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: Николаева Елизавета Сергеевна

Группа: М8О-201Б-20

Вариант: 20

Преподаватель: Миронов Евгений Сергеевич

Москва, 2021

**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/esnikolaeva/OS\_4

**Постановка задачи**

Задача: реализовать программу, в которой родительский процесс создает два дочерних процесса. Родительский процесс принимает строки, которые отправляются в тот или иной дочерний процесс. Оба процесса удаляют гласные из строк. Межпроцессорное взаимодействие осуществляется посредством отображаемых файлов (memory-mapped files).

Вариант 20) Правило фильтрации: если длина строки больше 10 символов, то строка отправляется во второй дочерний процесс, в противном случае в первый дочерний процесс.

**Общие сведения о программе**

Для реализации этой задачи нам нужны следующие библиотеки:

<unistd.h> - для работы с системными вызовами в Linux.

<stdlib.h> - для того, чтобы можно было пользоваться функциями, отвечающими за работу с памятью.

<limits.h> - для определения характеристик общих типов переменных.

<sys/mman.h> - для работы с memory-mapped files.  
<pthread.h> - для работы с потоками.

<ctype.h> - для классификации и преобразования отдельных символов.

<sys/stat.h> - для доступа к файлам.

<fcntl.h> - для работы с файловым дескриптором.

<sys/wait.h> - для использования символических констант.

<fstream> - для работы с файлами С++.

<string.h> - для использования функций над строками.

<stdio.h> - для использования взаимодействия с физическими устройствами (клавиатура и т.д)  
<iostream> - использования потока ввода и вывода  
<signal.h> - для указания того, как программа обрабатывает сигналы во время ее выполнения

<sstream> - для организации работы со строками  
Данная лабораторная работа сделана на основе второй лабораторной работы, посвященной работе с процессами. Для работы с memory-mapped files, согласно заданию, помимо основы второй лабораторной работы и использования специальных библиотек, у меня в программе также есть использование следующих системных вызовов:

mmap(...) - системный вызов, позволяющий выполнить отображение файла или устройства на память. принимающий следующие аргументы: адрес памяти для размещения, текущий размер файла, права на чтение и запись, права на то, чтобы делиться данным маппингом, сам файловый дескриптор и начальную позицию (с которой пойдет считывание).

munmap(...) - системный вызов, удаляющий маппинг из адресного пространства.

ftruncate(filedesc, size\_t bites) - системный вызов, увеличивающий память файла до size\_t bites.

**Общий метод и алгоритм решения**

С самого начала выполнения программы требуется 2 названия для дочерних процессов - куда они будут писать строки без гласных.  
Далее создаются 2 файла: f1.txt и f2.txt. Это те самые файлы, куда мы посредством file-mapping будем писать файлы для потомков. Строки длиной не более 10 символов будут идти в f1.txt, иначе в f2.txt. При этом посредством системного вызова ftruncate память всегда будет увеличиваться динамически после добавления каждой строки.  
После считывания всех строк дочерние процессы принимают из map-files строки и удаляют в них гласные, выводя строки без гласных в каждый из своих файлов. После завершения работы mapped-files удаляются из памяти при помощи системного вызова munmap.

**Исходный код**

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <ctype.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <stdint.h>

#include <fstream>

#include <iostream>

#include <semaphore.h>

sem\_t semaphore1;

sem\_t semaphore2;

bool isVowel(char a) {

a = tolower(a);

if (a == 'a' || a == 'e' || a == 'i' ||

a == 'o' || a == 'u' || a == 'y') {

return true;

}

return false;

}

char\* stringToString(char\* stream) {

unsigned int maxLength = 64, size = 64;

char\* buffer = (char\*)malloc(maxLength);

if (buffer != NULL) {

char character = EOF;

int index = 0;

while ((character = \*(stream + index)) != '\n' && character != EOF) {

buffer[index] = character;

index++;

if (index == size) {

size = index + maxLength;

buffer = (char\*)realloc(buffer, size);

}

}

buffer[index] = '\0'; //терминальный нуль -для обозначения конца строки

}

return buffer;

}

int main() {

std::fstream fos1, fos2;

int fileDescriptor;

struct stat statusBuffer;

if (sem\_init(&semaphore1, 1, 1) < 0) {

std::cout << "SEM1 ERROR" << std::endl;

exit(-1);

}

if (sem\_init(&semaphore2, 1, 1) < 0) {

std::cout << "SEM2 ERROR" << std::endl;

exit(-1);

}

if ((fileDescriptor = open("test.txt", O\_RDWR)) < 0) {

std::cout << "SRC FILE OPEN ERROR" << std::endl;

exit(-1);

}

char\* source;

if (fstat(fileDescriptor, &statusBuffer) < 0) {

std::cout << "FSTAT ERROR" << std::endl;

exit(-1);

}

source = (char\*)mmap(NULL, statusBuffer.st\_size, PROT\_READ, MAP\_SHARED,

fileDescriptor, 0);

if (source == MAP\_FAILED) {

std::cout << "MAPPING ERROR" << std::endl;

exit(-1);

}

int child1, child2;

if ((child1 = fork()) == -1) {

std::cout << "FORK1 ERROR" << std::endl;

exit(-1);

}

else if (child1 == 0) {

sem\_wait(&semaphore1);

fos1.open("1.txt", std::fstream::out);

int index = 0;

char\* string;

while ((string = stringToString(source + index)) && string[0] != '\0') {

int lenght = strlen(string);

if (lenght <= 10) {

for (int i = 0; i < lenght; i++) {

if (!isVowel(string[i])) {

fos1 << string[i];

}

}

fos1 << '\n';

}

index += lenght;

if (source[index] == '\n') index++;

free(string);

}

free(string);

sem\_post(&semaphore1);

}

else {

if ((child2 = fork()) == -1) {

std::cout << "FORK2 ERROR" << std::endl;

exit(-1);

}

else if (child2 == 0) {

sem\_wait(&semaphore2);

fos2.open("2.txt", std::fstream::out);

int index = 0;

char\* string;

while ((string = stringToString(source + index)) && string[0] != '\0'){

int length = strlen(string);

if (length > 10) {

for (int i = 0; i < length; i++) {

if (!isVowel(string[i])) {

fos2 << string[i];

}

}

fos2 << '\n';

}

index += strlen(string);

if (source[index] == '\n') index++;

free(string);

}

free(string);

sem\_post(&semaphore2);

}

else {

sem\_wait(&semaphore1);

sem\_wait(&semaphore2);

close(fileDescriptor);

if (munmap(source, statusBuffer.st\_size) < 0) {

std::cout << "UNMAPPING ERROR" << std::endl;

exit(-1);

}

if (sem\_destroy(&semaphore1) < 0) {

std::cout << "SEMDEL1 ERROR" << std::endl;

exit(-1);

}

if (sem\_destroy(&semaphore2) < 0) {

std::cout << "SEMDEL2 ERROR" << std::endl;

exit(-1);

}

}

}

}

**Выводы**

Данная лабораторная работа, на мой взгляд, служит отличным дополнением ко второй. Благодаря поставленному заданию я расширила свой функционал работы с процессами и освоила принцип реализации file-mapping.