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: Рожков И.С.

Преподаватель: Бахарев В.Д.

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

Дата: 15.10.24

Москва, 2024

**Постановка задачи**

**Вариант 21.**

**Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.**

**Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.**

**Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
* pid\_t getpid(void); – возвращает ID вызывающего процесса.
* int open(const char \*\_\_file, int \_\_oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
* ssize\_t write(int \_\_fd, const void \*\_\_buf, size\_t \_\_n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
* void exit(int \_\_status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
* int close(int \_\_fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
* int dup2(int \_\_fd, int \_\_fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
* int execv(const char \*\_\_path, char \*const \*\_\_argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
* ssize\_t read(int \_\_fd, void \*\_\_buf, size\_t \_\_nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
* pid\_t wait(int \*\_\_stat\_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа server.c получает на вход два аргумента – пути к файлам, в которые требуется записать результат работы. С помощью readlink() сохраняем полный путь до файла. После создаём два канала с помощью pipe для общения с двумя дочерними процессами. Далее выполняется fork() для создания первого дочернего процесса, с помощью конструкции switch/case определяем в каком процессе мы находимся.

Если процесс дочерний, то используем dup2() для копирования файлового дескриптора канала и с помощью execv() подменяем образ текущего процесса на новый(client).

Если процесс – родитель, то делаем ещё один fork(), далее повторяем те же действия, если мы в дочернем процессе. Если же мы родитель, то начинаем читать строки из потока ввода и по очереди передавать то первому дочернему процессу, то второму. После окончания ввода ждём завершения обоих дочерних процессов и программа завершается.

Программа client открывает переданный в качестве аргумента файл, после этого считывает строки из потока ввода(подменён на вывод канала сервера), переворачивает и записывает в открытый файл. При окончании ввода строк файл закрывается, программа завершается.

**Код программы**

**server.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";

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

{

if (argc == 1)

{

char msg[1024];

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

write(STDERR\_FILENO, msg, len);

exit(EXIT\_SUCCESS);

}

char progpath[1024];

{

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);

}

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

--len;

progpath[len] = '\0';

}

int channel\_1[2], channel\_2[2];

if (pipe(channel\_1) == -1)

{

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

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

exit(EXIT\_FAILURE);

}

if (pipe(channel\_2) == -1)

{

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

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

exit(EXIT\_FAILURE);

}

const pid\_t child\_1 = fork();

switch (child\_1)

{

case -1:

{

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

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

exit(EXIT\_FAILURE);

}

break;

case 0:

{

pid\_t pid = getpid();

if (dup2(channel\_1[STDIN\_FILENO], STDIN\_FILENO) == -1)

{

const char msg[128];

snprintf(msg, sizeof(msg), "%d: failed to use dup2\n", pid);

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

exit(EXIT\_FAILURE);

}

if (close(channel\_1[STDOUT\_FILENO]) == -1)

{

const char msg[128];

snprintf(msg, sizeof(msg), "%d: failed to close pipe\n", pid);

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

exit(EXIT\_FAILURE);

}

{

char msg[64];

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

"%d: I'm a child1\n", pid);

write(STDOUT\_FILENO, msg, length);

}

{

char path[1024];

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

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

}

}

}

break;

default:

{

const pid\_t child\_2 = fork();

switch (child\_2)

{

case -1:

{

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

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

exit(EXIT\_FAILURE);

}

break;

case 0:

{

pid\_t pid = getpid();

if (dup2(channel\_2[STDIN\_FILENO], STDIN\_FILENO) == -1)

{

const char msg[128];

snprintf(msg, sizeof(msg), "%d: failed to use dup2\n", pid);

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

exit(EXIT\_FAILURE);

}

if (close(channel\_2[STDOUT\_FILENO]) == -1)

{

const char msg[128];

snprintf(msg, sizeof(msg), "%d: failed to close pipe\n", pid);

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

exit(EXIT\_FAILURE);

}

{

char msg[64];

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

"%d: I'm a child2\n", pid);

write(STDOUT\_FILENO, msg, length);

}

{

char path[1024];

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

char \*const args[] = {CLIENT\_PROGRAM\_NAME, argv[2], 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);

