Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Группа: М8О-211Б-23

Студент: Тимофеева И.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 14.11.2024

Москва, 2024

**Постановка задачи**

**Вариант 4.**

**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 float. Количество чисел может быть произвольным.**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

● pid\_t fork(void); – создает дочерний процесс.

● pid\_t getpid(void); – функция возвращает идентификатор (PID) вызвавшего процесса.

● int pipe(int fd[2]) – создание неименованного канала для передачи данных между

процессами.

● ssize\_t read(int fd, void \*buf, size\_t count); – считывает до count байт из файлового

дескриптора fd в буфер buf.

● int32\_t snprintf(char \*msg, size\_t buffer\_size, const char \*restrict format, int x); - функция для преобразования числа x в отформатированную строку.

● ssize\_t write(int fd, const void \*buf, size\_t count); – записывает до count байт из буфера

buf в файловый дескриптор fd.

● ssize\_t readlink(const char \*restrict pathname, char \*restrict buf,size\_t bufsize); – считывает значение символьной ссылки pathname в буфер buf, у которого размер bufsize.

● int dup2(int oldfd, int newfd); – переназначение файлового дескриптора.

● int32\_t execv(char \*fname, const char \*argv, NULL); – заменяет текущий образ процесса новым образом процесса, загружает и выполняет новый дочерний процесс с исполняемым файлом fname и строкой аргументов argv.

● pid\_t wait(int \*stat\_loc); ожидание изменений состояния программы в дочерних процессах и получения соответствующей информации. Если дочерний процесс завершается, wait() возвращает PID завершённого дочернего процесса.

● float strtof(const char\* str, char\*\* endptr); - преобразует строку с представлением числа с плавающей точкой в значение типа float.

● int open(const char \*pathname, int flags, mode\_t mode) – открытие\создание файла.

● int close(int fd); – Закрывает файловый дескриптор fd.

● void exit(int status) – завершения выполнения процесса и возвращение статуса.

Создание каналов: Программа создает два канала (массив pipe и каналы pipe[1] и pipe[2]), чтобы организовать передачу данных между родительским процессом и одним дочерним процесом.

Ввод и открытие файлов: Программа запрашивает у пользователя имя файла, в который будут записаны выходные данные дочернего процесса. Открывает эти файлы с режимом O\_WRONLY | O\_CREAT | O\_TRUNC | O\_APPEND, что позволяет создавать новый файл или перезаписывать существующий.

Создание дочерних процессов:

Дочерний процесс (child): Порождается с помощью вызова fork() и перенастраивает стандартный ввод на чтение из pipe[1] и стандартный вывод на запись в file. Запускает внешний исполняемый файл ./client\_prog с помощью execv(). Программа обрабатывает входные данные и производит деление первого введенного числа на все остальные. Останавливает работу при попытке деления на 0 или после последнего числа. Результат записывает в файл file, который указан как аргумент программы.

Родительский процесс: ожидает завершения дочернего процесса и также завершает работу.

Ожидание завершения дочернего процесса: Родительский процесс использует wait(), чтобы дождаться завершения дочернего процесса.

**Код программы**

**serv\_prog.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

static char CLIENT\_PROGRAM\_NAME[] = "client\_prog";

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

if (argc == 1) {

char msg[1024];

//printf("usage\n");

uint32\_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);

write(STDERR\_FILENO, msg, len);

exit(EXIT\_SUCCESS);

}

pid\_t ppid = getpid(); // NOTE: Get parent PID

{

char msg[128];

int32\_t len = snprintf(msg, sizeof(msg) - 1, "%d: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit\n", ppid);

write(STDOUT\_FILENO, msg, len);

}

char buf[4096];

ssize\_t bytes;

while (bytes = read(STDIN\_FILENO, buf, sizeof(buf))) {

if (bytes < 0) {

const char msg[] = "error: failed to read from stdin\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

} else if (buf[0] == '\n' || buf[bytes-1] == '\n') {

// NOTE: When Enter is pressed with no input, then exit

break;

