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: Железнов Илья Васильевич
Группа: М8О–210Б–22
Вариант: 16
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель работы

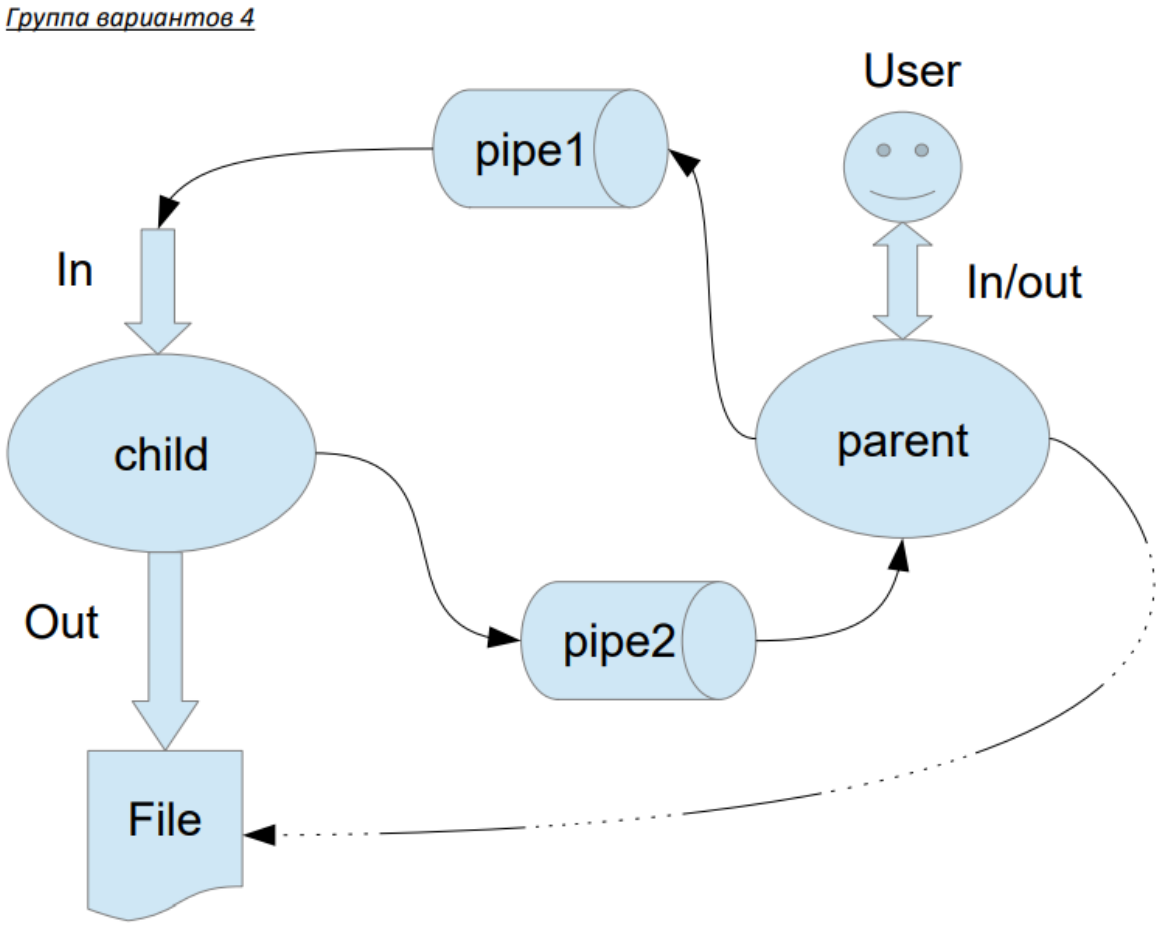
Целью является приобретение практических навыков в:

- 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.

Группа вариантов 4



Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись.

Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

16) Правило проверки: строка должна оканчиваться на «.» или «;»

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: iostream, stdio.h, fcntl.h, unistd.h, sys/wait.h. sys/mman.h, sys/stat.h, string.h, string.

