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-216БВ-24

Студент: Попов К.А.

Преподаватель: Бахарев В.Д.

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

Дата: 06.10.25

Москва, 2025

**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:

* ssize\_t write(int fd, const void \*buf, size\_t n) - пишет N байт из BUF в дескриптор FD. Возвращает количество записанных байт или -1 в случае ошибки.
* ssize\_t read(int fd, void \*buf, size\_t nbytes) - читает NBYTES байт, записывая в BUF из дескриптора FD. Возвращает число прочитанных байт, -1 в случае ошибки и 0 для EOF.
* pid\_t fork(void) клонирует вызывающий процесс, создав его точную копию. Возвращает -1 в случае ошибки, 0 для нового процесса и pid нового процесса для старого процесса.
* int pipe(int \*pipedes) - создаёт односторонний канал связи (pipe). В случае успеха два файловых дескриптора сохраняются в PIPEDES; байты, записанные в PIPEDES[1], могут быть прочитаны из PIPEDES[0]. Возвращает 0 в случае успеха, -1 в противном случае.
* int execv(const char \*path, char \*const \*argv) - выполнить PATH с аргументами ARGV. В случае успеха происходит замена образа памяти процесса, иначе возвращает -1.
* pid\_t getpid(void) - возвращает pid вызывающего процесса.
* pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options) - ожидает завершения дочернего процесса с соответствующим pid.
* void exit(int status) - завершает выполнение процесса с возвращением соответствующего статуса.
* int dup2(int fd, int fd2) - дублирует FD в FD2, закрыв FD2 и открыв FD2 в том же файле, что и FD.
* int open(const char \*pathname, int flags, mode\_t mode) - открывает файл PATHNAME в режиме, определенном FLAGS и MODE. создание файла Возвращает файловый дескриптор.
* int close(int fd) - закрывает файловый дескриптор FD.

В рамках лабораторной работы были написаны две программы: client.c и server.c. client.c отвечает за логику работы родительского процесса и создание дочерних процессов, а server.c - за работу дочерних процессов. В файле config.h определяется размер буфера для принимаемых от пользователя строк.

Программа (client) определяет свою директорию, принимает от пользователя пути к файлам относительно своей директории. В первый файл будут записаны результаты обработки нечетных строк, во второй - результаты обработки четных строк. Программа открывает данные файлы (системный вызов open) и два pipe (системный вызов pipe) для пересылки строк дочерним процессам. Затем с помощью fork создаются два дочерних процесса. Дочерние процессы получают копии всех дескрипторов родительского процесса. Выход первого pipe перенаправляется в стандартный поток ввода первого дочернего процесса, стандартный поток вывода дочернего процесса перенаправляется в первый файл (системные вызовы dup2). Аналогично для другого дочернего процесса. Все неиспользуемые дескрипторы закрываются с помощью close. Затем управление в дочерних процессах передается коду в server.c с помощью execv. Родительский процесс закрывает неиспользуемые дескрипторы: выходы pipe и файловые дескрипторы. Он принимает произвольное количество строк от пользователя через стандартный поток ввода и отправляет нечетные первому дочернему процессу посредством одного pipe, а четные - второму посредством другого pipe. Дочерние процессы инвертируют полученные строки и записывают результаты в соответствующие файлы. Процессы завершаются при получении от пользователя пустой строки, при этом родительский процесс ожидает завершение дочерних посредством waitpid.

**Код программы**

**client.c:**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include "../include/config.h"

#define WORKDIR\_BUFFERSIZE 1024

#define FILENAME\_BUFFERSIZE 256

static char SERVER\_PROGRAM\_NAME[] = "server";

int readFilename(char \*filename, const ssize\_t size) {

    ssize\_t bytes\_read = read(STDIN\_FILENO, filename, size);

    if (bytes\_read < 0) {

        return 1;

    }

    if (bytes\_read == 0 || filename[0] == '\n') {

        return 2;

    }

    if (bytes\_read == size && filename[size - 1] != '\n') {

        return 3;

    }

    if (filename[bytes\_read - 1] == '\n') {

        filename[bytes\_read - 1] = '\0';

    } else {

        filename[bytes\_read] = '\0';