}

}

}

break;

default:

{

pid\_t pid = getpid();

{

char msg[128];

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

"%d: I'm a parent, my child1 & child2 has PID %d %d\n", pid, child\_1, child\_2);

write(STDOUT\_FILENO, msg, length);

}

if (close(channel\_1[STDIN\_FILENO]) == -1 || close(channel\_2[STDIN\_FILENO]) == -1)

{

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

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

exit(EXIT\_FAILURE);

}

char buf[4096];

ssize\_t bytes;

int odd = 1;

{

const char msg[] = "Input strings:\n";

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

}

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')

{

break;

}

{

buf[bytes - 1] = '\0';

int32\_t written;

if (odd)

written = write(channel\_1[STDOUT\_FILENO], buf, bytes);

else

written = write(channel\_2[STDOUT\_FILENO], buf, bytes);

odd = abs(odd - 1);

if (written != bytes)

{

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

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

exit(EXIT\_FAILURE);

}

}

}

buf[0] = '\n';

if (write(channel\_1[STDOUT\_FILENO], buf, sizeof(char)) == -1 || write(channel\_2[STDOUT\_FILENO], buf, sizeof(char)) == -1)

{

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

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

exit(EXIT\_FAILURE);

}

if (close(channel\_1[STDOUT\_FILENO]) == -1 || close(channel\_2[STDOUT\_FILENO] == -1))

{

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

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

exit(EXIT\_FAILURE);

}

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

int child\_status;

pid\_t wpid;

while ((wpid = wait(&child\_status) > 0))

{

if (child\_status != EXIT\_SUCCESS)

{

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

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

exit(child\_status);

}

}

}

break;

}

break;

}

}

return 0;

}

**client.c**

#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

void str\_reverse(char \*str)

{

int len = strlen(str);

for (int i = 0; i < len / 2; ++i)

{

char temp = str[i];

str[i] = str[len - 1 - i];

str[len - 1 - i] = temp;

}

}

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

{

char buf[4096];

ssize\_t bytes;

pid\_t pid = getpid();

// 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);

}

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')

{

break;

}

{

buf[bytes - 1] = '\0';

str\_reverse(buf);

buf[bytes - 1] = '\n';

int32\_t written = write(file, buf, bytes);

if (written != bytes)

{

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

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

exit(EXIT\_FAILURE);

}

}

}

if (close(file) == -1)

{

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

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

exit(EXIT\_FAILURE);

}

return 0;

}

**Протокол работы программы**

$ ./server f1.txt f2.txt

295436: I'm a parent, my child1 & child2 has PID 295437 295438

Input strings:

295437: I'm a child1

295438: I'm a child2

string 1

string 2

string 3

string 4

string 5

string 6

last string

$ cat f1.txt

1 gnirts

3 gnirts

5 gnirts

gnirts tsal

$ cat f2.txt

2 gnirts

4 gnirts

6 gnirts

$ strace -f ./server f1.txt f2.txt

execve("./server", ["./server", "f1.txt", "f2.txt"], 0x7fff3983ce38 /\* 35 vars \*/) = 0

brk(NULL) = 0x55b7f5969000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdafaca6d0) = -1 EINVAL (Invalid argument)

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

0x7fe6157c6000

access("/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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=17915, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 17915, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fe6157c1000

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\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68,

896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, 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, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fe615598000

mprotect(0x7fe6155c0000, 2023424, PROT\_NONE) = 0

mmap(0x7fe6155c0000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE,

3, 0x28000) = 0x7fe6155c0000

mmap(0x7fe615755000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x1bd000) = 0x7fe615755000

mmap(0x7fe6157ae000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE,

3, 0x215000) = 0x7fe6157ae000

mmap(0x7fe6157b4000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS,

-1, 0) = 0x7fe6157b4000

close(3) = 0

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

0x7fe615595000

arch\_prctl(ARCH\_SET\_FS, 0x7fe615595740) = 0

set\_tid\_address(0x7fe615595a10) = 296697

set\_robust\_list(0x7fe615595a20, 24) = 0

rseq(0x7fe6155960e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fe6157ae000, 16384, PROT\_READ) = 0

