресное пространство процесса
* sem\_open() - создание или открытие именованных семафоров
* fork() - создание нового процесса
* sem\_post() - увеличение значения(разблокировка) семафора
* sem\_wait() - уменьшение значения(блокировка) семафора
* sem\_close() - закрытие именованного семафора
* munmap() - отключение отображения объекта в адресное пространство процесса

**Исходный код**

lab3.h

#**pragma once**

#include <cstdio>

#include <iostream>

#define FILTER\_LEN (11U)

#define MODE (0644U)

**int** ParentRoutine(**const** **char**\* pathToChild, std::istream& streamIn, std::ostream& streamOut);

lab3.cpp

#include "lab3.h"

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <unistd.h>

#include <cmath>

#include <cstdlib>

#include <iostream>

#include "shared\_memory.h"

#include "utils.h"

**int** ParentRoutine(**const** **char**\* pathToChild, std::istream& streamIn,

std::ostream& streamOut) {

std::string filename;

streamIn >> filename;

**int** file = open(filename.c\_str(), O\_RDONLY);

**if** (file == -1) {

perror("open");

exit(EXIT\_FAILURE);

}

SharedMemory<Slot> shm(SHARED\_MEMORY\_NAME);

pid\_t pid = fork();

**if** (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

**if** (pid > 0) { // parent

close(file);

**while** (shm.readLock() && !shm.getData()->isEmpty) {

**float** res = shm.getData()->num;

**if** (std::isinf(res)) {

streamOut << "Division by zero\n";

**return** -1;

}

streamOut << res << '\n';

shm.writeUnlock();

}

wait(**nullptr**);

} **else** { // child

dup2(file, STDIN\_FILENO);

close(file);

execl(pathToChild, "child", NULL);

perror("exec");

exit(EXIT\_FAILURE);

}

**return** 0;

}

shared\_memory.h

#**pragma once**

#include <fcntl.h>

#include <semaphore.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <string>

#include <string\_view>

**template** <**class** T>

**class** WeakSharedMemory {

**public**:

**enum** {

C\_READ\_ONLY = O\_RDONLY,

C\_READ\_WRITE = O\_RDWR,

} CREATE\_MODE;

**enum** {

A\_READ = PROT\_READ,

A\_WRITE = PROT\_WRITE,

} ATTACH\_MODE;

WeakSharedMemory() : \_size{0} {}

WeakSharedMemory(std::string\_view name)

: \_name(name), \_size(**sizeof**(T)) {

construct(\_name, \_size);

}

~WeakSharedMemory();

WeakSharedMemory(**const** WeakSharedMemory<T>& other) = **delete**;

WeakSharedMemory(WeakSharedMemory<T>&& other) noexcept;

WeakSharedMemory<T>& **operator**=(WeakSharedMemory<T>&& other) noexcept;

**bool** writeLock(**bool** timed = **false**);

**void** writeUnlock();

**bool** readLock(**bool** timed = **false**);

**void** readUnlock();

**int** getFD() **const** { **return** \_FD; }

T\* getData() { **return** **static\_cast**<T\*>(\_ptr); };

**const** T\* getData() **const** { **return** **static\_cast**<**const** T\*>(\_ptr); }

**const** std::string& getName() **const** { **return** \_name; }

**bool** isEmpty() **const** { **return** \_size == 0; }

**private**:

std::string \_name;

**int** \_FD;

sem\_t\* \_wSemPtr;

sem\_t\* \_rSemPtr;

size\_t \_size;

**void**\* \_ptr;

**void** construct(std::string& name, std::size\_t size);

WeakSharedMemory<T>& **operator**=(**const** WeakSharedMemory<T>& other) = **default**;

};

**template** <**class** T>

**class** SharedMemory : **public** WeakSharedMemory<T> {

**public**:

**using** WeakSharedMemory<T>::WeakSharedMemory;

~SharedMemory();

};

shared\_memory.cpp

#include "shared\_memory.h"

#include <sys/mman.h>

#include <unistd.h>

#include <cerrno>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <iostream>

#include "utils.h"

**template** <**class** T>

**void** WeakSharedMemory<T>::construct(std::string& name, std::size\_t size) {

\_wSemPtr = sem\_open((name + "W").c\_str(), O\_CREAT, S\_IRUSR | S\_IWUSR, 1);

\_rSemPtr = sem\_open((name + "R").c\_str(), O\_CREAT, S\_IRUSR | S\_IWUSR, 0);