    }

    return 0;

}

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

    char workdir[WORKDIR\_BUFFERSIZE];

    {

        ssize\_t len = readlink("/proc/self/exe", workdir, WORKDIR\_BUFFERSIZE - 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 (workdir[len] != '/') {

            --len;

        }

        workdir[len] = '\0';

    }

    char filename1[FILENAME\_BUFFERSIZE];

    char filename2[FILENAME\_BUFFERSIZE];

    int errcode = readFilename(filename1, FILENAME\_BUFFERSIZE);

    if (!errcode) {

        errcode = readFilename(filename2, FILENAME\_BUFFERSIZE);

    }

    switch (errcode) {

    case 1: {

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

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

    } exit(EXIT\_FAILURE);

    case 2: {

        const char msg[] = "error: no filename\n";

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

    } exit(EXIT\_FAILURE);

    case 3: {

        const char msg[] = "error: filename too long\n";

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

    } exit(EXIT\_FAILURE);

    }

    char filepath1[WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE];

    snprintf(filepath1, WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE, "%s/%s", workdir, filename1);

    int32\_t file1d = open(filepath1, O\_WRONLY | O\_CREAT | O\_TRUNC, 0600);

    if (file1d == -1) {

        const char msg[] = "error: cannot open file1\n";

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

        exit(EXIT\_FAILURE);

    }

    char filepath2[WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE];

    snprintf(filepath2, WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE, "%s/%s", workdir, filename2);

    int32\_t file2d = open(filepath2, O\_WRONLY | O\_CREAT | O\_TRUNC, 0600);

    if (file2d == -1) {

        const char msg[] = "error: cannot open file2\n";

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

        exit(EXIT\_FAILURE);

    }

    int pipeClientToServer1[2];

    if (pipe(pipeClientToServer1) == -1) {

        const char msg[] = "error: failed to create pipe pipeClientToServer1\n";

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

        exit(EXIT\_FAILURE);

    }

    int pipeClientToServer2[2];

    if (pipe(pipeClientToServer2) == -1) {

        const char msg[] = "error: failed to create pipe pipeClientToServer2\n";

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

        exit(EXIT\_FAILURE);

    }

    const pid\_t server1\_pid = fork();

    switch (server1\_pid) {

    case -1: {

        const char msg[] = "error: failed to spawn new process server1\n";

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

    } exit(EXIT\_FAILURE);

    case 0: {

        {

            const pid\_t pid = getpid();

            char msg[64];

            const int32\_t length = snprintf(msg, sizeof(msg), "PID %d: I'm a server1\n", pid);

            write(STDOUT\_FILENO, msg, length);

        }

        close(pipeClientToServer1[1]);

        dup2(pipeClientToServer1[0], STDIN\_FILENO);

        close(pipeClientToServer1[0]);

        dup2(file1d, STDOUT\_FILENO);

        close(file1d);

        close(pipeClientToServer2[0]);

        close(pipeClientToServer2[1]);

        close(file2d);

        char serverpath[WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE];

        snprintf(serverpath, sizeof(serverpath) - 1, "%s/%s", workdir, SERVER\_PROGRAM\_NAME);

        char \*const args[] = {SERVER\_PROGRAM\_NAME, NULL};

        int32\_t status = execv(serverpath, 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);

        }

    }

    }

    const pid\_t server2\_pid = fork();

    switch (server2\_pid) {

    case -1: {

        const char msg[] = "error: failed to spawn new process server2\n";

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

    } exit(EXIT\_FAILURE);

    case 0: {

        {

            const pid\_t pid = getpid();

            char msg[64];

            const int32\_t length = snprintf(msg, sizeof(msg), "PID %d: I'm a server2\n", pid);

            write(STDOUT\_FILENO, msg, length);

        }

        close(pipeClientToServer2[1]);

        dup2(pipeClientToServer2[0], STDIN\_FILENO);

        close(pipeClientToServer2[0]);

        dup2(file2d, STDOUT\_FILENO);

        close(file2d);

        close(pipeClientToServer1[0]);

        close(pipeClientToServer1[1]);

        close(file1d);

        char serverpath[WORKDIR\_BUFFERSIZE + FILENAME\_BUFFERSIZE];

        snprintf(serverpath, sizeof(serverpath) - 1, "%s/%s", workdir, SERVER\_PROGRAM\_NAME);

        char \*const args[] = {SERVER\_PROGRAM\_NAME, NULL};

        int32\_t status = execv(serverpath, 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);

