Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**МАППИРОВАНИЕ ФАЙЛОВ**

Студент: Куценко Максим Дмитриевич

Группа: М8О–212Б-22

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в: 

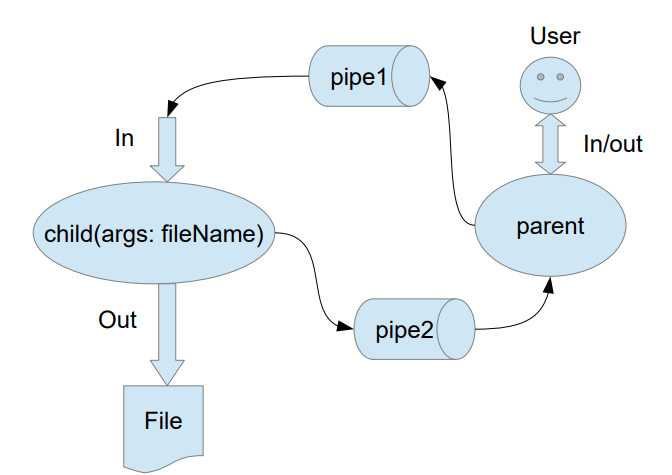
* Освоение принципов работы с файловыми системами.
* Обеспечение обмена данных между процессами посредством технологии «File Mapping».

**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 отображаемые файлы (memory-mapped files). Допускается использование системных сигналов/событий.

*Вариант задания 3:*

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. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число<endline>» Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

**Общие сведения о программе**

При помощи Makefile собираются программы — main и divider. В main происходит создание и маппирование файла, а также создание процесса-потомка при помощи fork, чтение из пользовательского ввода, обработка ошибок процесса-потомка и вывод результата. В divider происходит считывание из файла, с которым происходит работа, деление чисел, запись результата в файл и уменьшение размера файла. Пока проходит ввод в родительском файле, процесс-потомок останавливается при помощи сигнала.

Программа компилируется из файла main.c. Также используется заголовочные файлы: "stdlib.h", "stdio.h", "unistd.h", "fcntl.h", "sys/mman.h", "sys/wait.h", "signal.h", "string.h", "errno.h".

В программе используются следующие системные вызовы:

1. **open** — открытие файла.
2. **close —** закрытие файла.
3. **mmap —** маппирование файла.
4. **munmap —** прекращение маппирования.
5. **raise —** передача сигнала вызвавшему команду процессу.
6. **kill —** передача сигнала другому процессу.
7. **fork** — создание дочернего процесса.
8. **exec** — запуск исполняемого файла.
9. **wait** — ожидание окончания работы дочернего процесса родительским. Получает статус выхода.
10. **ftruncate** — установка размера файла.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы маппирования.
2. Написать main.c. В нём обеспечить обработку всех способных возникнутб ошибок, связанных с системными вызовами, а также то, что процесс-потомок должен запуститься только после ввода из родителя.
3. Написать divider.c. Там при помощи strtok делим строки получая числа для деления. Проверяем аргументы на правильность, печатаем в файл при помощи sptrinf и сжимаем файл при помощи ftruncate.

**Основные файлы программы**

**CmakeLists.txt**

cmake\_minimum\_required(VERSION 3.0)

project(Memory\_map)

set(CMAKE\_CXX\_STANDARD 20)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

add\_executable(main main.c)

add\_executable(divider divider.c)

**main.c:**

#include "stdlib.h"

#include "stdio.h"

#include "unistd.h"

#include "fcntl.h"

#include "sys/mman.h"

#include "sys/wait.h"

#include "signal.h"

#define filename "file.txt"

int main() {

int file\_write = open(filename, O\_CREAT | O\_RDWR | O\_TRUNC, 0777); // creating file

if (file\_write == -1) {

perror("File creation failure");

return 1;

}

posix\_fallocate(file\_write, 0, 4096); // ensuring that file has a size of one page

char\* buffer = mmap(NULL, 4096, PROT\_WRITE, MAP\_SHARED, file\_write, 0); // buffering the file

if (buffer == MAP\_FAILED) {

perror("Parent mapping for writing failure");

return 2;

}

close(file\_write);

pid\_t pid = fork(); // creating child process

if (pid == -1) {

perror("Child process creation failure");

return 3;

} else if (pid == 0) { // going in child process

int stop\_signal = raise(SIGSTOP); //stopping the child to avoid mapping intersection

if (stop\_signal != 0) {

perror("Child process stopping failure");

return 4;

}

execle("divider", "divider", filename, NULL); // executing program to divide the numbers

perror("Exec error");

return 5;

