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:Маринин И.С.
Группа: М8О-208Б-20
Вариант: 22
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

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: с вероятностью 80% строки отправляются в процесс 1, иначе в процесс 2. Оба процесса инвертируют строки. Межпроцессорное взаимодействие осуществляется посредством отображаемых файлов (memory-mapped files).

Общие сведения о программе

Для реализации поставленной 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 C++.
<string> - 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 которого 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. 80% строк будет 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.
Собирается программа при помощи команды g++ lab4.cpp, запускается при помощи команды ./a.out.

Исходный код

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fstream>
#include <string>
#include <sstream>

const int MAX_LENGTH = 50;
const int NUMBER_OF_BYTES = MAX_LENGTH * sizeof(char);

using namespace std;

int main() {

    int two_path = 0;
    int one_path = 0;
    int first_pos = 0;
    int second_pos = 0;
    int first_length = 0;
    int second_length = 0;
    int fd1;
    int fd2;
    fstream fs;
    string path_child1, path_child2;

    cout << "Enter the file names: " << endl;
    cin >> path_child1 >> path_child2;

    string str;
```

```
    if ((fd1 = open("f1.txt", O_RDWR| O_CREAT, S_IRWXU)) == -1)
{
        cout << "Error: can not open the f1.txt. Try again lat-
er." << endl;
        exit(EXIT_FAILURE);
    }

    if ((fd2 = open("f2.txt", O_RDWR| O_CREAT, S_IRWXU)) == -1)
{
        cout << "Error: can not open the f2.txt. Try again lat-
er." << endl;
        exit(EXIT_FAILURE);
    }

    char *mapped_file1 = (char *)mmap(0, NUMBER_OF_BYTES, PRO-
T_READ | PROT_WRITE, MAP_SHARED, fd1, 0);
    char *mapped_file2 = (char *)mmap(0, NUMBER_OF_BYTES, PRO-
T_READ | PROT_WRITE, MAP_SHARED, fd2, 0);

    if (mapped_file1 == MAP_FAILED ||mapped_file2 == MAP_FAILED)
{
        cout << "An error with mmap function one has been de-
tected" << endl;
        exit(EXIT_FAILURE);
    }

    cout << "Enter your strings: " << endl;

    while (cin >> str) {

        str = str + "\n";

        int r = rand() % 10 + 1;
        if (r >= 2) {
            one_path++;
            first_length += str.size();
            if (ftruncate(fd1, first_length)) { // устанавливаем
длину fd1 в first_length байт. При успешной работе функции воз-
вращаемое значение равно нулю; При ошибке возвращается -1
```

```
                cout << "Error during ftrancate with mf1 has
been detected" << endl;
                exit(EXIT_FAILURE);
            }
            for (int i = 0; i < str.size(); ++i) {
                mapped_file1[first_pos++] = str[i];
            };
        }

        else {
            two_path++;
            second_length += str.size();
            if (ftruncate(fd2, second_length)) {
                cout << "Error during ftrancate with mf2 has
been detected" << endl;
                exit(EXIT_FAILURE);
            }
            for (int i = 0; i < str.size(); ++i) {
                mapped_file2[second_pos++] = str[i];
            }
        }
    }

    int first_identificator = fork();
    if (first_identificator == -1) {
        cout << "Fork error!" << endl;
        exit(EXIT_FAILURE);
    }

    else if (first_identificator == 0) {
        ofstream ofs (path_child1, ios::out | ios::trunc);
        fs.open(path_child1, fstream::in | fstream::out |
fstream::app);
        if (!fs.is_open()) {
            exit(EXIT_FAILURE);
        }

        int i = 0;
        while (one_path > 0) {
            string str;
```

```
            while (mapped_file1[i] != '\n') {
                str += mapped_file1[i];
                i++;
            }
            if (mapped_file1[i] == '\n')
                i++;

            int j = (int)str.size() - 1;

            for (int i = 0; i < str.size() / 2; ++i) {
                char tmp = str[i];
                str[i] = str[j];
                str[j] = tmp;
                j--;
            }

            fs << str << endl;
            one_path--;
        }
    }
    else {
        int second_identificator = fork();
        if (second_identificator == -1) {
            cout << "Fork error!" << endl;
            return 4;
        }
        else if (second_identificator == 0) {
            ofstream ofs (path_child2, ios::out | ios::trunc);
            fs.open(path_child2, fstream::in | fstream::out |
fstream::app);
            if (!fs.is_open()) {
                exit(EXIT_FAILURE);
            }

            int i = 0;
            while (two_path > 0) {
                string str;
                while (mapped_file2[i] != '\n') {
                    str += mapped_file2[i];
                    i++;
```

```
                }
                if (mapped_file2[i] == '\n')
                    i++;

                int j = (int)str.size() - 1;

                for (int i = 0; i < str.size() / 2; ++i) {
                    char tmp = str[i];
                    str[i] = str[j];
                    str[j] = tmp;
                    j--;
                }

                fs << str << endl;
                two_path--;
            }
        }
        else
        {
            if (munmap(mapped_file1, NUMBER_OF_BYTES) == -1)
{ // Проверяем отражается ли NUMBER_OF_BYTES байт, определенного
файловым описателем, в память
                cout << "Munmap1 error has been dected!" <<
endl;
                exit(EXIT_FAILURE);
            }
            if (munmap(mapped_file2, NUMBER_OF_BYTES) == -1) {
                cout << "Munmap2 error has been dected!" <<
endl;
                exit(EXIT_FAILURE);
            }
            close(fd1);
            close(fd2);
            remove("f1.txt");
            remove("f2.txt");
            return 0;
        }
    }
}
```

Демонстрация работы программы

ivanmarinin@MacBook-Air-Ivan src % g++ task_22.cpp

ivanmarinin@MacBook-Air-Ivan src % ./a.out

Enter the file names:

test

test2

Enter your strings:

ooopppp

iii

jrjririirrrrr

kfkkkkkkffffff

fjjfjfjff

qqqeeee

ivanmarinin@MacBook-Air-Ivan src % cat test

ppppooo

iii

rrrrriirirjrj

ffffffkkkkkkfk

eeeeqqq

ivanmarinin@MacBook-Air-Ivan src % cat test2

ffjfjfjjf

Выводы

Mmap еще один способ взаимодействия между процессами, такой подход в отличие от pipe ускоряет работу программы, засчет того, что нет вызовов read, write, помимо всего прочего мы возвращаем void *, который в итоге можем конвертировать под свои типы данных.