        }

    }

    }

    {

        const pid\_t pid = getpid();

        char msg[128];

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

            "PID %d: I'm a parent, my children has PID %d and %d\n", pid, server1\_pid, server2\_pid);

        write(STDOUT\_FILENO, msg, length);

    }

    close(pipeClientToServer1[0]);

    close(file1d);

    close(pipeClientToServer2[0]);

    close(file2d);

    char buf[USER\_INPUT\_BUFFERSIZE];

    ssize\_t bytes\_read;

    bool oddLine = true;

    while (bytes\_read = read(STDIN\_FILENO, buf, USER\_INPUT\_BUFFERSIZE)) {

        if (bytes\_read < 0) {

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

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

            exit(EXIT\_FAILURE);

        }

        if (buf[0] == '\n') {

            write(pipeClientToServer1[1], buf, bytes\_read);

            write(pipeClientToServer2[1], buf, bytes\_read);

            break;

        }

        if (bytes\_read == USER\_INPUT\_BUFFERSIZE && buf[bytes\_read - 1] != '\n') {

            char msg[] = "error: received line too long\n";

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

            exit(EXIT\_FAILURE);

        }

        ssize\_t bytes\_written;

        if (oddLine) {

            bytes\_written = write(pipeClientToServer1[1], buf, bytes\_read);

        } else {

            bytes\_written = write(pipeClientToServer2[1], buf, bytes\_read);

        }

        if (bytes\_written != bytes\_read) {

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

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

            exit(EXIT\_FAILURE);

        }

        oddLine = !oddLine;

    }

    close(pipeClientToServer1[1]);

    close(pipeClientToServer2[1]);

    waitpid(server1\_pid, NULL, 0);

    waitpid(server2\_pid, NULL, 0);

    {

        const char msg[] = "Parent exit successfully\n";

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

    }

    return 0;

}

**server.c:**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdint.h>

#include "../include/config.h"

#define MSG\_BUFFERSIZE 256

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

    const pid\_t pid = getpid();

    char buf[USER\_INPUT\_BUFFERSIZE];

    ssize\_t bytes\_read;

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

        if (bytes\_read < 0) {

            char msg[MSG\_BUFFERSIZE];

            const int32\_t msg\_size = snprintf(msg, MSG\_BUFFERSIZE, "error: failed to read from stdin (PID: %d)\n", pid);

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

            exit(EXIT\_FAILURE);

        }

        if (buf[0] == '\n') {

            break;

        }

        for (ssize\_t i = 0, j = bytes\_read - 2; i < j; ++i, --j) {

            char tmp = buf[i];

            buf[i] = buf[j];

            buf[j] = tmp;

        }

        ssize\_t bytes\_written = write(STDOUT\_FILENO, buf, bytes\_read);

        if (bytes\_written != bytes\_read) {

            char msg[MSG\_BUFFERSIZE];

            const int32\_t msg\_size = snprintf(msg, MSG\_BUFFERSIZE, "error: failed to write to stdout (PID: %d)\n", pid);

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

            exit(EXIT\_FAILURE);

        }

    }

    return 0;

}

**Протокол работы программы**

**Тестирование:**

root@1d8265a9b512:/workspaces/OS\_labs/lab1/build# ./client

file1.txt

file2.txt

PID 12066: I'm a server1

PID 12007: I'm a parent, my children has PID 12066 and 12067

PID 12067: I'm a server2

string1

string2

string with spaces

qwerty

123456789

Parent exit successfully

root@1d8265a9b512:/workspaces/OS\_labs/lab1/build# cat < file1.txt

1gnirts

secaps htiw gnirts

987654321

root@1d8265a9b512:/workspaces/OS\_labs/lab1/build# cat < file2.txt

2gnirts

ytrewq

**Strace:**

root@a71f0c5ef53a:/workspaces/OS\_labs/lab1/build# strace -f ./client

execve("./client", ["./client"], 0x7ffd53089c68 /\* 27 vars \*/) = 0

brk(NULL) = 0x361ce000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7690022e0000

