Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Тема работы**

Студент: Савинова Екатерина Ильинична

Группа: М8О-207Б-21

Вариант: 15

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/savinova-kati/operating-systems/tree/main/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 4:**

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

**Вариант 15**:

Правило проверки: строка должна начинаться с заглавной буквы

**Общие сведения о программе**

Программа представлена файлом lab.cpp.

**Общий метод и алгоритм решения**

Опишу новые для себя системные вызовы:

**-sem\_open**

Функция sem\_open() создаёт новый семафор POSIX или открывает существующий семафор.

**-sem\_wait**

Функция sem\_wait() уменьшает (блокирует) семафор, на который указывает sem. Если значение семафор больше нуля, то выполняется уменьшение и функция сразу завершается. Если значение семафора равно нулю, то вызов блокируется до тех пор, пока не станет возможным выполнить уменьшение

**-sem\_post**

Функция sem\_post() увеличивает (разблокирует) семафор, на который указывает sem. Если значение семафора после этого становится больше нуля, то другой процесс или нить заблокированная в вызове sem\_wait, проснётся и заблокирует семафор.

**Алгоритм решения:**

В родительском процессе принимаем из ввода строчку пользователя, затем открываем объект общей памяти, устанавливаем ему размер текста и отображаем на него текст.

Далее создаем семафор, вызываем sem\_post и переходим в дочерний процесс для проверки верности правилу строчки.

**Исходный код**

**lab.cpp**

#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 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;

if (mas[0] >= 65 && mas[0] <= 90) {

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

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

file\_1 << mas[i];

}

file\_1 << endl;

cout << "Added string " << result << " 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 name of file ";

cin >> filename;

cout << endl;

cout<<"Enter amount of strings: ";

int amount;

cin >> amount;

cout << endl;

const int mapsize = amount\*256;

int flaccess = S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH; //права семафора

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;

//execl("./child\_", to\_string(truba[0]).c\_str(), to\_string(truba[1]).c\_str(), to\_string(truba\_2[0]).c\_str(), to\_string(truba\_2[1]).c\_str(), name.c\_str(), NULL);

}

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 string does not fit the rule" << 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); //разблакировка семафора

sem\_wait(semaphore);

if (mapped[0] == 1) {

cout << "The string does not fit the rule" << endl;

mapped[0] = 0;

}

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

}

}

munmap(mapped, mapsize);

sem\_close(semaphore);

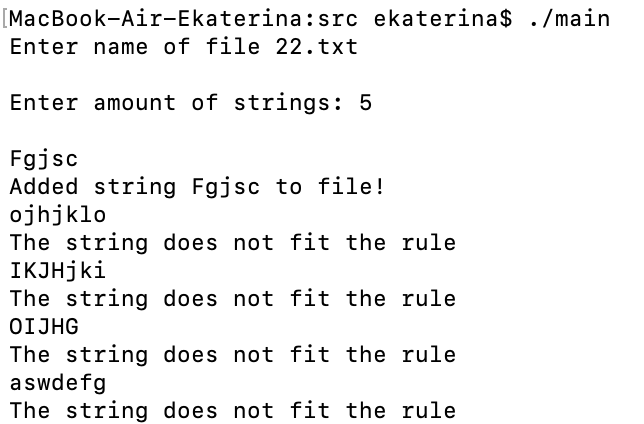
sem\_unlink(sem\_file.c\_str());

return 0;

}

**Демонстрация работы программы**

Ввод в консоль:



**Выводы**

Благодаря данной лабораторной работе, я получила больше информации о работе с отображаемой памятью и семафорами.