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: Корнев Максим Сергеевич

Группа: М8О–212Б–22

Вариант: 16

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

* Освоение 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.

Вариант 16:

В файле записана строка, необходимо проверить оканчивается ли она на «;» или «.». В случае выполнения условия вывести ее в стандартный поток вывода, иначе вывести сообщение об ошибке.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Помимо этого, есть программа child.cpp, которая запускается при помощи execlp. Обе программы собираются при помощи системы сборки CMake.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Считать название тестового файла
2. C помощью функции fstat заполнить структуру stat; узнать размер файла
3. С помощью функции mmap отразить файл в память
4. Создать дочерний процесс
5. Дождаться завершения работы алгоритма в дочернем процессе
6. Очистить память и закрыть файл

**Основные файлы программы**

**Main.cpp**

#include <iostream>

#include <string>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <signal.h>

int main(int argc, char\* argv[]) {

    if(argc != 2) {

        std::cerr << "Opening file error" << std::endl;

        return 1;

    }

    const char\* CHILD = "child";

    std::string path {"../test/"};

    std::string name {argv[1]};

    path += name;

    //std::cout << path << std::endl;

    int fd = open(path.c\_str(), O\_RDONLY);

    if (fd == -1) {

        std::cerr << "Error not open file" << std::endl;

        return 1;

    }

    //определяем размер файла

    struct stat fileStats;

    if(fstat(fd, &fileStats) == -1) {

        std::cerr << "Failed to get file information" << std::endl;

        close(fd);

        return 1;

    }

    char\* fileData = (char\*)mmap(NULL,fileStats.st\_size, PROT\_READ, MAP\_SHARED, fd, 0);

    if(fileData == MAP\_FAILED) {

        std::cerr << "Failed to map the file into memory" << std::endl;

        close(fd);

        return 1;

    }

    pid\_t pid = fork();

    if(pid == -1) {

        perror("Fork Error");

        munmap(fileData, fileStats.st\_size);

        close(fd);

        return 1;

    }

    if(pid == 0) {

        execl(CHILD, fileData, std::to\_string(fileStats.st\_size).c\_str(),NULL);

        perror("Execl Error");

        munmap(fileData, fileStats.st\_size);

        return 1;

    } else {

        int status;

        waitpid(pid, &status, 0);

        if(WIFEXITED(status)) {

            std::cout << "Child procces was finished by code:  " << WEXITSTATUS(status) << std::endl;

        }

    }

    if(munmap(fileData, fileStats.st\_size) == -1) {

        std::cerr << "Failed to free memory" << std::endl;

    }

    close(fd);

    return 0;

}

**Child.cpp**

#include <iostream>

#include <cstring>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

int main(int argc, char\* argv[]) {

    std::string path = argv[1];

    int size = atoi(argv[2]);

    //std::cout << path << "\t" << size << std::endl;

    int fd = open(path.c\_str(), O\_RDONLY);

    if (fd == -1) {

        std::cerr << "Error not open file" << std::endl;

        return 1;

    }

    struct stat fileStats;

    if (fstat(fd, &fileStats) == -1) {

        std::cerr << "Failed to get file information" << std::endl;

        close(fd);

        return 1;

    }

    char\* fileData = (char\*)mmap(NULL, fileStats.st\_size, PROT\_READ, MAP\_SHARED, fd, 0);

    if (fileData == MAP\_FAILED) {

        std::cerr << "Failed to map the file into memory" << std::endl;

        close(fd);

        return 1;

    }

    if (fileData[size - 1] == ';' || fileData[size - 1] == '.') {

        std::cout << "Valid: " << fileData << std::endl;

    } else {

        std::cerr << "The string is not valid." << std::endl;

        return 1;

    }

    if (munmap(fileData, fileStats.st\_size) == -1) {

        std::cerr << "Failed to free memory" << std::endl;

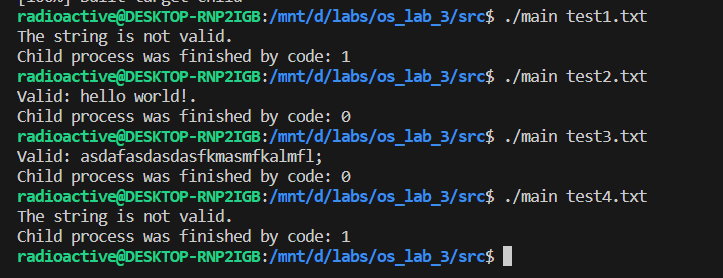
    }

    close(fd);

    return 0;

}

**Пример работы**

****

**Вывод**

В данной лабораторной работе были изучены принципы работы с файловыми системами и освоены навыки обмена данными между процессами с использованием технологии "File mapping".

Для достижения поставленных целей была составлена и отлажена программа на языке C++, которая осуществляет работу с процессами и обменивается данными между ними в выбранной операционной системе. Основной процесс создает один или несколько дочерних процессов для решения поставленной задачи. Взаимодействие между процессами происходит через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

В результате выполнения лабораторной работы были достигнуты поставленные цели - были освоены принципы работы с файловыми системами и технология "File mapping", а также была разработана и отлажена программа, обеспечивающая работу с процессами и взаимодействие между ними.