Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**ОТОБРАЖАЕМЫЕ ФАЙЛЫ**

Студент: Ильин И. О.

Группа: М8О–206Б–19

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

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 со стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости 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.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, stdbool.h, sys/types.h, sys/wait.h, fcntl.h, sys/mman.h. В программе используются следующие системные вызовы:

1. **mmap** ­­–– отражает *length* байтов, начиная со смещения *offset*файла (или другого объекта), определенного файловым дескриптором *fd*, в память, начиная с адреса *start*. При удачном выполнении mmap возвращает указатель на область с отраженными данными. При ошибке возвращается значение MAP\_FAILED (-1), а переменная *errno* приобретает соответствующее значение.
2. **munmap** ­­–– удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти". При удачном выполнении munmap возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная *errno* приобретает соответствующее значение.
3. **fork ––** создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
4. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
5. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с mmap.
2. Написать функцию для считывания входной строки и конвертации ее в массив слов через пробел.
3. Написать функцию преобразования строки в int, с учётом неправильных данных.
4. Организовать считывание пути до файла и чисел до конца ввода.
5. Организовать передачу данных между процессами отраженный данных
6. Организовать обработку данных в дочернем процессе.
7. Реализовать обработку системных ошибок на стороне родительского процесса

**Основные файлы программы**

**main.c**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <stdbool.h>

#include <sys/mman.h>

#define ERR\_MMAP 2

#define ERR\_MALLOC 3

#define ERR\_REALLOC 4

#define ERR\_READ 5

#define ERR\_WRITE 6

#define ERR\_OPEN 7

#define ERR\_CLOSE 8

#define ERR\_PIPE 9

#define ERR\_FORK 10

#define ERR\_WAIT 11

#define ERR\_DIV\_0 20

#define ERR\_INV\_INT 21

#define ERR\_NO\_STATUS 22

#define ERR\_SIZE 23

#define CHILD\_BLOCK 24

#define PAR\_BLOCK 25

#define PAGESIZE 2048

/\*

\* 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.

\*/

int get\_int\_length(int num) { // Считает длину числа

int length = 0;

if (num < 0) {

++length;

}

do {

num /= 10;

++length;

} while (num != 0);

return length;

}

void int\_to\_string(char\* s, int x) { // Преобразует число в строку

bool is\_negative = x < 0;

if (is\_negative) {

x \*= -1;

}

bool is\_first = true;

char\* p;

do {

if (is\_first) {

p = s;

is\_first = false;

} else {

++p;

}

\*p = '0' + (x % 10);

x /= 10;

} while (x != 0);

if (is\_negative) {

++p;

\*p = '-';

}

while (s < p) {

char tmp = \*s;

\*s = \*p;

\*p = tmp;

++s;

--p;

}

}

void read\_strings(int fd, char\*\*\* ptr\_buffer, int\* res\_size) { // Считывает строки через пробел до конца ввода

char c; // Текущий считываемый символ

int str\_idx = 0; // Индекс текущего слова

size\_t buffer\_size = 1; // Кол-во строк

char\*\* buffer = malloc(sizeof(char\*) \* buffer\_size);

if (!buffer) {

perror("Malloc error");

exit(ERR\_MALLOC);

}

int\* lengths = malloc(sizeof(int) \* buffer\_size);

if (!lengths) {

perror("Malloc error");

exit(ERR\_MALLOC);

}

lengths[0] = 0;

int str\_size = 1; // Длина текущей строки

buffer[0] = malloc(sizeof(char) \* str\_size);

if (!buffer[0]) {

perror("Malloc error");

exit(ERR\_MALLOC);

}

size\_t read\_bytes = 0;

while ((read\_bytes = read(fd, &c, sizeof(char))) > 0) {

if ((read\_bytes != sizeof(char)) && (read\_bytes != 0)) {

perror("Read error");

exit(ERR\_READ);

}

if (c == '\n') { // Конец ввода слов

break;

}

if (c == ' ') { // Следующее слово

++str\_idx;

str\_size = 1;

continue;

}

// Увеличиваем буфер строк и длин

if (str\_idx >= buffer\_size) {

++buffer\_size;

str\_size = 1;

buffer = realloc(buffer, sizeof(char\*) \* buffer\_size);

if (!buffer) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

buffer[buffer\_size - 1] = malloc(sizeof(char) \* str\_size);

if (!buffer[buffer\_size - 1]) {

perror("Malloc error");

exit(ERR\_MALLOC);

}

for (size\_t i = 0; i < str\_size; ++i) {

buffer[buffer\_size - 1][i] = 0;

}

lengths = realloc(lengths, sizeof(int) \* buffer\_size);

if (!lengths) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

lengths[buffer\_size - 1] = 0;

}

// Увеличиваем длину строки

if (lengths[str\_idx] == str\_size) {

str\_size += 16;

buffer[str\_idx] = realloc(buffer[str\_idx], sizeof(char) \* str\_size);

if (!buffer[str\_idx]) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

}

buffer[str\_idx][lengths[str\_idx]] = c;

++lengths[str\_idx];

