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**

**“Межпроцессорное взаимодействие через memory-mapped files”**

Студент: Фаттяхетдинов Сильвестр Динарович

Группа: М8О-208Б-20

Вариант: 17

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

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/silverfatt/OS

**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 в зависимости от следующего правила: если длина строки больше 10 символов, то строка отправляется во второй дочерний процесс, в противном случае в первый дочерний процесс. Оба процесса удаляют гласные из строк. Межпроцессорное взаимодействие осуществляется посредством отображаемых файлов (memory-mapped files).

**Общие сведения о программе**

Вся программа содержится в одном файле lab4.cpp

**Общий метод и алгоритм решения**

Ввод строк осуществляется вручную в файл с названием test.txt – это требуется для корректной работы программы.

В начале инициализируются семафоры, получается файловый дескриптор файла со строками а так же информация о нём (в дальнейшем понадобится размер файла).

Файл отображается на память и приводится к типу char\* - теперь с ним можно работать как с обыкновенным массивом.

При помощи fork() создаются два дочерних процесса, работа которых в принципе идентична, отличие одно – первый обрабатывает строки не длиннее 10 символов, второй – остальные, и первый пишет в файл 1.txt, второй – в 2.txt

Всё нижесказанное будет относиться к обоим дочерним процессам.

Открывается файл на вывод с фиксированным имя, запускается цикл while, который получает по одной строке из файла и, если она удовлетворяет условию – обрабатывает её, удаляя гласные.

В это время родительский процесс ожидает, пока дочерние процессы закончат свою работу (регулирование осуществляется при помощи семафоров). После завершения ожидания он освобождает отображённую память и завершает работу программы.

**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 \*getstr(char \*stream){

    unsigned int maxlen = 64, size = 64;

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

    if (buffer != NULL){ // Проверка на успешность выделения памяти

        char ch = EOF;

        int pos = 0;// Посимвольное чтение, при необходимости - реаллокация памяти

        while ((ch = \*(stream + pos)) != '\n' && ch != EOF){

            buffer[pos] = ch;

            //std::cout << buffer[pos];

            pos++;

            if (pos == size){ // Проверка на необходимость реаллокации памяти

                size = pos + maxlen;

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

            }

        }

        buffer[pos] = '\0'; // Добавление терминального нуля для обозначения конца строки

    }

    return buffer;

}

int main(){

    std::fstream fos1, fos2;

    int fd\_src;

    struct stat statbuf;

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

        printf("SEM1 ERROR\n");

        exit(-1);

    };

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

        printf("SEM2 ERROR\n");

        exit(-1);

    }

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

        printf("SRC FILE OPEN ERROR\n");

        exit(-1);

    }

    char \*src;

    if (fstat(fd\_src, &statbuf) < 0){

        printf("FSTAT ERROR\n");

        exit(-1);

    }

    src = (char\*)mmap(NULL, statbuf.st\_size, PROT\_READ, MAP\_SHARED, fd\_src, 0);

    if (src == MAP\_FAILED){

        printf("MAPPING ERROR\n");

        exit(-1);

    }

    int cid1,cid2;

    if ((cid1 = fork()) == -1){

        printf("FORK1 ERROR\n");

        exit(-1);

    }

    else if(cid1 == 0){//-------------------CHILD1------------------- <=10

    //std::cout << "CHILD1 ID:" << getpid();

    sem\_wait(&semaphore1);

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

    int pos = 0;

    char\* str;

    while((str = getstr(src+pos)) && str[0] != '\0'){

       int length = strlen(str);

       if(length <= 10){

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

               if(!IsVowel(str[i])){

                   fos1 << str[i];

                   //std::cout << str[i];

                   }

           }

           fos1 << '\n';

       }

       pos += length;

       if (src[pos] == '\n') pos++;

       free(str);

    }

    free(str);

    sem\_post(&semaphore1);

    }//-------------------CHILD1-------------------

    else{

        if ((cid2 = fork()) == -1){

            printf("FORK2 ERROR\n");

            exit(-1);

        } else if (cid2 == 0){//-------------------CHILD2------------------- >10

        //std::cout << "CHILD2 ID:" << getpid();

        sem\_wait(&semaphore2);

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

        int pos = 0;

        char\* str;

       while((str = getstr(src+pos)) && str[0] != '\0'){

       int length = strlen(str);

       if(length > 10){

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

               if(!IsVowel(str[i])){

                   fos2 << str[i];

                   }

           }

           fos2 << '\n';

       }

       pos += strlen(str);

       if (src[pos] == '\n') pos++;

       free(str);

    }

    free(str);

    sem\_post(&semaphore2);

        }//-------------------CHILD2-------------------

        else{//-------------------PARENT-------------------

        sem\_wait(&semaphore1);

        sem\_wait(&semaphore2);

        close(fd\_src);

        if(munmap(src, statbuf.st\_size) <0){

            printf("UNMAPPING ERROR\n");

            exit(-1);

        }

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

            printf("SEMDEL1 ERROR\n");

            exit(-1);

        }

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

            printf("SEMDEL2 ERROR\n");

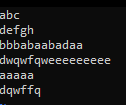
            exit(-1);

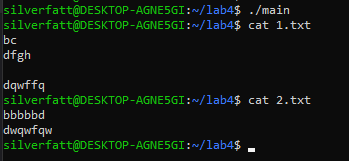
        }

        }//-------------------PARENT-------------------

    }

}

**Демонстрация работы программы**text.txt  
****

****

**Выводы**Данная лабораторная работа, на мой взгляд, служит отличным дополнением ко второй лабораторной работе. Благодаря поставленному заданию я расширил свои навыки работы с процессами и освоил технологию «файл маппинга» и научился грамотно её использовать.