Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Литовченко Анна Александровна

Группа: М80 – 207Б-21

Вариант: 5

Преподаватель:Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Annalitov/OS/lab4

**Постановка задачи**

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.

**Группа вариантов 1:**

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.

**Вариант 5**:

В программе child происходит проверка чисел на правило “Не отрицательное и не простое”

**Общие сведения о программе**

Программа представлена одним файлом: main.cpp

Операционная система: MacOs

**Общий метод и алгоритм решения**

**Используемые системные вызовы:**

fork(void) - создаёт новый процесс посредством копирования вызывающего

процесса. Новый процесс считается дочерним процессом. Вызывающий процесс

считается родительским процессом. Дочерний и родительский процессы находятся в

отдельных пространствах памяти. Сразу после fork() эти пространства имеют одинаковое

содержимое.

int close() - закрывает файловый дескриптор.

pid = fork(); Начиная с этого момента, процессов становится два. У каждого своя память. в

процессе-родителе pid хранит идентификатор ребёнка. в ребёнке в этой же переменной

лежит 0.

char \*mapped = (char \*)mmap(0, mapsize, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

- вызов mmap позволяет отобразить открытый файл в адресное пространство

процесса. Первый аргумент является нулевым указателем, при этом система сама

выбирает адрес начала отображаемого сегмента. Длина файла совпадает с размером

char. Устанавливается доступ на чтение и запись. Четвертый аргумент имеет

значение MAP\_SHARED, что позволяет процессам «видеть» изменения, вносимые

друг другом. Функция возвращает адрес начала участка разделяемой памяти.

Одним из способов добиться совместного использования памяти родительским и

дочерним процессами является вызов mmap с флагом MAP\_SHARED перед вызовом fork.

В этом случае все отображения памяти, установленные родительским процессом, будут

унаследованы дочерним. Более того, изменения в содержимом объекта, вносимые

родительским процессом, будут видны дочернему, и наоборот.

**Исходный код**

#include <iostream>

#include <string>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <unistd.h>

#include <fstream>

#include <errno.h>

#include <sys/mman.h>

#include <cstdio>

using namespace std;

int flag\_ = 0;

int isNotPrime(int n)

{

if (n < 2) {

return 0;

} else {

for (int i = 2; i \* i < n + 1; i++) {

if (n % i == 0) {

return 1;

}

}

}

return 0;

}

int child(string filename, char \*mapped, string sem\_file) {

int count = 1;

fstream file\_1;

file\_1.open(filename, fstream::in | fstream::out | fstream::app);

sem\_t \*semaphore = sem\_open(sem\_file.c\_str(), 1);

while (true) {

if (sem\_wait(semaphore) == -1) {

perror("Semaphore error");

exit(EXIT\_FAILURE);

}

if (mapped[count] == '!') {

break;

}

int str\_size = (int)mapped[count];

int start = count;

char mas[str\_size];

int i = 0;

for(; count < start + str\_size; count++) {

mas[i] = mapped[count + 1];

i += 1;

}

string result;

for(int i = 0; i < str\_size; i++) {

result.push\_back(mas[i]);

}

int number;

int number\_1;

number = stoi(result);

number\_1 = abs(number);

if ( number > 0 && isNotPrime(number\_1) > 0 ) {

file\_1 << result << endl;

file\_1 << endl;

cout << "A number " << result << " is added to file!" << endl;

} else {

mapped[0] = 1;

}

sem\_post(semaphore);

count++;

}

return 0;

}

int main ()

{

string filename;

int flag;

int strings\_size;

string sem\_file = "a.semaphore";

cout << "Enter a filename: ";

cin >> filename;

cout << endl;

cout<<"Enter number of operations: ";

int amount;

cin >> amount;

cout << endl;

const int mapsize = amount\*256;

int flaccess = S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH;

sem\_t \*semaphore = sem\_open(sem\_file.c\_str(), O\_CREAT, flaccess, 0);

if (semaphore == SEM\_FAILED) {

perror("Semaphore error");

exit(EXIT\_FAILURE);

}

char \*mapped = (char \*)mmap(0, mapsize, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

pid\_t id = fork();

if (id == -1){

perror("fork");

cout << "1";

exit(EXIT\_FAILURE);

}

else if (id == 0) {

child(filename, mapped, sem\_file);

return 0;

}

if (id != 0) {

string string\_r;

int start = 1;

mapped[0] = 0;

for (int i = 0; i < amount + 1; ++i) {

if (i == amount) {

mapped[start] = '!';

if (mapped[0] == 1) {

cout << "The number is prime or negative" << endl;

mapped[0] = 0;

}

sem\_post(semaphore);

break;

}

cin >> string\_r;

for (int j = 0; j < string\_r.size() + 1; j++){

if (j == 0) {

mapped[start] = (char)string\_r.size();

continue;

}

mapped[start + j] = string\_r[j - 1];

}

sem\_post(semaphore);

sem\_wait(semaphore);

start += string\_r.size() + 1;

if (mapped[0] == 1) {

cout << "The number is prime or negative" << endl;

mapped[0] = 0;

}

}

}

munmap(mapped, mapsize);

sem\_close(semaphore);

sem\_unlink(sem\_file.c\_str());

return 0;

}

**Демонстрация работы программы**

litann@Annalit lab4 % ./main

Enter a filename: 23.txt

Enter number of operations: 2

0

The number is prime or negative

14

A number 14 is added to file!

**Вывод**

Отображение в память содержимого файла, который сначала открывается вызовом open, а

затем отображается вызовом mmap удобно тем, не приходится вызывать read,

write, поскольку все считывается в некоторую область памяти . Часто это заметно упрощает код.