}

}

// NOTE: Get full path to the directory, where program resides

char progpath[1024];

{

// NOTE: Read full program path, including its name

ssize\_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);

if (len == -1) {

const char msg[] = "error: failed to read full program path\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

// NOTE: Trim the path to first slash from the end

while (progpath[len] != '/')

--len;

progpath[len] = '\0';

}

// NOTE: Open pipe

int channel[2];

if (pipe(channel) == -1) {

const char msg[] = "error: failed to create pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

// NOTE: Spawn a new process

const pid\_t child = fork();

switch (child) {

case -1: { // NOTE: Kernel fails to create another process

const char msg[] = "error: failed to spawn new process\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

} break;

case 0: { // NOTE: We're a child, child doesn't know its pid after fork

pid\_t pid = getpid(); // NOTE: Get child PID

//printf("We got child PID: %d\n", pid);

// NOTE: Connect parent stdin to child stdin

dup2(STDIN\_FILENO, channel[STDIN\_FILENO]);

close(channel[STDOUT\_FILENO]);

{

char msg[64];

const int32\_t length = snprintf(msg, sizeof(msg), "%d: I'm a child\n", pid);

write(STDOUT\_FILENO, msg, length);

//printf("We wrote: I'm a child!\n");

}

{

//printf("Client program execution...\n");

char path[1024];

//snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT\_PROGRAM\_NAME);

snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT\_PROGRAM\_NAME) < 0 ? abort() : (void)0;

// NOTE: args[0] must be a program name, next the actual arguments

// NOTE: `NULL` at the end is mandatory, because `exec\*`

// expects a NULL-terminated list of C-strings

char \*const args[] = {CLIENT\_PROGRAM\_NAME, argv[1], buf, NULL};

int32\_t status = execv(path, args);

if (status == -1) {

const char msg[] = "error: failed to exec into new exectuable image\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

//printf("Client program executied!\n");

}

} break;

default: { // NOTE: We're a parent, parent knows PID of child after fork

pid\_t pid = getpid(); // NOTE: Get parent PID

{

char msg[64];

const int32\_t length = snprintf(msg, sizeof(msg),

"%d: I'm a parent, my child has PID %d\n", pid, child);

write(STDOUT\_FILENO, msg, length);

}

// NOTE: `wait` blocks the parent until child exits

int child\_status;

wait(&child\_status);

if (child\_status != EXIT\_SUCCESS) {

const char msg[] = "error: child exited with error\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(child\_status);

}

} break;

}

}

**client\_prog.c**

#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

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

char buf[4096];

ssize\_t bytes;

// Open file for saving result

// NOTE: `O\_WRONLY` only enables file for writing

// NOTE: `O\_CREAT` creates the requested file if absent

// NOTE: `O\_TRUNC` empties the file prior to opening

// NOTE: `O\_APPEND` subsequent writes are being appended instead of overwritten

int32\_t file = open(argv[1], O\_WRONLY | O\_CREAT | O\_TRUNC | O\_APPEND, 0600);

if (file == -1) {

const char msg[] = "error: failed to open requested file\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

pid\_t pid = getpid();

char messg[300];

int32\_t len = snprintf(messg, sizeof(messg), "%d: Solution of expression (", pid);

write(STDERR\_FILENO, messg, len);

write(file, messg, len);

//преобразование аргументов программы в числа

int i;

for (i = 2; i<argc; i++)

{

//printf("%s\n", argv[i]);

//const char messg[] = "Solution of expression (";

//write(STDERR\_FILENO, messg, sizeof(messg));

char\* str = argv[i];

// Указатель, на непреобразованный остаток строки str,

// в начале указывает на начало строки str

char\* nstr = str;

// Переменная для сохранения результата преобразования

float d = 0, res = 0;

// Организуем бесконечный цикл

int i = 0;

while (++i)