mprotect(0x55b7f4cdb000, 4096, PROT\_READ) = 0

mprotect(0x7fe615800000, 8192, PROT\_READ) = 0

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

munmap(0x7fe6157c1000, 17915) = 0

readlink("/proc/self/exe", "/home/empress/OS\_labs/lab\_1/serv"..., 1023) = 34

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

**pipe2([5, 6], 0) = 0**

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

**Process 296698 attached**

**, child\_tidptr=0x7fe615595a10) = 296698**

[pid 296698] set\_robust\_list(0x7fe615595a20, 24 <unfinished ...>

[pid 296697] clone(child\_stack=NULL,

flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD <unfinished ...>

[pid 296698] <... set\_robust\_list resumed>) = 0

**[pid 296698] getpid(strace: Process 296699 attached**

**) = 296698**

[pid 296697] <... clone resumed>, child\_tidptr=0x7fe615595a10) = 296699

[pid 296699] set\_robust\_list(0x7fe615595a20, 24 <unfinished ...>

**[pid 296697] getpid( <unfinished ...>**

**[pid 296698] dup2(3, 0 <unfinished ...>**

[pid 296699] <... set\_robust\_list resumed>) = 0

[pid 296697] <... getpid resumed>) = 296697

[pid 296698] <... dup2 resumed>) = 0

[pid 296697] write(1, "296697: I'm a parent, my child1 "..., 63 <unfinished ...>

[pid 296699] getpid(296697: I'm a parent, my child1 & child2 has PID 296698 296699

<unfinished ...>

[pid 296697] <... write resumed>) = 63

**[pid 296698] close(4 <unfinished ...>**

**[pid 296697] close(3 <unfinished ...>**

[pid 296699] <... getpid resumed>) = 296699

[pid 296697] <... close resumed>) = 0

[pid 296698] <... close resumed>) = 0

**[pid 296697] close(5 <unfinished ...>**

**[pid 296699] dup2(5, 0 <unfinished ...>**

**[pid 296697] <... close resumed>) = 0**

**[pid 296699] <... dup2 resumed>) = 0**

[pid 296698] write(1, "296698: I'm a child1\n", 21 <unfinished ...>

296698: I'm a child1

[pid 296697] write(1, "Input strings:\n\0", 16 <unfinished ...>

Input strings:

[pid 296699] close(6 <unfinished ...>

[pid 296697] <... write resumed>) = 16

[pid 296698] <... write resumed>) = 21

[pid 296697] read(0, <unfinished ...>

[pid 296699] <... close resumed>) = 0

[pid 296699] write(1, "296699: I'm a child2\n", 21 <unfinished ...>

296699: I'm a child2

[pid 296698] execve("/home/empress/OS\_labs/lab\_1/client", ["client", "f1.txt"],

0x7ffdafaca8b8 /\* 35 vars \*/ <unfinished ...>

[pid 296699] <... write resumed>) = 21

[pid 296699] execve("/home/empress/OS\_labs/lab\_1/client", ["client", "f2.txt"],

0x7ffdafaca8b8 /\* 35 vars \*/ <unfinished ...>

[pid 296698] <... execve resumed>) = 0

[pid 296699] <... execve resumed>) = 0

[pid 296699] brk(NULL <unfinished ...>

[pid 296698] brk(NULL <unfinished ...>

[pid 296699] <... brk resumed>) = 0x556e59557000

[pid 296698] <... brk resumed>) = 0x558eb5c48000

[pid 296699] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffd492f8230) = -1 EINVAL (Invalid

argument)

[pid 296698] arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffced657770 <unfinished ...>

[pid 296699] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0

<unfinished ...>

[pid 296698] <... arch\_prctl resumed>) = -1 EINVAL (Invalid argument)

[pid 296699] <... mmap resumed>) = 0x7fc1cdee1000

[pid 296698] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0

<unfinished ...>

[pid 296699] access("/etc/ld.so.preload", R\_OK <unfinished ...>

[pid 296698] <... mmap resumed>) = 0x7fd228278000

[pid 296699] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 296698] access("/etc/ld.so.preload", R\_OK <unfinished ...>

