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: Друхольский А.К.

Группа: М8О-207Б-21

Вариант: 22

Преподаватель: Черемисинов Максим

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/ssForz/OS-labs

**Постановка задачи**

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

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).

**Общие сведения о программе**

Оба процесса представлены в одном файле lab.cpp. Компилируется при помощи cmake

* sem\_open() - инициализируем и открываем семафор
* (char \*)mmap(0, mapsize, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0) — создаём memory map для передачи информации между процессами, настраивая определёнными флагами
* munmap(mapped, mapsize) — закрытие memory map
* sem\_close(semaphore) — отключение семафора
* sem\_unlink(sem\_file) — закрываем семафор по данному имени

**Общий метод и алгоритм решения**

Главная идея в том, чтобы изначально создать mmap размера под нашу задачу. В таком случае не придётся перезаписывать конкретные участки памяти под новые строки. В начале каждого блока строки будет записывать длину строки, что нужно для упрощения алгоритмаю. Итак мы имеем mmap, в котором по порядку идут: размер-строка-размер-строка…

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

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

int counter = 0;

string filename = path;

fstream cur\_file;

cur\_file.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[counter] == '!') {

cout<<"ITS OVER"<<endl;

break;

}

int cur\_size = (int)mapped[counter];

int start = counter;

char str\_array[cur\_size];

int i = 0;

for (; counter < start + cur\_size ; counter++) {

str\_array[i] = mapped[counter+1];

i++;

}

string result\_str;

for (int j = cur\_size - 1; j >= 0; j--) {

cur\_file << str\_array[j];

}

cur\_file<<endl;

cout<<"Added result stroke to "<<filename<<endl;

counter++;

}

return 0;

}

int main()

{

string current\_str;

string sem\_file1 = "a.semaphore";

string sem\_file2 = "b.semaphore";

string filename1, filename2;

cout<<"Enter the name for first child file: ";

cin>>filename1;

cout<<endl;

cout<<"Enter the name for second child file: ";

cin>>filename2;

cout<<endl;

cout<<"Enter amount of strings: ";

int cnt;

cin>>cnt;

const int mapsize = cnt\*256;

sem\_t \*semaphore1 = sem\_open(sem\_file1.c\_str(), O\_CREAT, flaccess, 0);

if (semaphore1 == SEM\_FAILED) {

perror("Semaphore error");

exit(EXIT\_FAILURE);

}

sem\_t \*semaphore2 = sem\_open(sem\_file2.c\_str(), O\_CREAT, flaccess, 0);

if (semaphore2 == SEM\_FAILED) {

perror("Semaphore error");

exit(EXIT\_FAILURE);

}

char \*mapped1 = (char \*)mmap(0, mapsize, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

char \*mapped2 = (char \*)mmap(0, mapsize, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

pid\_t f\_id1 = fork();

if (f\_id1 == -1) {

cout<<"Fork error with code -1 returned in the parent, no child\_1 process is created"<<endl;

exit(EXIT\_FAILURE);

} else if (f\_id1 == 0) { //child1

string childfile = filename1;

child\_execute(childfile, mapped1, sem\_file1);

return 0;

}

pid\_t f\_id2 = fork();

if (f\_id2 == -1) {

cout<<"Fork error with code -1 returned in the parent, no child\_2 process is created"<<endl;

exit(EXIT\_FAILURE);

} else if (f\_id2 == 0) { //child2

string childfile = filename2;

child\_execute(childfile, mapped2, sem\_file2);

return 0;

}

if (f\_id1 != 0 && f\_id2 != 0){ //parent

cout<<endl;

int stat\_counter1 = 0;

int stat\_counter2 = 0;

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

if (i == cnt) {

mapped1[stat\_counter1] = '!';

mapped2[stat\_counter2] = '!';

sem\_post(semaphore1);

sem\_post(semaphore2);

break;

}

cin>>current\_str;

while (current\_str.size() >= 20) {

cout<<"Wrong size of string, try again (<20): "<<endl;

cin>>current\_str;

}

if (rand() % 6 < 5) {

for (int j = 0; j < current\_str.size()+1; j++) {

if (j == 0) {

mapped1[stat\_counter1] = (char)current\_str.size();

continue;

}

mapped1[stat\_counter1 + j] = current\_str[j - 1];

}

sem\_post(semaphore1);

stat\_counter1 += current\_str.size() + 1;

} else {

for (int j = 0; j < current\_str.size()+1; j++) {

if (j == 0) {

mapped2[stat\_counter2] = (char)current\_str.size();

continue;

}

mapped2[stat\_counter2 + j] = current\_str[j - 1];