В программе используются следующие системные вызовы:

1. **mmap** – создает новое сопоставление в виртуальном адресном пространстве вызывающий процесс. Начальный адрес нового сопоставления: указан в addr. Аргументы функции: **void* addr** — желаемый адрес начала участка отображенной памяти, передаём 0 — тогда ядро само выберет этот адрес. **size_t len** – количество байт, которое нужно отобразить в память, **int prot** — число, определяющее степень защищённости отображенного участка памяти(только чтение, только запись, исполнение, область недоступна). Обычные значения — PROT_READ, PROT_WRITE(можно кобминировать через ИЛИ), **int flag —** описывает атрибуты области. Обычное значение — MAP_SHARED, **int filedes –** дескриптор файла, который нужно отобразить, **off_t off –** смещение тображенного участка от начала файла
2. **munmap** – функция должна удалить любые сопоставления для всех страниц, содержащих любую часть адресного пространства процесса, начиная с addr и продолжая len байт. Аргументы функции: **void* addr** – указатель на виртуальное адресное пространство, **size_t len** - его размер в байтах.

3. **mremap** – функция переназначает адрес виртуальной памяти. Аргументы функции: **void* old_adress –** указатель на старое виртуальное адресное пространство, **size_t old_size** - старый размер блока виртуальной памяти, **size_t new_size –** требуемый размер блока виртуальной памяти, **unsigned long flags –** параметр, контролирующий работу с памятью (MREMAP_MAYMOVE, MREMAP_FIXED, MREMAP_DONTUNMAP).

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap, malloc.
2. Переписать вариант первой лабараторной, работующей на pipe, используя mmap (munmap, mremap).
3. Реализовать простой интерфейс ввода и вывода результата.
4. Путем двух процессов работать со строками, сообщая между собой информацию.
5. В созданный по ходу работы с программой файл, записать строки, прошедшие на валидность.

