Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Студент: Лизунов К. Р

Группа: М8О-215Б-23

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

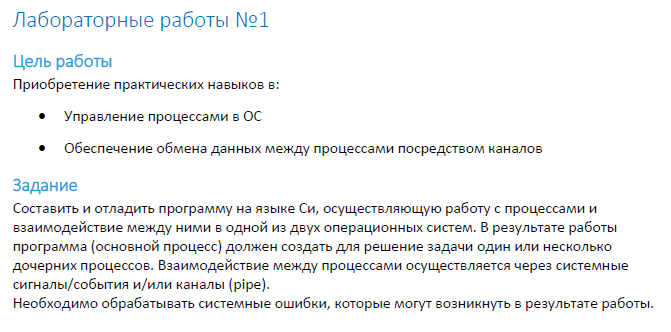
Дата: \_\_\_\_\_\_\_\_\_\_\_

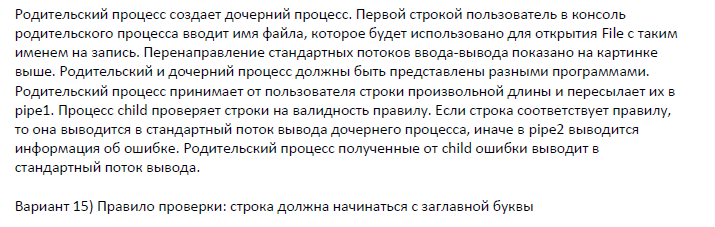
Москва, 2024

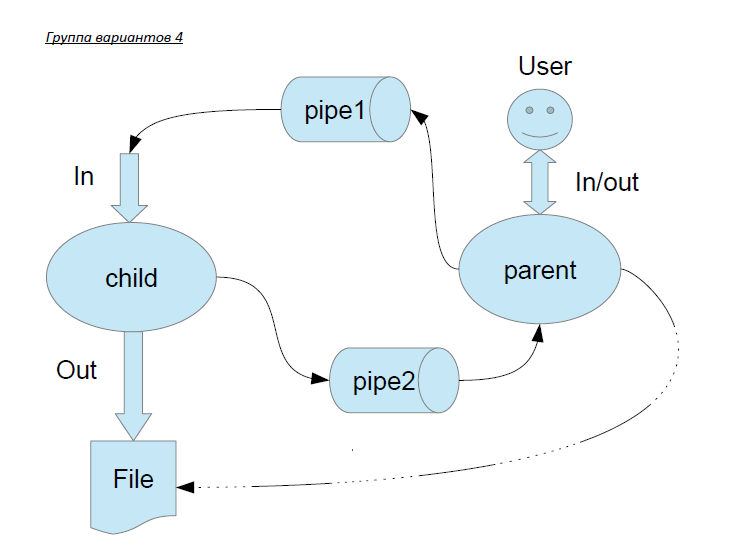
# **Постановка задачи**

## Вариант 15.

**Задание**







**Общий метод и алгоритм решения**

В данной лабораторной работе реализовано взаимодействие между родительским (parent.c) и дочерним (child.c) процессами с использованием механизма межпроцессного взаимодействия pipe(). Родительский процесс получает от пользователя имя файла и команду, передает их в дочерний процесс через канал. Дочерний процесс записывает данные в файл, выполняет арифметические операции и отправляет результат обратно родителю через другой канал.

Использованные системные вызовы:

* **fork()** – создает дочерний процесс (копия родительского).
* **pipe()** – создает канал (pipe) для обмена данными между процессами.
* **exit()** – завершает выполнение процесса.
* **execl()** – заменяет текущий процесс новым (загрузка программы).
* **close()** – закрывает файловый дескриптор (например, конец канала).
* **dup2()** – перенаправляет дескриптор (например, pipe на stdin/stdout).
* **perror()** – выводит сообщение об ошибке (на основе errno).
* **wait()** – ожидает завершения дочернего процесса.