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=33091, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 33091, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7690022d7000

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\0\20t\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

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1926232, ...}, 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, 1974096, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7690020f5000

mmap(0x76900211b000, 1400832, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x76900211b000

mmap(0x769002271000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17c000) = 0x769002271000

mmap(0x7690022c4000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1cf000) = 0x7690022c4000

mmap(0x7690022ca000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7690022ca000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7690020f2000

arch\_prctl(ARCH\_SET\_FS, 0x7690020f2740) = 0

set\_tid\_address(0x7690020f2a10) = 1820

set\_robust\_list(0x7690020f2a20, 24) = 0

rseq(0x7690020f3060, 0x20, 0, 0x53053053) = 0

mprotect(0x7690022c4000, 16384, PROT\_READ) = 0

mprotect(0x403000, 4096, PROT\_READ) = 0

mprotect(0x769002313000, 8192, PROT\_READ) = 0

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

munmap(0x7690022d7000, 33091) = 0

**readlink("/proc/self/exe", "/workspaces/OS\_labs/lab1/build/c"..., 1023) = 37**

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

read(0, 0x7ffe9b6700f0, 256) = ? ERESTARTSYS (To be restarted if SA\_RESTART is set)

--- SIGWINCH {si\_signo=SIGWINCH, si\_code=SI\_KERNEL} ---

**read(0, file1.txt**

**"file1.txt\n", 256) = 10**

**read(0, file2.txt**

**"file2.txt\n", 256) = 10**

**openat(AT\_FDCWD, "/workspaces/OS\_labs/lab1/build/file1.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 3**

**openat(AT\_FDCWD, "/workspaces/OS\_labs/lab1/build/file2.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 4**

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

**pipe2([7, 8], 0) = 0**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 2036 attached**

**, child\_tidptr=0x7690020f2a10) = 2036**

[pid 2036] set\_robust\_list(0x7690020f2a20, 24 <unfinished ...>

**[pid 1820] clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD <unfinished ...>**

[pid 2036] <... set\_robust\_list resumed>) = 0

**[pid 2036] getpid(strace: Process 2037 attached**

**<unfinished ...>**

[pid 1820] <... clone resumed>, child\_tidptr=0x7690020f2a10) = 2037

[pid 2036] <... getpid resumed>) = 2036

**[pid 1820] getpid( <unfinished ...>**

[pid 2037] set\_robust\_list(0x7690020f2a20, 24 <unfinished ...>

[pid 1820] <... getpid resumed>) = 1820

[pid 2037] <... set\_robust\_list resumed>) = 0

**[pid 2036] write(1, "PID 2036: I'm a server1\n", 24 <unfinished ...>**

PID 2036: I'm a server1

**[pid 1820] write(1, "PID 1820: I'm a parent, my child"..., 58 <unfinished ...>**

PID 1820: I'm a parent, my children has PID 2036 and 2037

**[pid 2037] getpid( <unfinished ...>**

[pid 2036] <... write resumed>) = 24

[pid 1820] <... write resumed>) = 58

[pid 2037] <... getpid resumed>) = 2037

**[pid 1820] close(5 <unfinished ...>**

**[pid 2036] close(6 <unfinished ...>**

[pid 1820] <... close resumed>) = 0

**[pid 2037] write(1, "PID 2037: I'm a server2\n", 24 <unfinished ...>**

PID 2037: I'm a server2

**[pid 1820] close(3 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 1820] <... close resumed>) = 0

[pid 2037] <... write resumed>) = 24

**[pid 1820] close(7 <unfinished ...>**

**[pid 2036] dup2(5, 0 <unfinished ...>**

[pid 1820] <... close resumed>) = 0

**[pid 2037] close(8 <unfinished ...>**

**[pid 1820] close(4 <unfinished ...>**

[pid 2036] <... dup2 resumed>) = 0

[pid 1820] <... close resumed>) = 0