[pid 296699] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 296698] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 296699] <... openat resumed>) = 6

[pid 296699] newfstatat(6, "", <unfinished ...>

[pid 296698] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 296699] <... newfstatat resumed>{st\_mode=S\_IFREG|0644, st\_size=17915, ...},

AT\_EMPTY\_PATH) = 0

[pid 296698] <... openat resumed>) = 4

[pid 296699] mmap(NULL, 17915, PROT\_READ, MAP\_PRIVATE, 6, 0 <unfinished ...>

[pid 296698] newfstatat(4, "", <unfinished ...>

[pid 296699] <... mmap resumed>) = 0x7fc1cdedc000

[pid 296698] <... newfstatat resumed>{st\_mode=S\_IFREG|0644, st\_size=17915, ...},

AT\_EMPTY\_PATH) = 0

**[pid 296699] close(6 <unfinished ..**.>

[pid 296698] mmap(NULL, 17915, PROT\_READ, MAP\_PRIVATE, 4, 0 <unfinished ...>

[pid 296699] <... close resumed>) = 0

[pid 296698] <... mmap resumed>) = 0x7fd228273000

[pid 296699] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC

<unfinished ...>

**[pid 296698] close(4 <unfinished ...>**

[pid 296699] <... openat resumed>) = 6

[pid 296698] <... close resumed>) = 0

[pid 296699] read(6, <unfinished ...>

[pid 296698] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC

<unfinished ...>

[pid 296699] <... read

resumed>"\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 296698] <... openat resumed>) = 4

[pid 296699] pread64(6,

"\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 296698] read(4, <unfinished ...>

[pid 296699] pread64(6, <unfinished ...>

[pid 296698] <... read

resumed>"\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 296699] <... pread64 resumed>"\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 296698] pread64(4, <unfinished ...>

[pid 296699] pread64(6, <unfinished ...>

[pid 296698] <... pread64

resumed>"\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 296699] <... pread64

resumed>"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...

, 68, 896) = 68

[pid 296698] pread64(4, <unfinished ...>

[pid 296699] newfstatat(6, "", <unfinished ...>

[pid 296698] <... pread64 resumed>"\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 296699] <... newfstatat resumed>{st\_mode=S\_IFREG|0755, st\_size=2220400, ...},

AT\_EMPTY\_PATH) = 0

[pid 296698] pread64(4, <unfinished ...>

[pid 296699] pread64(6, <unfinished ...>

[pid 296698] <... pread64

resumed>"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...

, 68, 896) = 68

[pid 296699] <... pread64

resumed>"\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 296698] newfstatat(4, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...},

AT\_EMPTY\_PATH) = 0

[pid 296698] pread64(4, <unfinished ...>

[pid 296699] mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 6, 0 <unfinished ...>

[pid 296698] <... pread64

resumed>"\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 296699] <... mmap resumed>) = 0x7fc1cdcb3000

[pid 296698] mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 4, 0 <unfinished ...>

[pid 296699] mprotect(0x7fc1cdcdb000, 2023424, PROT\_NONE <unfinished ...>

[pid 296698] <... mmap resumed>) = 0x7fd22804a000

[pid 296699] <... mprotect resumed>) = 0

[pid 296698] mprotect(0x7fd228072000, 2023424, PROT\_NONE <unfinished ...>

[pid 296699] mmap(0x7fc1cdcdb000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x28000 <unfinished ...>

[pid 296698] <... mprotect resumed>) = 0

[pid 296699] <... mmap resumed>) = 0x7fc1cdcdb000

[pid 296698] mmap(0x7fd228072000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x28000 <unfinished ...>

[pid 296699] mmap(0x7fc1cde70000, 360448, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x1bd000 <unfinished ...>

[pid 296698] <... mmap resumed>) = 0x7fd228072000

[pid 296699] <... mmap resumed>) = 0x7fc1cde70000

[pid 296698] mmap(0x7fd228207000, 360448, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x1bd000 <unfinished ...>

[pid 296699] mmap(0x7fc1cdec9000, 24576, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 6, 0x215000 <unfinished ...>