\_FD = shm\_open(name.c\_str(), O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR);

**if** (\_FD < 0) {

**switch** (errno) {

**case** EACCES:

**throw** std::runtime\_error("Permission Exception");

**break**;

**case** EINVAL:

**throw** std::runtime\_error("Invalid shared memory name passed");

**break**;

**case** EMFILE:

**throw** std::runtime\_error(

"The process already has the maximum number of files "

"open");

**break**;

**case** ENAMETOOLONG:

**throw** std::runtime\_error("The length of name exceeds PATH\_MAX");

**break**;

**case** ENFILE:

**throw** std::runtime\_error(

"The limit on the total number of files open on the "

"system "

"has been reached");

**break**;

**default**:

**throw** std::runtime\_error(

"Invalid exception occurred in shared memory creation");

**break**;

}

}

ftruncate(\_FD, size);

\_ptr = mmap(**nullptr**, size, O\_RDWR, MAP\_SHARED, \_FD, 0);

**if** (\_ptr == **nullptr**) {

**throw** std::runtime\_error(

"Exception in attaching the shared memory region");

}

}

**static** **bool** SemTimedWait(sem\_t\* sem) {

**struct** timespec absoluteTime;

**if** (clock\_gettime(CLOCK\_REALTIME, &absoluteTime) == -1) {

**return** **false**;

}

absoluteTime.tv\_sec += 5;

**return** sem\_timedwait(sem, &absoluteTime) == 0;

}

**template** <**class** T>

**bool** WeakSharedMemory<T>::writeLock(**bool** timed) {

**if** (timed) **return** SemTimedWait(\_wSemPtr);

sem\_wait(\_wSemPtr);

**return** **true**;

}

**template** <**class** T>

**void** WeakSharedMemory<T>::writeUnlock() {

sem\_post(\_wSemPtr);

}

**template** <**class** T>

**bool** WeakSharedMemory<T>::readLock(**bool** timed) {

**if** (timed) **return** SemTimedWait(\_rSemPtr);

sem\_wait(\_rSemPtr);

**return** **true**;

}

**template** <**class** T>

**void** WeakSharedMemory<T>::readUnlock() {

sem\_post(\_rSemPtr);

}

**template** <**class** T>

WeakSharedMemory<T>::~WeakSharedMemory() {

**if** (sem\_close(\_rSemPtr) < 0) {

std::perror("sem\_close");

std::abort();

}

**if** (sem\_close(\_wSemPtr) < 0) {

std::perror("sem\_close");

std::abort();

}

**if** (munmap(\_ptr, \_size) < 0) {

std::perror("munmap");

std::abort();

}

}

**template** <**class** T>

SharedMemory<T>::~SharedMemory() {

**if** (**this**->isEmpty()) {

**return**;

}

**if** (shm\_unlink(**this**->getName().c\_str()) < 0) {

std::perror("shm\_unlink");

std::abort();

}

**if** (sem\_unlink((**this**->getName() + "W").c\_str()) < 0) {

std::perror("sem\_unlink");

std::abort();

}

**if** (sem\_unlink((**this**->getName() + "R").c\_str()) < 0) {

std::perror("sem\_unlink");

std::abort();

}

}

**template** <**class** T>

WeakSharedMemory<T>::WeakSharedMemory(WeakSharedMemory<T>&& other) noexcept {

other = \***this**;

\_size = 0;

}

**template** <**class** T>

WeakSharedMemory<T>& WeakSharedMemory<T>::**operator**=(

WeakSharedMemory<T>&& other) noexcept {

other = \***this**;

\_size = 0;

**return** \***this**;

}

**template** **class** WeakSharedMemory<Slot>;

**template** **class** SharedMemory<Slot>;

child.cpp

#include <unistd.h>

#include <iostream>

#include <sstream>

#include <string>

#include "shared\_memory.h"

#include "utils.h"

**int** main() {

std::string line;

WeakSharedMemory<Slot> shm(SHARED\_MEMORY\_NAME);

**while** (std::getline(std::cin, line)) {

std::istringstream iss(line);

**float** dividend;

**if** (!(iss >> dividend)) {

dividend = 0;

}

**float** divisor;

**while** (iss >> divisor) {

dividend /= divisor;

**if** (divisor == 0) {

shm.writeLock();

shm.getData()->isEmpty = **false**;

shm.getData()->num = dividend;

shm.readUnlock();