[pid 2037] <... close resumed>) = 0

**[pid 1820] read(0, <unfinished ...>**

**[pid 2036] close(5 <unfinished ...>**

**[pid 2037] dup2(7, 0 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 2037] <... dup2 resumed>) = 0

**[pid 2036] dup2(3, 1 <unfinished ...>**

**[pid 2037] close(7 <unfinished ...>**

[pid 2036] <... dup2 resumed>) = 1

[pid 2037] <... close resumed>) = 0

**[pid 2036] close(3 <unfinished ...>**

**[pid 2037] dup2(4, 1 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 2037] <... dup2 resumed>) = 1

**[pid 2036] close(7 <unfinished ...>**

**[pid 2037] close(4 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 2037] <... close resumed>) = 0

**[pid 2036] close(8 <unfinished ...>**

**[pid 2037] close(5 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 2037] <... close resumed>) = 0

**[pid 2036] close(4 <unfinished ...>**

**[pid 2037] close(6 <unfinished ...>**

[pid 2036] <... close resumed>) = 0

[pid 2037] <... close resumed>) = 0

**[pid 2036] execve("/workspaces/OS\_labs/lab1/build/server", ["server"], 0x7ffe9b670768 /\* 27 vars \*/ <unfinished ...>**

**[pid 2037] close(3) = 0**

**[pid 2037] execve("/workspaces/OS\_labs/lab1/build/server", ["server"], 0x7ffe9b670768 /\* 27 vars \*/) = 0**

[pid 2036] <... execve resumed>) = 0

[pid 2037] brk(NULL <unfinished ...>

[pid 2036] brk(NULL <unfinished ...>

[pid 2037] <... brk resumed>) = 0x1af3a000

[pid 2036] <... brk resumed>) = 0x21d98000

[pid 2037] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2036] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2037] <... mmap resumed>) = 0x7e9570069000

[pid 2036] <... mmap resumed>) = 0x7ea77d6f8000

[pid 2037] access("/etc/ld.so.preload", R\_OK <unfinished ...>

[pid 2036] access("/etc/ld.so.preload", R\_OK <unfinished ...>

[pid 2037] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 2036] <... access resumed>) = -1 ENOENT (No such file or directory)

[pid 2037] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 2036] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 2037] <... openat resumed>) = 3

[pid 2037] newfstatat(3, "", <unfinished ...>

[pid 2036] <... openat resumed>) = 3

[pid 2037] <... newfstatat resumed>{st\_mode=S\_IFREG|0644, st\_size=33091, ...}, AT\_EMPTY\_PATH) = 0

[pid 2036] newfstatat(3, "", <unfinished ...>

[pid 2037] mmap(NULL, 33091, PROT\_READ, MAP\_PRIVATE, 3, 0 <unfinished ...>

[pid 2036] <... newfstatat resumed>{st\_mode=S\_IFREG|0644, st\_size=33091, ...}, AT\_EMPTY\_PATH) = 0

[pid 2037] <... mmap resumed>) = 0x7e9570060000

[pid 2036] mmap(NULL, 33091, PROT\_READ, MAP\_PRIVATE, 3, 0 <unfinished ...>

[pid 2037] close(3 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d6ef000

[pid 2037] <... close resumed>) = 0

[pid 2036] close(3 <unfinished ...>

[pid 2037] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 2036] <... close resumed>) = 0

[pid 2037] <... openat resumed>) = 3

[pid 2036] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC <unfinished ...>

[pid 2037] read(3, <unfinished ...>

[pid 2036] <... openat resumed>) = 3

[pid 2037] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

[pid 2036] read(3, <unfinished ...>

[pid 2037] pread64(3, <unfinished ...>

[pid 2036] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) = 832

[pid 2037] <... 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 2036] pread64(3, <unfinished ...>

[pid 2037] newfstatat(3, "", <unfinished ...>

[pid 2036] <... 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 2037] <... newfstatat resumed>{st\_mode=S\_IFREG|0755, st\_size=1926232, ...}, AT\_EMPTY\_PATH) = 0

[pid 2036] newfstatat(3, "", <unfinished ...>

[pid 2037] pread64(3, <unfinished ...>

[pid 2036] <... newfstatat resumed>{st\_mode=S\_IFREG|0755, st\_size=1926232, ...}, AT\_EMPTY\_PATH) = 0