Основные файлы программы

main.cpp

```
#include <iostream>

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <string.h>
#include <string>

int main()
{
    char symbol;
    char* in = (char* )malloc(sizeof(char));
    char* filePath = (char* )malloc(sizeof(char));

    int* size = (int* )mmap(NULL, sizeof(int), PROT_READ |
PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, 0, 0);
    int counter = 0;

    if (size == MAP_FAILED) {
        std::perror("ERROR! mmap: int size");
        exit(EXIT_FAILURE);
    }

    *size = 1;

    std::cout << "Enter file path" << std::endl;
```

```
    while ((symbol = getchar()) !=  '\n') {
        filePath[counter++] = symbol;

        if (counter == *size) {
            *size *= 2;
            filePath = (char* )realloc(filePath, (counter + 1) *
sizeof(char));
        }
    }

    filePath = (char* )realloc(filePath, (*size) *
sizeof(char));
    filePath[counter] = '\0';
    counter = 0;
    *size = 1;

    std::cout << "Enter something strings. If you want to
stoped, press enter Ctrl+D" << std::endl;

    while ((symbol = getchar()) != EOF) {
        in[counter++] = symbol;

        if (counter == *size) {
            *size *= 2;
            in = (char* )realloc(in, (*size) * sizeof(char));
        }
    }

    *size = counter + 1;
    in = (char* )realloc(in, (*size) * sizeof(char));
    in[(*size) - 1] = '\0';

    char* ptr = (char* )mmap(NULL, (*size) * sizeof(char),
PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, 0 , 0);

    if (ptr == MAP_FAILED) {
        std::perror("ERROR! mmap: array symbol");
        free(in);
        free(filePath);

        int err = munmap(size, sizeof(int));

        if (err != 0) {
            std::perror("ERROR! munmap: delete");
        }

        exit(EXIT_FAILURE);
    }

    strcpy(ptr, in);

    int flags = O_RDWR | O_CREAT;
    int mods = S_IRWXU | S_IRWXG | S_IRWXO;
```

```
    int fd = open(filePath, flags, mods);

    if (fd < 0) {
        std::perror("ERROR! file: not opened");
        free(in);
        free(filePath);
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));

        if ((err1 != 0) || (err2 != 0)) {
            std::perror("ERROR! munmap: delete");
        }

        exit(EXIT_FAILURE);
    }

    char* f = (char* )mmap(NULL, sizeof(char), PROT_READ |
PROT_WRITE, MAP_SHARED, fd, 0);

    if (f == MAP_FAILED) {
        std::perror("ERROR! mmap: file create");
        free(in);
        free(filePath);

        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));

        if ((err1 != 0) || (err2 != 0)) {
            std::perror("ERROR! munmap: delete");
        }

        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        std::perror("ERROR! fork: child didn't created");

        free(in);
        free(filePath);

        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));

        if ((err1 != 0) || (err2 != 0)) {
            std::perror("ERROR! munmap: delete");
        }

        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // child
```

```
        std::string str, fileStr, outStr;

        for (size_t i = 0; i < *size; ++i) {
            if (i != (*size) - 1) {
                str += ptr[i];
            }

            if ((ptr[i] == '\n') || (i == (*size) - 1)) {
                if ((i > 0) && ((ptr[i - 1] == '.') || (ptr[i -
1] == ';'))) {
                    fileStr += str;
                } else {
                    outStr += str;
                }

                str.clear();
            }
        }

        if ((fileStr.length()) && (fileStr[fileStr.length() - 1]
!= '\n')) {
            fileStr += '\n';
        }

        if (fileStr.length() != 0) {
            if (ftruncate(fd, std::max((int)fileStr.length(), 1)
* sizeof(char)) == -1) {
                std::perror("ERROR! ftruncate: file is not
cut");

                free(in);
                free(filePath);

                exit(EXIT_FAILURE);
            }

            if ((f = (char*)mremap(f, sizeof(char),
(fileStr.length() + 1) * sizeof(char), MREMAP_MAYMOVE)) ==
(void* ) -1) {
                std::perror("ERROR! mremap: not resize
memmory");

                free(in);
                free(filePath);
                exit(EXIT_FAILURE);
            }

            sprintf(f, "%s", fileStr.c_str());
        }

        if ((outStr.length()) && (outStr[outStr.length() - 1] !=
'\n')) {
            outStr += '\n';
```

```
        }

        if ((ptr = (char* )mremap(ptr, (*size) * sizeof(char),
outStr.length() + 1, MREMAP_MAYMOVE)) == (void* )-1) {
            std::perror("ERROR! mremap: Failed to cut file by
line");

            free(in);
            free(filePath);
            exit(EXIT_FAILURE);
        }

        *size = outStr.length() + 1;
        sprintf(ptr, "%s", outStr.c_str());
    } else { // parent
        int wstatus;
        waitpid(pid, &wstatus, 0);

        if (wstatus) {
            free(in);
            free(filePath);

            int err1 = munmap(ptr, (*size) * sizeof(char));
            int err2 = munmap(f, counter * sizeof(char));
            int err3 = munmap(size, sizeof(int));

            if ((err1 != 0) || (err2 != 0) || (err3 != 0)) {
                std::perror("ERROR! munmap: delete");
            }

            exit(EXIT_FAILURE);
        }

        struct stat statbuf;
        if (fstat(fd, &statbuf) < 0) {
            std::perror("ERROR! fstat: cannot open file");
            exit(EXIT_FAILURE);
        }

        counter = std::max((int)statbuf.st_size, 1);

        std::cout << "This strings end to '.' or ';' :" <<
std::endl;

        if (statbuf.st_size > 1) {
            std::cout << f << std::endl;
        }

        std::cout << "This strings not end to '.' or ';' :" <<
std::endl;
        std::cout << ptr;
        close(fd);
```

```
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(f, counter * sizeof(char));
        int err3 = munmap(size, sizeof(int));

        if ((err1 != 0) || (err2 != 0) || (err3 != 0)) {
            std::perror("ERROR! munmap: delete");
            free(in);
            free(filePath);

            exit(EXIT_FAILURE);
        }
    }

    free(in);
    free(filePath);

    return 0;
}
```

Пример работы

```
keinpop@DESKTOP-T6SLHUS:/mnt/c/oc_lab3/src$ g++ main.cpp
keinpop@DESKTOP-T6SLHUS:/mnt/c/oc_lab3/src$ ./a.out
Enter file path
1
Enter something strings. If you want to stoped, press enter
Ctrl+D
asdasd
ccccccccccccccccccc.
zxc;;.d
asdsadsad;
aqwerga..
Hello, World!
asdsd   dasdsa f .
z   .
   l;
This strings end to '.' or ';' :
ccccccccccccccccccc.
```

```
asdsadsad;
aqwerga..
asdsd   dasdsa f .
z   .
   l;

This strings not end to '.' or ';' :
asdasd
zxc;;.d
Hello, World!
```

Вывод

Изучив принцип работы виртуальной памяти на низком уровне работы, я смог разобраться в ее работе и удобстве. Переписав первую лабораторную работу с технологии pipe на технологию mmap, я понял, что можно работать с передачей данных по-разному. Технология mmap очень удобна в работе, хоть и не так проста на первый взгляд. Важный аспект при работе с ней, грамотное выделение памяти с правами доступа и ее удаление, дабы избежать утечек и ошибок. В будущем мне может пригодиться умение работать с mmap, так как это очень актуальная технология на низкоуровневых разработках.