Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**МЕЖПРОЦЕССОРНОЕ ВЗАИМОДЕЙСТВИЕ**

Студент: Ильин Илья Олегович

Группа: М8О–206Б–19

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

## Приобретение практических навыков в:

## Управление процессами в ОС

## Обеспечение обмена данных между процессами посредством каналов

## 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 каналы (pipe).

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: unistd.h, stdio.h, stdlib.h, sys/types.h, sys/wait.h, fcntl.h, stdbool.h . В программе используются следующие системные вызовы:

1. **pipe** ­­–– принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а так же вернется 0. В случае неуспеха вернется -1.
2. **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.
3. **close ––** принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
4. **open ––** создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа ( при создании можно указать права для файла ). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
5. **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.
6. **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.
7. **waitpid** — приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс, указанный в параметре pid, не завершит выполнение, или пока не появится сигнал, который либо завершает текущий процесс либо требует вызвать функцию-обработчик. Если указанный дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби"), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe, fork, waitpid.
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 посредством каналов.
6. Организовать обработку данных в дочернем процессе.
7. Реализовать обработку системных ошибок на стороне родительского процесса, используя waitpid.

**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>

#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

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, size\_t\* 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);

}

// Конец ввода слов

if (c == '\n') { break; }

// Следующее слово

if (c == ' ') {

++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) {

//printf("Increasing str len\n");

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;

}

// Считывает одну строку

void read\_string(int fd, char\*\* str, size\_t\* size) {

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') { 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;

}

// Преобразует массив строк в массив int'ов

void str\_array\_to\_int(char\*\* strs, int nums[], size\_t n) {

for (size\_t i = 0; i < n; ++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 fd[2]; // массив файловых дескрипторов

// создаем pipe

if (pipe(fd) != 0) { perror("Piping error"); exit(ERR\_PIPE); }

int status;

int id = fork();

if (id == -1) {

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

exit(ERR\_FORK);

} else if (id == 0) {

// Ребенок

size\_t path\_size;

size\_t size;

int divident;

int divisor = 1;

// Читаем размер строки и строку

read\_from\_fd(fd[0], &path\_size, sizeof(size\_t));

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

if (!path) { exit(ERR\_REALLOC); }

read\_from\_fd(fd[0], path, sizeof(char) \* path\_size);

// Читаем числа

read\_from\_fd(fd[0], &size, sizeof(size\_t));

read\_from\_fd(fd[0], &divident, sizeof(int));

for (int i = 0; i < size - 1; ++i) {

int num;

read\_from\_fd(fd[0], &num, sizeof(int));

divisor \*= num;

}

close(fd[0]);

close(fd[1]);

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); // 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);

free(path);

exit(0);

} else {

// Родитель, id = child\_id

char\*\* strs = NULL;

size\_t size = 0;

char\* path;

size\_t path\_size = 0;

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

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

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

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

}

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

write\_to\_fd(fd[1], &path\_size, sizeof(size\_t));

write\_to\_fd(fd[1], path, sizeof(char) \* path\_size);

write\_to\_fd(fd[1], &size, sizeof(size\_t));

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

write\_to\_fd(fd[1], nums + i, sizeof(int));

}

close(fd[0]);

close(fd[1]);

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

free(strs);

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);

}

}

return 0;

}

**Пример работы**

**Test1** — проверка корректной работы на 2 числах.

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in01.txt

Tests/out01.txt

10 2

MacBook-Pro:os\_lab\_02 mr-ilin$ ./a.out < Tests/in01.txt

[1297] Reading...

[1297] Red all values.

0) 10

1) 2

[1297] Writing...

[1297] Wrote all values.

[1297] Child ended with status: 0

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/out01.txt

5

**Test2** — проверка корректной работы на нескольких числах.

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in02.txt

Tests/out02.txt

864 8 3 3 4MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in02.txt

Tests/out02.txt

864 8 3 3 4

MacBook-Pro:os\_lab\_02 mr-ilin$ ./a.out < Tests/in02.txt

[1351] Reading...

[1351] Red all values.

0) 864

1) 8

2) 3

3) 3

4) 4

[1351] Writing...

[1351] Wrote all values.

[1351] Child ended with status: 0

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/out02.txt

3

**Test3** — проверка корректной работы парсера на отрицательных числах.

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in03.txt

Tests/out03.txt

-1 -1 -1 -1 -1

MacBook-Pro:os\_lab\_02 mr-ilin$ ./a.out < Tests/in03.txt

[1405] Reading...

[1405] Red all values.

0) -1

1) -1

2) -1

3) -1

4) -1

[1405] Writing...

[1405] Wrote all values.

[1405] Child ended with status: 0

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/out03.txt

-1

**Test4** — проверка работы при делении на ноль.

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in04.txt

Tests/out04.txt

124 5 3 4 0 8 7 6

MacBook-Pro:os\_lab\_02 mr-ilin$ ./a.out < Tests/in04.txt

[1446] Reading...

[1446] Red all values.

0) 124

1) 5

2) 3

3) 4

4) 0

5) 8

6) 7

7) 6

[1446] Writing...

[1446] Wrote all values.

error: you cant divide by zer0

: Undefined error: 0

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/out04.txt

**Test5** — проверка работы с файлом, для которого у программы нет прав на запись.

MacBook-Pro:os\_lab\_02 mr-ilin$ chmod -w Tests/out05.txt

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/in05.txt

Tests/out05.txt

121 11 11

MacBook-Pro:os\_lab\_02 mr-ilin$ ./a.out < Tests/in05.txt

[1505] Reading...

[1505] Red all values.

0) 121

1) 11

2) 11

[1505] Writing...

[1505] Wrote all values.

error: can't open file

: Undefined error: 0

MacBook-Pro:os\_lab\_02 mr-ilin$ cat Tests/out05.txt

**Вывод**

Выполняя данную лабораторную работу, я познакомился и научился работать с процессами и каналами. Процессы являются очень важной абстракцией, которая позволяет нам выполнять несколько параллельные операции, даже при работе на одноядерном процессоре. Однако, концепция каналов является не менее важной, ведь каналы играют очень большую роль в программах, которые состоят из множества процессов или потоков. Они позволяют передавать информацию между процессами, что позволяет синхронизировать их работу, добиваясь в итоге решения общей задачи.

Полученные знания непременно пригодятся в дальнейших лабораторных работах.