#### **Родительский процесс (**parent.c**):**

1. Создает два канала (pipe1 для передачи данных в child, pipe2 для получения результата от child).
2. Выполняет fork() для создания дочернего процесса.
3. В дочернем процессе:
   * Перенаправляет STDIN на pipe1[0] (чтение из канала).
   * Перенаправляет STDOUT на pipe2[1] (запись результата в канал).
   * Закрывает ненужные файловые дескрипторы.
   * Запускает execl() для выполнения child.c.
4. В родительском процессе:
   * Закрывает ненужные файловые дескрипторы.
   * Запрашивает у пользователя имя файла и команду (список чисел).
   * Передает эти данные в pipe1[1] (запись в канал).
   * Ждет завершения дочернего процесса (waitpid()).
   * Считывает результат из pipe2[0] и выводит на экран.

#### **Дочерний процесс (**child.c**):**

1. Читает из stdin (канал pipe1) имя файла.
2. Открывает файл для записи (fopen()).
3. Читает из stdin команду (список чисел).
4. Разбирает команду, выполняет последовательное деление чисел.
   * Если встречается деление на ноль, записывает ошибку в файл и выводит сообщение об ошибке.
5. Записывает результат в файл и stdout (который перенаправлен в pipe2).
6. Закрывает файл и завершает работу.

**Код Программы**

**Child.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <ctype.h>

#define MAX\_INPUT 512

int main() {

char str\_[MAX\_INPUT]; // Буфер для хранения входной строки

while (fgets(str\_, MAX\_INPUT, stdin) != NULL){

if (isupper(str\_[0])){ // Если строка корректна, она передаётся в STDOUT (перенаправлен в файл).

printf("%s", str\_);

} else {

fprintf(stderr, "error %s", str\_); // в STDERR (связанный с pipe2).

}

}

return 0;

}

**Parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <fcntl.h>

#define MAX\_INPUT 512

//Создаёт макрос, который устанавливает максимальную длину

// строки (512 символов).

// Это помогает управлять памятью и избежать переполнения буфера.

int main() {

int pipe1[2], pipe2[2];

char filepath[MAX\_INPUT];

char str\_[MAX\_INPUT];

char error[MAX\_INPUT];

// Объявление переменных и каналов. pipe1 — передача

// данных родитель → дочерний.

// pipe2 — возврат ошибок дочерний → родитель.

// Данные и ошибки хранятся в буферах (учётная информация о процессе).

printf("File name: ");

fgets(filepath, MAX\_INPUT, stdin);

filepath[strlen(filepath) - 1] = 0;

// Запрашивает имя файла, в который будут записываться корректные строки.

// Использует адресное пространство для хранения введённого имени файла.

int file = open(filepath, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (file == -1) {

perror("file open");

exit(1);

}

// Открывает файл на запись. Если файла нет, он создаётся

// (O\_CREAT), если файл существует — очищается (O\_TRUNC).

// В случае ошибки вызывается perror() — выводит сообщение о проблеме.

if (pipe(pipe1) == -1 || pipe(pipe2) == -1){

perror("pipe");

exit(1);

}

// pipe() создаёт два канала.

// Каждый канал содержит два файловых дескриптора:

// pipe1[0] — для чтения (родитель читает из него).

// pipe1[1] — для записи (родитель пишет в него).

switch (fork()){

// fork() — системный вызов, который создаёт дочерний процесс.

// Родительский процесс получает PID дочернего процесса.

// Дочерний процесс получает 0.

case -1:

perror("fork");

exit(1);

case 0:

dup2(pipe1[0], STDIN\_FILENO); //чтение данных из канала

dup2(pipe2[1], STDERR\_FILENO); //запись ошибок в канал

dup2(file, STDOUT\_FILENO); // запись корректных данных в файл

// dup2() перенаправляет стандартные потоки:

// TDIN → pipe1[0] (чтение данных из канала).

// STDERR → pipe2[1] (запись ошибок в канал).

// STDOUT → file (запись корректных данных в файл).

close(file);

close(pipe1[0]);

close(pipe1[1]);

close(pipe2[0]);

close(pipe2[1]);

execlp("./build/child", "child", NULL);

perror("execlp");

exit(1);

// execlp() заменяет текущий процесс (child.cpp).

// Если замена не удалась, программа выводит сообщение об ошибке

//и завершает процесс.

default:

close(pipe1[0]);

close(pipe2[1]);

while (1){

printf("String or 'e' for exit: ");

fgets(str\_, MAX\_INPUT, stdin);

if (strncmp(str\_, "e", 1) == 0) break;

write(pipe1[1], str\_, strlen(str\_));

}

// Родитель принимает строки от пользователя и отправляет

// их дочернему процессу через pipe1.