}

sem\_post(semaphore2);

stat\_counter2 += current\_str.size() + 1;

}

}

}

munmap(mapped1, mapsize);

munmap(mapped2, mapsize);

sem\_close(semaphore1);

sem\_close(semaphore2);

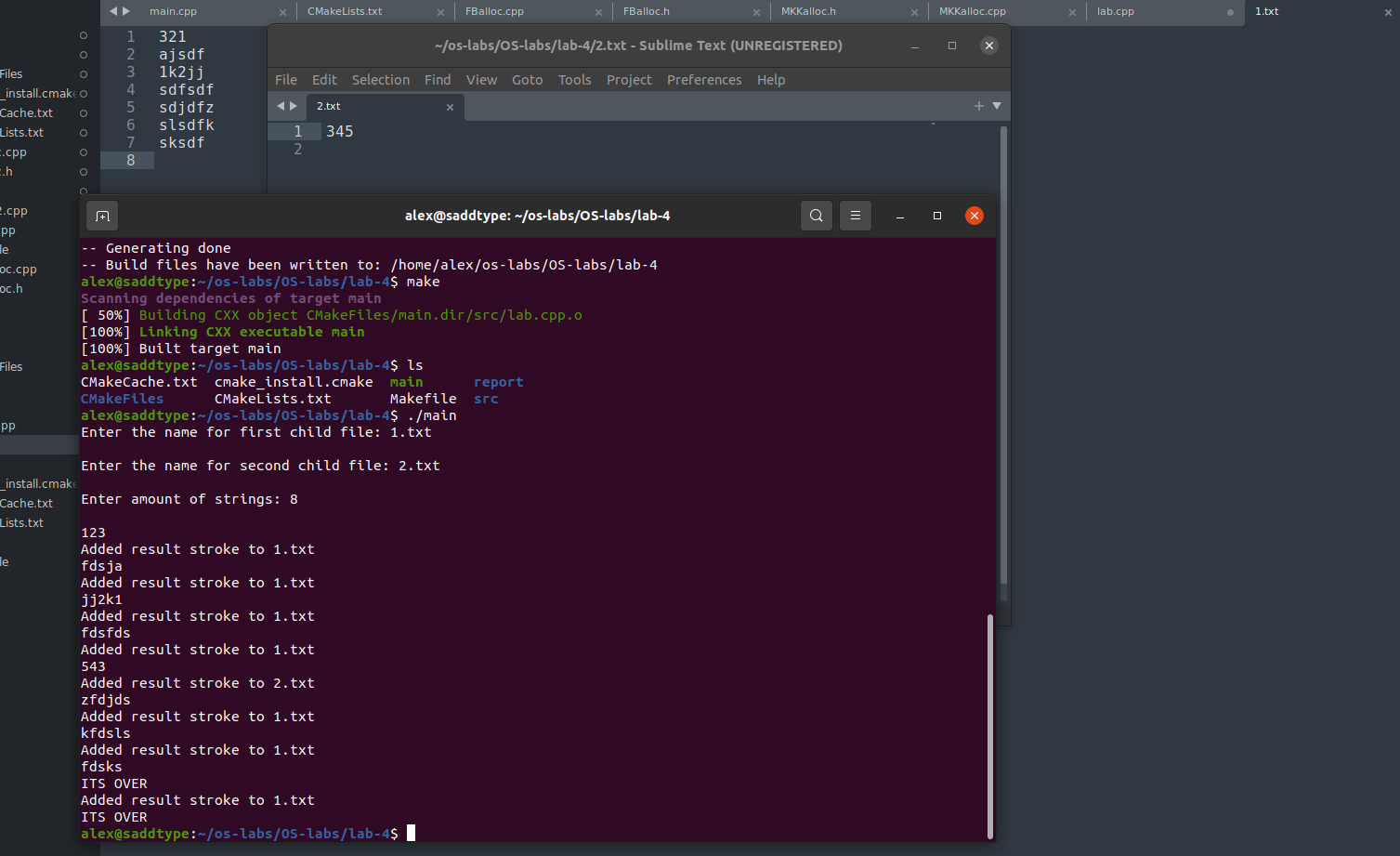
sem\_unlink(sem\_file1.c\_str());

sem\_unlink(sem\_file2.c\_str());

return 0;

}

**Демонстрация работы программы**



**Выводы**

В процессе работы я познакомился с memory map как со способом передачи информации между процессами. Я решил использовать ANON map, что позволило обойтись без file descriptor, но я столкнулся с проблемой. При передаче указателя между процессами он меняется. Это решается, если оба процесса выполняются одним исполняемым файлом.