[pid 2037] <... 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 2036] pread64(3, <unfinished ...>

[pid 2037] mmap(NULL, 1974096, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0 <unfinished ...>

[pid 2036] <... 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 2037] <... mmap resumed>) = 0x7e956fe7e000

[pid 2036] mmap(NULL, 1974096, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0 <unfinished ...>

[pid 2037] mmap(0x7e956fea4000, 1400832, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d50d000

[pid 2037] <... mmap resumed>) = 0x7e956fea4000

[pid 2036] mmap(0x7ea77d533000, 1400832, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000 <unfinished ...>

[pid 2037] mmap(0x7e956fffa000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17c000 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d533000

[pid 2037] <... mmap resumed>) = 0x7e956fffa000

[pid 2036] mmap(0x7ea77d689000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17c000 <unfinished ...>

[pid 2037] mmap(0x7e957004d000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1cf000 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d689000

[pid 2037] <... mmap resumed>) = 0x7e957004d000

[pid 2036] mmap(0x7ea77d6dc000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1cf000 <unfinished ...>

[pid 2037] mmap(0x7e9570053000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d6dc000

[pid 2037] <... mmap resumed>) = 0x7e9570053000

[pid 2036] mmap(0x7ea77d6e2000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2037] close(3 <unfinished ...>

[pid 2036] <... mmap resumed>) = 0x7ea77d6e2000

[pid 2037] <... close resumed>) = 0

[pid 2036] close(3 <unfinished ...>

[pid 2037] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2036] <... close resumed>) = 0

[pid 2037] <... mmap resumed>) = 0x7e956fe7b000

[pid 2036] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0 <unfinished ...>

[pid 2037] arch\_prctl(ARCH\_SET\_FS, 0x7e956fe7b740) = 0

[pid 2036] <... mmap resumed>) = 0x7ea77d50a000

[pid 2037] set\_tid\_address(0x7e956fe7ba10 <unfinished ...>

[pid 2036] arch\_prctl(ARCH\_SET\_FS, 0x7ea77d50a740 <unfinished ...>

[pid 2037] <... set\_tid\_address resumed>) = 2037

[pid 2036] <... arch\_prctl resumed>) = 0

[pid 2037] set\_robust\_list(0x7e956fe7ba20, 24 <unfinished ...>

[pid 2036] set\_tid\_address(0x7ea77d50aa10 <unfinished ...>

[pid 2037] <... set\_robust\_list resumed>) = 0

[pid 2036] <... set\_tid\_address resumed>) = 2036

[pid 2037] rseq(0x7e956fe7c060, 0x20, 0, 0x53053053 <unfinished ...>

[pid 2036] set\_robust\_list(0x7ea77d50aa20, 24 <unfinished ...>

[pid 2037] <... rseq resumed>) = 0

[pid 2036] <... set\_robust\_list resumed>) = 0

[pid 2037] mprotect(0x7e957004d000, 16384, PROT\_READ <unfinished ...>

[pid 2036] rseq(0x7ea77d50b060, 0x20, 0, 0x53053053 <unfinished ...>

[pid 2037] <... mprotect resumed>) = 0

[pid 2036] <... rseq resumed>) = 0

[pid 2037] mprotect(0x403000, 4096, PROT\_READ) = 0

[pid 2036] mprotect(0x7ea77d6dc000, 16384, PROT\_READ) = 0

[pid 2036] mprotect(0x403000, 4096, PROT\_READ) = 0

[pid 2036] mprotect(0x7ea77d72b000, 8192, PROT\_READ <unfinished ...>

[pid 2037] mprotect(0x7e957009c000, 8192, PROT\_READ <unfinished ...>

[pid 2036] <... mprotect resumed>) = 0

[pid 2036] prlimit64(0, RLIMIT\_STACK, NULL, <unfinished ...>

[pid 2037] <... mprotect resumed>) = 0

[pid 2036] <... prlimit64 resumed>{rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 2037] prlimit64(0, RLIMIT\_STACK, NULL, <unfinished ...>