// Если введено "e", программа завершает ввод.

close(pipe1[1]);

while(read(pipe2[0], error, MAX\_INPUT) > 0) {

printf("error: %s \n", error);

}

// read() читает сообщения об ошибках из pipe2 и выводит их на экран.

close(pipe2[0]);

wait(NULL);

}

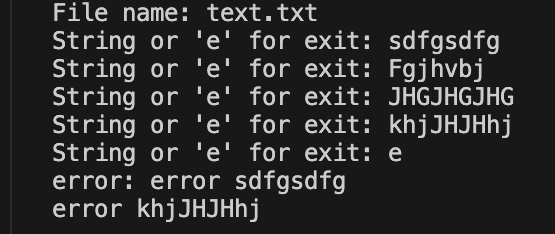
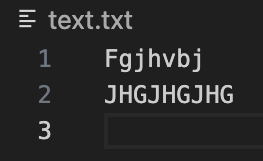
// Родитель вызывает wait(), чтобы дождаться завершения дочернего процесса.

// Это предотвращает появление "зомби"-процессов.

return 0;

}

**Протокол работы программы**

** **

**Strace**

execve("./parent", ["./parent"], 0xfffff2393a00 /\* 4 vars \*/) = 0

brk(NULL) = 0xaaaaeebb8000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0xffffaeac2000

faccessat(AT\_FDCWD, "/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=9167, ...}) = 0

mmap(NULL, 9167, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0xffffaeabf000

close(3) = 0

openat(AT\_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\360\206\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=1722920, ...}) = 0

mmap(NULL, 1892240, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_DENYWRITE, -1, 0) = 0xffffae8bc000

mmap(0xffffae8c0000, 1826704, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0) = 0xffffae8c0000

munmap(0xffffae8bc000, 16384) = 0

munmap(0xffffaea7e000, 49040) = 0

mprotect(0xffffaea5a000, 77824, PROT\_NONE) = 0

mmap(0xffffaea6d000, 20480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0xffffaea6d000

mmap(0xffffaea72000, 49040, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0xffffaea72000

close(3) = 0

set\_tid\_address(0xffffaeac2fb0) = 9

set\_robust\_list(0xffffaeac2fc0, 24) = 0

rseq(0xffffaeac3600, 0x20, 0, 0xd428bc00) = 0

mprotect(0xffffaea6d000, 12288, PROT\_READ) = 0

mprotect(0xaaaab6d3f000, 4096, PROT\_READ) = 0

mprotect(0xffffaeac8000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0xffffaeabf000, 9167) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

getrandom("\x3b\x1c\xff\x5c\x3c\x6b\x10\x19", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0xaaaaeebb8000

brk(0xaaaaeebd9000) = 0xaaaaeebd9000

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(1, "File name: ", 11) = 11

read(0, "t.txt\n", 1024) = 6

openat(AT\_FDCWD, "t.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0644) = 3

pipe2([4, 5], 0) = 0

pipe2([6, 7], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0xffffaeac2fb0) = 10

close(4) = 0

close(7) = 0

write(1, "String or 'e' for exit: ", 24) = 24

read(0, "ghgh\n", 1024) = 5

write(5, "ghgh\n", 5) = 5

write(1, "String or 'e' for exit: ", 24) = 24

read(0, "HJJH\n", 1024) = 5

write(5, "HJJH\n", 5) = 5

write(1, "String or 'e' for exit: ", 24) = 24

read(0, "e\n", 1024) = 2

close(5) = 0

read(6, "error ghgh\n", 512) = 11

write(1, "error ghgh\n \n", 13) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=10, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

write(1, "error ghgh\n \n", 13) = 13

read(6, "", 512) = 0

close(6) = 0

wait4(-1, NULL, 0, NULL) = 10

exit\_group(0) = ?

+++ exited with 0 +++

**Выводы**

Выполнив лабораторную работу, я приобрел навыки работы с процессами и взаимодействия между ними. Изучил основные термины, принципы взаимодействия и программную реализацию.