}

for (int i = 0; i < buffer\_size; ++i) {

buffer[i] = realloc(buffer[i], sizeof(char) \* lengths[i]);

if (!buffer[i]) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

buffer[i][lengths[i]] = '\0';

}

\*res\_size = buffer\_size;

\*ptr\_buffer = buffer;

}

// str[4] = get\_string();

void read\_string(int fd, char\*\* str, size\_t\* size) { // Считывает одну строку

char c; // Текущий считываемый символ

size\_t str\_len = 0;

size\_t str\_size = 1; // Длина текущей строки

char\* buffer = malloc(sizeof(char) \* str\_size);

if (!buffer) {

perror("Malloc error");

exit(ERR\_MALLOC);

}

size\_t read\_bytes = 0;

while ((read\_bytes = read(fd, &c, sizeof(char))) > 0) {

if ((read\_bytes != sizeof(char)) && (read\_bytes != 0)) {

perror("Read error");

exit(ERR\_READ);

}

if (c == '\n') { // Конец ввода слов

break;

}

// Увеличиваем длину строки

if (str\_len == str\_size) {

str\_size += 16;

buffer = realloc(buffer, sizeof(char) \* str\_size);

if (!buffer) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

}

buffer[str\_len] = c;

++str\_len;

}

buffer = realloc(buffer, sizeof(char) \* str\_len);

if (!buffer) {

perror("Realloc error");

exit(ERR\_REALLOC);

}

buffer[str\_len] = '\0';

\*size = str\_len;

\*str = buffer;

}

void str\_array\_to\_int(char\*\* strs, int nums[], size\_t n) { // Преобразует массив строк в массив int'ов

for (size\_t i = 0; i < n; ++i) {

///nums[i] = atoi(strs[i]);

char\* p = strs[i];

bool is\_negative = false;

int result = 0;

while(\*p != '\0') {

if (\*p == '-' || \*p == '+') {

if (p != strs[i]) {

perror("Invalid int");

exit(ERR\_INV\_INT);

}

if (\*p == '-') {

is\_negative = true;

}

} else if ('0' <= \*p && \*p <= '9') {

result = result \* 10 + (\*p - '0');

} else {

perror("Invalid int");

exit(ERR\_INV\_INT);

}

++p;

}

if (is\_negative) {

result \*= -1;

}

nums[i] = result;

}

}

void write\_to\_fd(int fd, const void \*buf, size\_t nbytes) {

if (write(fd, buf, nbytes) != nbytes) {

perror("Write error");

exit(ERR\_WRITE);

}

}

void read\_from\_fd(int fd, void \*buf, size\_t nbytes) {

if (read(fd, buf, nbytes) != nbytes) {

perror("Read error");

exit(ERR\_READ);

}

}

int main() {

int\* m\_file = mmap(NULL, PAGESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

if (m\_file == MAP\_FAILED) {

perror("mmap error");

exit(ERR\_MMAP);

}

int\* sig = mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

if (sig == MAP\_FAILED) {

perror("mmap error");

exit(ERR\_MMAP);

}

\*sig = CHILD\_BLOCK;

printf("Enter path to file:\n");

char\* path;

size\_t path\_size = 0;

read\_string(0, &path, &path\_size); // Читает имя выходного файла

int status;

int id = fork();

if (id == -1) {

perror("fork() error\n");

exit(ERR\_FORK);

} else if (id == 0) {

// Ребенок

while (\*sig == CHILD\_BLOCK) {}

size\_t size = m\_file[0];

int divident = m\_file[1];

int divisor = 1;

// Читаем числа

for (int i = 2; i < size + 1; ++i) {

divisor \*= m\_file[i];

}

if (divisor == 0) {

exit(ERR\_DIV\_0);

}

int res = divident / divisor;

int length = get\_int\_length(res);

char s[length];

int\_to\_string(s, res);

int fp = open(path, O\_CREAT | O\_WRONLY, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH); // создаем, если не сущ., только для записи

if (fp == -1) { // Если не получилось открыть файл

free(path);

close(fp);

exit(ERR\_OPEN);

}

write\_to\_fd(fp, s, sizeof(char) \* length);

close(fp);

\*sig = CHILD\_BLOCK;

exit(0);

} else {

// Родитель, id = child\_id

char\*\* strs = NULL;

int size = 0;

printf("[%d] Reading...\n", getpid()); fflush(stdout);

read\_strings(0, &strs, &size); // Читаем числа до \n в массив char\* и переводим в массив int'ов

if (size\*sizeof(int) > PAGESIZE ) {

perror("size is to big");

exit(ERR\_SIZE);

}

int nums[size];

str\_array\_to\_int(strs, nums, size);

printf("[%d] Red all values.\n", getpid()); fflush(stdout);

for (size\_t i = 0; i < size; ++i) {

printf("nums[%zu] = %d\n", i, nums[i]);

}

printf("[%d] Writing...\n", getpid()); fflush(stdout);

m\_file[0] = size;

for (size\_t i = 1; i < size + 1; ++i) {

m\_file[i] = nums[i - 1];