**return** -1;

}

}

shm.writeLock();

shm.getData()->isEmpty = **false**;

shm.getData()->num = dividend;

shm.readUnlock();

}

shm.writeLock();

shm.getData()->isEmpty = **true**;

shm.readUnlock();

**return** 0;

}

utils.h

#**pragma once**

#include <semaphore.h>

#include <cstddef>

#include <stdexcept>

**inline** constexpr **int** MAP\_SIZE = **sizeof**(**bool**) + **sizeof**(**float**);

**inline** constexpr **const** **char**\* SHARED\_MEMORY\_NAME = "shared\_memory";

**struct** Slot {

**bool** isEmpty;

**float** num;

};

main.cpp

#include <cstdlib>

#include <iostream>

#include "lab3.h"

**int** main() {

**const** **char** \*childPath = getenv("PATH\_TO\_CHILD");

**if** (!childPath) {

std::cerr << "Set environment variable PATH\_TO\_CHILD\n";

std::exit(EXIT\_FAILURE);

}

**return** ParentRoutine(childPath, std::cin, std::cout);

}

lab3\_test.cpp

#include "lab3.h"

#include <gtest/gtest.h>

#include <array>

#include <filesystem>

#include <fstream>

#include <iostream>

#include <memory>

**namespace** fs = std::filesystem;

TEST(FirstLabTests, SimpleTest) {

**const** **char** \*childPath = getenv("PATH\_TO\_CHILD");

ASSERT\_TRUE(childPath);

**const** std::string filenameInput = "in.txt";

**const** std::string filenameTest = "test.txt";

**const** std::string filenameOutput = "out.txt";

constexpr **int** testSize = 5;

constexpr **int** outputSize = testSize;

**const** std::array<std::string, outputSize> testLines = {

"10 1",

"100 10",

"",

"1 9 9",

"100 3 4 6",

};

**const** std::array<std::string, outputSize> expectedOutputLines = {

"10",

"10",

"0",

"0.0123457",

"1.38889",

};

{

**auto** input = std::ofstream(filenameInput);

input << filenameTest << '\n';

**auto** test = std::ofstream(filenameTest);

**for** (**const** **auto** &line : testLines) {

test << line << '\n';

}

}

{

**auto** input = std::ifstream(filenameInput);

**auto** output = std::ofstream(filenameOutput);

ParentRoutine(childPath, input, output);

}

**auto** output = std::ifstream(filenameOutput);

std::string result;

**for** (**const** std::string &line : expectedOutputLines) {

std::getline(output, result);

EXPECT\_EQ(result, line);

}

**auto** removeIfExists = [](**const** std::string &path) {

**if** (fs::exists(path)) {

fs::remove(path);

}

};

removeIfExists(filenameInput);

removeIfExists(filenameTest);

removeIfExists(filenameOutput);

}

Консоль

anna@anna-**virtual**-machine:~/labs\_3sem/os\_labs/build/lab3$ PATH\_TO\_CHILD=../lab3/child\_lab3 ./lab3

lab3.txt

25

2

11

1

3

Division by zero

### Запуск тестов

anna@anna-**virtual**-machine:~/labs\_3sem/os\_labs/build/tests$ PATH\_TO\_CHILD=../lab3/child\_lab3 ./lab3\_test

Running main() from /home/anna/labs\_3sem/os\_labs/build/\_deps/googletest-src/googletest/src/gtest\_main.cc

[==========] Running 1 test from 1 test suite.

[----------] Global test environment set-up.

[----------] 1 test from FirstLabTests

[ RUN ] FirstLabTests.SimpleTest

[ OK ] FirstLabTests.SimpleTest (9 ms)

[----------] 1 test from FirstLabTests (9 ms total)

[----------] Global test environment tear-down

[==========] 1 test from 1 test suite ran. (9 ms total)

[ PASSED ] 1 test.

Выводы

В ходе выполнения данной лабораторной работы я изучила принципы работы с файловыми системами и технологию обмена данными между процессами с использованием файловых отображений в память. Я познакомилась с системными вызовами, такими как open, mmap, munmap, shm\_open, ftruncate, sem\_open, sem\_wait, sem\_post. Также изучила работу с семафорами для обеспечения синхронизации доступа к общим ресурсам. При этом я узнала о понятиях таблицы страниц, page hit и page fault, которые играют важную роль при работе с общими сегментами памяти.