[pid 296698] <... mmap resumed>) = 0x7fd228207000

[pid 296699] <... mmap resumed>) = 0x7fc1cdec9000

[pid 296698] mmap(0x7fd228260000, 24576, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 4, 0x215000 <unfinished ...>

[pid 296699] mmap(0x7fc1cdecf000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 296698] <... mmap resumed>) = 0x7fd228260000

[pid 296699] <... mmap resumed>) = 0x7fc1cdecf000

[pid 296698] mmap(0x7fd228266000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

**[pid 296699] close(6 <unfinished ...>**

[pid 296698] <... mmap resumed>) = 0x7fd228266000

[pid 296699] <... close resumed>) = 0

**[pid 296698] close(4 <unfinished ...>**

[pid 296699] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 296698] <... close resumed>) = 0

[pid 296699] <... mmap resumed>) = 0x7fc1cdcb0000

[pid 296699] arch\_prctl(ARCH\_SET\_FS, 0x7fc1cdcb0740 <unfinished ...>

[pid 296698] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 296699] <... arch\_prctl resumed>) = 0

[pid 296698] <... mmap resumed>) = 0x7fd228047000

[pid 296699] set\_tid\_address(0x7fc1cdcb0a10) = 296699

[pid 296698] arch\_prctl(ARCH\_SET\_FS, 0x7fd228047740 <unfinished ...>

[pid 296699] set\_robust\_list(0x7fc1cdcb0a20, 24) = 0

[pid 296698] <... arch\_prctl resumed>) = 0

[pid 296699] rseq(0x7fc1cdcb10e0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 296698] set\_tid\_address(0x7fd228047a10 <unfinished ...>

[pid 296699] <... rseq resumed>) = 0

[pid 296698] <... set\_tid\_address resumed>) = 296698

[pid 296699] mprotect(0x7fc1cdec9000, 16384, PROT\_READ <unfinished ...>

[pid 296698] set\_robust\_list(0x7fd228047a20, 24 <unfinished ...>

[pid 296699] <... mprotect resumed>) = 0

[pid 296698] <... set\_robust\_list resumed>) = 0

[pid 296699] mprotect(0x556e593da000, 4096, PROT\_READ <unfinished ...>

[pid 296698] rseq(0x7fd2280480e0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 296699] <... mprotect resumed>) = 0

[pid 296699] mprotect(0x7fc1cdf1b000, 8192, PROT\_READ <unfinished ...>

[pid 296698] <... rseq resumed>) = 0

[pid 296699] <... mprotect resumed>) = 0

[pid 296699] prlimit64(0, RLIMIT\_STACK, NULL, <unfinished ...>

[pid 296698] mprotect(0x7fd228260000, 16384, PROT\_READ <unfinished ...>

[pid 296699] <... prlimit64 resumed>{rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 296698] <... mprotect resumed>) = 0

[pid 296698] mprotect(0x558eb4dd5000, 4096, PROT\_READ <unfinished ...>

[pid 296699] munmap(0x7fc1cdedc000, 17915 <unfinished ...>

[pid 296698] <... mprotect resumed>) = 0

[pid 296699] <... munmap resumed>) = 0

[pid 296698] mprotect(0x7fd2282b2000, 8192, PROT\_READ <unfinished ...>

[pid 296699] getpid( <unfinished ...>

[pid 296698] <... mprotect resumed>) = 0

[pid 296698] prlimit64(0, RLIMIT\_STACK, NULL, <unfinished ...>

[pid 296699] <... getpid resumed>) = 296699

[pid 296698] <... prlimit64 resumed>{rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 296699] openat(AT\_FDCWD, "f2.txt", O\_WRONLY|O\_CREAT|O\_TRUNC|O\_APPEND, 0600 <unfinished ...>

[pid 296698] munmap(0x7fd228273000, 17915) = 0

[pid 296699] <... openat resumed>) = 6

[pid 296699] read(0, <unfinished ...>

[pid 296698] getpid() = 296698

[pid 296698] openat(AT\_FDCWD, "f1.txt", O\_WRONLY|O\_CREAT|O\_TRUNC|O\_APPEND, 0600) = 4