{

d = strtof(nstr, &nstr);

if (d==0) break;

if (res == 0)

res = d;

else

res = res / d;

char msg[32];

int32\_t len;

if (i==1)

len = snprintf(msg, sizeof(msg) - 1, "%0.3f", d);

else

len = snprintf(msg, sizeof(msg) - 1, " / %0.3f", d);

write(STDERR\_FILENO, msg, len);

write(file, msg, len);

//printf("%0.3f / ", d);

}

int32\_t len = snprintf(messg, sizeof(messg) - 1, ") is %0.3f\n",res);

write(STDERR\_FILENO, messg, len);

//printf ("\nresult = %0.3f\n", res);

int32\_t written = write(file, messg, len);

if (written != len) {

const char msg[] = "error: failed to write to file\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

}

}

**Протокол работы программы**

**Тестирование:**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab01$ ./a.out res.txt

7885: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit

15 2 1.2 3.2 0.01

7885: I'm a parent, my child has PID 7886

7886: I'm a child

7886: Solution of expression (15.000 / 2.000 / 3.100 / 1.800 / 0.010) is 134.409

**Strace:**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab01$ strace -f ./a.out res.txt

execve("./a.out", ["./a.out", "res.txt"], 0x7ffd58fab160 /\* 46 vars \*/) = 0

brk(NULL) = 0x55b78d066000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe46c697b0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f790ef09000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=64091, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 64091, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f790eef9000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-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>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\244;\374\204(\337f#\315I\214\234\f\256\271\32"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, AT\_EMPTY\_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f790ec00000

mmap(0x7f790ec28000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f790ec28000

mmap(0x7f790edbd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f790edbd000

mmap(0x7f790ee15000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7f790ee15000

mmap(0x7f790ee1b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f790ee1b000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f790eef6000

arch\_prctl(ARCH\_SET\_FS, 0x7f790eef6740) = 0

set\_tid\_address(0x7f790eef6a10) = 7885

set\_robust\_list(0x7f790eef6a20, 24) = 0

rseq(0x7f790eef70e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f790ee15000, 16384, PROT\_READ) = 0

mprotect(0x55b78b130000, 4096, PROT\_READ) = 0

mprotect(0x7f790ef43000, 8192, PROT\_READ) = 0

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

munmap(0x7f790eef9000, 64091) = 0

getpid() = 7885

write(1, "7885: Start typing row of number"..., 827885: Start typing row of number. Press 'Ctrl-D' or 'Enter' with no input to exit

) = 82

read(0, 15 2 3.1 1.8 0.01

"15 2 3.1 1.8 0.01\n", 4096) = 18

readlink("/proc/self/exe", "/home/irina/Prog/Prog\_C/Kurs2/La"..., 1023) = 47

pipe2([3, 4], 0) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 7886 attached

, child\_tidptr=0x7f790eef6a10) = 7886

[pid 7886] set\_robust\_list(0x7f790eef6a20, 24 <unfinished ...>

[pid 7885] getpid( <unfinished ...>

[pid 7886] <... set\_robust\_list resumed>) = 0

[pid 7885] <... getpid resumed>) = 7885

[pid 7885] write(1, "7885: I'm a parent, my child has"..., 42 <unfinished ...>

[pid 7886] getpid(7885: I'm a parent, my child has PID 7886

<unfinished ...>

[pid 7885] <... write resumed>) = 42

[pid 7886] <... getpid resumed>) = 7886

[pid 7885] wait4(-1, <unfinished ...>

[pid 7886] dup2(0, 3) = 3

[pid 7886] close(4) = 0

[pid 7886] write(1, "7886: I'm a child\n", 187886: I'm a child

) = 18

[pid 7886] execve("/home/irina/Prog/Prog\_C/Kurs2/LabOS/Lab01/client\_prog", ["client\_prog", "res.txt", "15 2 3.1 1.8 0.01\n row of number"...], 0x7ffe46c69990 /\* 46 vars \*/) = 0

