**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

**Лабораторная работа по предмету "Операционные системы" №3**

Студент: Федоров М. А.

Преподаватель: Миронов Е.С.

Группа: М8О-207Б-22

Дата: 15.09.2022

Оценка:

Подпись:

Оглавление

[Цель работы 3](#__RefHeading__738_2004454639)

[Постановка задачи 3](#__RefHeading__740_2004454639)

[Общие сведения о программе 4](#__RefHeading__742_2004454639)

[Общий алгоритм решения 5](#__RefHeading__744_2004454639)

[Реализация 5](#__RefHeading__746_2004454639)

[Пример работы 8](#__RefHeading__748_2004454639)

[Вывод 9](#__RefHeading__750_2004454639)

# **Цель работы**

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

# **Постановка задачи**

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.*Изображение выглядит как текст, диаграмма, круг, снимок экрана

Автоматически созданное описание*

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами.

# **Общие сведения о программе**

Программа представлена двумя файлами **–** l1\_parent.c и l1\_child.c.

В программе используются следующие системные вызовы:

**signal() –** дает указание выполнить функцию, на которую указывает sigfunc, в случае получения сигнала signal.

**fopen() –**  создание потока для записи в файл

**mmap() –** открытие MMF файла и получение его адреса

**kill() –** отправка сигнала

**ftruncate() –** задает размер файла.

**execl () –** запуск второстепенной программы, указанной в параметрах

**fork ()** – создание дочернего процесса.

# **Общий алгоритм решения**

# **Реализация**

**l1\_child.c**

#include<stdio.h>

#include<stdlib.h>

#include<signal.h>

#include<unistd.h>

#include<string.h>

#include<sys/wait.h>

#include<sys/mman.h>

#include<fcntl.h>

#include<ctype.h>

#include<stdbool.h>

void continuation(int){

    //printf("continuation\n");

}

void ending(int){

    printf("ending child process\n");

    exit(0);

}

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

    signal(SIGUSR1, continuation);

    signal(SIGUSR2, ending);

    signal(SIGTERM, ending);

    char\* filename = argv[1];

    int mmap\_file = atoi(argv[2]);

    char\* mmap\_pointer;

    int offset;

    int iterator;

    int readlen = 10;

    FILE\* file\_stream = fopen(filename, "w");

    while(1){

        pause();

        //printf("%s\n", filename);

        offset = 0;

        mmap\_pointer = mmap(0, readlen, PROT\_WRITE, MAP\_SHARED, mmap\_file, offset);

        if (isupper(mmap\_pointer[0]) == false){

            kill(0, SIGTERM);

        }

        iterator = 0;

        while(mmap\_pointer[iterator] != '\0'){

            printf("writing:%c    ", mmap\_pointer[iterator]);

            fprintf(file\_stream, "%c", mmap\_pointer[iterator]);

            ++iterator;

            printf("%d - %d\n", iterator, readlen);

            if (iterator >= readlen+offset){

                printf("changing offset\n");

                offset += readlen;

                mmap\_pointer = mmap(0, readlen+offset, PROT\_WRITE, MAP\_SHARED, mmap\_file, 0);

                printf("offset changed. Next letter:%c\n", mmap\_pointer[iterator]);

            }

        }

        kill(0, SIGUSR1);

    }

    printf("\nfilename:%s\n", filename);

    printf("\ndescriptor:%d\n", mmap\_file);

    printf("\nfirst char:%c\n", mmap\_pointer[0]);

    return 0;

}

**-----------------------------------------------------------**

**l1\_parent.c**

#include<stdio.h>

#include<stdlib.h>

#include<signal.h>

#include<unistd.h>

#include<string.h>

#include<sys/wait.h>

#include<sys/mman.h>

#include<fcntl.h>

void filler(int){

}

void inputError(int){

    printf("finishing programm because of input error\n");

    exit(-1);

}

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

    signal(SIGUSR1, filler);

    signal(SIGUSR2, filler);

    signal(SIGTERM, inputError);

    printf("max size of input is 100 symbols. If input has more symbols, programm may work incorrect. Every input must start with hight letter\n");

    char\* filename = malloc(100);

    char\* std\_input = malloc(100);

    char child\_status[1];

    char child\_parameters[4];

    char\* mmap\_pointer;

    int mmap\_file = open("mmapfile.txt", O\_RDWR | O\_TRUNC);

    size\_t len;

    printf("input filename to write:");

    scanf("%s", filename);

    pid\_t pid = fork();

    if (pid == -1){

        perror("fork");

        return -1;

    } else if (pid == 0) {

        printf("\n\n\nstarted child process\n");

        char descriptor[1];

        descriptor[0] = mmap\_file+'0';

        execl("./child", "./child", filename, descriptor, NULL);

        printf("\n\n\nafg\n\n\n");

        return 0;

    } else {

        getc(stdin);

        printf("scanning:");

        while (fgets(std\_input, 100, stdin)!=NULL && \*std\_input != '\n') {

            len = strlen(std\_input);

            if (ftruncate(mmap\_file, len) == -1){

                perror("ftruncate");

                return -1;

            }

            mmap\_pointer = mmap(0, len, PROT\_WRITE, MAP\_SHARED, mmap\_file, 0);

            if (mmap\_pointer == MAP\_FAILED){

                perror("mmap");

                return -1;

            }

            printf("string:%s", std\_input);

            //printf("%d\n", mmap\_pointer);

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

                //printf("writing:%c\n", std\_input[i]);

                mmap\_pointer[i] = std\_input[i];

            }

            printf("signaling\n");

            kill(0, SIGUSR1);

            pause();

            printf("scanning:");

        }

        printf("\n");

    }

    printf("waiting for end of child process...\n");

    kill(0, SIGUSR2);

    wait(NULL);

    printf("parent process succesfully end work\n");

    return 0;

}

**-----------------------------------------------------------**

# **Пример работы**

**Test 1**

|  |  |
| --- | --- |
| Input | Output |
| 1  Asd  Rog |  |

# **Вывод**