} else { // going in parent process

waitpid(pid, NULL, WUNTRACED); // waiting for a child to be stopped

char c = getchar();

for(int i=0;; ++i) { // writing arguments from input to buffer

buffer[i] = c;

c = getchar();

if (c=='\n') {

buffer[i+1] = '\0';

break;

}

}

if (munmap(buffer, 4096) == -1) { // checking for error in child

return 6;

}

int cont\_signal = kill(pid, SIGCONT); // sending the signal for child to continue

if (cont\_signal != 0) {

perror("Child process continuation failure");

return 7;

}

int status;

wait(&status); // waiting for child process to finish

int return\_value = WEXITSTATUS(status);

if (return\_value!=0) { // stopping the program in case the child process had an error

return 8;

}

int file\_read = open(filename, O\_RDONLY); // opening file for reading result

if (file\_read == -1) {

perror("File opening for result failure");

return 9;

}

buffer = mmap(NULL, 4096, PROT\_READ, MAP\_SHARED, file\_read, 0);

if (buffer == MAP\_FAILED) {

perror("Parent mapping for reading failure");

return 10;

}

printf("Result: %s\n", buffer);

}

return 0;

}

**divider.c**

#include "stdlib.h"

#include "stdio.h"

#include "unistd.h"

#include "fcntl.h"

#include "sys/mman.h"

#include "string.h"

#include "errno.h"

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

int file\_rw = open(argv[1], O\_RDWR); // opening file for reading arguments

if (file\_rw == -1) {

perror("File opening in child failure");

\_exit(-1);

}

char\* buffer = mmap(NULL, 4096, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file\_rw, 0);

if (buffer == MAP\_FAILED) {

perror("Mapping in child failure");

\_exit(-2);

}

char\* var = strtok(buffer, "\0"); // using strtok to get numbers

var = strtok(var, " ");

int result = atoi(var);

if (var == NULL || (var[0]!='0' && result==0)) {

errno = EINVAL;

perror("Non-numerical input");

\_exit(-3);

}

var = strtok(NULL, " ");

while(var != NULL) {

int del = atoi(var);

if (del == 0) {

errno = EINVAL;

if (var[0]=='0') perror("Zero divisor");

else perror("Non-numerical input");

\_exit(-4);

}

result /= del;

var = strtok(NULL, " ");

}

int num\_len = 1, num = result/10;

while (num>0) {

num/=10;

++num\_len;

}

char res\_str[num\_len];

sprintf(res\_str, "%d", result);

memcpy(buffer, res\_str, num\_len); // writing the result in file

ftruncate(file\_rw, num\_len); // shrinking the file

}

**Примеры работы**

**[Maxim@HONOR-MB15 build]$ ./main**

1000 5 2 4

Result: 25

**[Maxim@HONOR-MB15 build]$ ./main**

1000000 2 3 4 5 6

Result: 1388

**[Maxim@HONOR-MB15 build]$ ./main**

30 5 3 2

Result: 1

**[Maxim@HONOR-MB15 build]$ ./main**

123456789 9 8 7 6 5 4 3 2 1

Result: 340

**[Maxim@HONOR-MB15 build]$ ./main**

2

Result: 2

**[Maxim@HONOR-MB15 build]$ ./main**

0 1 2 3

Result: 0

**[Maxim@HONOR-MB15 build]$ ./main**

100 2 3 0

Zero divisor: Invalid argument

**[Maxim@HONOR-MB15 build]$ ./main**

p 2

Non-numerical input: Invalid argument

**[Maxim@HONOR-MB15 build]$ ./main**

2 p

Non-numerical input: Invalid argument

**[Maxim@HONOR-MB15 build]$ ./main**

Non-numerical input: Invalid argument

**Вывод**

Изучил, как работает mmap и его преимущества в сравнении с использование pipe и работой с дескрипторами. Познакомился с такой функцией как ftruncate, впервые применил сигналы.

Использование маппинга куда проще, чем использование pipe. Запись и считывание из pipe является достаточно муторным и требует куда больше операций для организации правильной связи процессов.

Сигналы для себя нашёл более полезными в данной ЛР, чем мьютекс, поскольку в данном случае требуется остановка только одного процесса на время пользовательского ввода, а с помощью сигналов это делается куда проще.

Впервые воспользовался глобальной переменной errno и с помощью её изменения избавился от ERROR: SUCCESS при использовании perror() без наличия явной ошибки.

Встретил ошибку, связанную с невозможностью маппинга при создании файла с разрешением другим чем O\_RDWR.