[pid 7886] brk(NULL) = 0x56082ff67000

[pid 7886] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffde31f8000) = -1 EINVAL (Недопустимый аргумент)

[pid 7886] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff2c6b9d000

[pid 7886] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

[pid 7886] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 4

[pid 7886] newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=64091, ...}, AT\_EMPTY\_PATH) = 0

[pid 7886] mmap(NULL, 64091, PROT\_READ, MAP\_PRIVATE, 4, 0) = 0x7ff2c6b8d000

[pid 7886] close(4) = 0

[pid 7886] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 4

[pid 7886] read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

[pid 7886] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 7886] pread64(4, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

[pid 7886] pread64(4, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\244;\374\204(\337f#\315I\214\234\f\256\271\32"..., 68, 896) = 68

[pid 7886] newfstatat(4, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, AT\_EMPTY\_PATH) = 0

[pid 7886] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 7886] mmap(NULL, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0) = 0x7ff2c6800000

[pid 7886] mmap(0x7ff2c6828000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x28000) = 0x7ff2c6828000

[pid 7886] mmap(0x7ff2c69bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1bd000) = 0x7ff2c69bd000

[pid 7886] mmap(0x7ff2c6a15000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x214000) = 0x7ff2c6a15000

[pid 7886] mmap(0x7ff2c6a1b000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff2c6a1b000

[pid 7886] close(4) = 0

[pid 7886] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff2c6b8a000

[pid 7886] arch\_prctl(ARCH\_SET\_FS, 0x7ff2c6b8a740) = 0

[pid 7886] set\_tid\_address(0x7ff2c6b8aa10) = 7886

[pid 7886] set\_robust\_list(0x7ff2c6b8aa20, 24) = 0

[pid 7886] rseq(0x7ff2c6b8b0e0, 0x20, 0, 0x53053053) = 0

[pid 7886] mprotect(0x7ff2c6a15000, 16384, PROT\_READ) = 0

[pid 7886] mprotect(0x56082f626000, 4096, PROT\_READ) = 0

[pid 7886] mprotect(0x7ff2c6bd7000, 8192, PROT\_READ) = 0

[pid 7886] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 7886] munmap(0x7ff2c6b8d000, 64091) = 0

[pid 7886] openat(AT\_FDCWD, "res.txt", O\_WRONLY|O\_CREAT|O\_TRUNC|O\_APPEND, 0600) = 4

[pid 7886] getpid() = 7886

[pid 7886] write(2, "7886: Solution of expression (", 307886: Solution of expression () = 30

[pid 7886] write(4, "7886: Solution of expression (", 30) = 30

[pid 7886] write(2, "15.000", 615.000) = 6

[pid 7886] write(4, "15.000", 6) = 6

[pid 7886] write(2, " / 2.000", 8 / 2.000) = 8

[pid 7886] write(4, " / 2.000", 8) = 8

[pid 7886] write(2, " / 3.100", 8 / 3.100) = 8

[pid 7886] write(4, " / 3.100", 8) = 8

[pid 7886] write(2, " / 1.800", 8 / 1.800) = 8

[pid 7886] write(4, " / 1.800", 8) = 8

[pid 7886] write(2, " / 0.010", 8 / 0.010) = 8

[pid 7886] write(4, " / 0.010", 8) = 8

[pid 7886] write(2, ") is 134.409\n", 13) is 134.409

) = 13

[pid 7886] write(4, ") is 134.409\n", 13) = 13

[pid 7886] exit\_group(0) = ?

[pid 7886] +++ exited with 0 +++

<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 7886

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=7886, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0) = ?

+++ exited with 0 +++

**res.txt:**

7886: Solution of expression (15.000 / 2.000 / 3.100 / 1.800 / 0.010) is 134.409

**Вывод**

**В ходе лабораторной работы я научилась создавать процессы, каналы и другие конструкции. Таким образом, программа иллюстрирует использование базовых системных вызовов для создания процессов и организации их работы с файлами и межпроцессным взаимодействием с помощью каналов.**