}

printf("[%d] Wrote all values.\n", getpid());

for (size\_t i = 0; i < size + 1; ++i) {

printf("m\_file[%zu] %d\n", i, m\_file[i]);

}

free(strs);

\*sig = PAR\_BLOCK;

if (waitpid(id, &status, 0) == -1) {

perror("wait() error\n");

exit(ERR\_WAIT);

}

if (WIFEXITED(status)) { // Если ребенок завершился со статусом завершения

int exit\_code = WEXITSTATUS(status);

switch (exit\_code) {

case ERR\_DIV\_0:

perror("error: you cant divide by zer0\n");

exit(exit\_code);

break;

case ERR\_OPEN:

perror("error: can't open file\n");

exit(exit\_code);

break;

case ERR\_CLOSE:

perror("error: can't close file\n");

exit(exit\_code);

break;

case ERR\_WRITE:

perror("error: can't write to file\n");

exit(exit\_code);

break;

case ERR\_READ:

perror("error: can't read from file\n");

exit(exit\_code);

break;

case ERR\_REALLOC:

perror("error: realloc error\n");

exit(exit\_code);

break;

default:

printf("[%d] Child ended with status: %d\n", getpid(), status); fflush(stdout);

break;

}

} else {

perror("error: child exit with no status\n");

exit(ERR\_NO\_STATUS);

}

}

free(path);

if (munmap(m\_file, PAGESIZE) == -1) {

perror("munmap error");

return ERR\_MMAP;

}

return 0;

}

**Пример работы**

MacBook-Pro:src mr-ilin$ make

gcc -pedantic -Wall -std=c11 main.c -o lab4

MacBook-Pro:src mr-ilin$ ./lab4 < ../test/in01.txt

Enter path to file:

[78467] Reading...

[78467] Red all values.

nums[0] = 10

nums[1] = 2

[78467] Writing...

[78467] Wrote all values.

m\_file[0] 2

m\_file[1] 10

m\_file[2] 2

[78467] Child ended with status: 0

MacBook-Pro:src mr-ilin$ cat ../test/out01.txt

5

MacBook-Pro:src mr-ilin$ ./lab4 < ../test/in02.txt

Enter path to file:

[78473] Reading...

[78473] Red all values.

nums[0] = 864

nums[1] = 8

nums[2] = 3

nums[3] = 3

nums[4] = 4

[78473] Writing...

[78473] Wrote all values.

m\_file[0] 5

m\_file[1] 864

m\_file[2] 8

m\_file[3] 3

m\_file[4] 3

m\_file[5] 4

[78473] Child ended with status: 0

MacBook-Pro:src mr-ilin$ cat ../test/out02.txt

3

MacBook-Pro:src mr-ilin$ ./lab4 < ../test/in03.txt

Enter path to file:

[78498] Reading...

[78498] Red all values.

nums[0] = -1

nums[1] = -1

nums[2] = -1

nums[3] = -1

nums[4] = -1

[78498] Writing...

[78498] Wrote all values.

m\_file[0] 5

m\_file[1] -1

m\_file[2] -1

m\_file[3] -1

m\_file[4] -1

m\_file[5] -1

[78498] Child ended with status: 0

MacBook-Pro:src mr-ilin$ cat ../test/out03.txt

-1

MacBook-Pro:src mr-ilin$ ./lab4 < ../test/in04.txt

Enter path to file:

[78505] Reading...

[78505] Red all values.

nums[0] = 124

nums[1] = 5

nums[2] = 3

nums[3] = 4

nums[4] = 0

nums[5] = 8

nums[6] = 7

nums[7] = 6

[78505] Writing...

[78505] Wrote all values.

m\_file[0] 8

m\_file[1] 124

m\_file[2] 5

m\_file[3] 3

m\_file[4] 4

m\_file[5] 0

m\_file[6] 8

m\_file[7] 7

m\_file[8] 6

error: you cant divide by zer0

: Undefined error: 0

MacBook-Pro:src mr-ilin$ cat ../test/out04.txt

MacBook-Pro:src mr-ilin$

**Вывод**

В процессе выполнения данной лабораторной работы я познакомился с Memory Mapped Files, который представляют собой отображение данных из файлов в участок памяти, причём при работе с данным участком памяти все изменения тут же вносятся в сам файл. Таким образом, использование отображения файла в память позволяет существенно уменьшить время работы с файлами, особенно с большими, поскольку уменьшается число системных вызовов, а также не производится лишнее копирование данных в буфер.

В контексте данной лабораторной работы File Mapping использовался для создания участка памяти, которым могут совместно пользоваться несколько процессов, что позволяет производить обмен данными между ними. Обязательно стоит учитывать, что обращение к данному участку памяти не является блокирующим, поэтому необходимо использовать различные примитивы синхронизации, для корректной работы программы.

В качестве вывода хочется отметить, что в данном варианте лабораторной работы лучше использовать каналы, поскольку их проще использовать и они являются блокирующими, особенно учитывая то, что использовалось всего 2 процесса, которые не выполнялись параллельно. В случае работы с множеством процессов лучше и проще использовать разделяемую память.