[pid 2036] munmap(0x7ea77d6ef000, 33091 <unfinished ...>

[pid 2037] <... prlimit64 resumed>{rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 2036] <... munmap resumed>) = 0

[pid 2037] munmap(0x7e9570060000, 33091 <unfinished ...>

**[pid 2036] getpid( <unfinished ...>**

[pid 2037] <... munmap resumed>) = 0

[pid 2036] <... getpid resumed>) = 2036

**[pid 2037] getpid( <unfinished ...>**

**[pid 2036] read(0, <unfinished ...>**

[pid 2037] <... getpid resumed>) = 2037

**[pid 2037] read(0, string1**

**<unfinished ...>**

[pid 1820] <... read resumed>"string1\n", 4096) = 8

**[pid 1820] write(6, "string1\n", 8) = 8**

[pid 2036] <... read resumed>"string1\n", 4096) = 8

**[pid 1820] read(0, <unfinished ...>**

**[pid 2036] write(1, "1gnirts\n", 8) = 8**

**[pid 2036] read(0, string2**

**<unfinished ...>**

[pid 1820] <... read resumed>"string2\n", 4096) = 8

**[pid 1820] write(8, "string2\n", 8) = 8**

[pid 2037] <... read resumed>"string2\n", 4096) = 8

**[pid 1820] read(0, <unfinished ...>**

**[pid 2037] write(1, "2gnirts\n", 8) = 8**

**[pid 2037] read(0, string with spaces**

**<unfinished ...>**

[pid 1820] <... read resumed>"string with spaces\n", 4096) = 19

**[pid 1820] write(6, "string with spaces\n", 19) = 19**

[pid 2036] <... read resumed>"string with spaces\n", 4096) = 19

**[pid 1820] read(0, <unfinished ...>**

**[pid 2036] write(1, "secaps htiw gnirts\n", 19) = 19**

**[pid 2036] read(0, qwerty**

**<unfinished ...>**

[pid 1820] <... read resumed>"qwerty\n", 4096) = 7

**[pid 1820] write(8, "qwerty\n", 7) = 7**

[pid 2037] <... read resumed>"qwerty\n", 4096) = 7

**[pid 1820] read(0, <unfinished ...>**

**[pid 2037] write(1, "ytrewq\n", 7) = 7**

**[pid 2037] read(0, 123456789**

**<unfinished ...>**

[pid 1820] <... read resumed>"123456789\n", 4096) = 10

**[pid 1820] write(6, "123456789\n", 10) = 10**

[pid 2036] <... read resumed>"123456789\n", 4096) = 10

**[pid 1820] read(0, <unfinished ...>**

**[pid 2036] write(1, "987654321\n", 10) = 10**

**[pid 2036] read(0,**

**<unfinished ...>**

[pid 1820] <... read resumed>"\n", 4096) = 1

**[pid 1820] write(6, "\n", 1) = 1**

[pid 2036] <... read resumed>"\n", 4096) = 1

**[pid 1820] write(8, "\n", 1 <unfinished ...>**

[pid 2036] exit\_group(0 <unfinished ...>

[pid 1820] <... write resumed>) = 1

[pid 2037] <... read resumed>"\n", 4096) = 1

[pid 2036] <... exit\_group resumed>) = ?

**[pid 1820] close(6 <unfinished ...>**

[pid 2037] exit\_group(0 <unfinished ...>

[pid 1820] <... close resumed>) = 0

[pid 2037] <... exit\_group resumed>) = ?

**[pid 1820] close(8) = 0**

**[pid 1820] wait4(2036, <unfinished ...>**

[pid 2036] +++ exited with 0 +++

[pid 1820] <... wait4 resumed>NULL, 0, NULL) = 2036

[pid 1820] --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=2036, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

**[pid 1820] wait4(2037, <unfinished ...>**

[pid 2037] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 2037

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=2037, si\_uid=0, si\_status=0, si\_utime=0, si\_stime=0} ---

write(1, "Parent exit successfully\n\0", 26Parent exit successfully

) = 26

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

Изучил системные вызовы в OC Linux на языке C. Приобрел практические навыки в управлении процессами в ОС Linux и обеспечении обмена данными между процессами посредством каналов, а также в использовании инструмента отладки strace.