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: Сектименко Ирина Владимировна

Группа: М8О–206Б–20

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Освоении 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.

## Общие сведения о программе

Программа компилируется из файла main.cpp. В программе используются следующие системные вызовы:

1. **mmap** – создает memory mapped file с разделяемой общей памятью на основе содержимого обычного файла.
2. **mremap** – изменяет размер memory mapped file.
3. **ftruncate** – изменяет (увеличивает) размер файла.
4. **munmap** – записывает все изменения из memory mapped file в обычный файл.

## Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы memory mapped file.
2. Переделать лабораторную работу №1 с использованием memory mapped file вместо pipe.

## Основные файлы программы

**main.cpp**

#include <iostream>

#include <stdio.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <string.h>

#include <string>

std::string reverse(std::string& s) {

std::string s1, s2;

int i = 0;

while (i < s.size()) {

while (s[i] != '\n') {

s1 = s[i] + s1;

++i;

}

s1 += "\n";

s2 = s2 + s1;

s1 = "";

++i;

}

return s2;

}

int main() {

int pageSize = getpagesize();

std::string fileName1, fileName2;

std::cin >> fileName1 >> fileName2;

const char \* name1 = fileName1.c\_str();

const char \* name2 = fileName2.c\_str();

int file1 = open(name1, O\_RDWR | O\_CREAT | O\_APPEND, S\_IRWXU | S\_IRWXG | S\_IRWXO);

struct stat size1;

fstat(file1, &size1);

if (size1.st\_size == 0) {

ftruncate(file1, 1);

fstat(file1, &size1);

}

int file2 = open(name2, O\_RDWR | O\_CREAT | O\_APPEND, S\_IRWXU | S\_IRWXG | S\_IRWXO);

struct stat size2;

fstat(file2, &size2);

if (size2.st\_size == 0) {

ftruncate(file2, 1);

fstat(file2, &size2);

}

char \* mFile1 = (char \*)mmap(NULL, size1.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file1, 0);

char \* mFile2 = (char \*)mmap(NULL, size2.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file2, 0);

pid\_t pid = fork();

if (pid == -1) {

perror("fork1");

return -1;

}

pid\_t pid2;

if (pid > 0) {

pid2 = fork();

if (pid2 == -1) {

perror("fork2");

return -1;

} else if (pid2 == 0) {

pid = 1;

}

}

if (pid == 0) {

pid2 = 1;

}

if (pid > 0 && pid2 > 0) {

std::string s, s1, s2;

int flag = 0;

while (std::cin >> s) {

if (flag == 0) {

s1 += s;

s1 += "\n";

flag = 1;

} else {

s2 += s;

s2 += "\n";

flag = 0;

}

}

mFile1 = (char \*)mremap(mFile1, size1.st\_size, s1.size(), MREMAP\_MAYMOVE);

if (mFile1 == (char \*)-1) {

perror("mmap1");

return -1;

}

size1.st\_size = s1.size();

memcpy(mFile1, s1.c\_str(), s1.size());

mFile2 = (char \*)mremap(mFile2, size2.st\_size, s2.size(), MREMAP\_MAYMOVE);

if (mFile1 == (char \*)-1) {

perror("mmap2");

return -1;

}

size2.st\_size = s2.size();

strcpy((char \*)mFile2, s2.c\_str());

ftruncate(file1, s1.size());

ftruncate(file2, s2.size());

if (munmap(mFile1, s1.size()) < 0) {

perror("close mmf1");

}

if (munmap(mFile2, s2.size()) < 0) {

perror("close mmf2");

}

} if (pid == 0) {

std::string s = mFile1;

std::string s1 = reverse(s);

memcpy(mFile1, s1.c\_str(), s1.size());

if (munmap(mFile1, s1.size()) < 0) {

perror("close mmf1");

}

} else if (pid2 == 0) {

std::string s = mFile2;

std::string s1 = reverse(s);

memcpy(mFile2, s1.c\_str(), s1.size());

if (munmap(mFile2, s1.size()) < 0) {

perror("close mmf2");

}

}

close(file1);

close(file2);

return 0;

}

## Пример работы

input

text1.txt

text2.txt

asdf

as

asd

asdfg

output (text1.txt)

fdsa

dsa

output (text2.txt)

sa

gfdsa

## Вывод

В ходе лабораторной работы я научилась работать с memory mapped files. Для меня это показалось ненамного сложнее, чем работа с pipes, пусть и механика работы немного отличалась. Самым обидным было, когда я забыла увеличить память выходных файлов и в итоге в них не записывались данным. Я долго искала ошибку, не могла понять в чем дело, но в итоге разобралась с помощью лучшего в мире способа отладки: принтов.