[pid 296698] read(0, string 1

<unfinished ...>

[pid 296697] <... read resumed>"string 1\n", 4096) = 9

[pid 296697] write(4, "string 1\0", 9) = 9

[pid 296698] <... read resumed>"string 1\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296698] write(4, "1 gnirts\n", 9) = 9

[pid 296698] read(0, string 2

<unfinished ...>

[pid 296697] <... read resumed>"string 2\n", 4096) = 9

[pid 296697] write(6, "string 2\0", 9) = 9

[pid 296699] <... read resumed>"string 2\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296699] write(6, "2 gnirts\n", 9) = 9

[pid 296699] read(0, string 3

<unfinished ...>

[pid 296697] <... read resumed>"string 3\n", 4096) = 9

[pid 296697] write(4, "string 3\0", 9) = 9

[pid 296698] <... read resumed>"string 3\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296698] write(4, "3 gnirts\n", 9) = 9

[pid 296698] read(0, string 4

<unfinished ...>

[pid 296697] <... read resumed>"string 4\n", 4096) = 9

[pid 296697] write(6, "string 4\0", 9) = 9

[pid 296699] <... read resumed>"string 4\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296699] write(6, "4 gnirts\n", 9) = 9

[pid 296699] read(0, string 5

<unfinished ...>

[pid 296697] <... read resumed>"string 5\n", 4096) = 9

[pid 296697] write(4, "string 5\0", 9) = 9

[pid 296698] <... read resumed>"string 5\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296698] write(4, "5 gnirts\n", 9) = 9

[pid 296698] read(0, string 6

<unfinished ...>

[pid 296697] <... read resumed>"string 6\n", 4096) = 9

[pid 296697] write(6, "string 6\0", 9) = 9

[pid 296699] <... read resumed>"string 6\0", 4096) = 9

[pid 296697] read(0, <unfinished ...>

[pid 296699] write(6, "6 gnirts\n", 9) = 9

[pid 296699] read(0, last string

<unfinished ...>

[pid 296697] <... read resumed>"last string\n", 4096) = 12

[pid 296697] write(4, "last string\0", 12) = 12

[pid 296698] <... read resumed>"last string\0", 4096) = 12

[pid 296697] read(0, <unfinished ...>

[pid 296698] write(4, "gnirts tsal\n", 12) = 12

[pid 296698] read(0,

<unfinished ...>

[pid 296697] <... read resumed>"\n", 4096) = 1

[pid 296697] write(4, "\n", 1) = 1

[pid 296698] <... read resumed>"\n", 4096) = 1

[pid 296697] write(6, "\n", 1 <unfinished ...>

**[pid 296698] close(4 <unfinished ...>**

[pid 296697] <... write resumed>) = 1

[pid 296699] <... read resumed>"\n", 4096) = 1

[pid 296698] <... close resumed>) = 0

**[pid 296697] close(4 <unfinished ...>**

**[pid 296699] close(6 <unfinished ...>**

[pid 296697] <... close resumed>) = 0

[pid 296698] exit\_group(0 <unfinished ...>

**[pid 296697] close(0 <unfinished ...>**

[pid 296699] <... close resumed>) = 0

[pid 296697] <... close resumed>) = 0

[pid 296698] <... exit\_group resumed>) = ?

**[pid 296697] wait4(-1, <unfinished ...>**

[pid 296699] exit\_group(0 <unfinished ...>

[pid 296698] +++ exited with 0 +++

[pid 296699] <... exit\_group resumed>) = ?

[pid 296697] <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 296698

[pid 296699] +++ exited with 0 +++

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=296698, si\_uid=1000,

si\_status=0, si\_utime=1, si\_stime=0} ---

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 296699

wait4(-1, 0x7ffdafac92b4, 0, NULL) = -1 ECHILD (No child processes)

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе написания данной лабораторной работы я научился работать с системными вызовами в СИ. Научился создавать программы, состоящие из нескольких процессов, и передавать данные между процессами по каналам. Во время отладки программы я познакомился с утилитой strace, она оказалась достаточно удобной для получения информации о работе многопоточных программ. Лабораторная работа была довольно интересна, так как я раньше не создавал программы на СИ, которые запускают несколько процессов параллельно.