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: Леоненкова Е.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.11.24

Москва, 2024

Постановка задачи

Вариант 18.

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 удаляют все гласные из строк.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает pipe, однонаправленный канал, для межпроцессного взаимодействия, и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
- pid_t getpid(void); – возвращает pid текущего или родительского процесса.
- int open(const char *__file, int __flag, …); –применяется для открытия файла с указанными флагами, для чтения и/или записи.
- 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); – используется для завершения программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – закрывает файловый дескриптор, освобождая ресурсы, связанные с ним.
- int dup2(int __fd, int __fd2); – используется для перенаправления дескрипторов. копирует FD в FD2.Если FD2 уже открыт, он будет закрыт перед операцией.
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса новым процессом, выполняющим программу, указанную по пути path.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает до nbytes байт из файла(FD) в буфер(BUF).
- pid_t wait(int *__stat_loc); – ожидает изменение состояния дочернего процесса и получает информацию об изменении.

Для выполнения данной лабораторной работы я детально изучила используемые системные вызовы и ознакомилась с примером решения подобной задачи, чтобы понять основные принципы работы.

Для выполнения данной лабораторной работы я изучила системные вызовы, используемые в программе, а также пример выполнения аналогичных задач.

Программа main.c запрашивает у пользователя два имени файлов, в которые будут записаны результаты работы дочерних процессов. После получения данных имён с помощью функции get_input, программа определяет полный путь к своему исполняемому файлу с помощью get_program_path. Затем создаются два канала для связи с дочерними процессами с использованием pipe ().

Сначала выполняется fork (), создающий первый дочерний процесс. Если текущий процесс является дочерним, то, перенаправляется стандартный ввод с помощью dup2 () на канал; с помощью execv () текущий процесс заменяется на выполнение программы child.c, передавая ей имя первого файла для записи.

Если процесс является родительским, то запускается второй fork (), создающий второй дочерний процесс. Для второго дочернего процесса выполняются те же действия, что и для первого, только передаётся имя второго файла.

Родительский процесс после создания дочерних процессов переходит к чтению строк из стандартного ввода. Полученные строки отправляются поочерёдно первому и второму дочернему процессу через соответствующие каналы. Если ввод завершается или встречается пустая строка, работа родительского процесса завершается. Родитель также ожидает завершения дочерних процессов с помощью waitpid ().

Программа child.c выполняет основную обработку данных. Она открывает переданный ей файл для записи. Данные, полученные из стандартного ввода (перенаправленного через канал), обрабатываются — из строк удаляются все гласные буквы. Результат записывается в файл и выводится на экран. Если встречается пустая строка, программа завершает работу.

Код программы

main.c

```c
#include "../inc/parent.h"

void get_input(char *input, size_t size) ;

int main() {

    char progpath[4096];
    get_program_path(progpath, sizeof(progpath));

    char file_1[4096], file_2[4096];

    {
        char msg[32];
        const int32_t length = snprintf(msg, sizeof(msg), "Print first file name:\n");
        write(STDOUT_FILENO, msg, length);
    }
    get_input(file_1, sizeof(file_1));
```

```
    {
        char msg[32];
        const int32_t length = snprintf(msg, sizeof(msg), "Print second file name:\n");
        write(STDOUT_FILENO, msg, length);
    }
    get_input(file_2, sizeof(file_2));

    int channel_1[2], channel_2[2];
    create_pipe(channel_1);
    create_pipe(channel_2);

    const pid_t child_1 = fork();
    if (child_1 == 0)
        handle_child_process(channel_1, progpath, file_1);

    const pid_t child_2 = fork();
    if (child_2 == 0)
        handle_child_process(channel_2, progpath,file_2);

    if (child_1 == -1 || child_2 == -1) {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    handle_parent_process(channel_1, channel_2, child_1, child_2);

    return 0;
}

void get_input(char *input, size_t size)
{
    if (fgets(input, size, stdin) == NULL) {
        char msg[32];
        const int32_t length = snprintf(msg, sizeof(msg), "error: failed to read input\n");
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
```

```
    }

    size_t len = strlen(input);
    if (len > 0 && input[len - 1] == '\n') {
        input[len - 1] = '\0';
    }

    if (len == 1 && input[0] == '\0') {
        const char msg[] = "Empty input detected, exiting...\n";
        write(STDOUT_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_SUCCESS);
    }
}
```

core.c

```
#include "../inc/parent.h"

void get_program_path(char *progpath, size_t size) {
    ssize_t len = readlink("/proc/self/exe", progpath, size - 1);
    if (len == -1) {
        const char msg[] = "error: failed to read program path\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    progpath[len] = '\0';

    char *last_slash = strrchr(progpath, '/');
    if (last_slash != NULL) {
        *last_slash = '\0';
    }
}

void create_pipe(int channel[2]) {
    if (pipe(channel) == -1) {
        const char msg[] = "error: failed to create pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
```

```
        exit(EXIT_FAILURE);
    }
}

void handle_child_process(int channel[2], char *progpath, char *filename) {
    pid_t pid = getpid();

    int file = open(filename, O_RDONLY);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    if (dup2(channel[0], STDIN_FILENO) == -1) {
        const char msg[] = "error: failed to duplicate channel to stdin\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        close(file);
        exit(EXIT_FAILURE);
    }

    close(channel[0]);
    close(channel[1]);

    {
        char msg[64];
        const int32_t length = snprintf(msg, sizeof(msg),
            "%d: I'm a child\n", pid);
        write(STDOUT_FILENO, msg, length);
    }

    char path[1024];
    snprintf(path, sizeof(path), "%s/%s", progpath, CLIENT_PROGRAM_NAME);

    char *const args[] = {CLIENT_PROGRAM_NAME, filename, NULL};
    if (execv(path, args) == -1) {
        const char msg[] = "error: failed to exec into new executable image\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
```

```
        close(file);
        exit(EXIT_FAILURE);
    }
}

void handle_parent_process(int channel_1[2], int channel_2[2], pid_t child_1, pid_t child_2) {
    pid_t pid = getpid();
    {
        char msg[64];
        const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a parent, my children have
PID: %d and %d\n", pid, child_1, child_2);
        write(STDOUT_FILENO, msg, length);
    }

    close(channel_1[0]);
    close(channel_2[0]);

    char buf[4096];
    ssize_t bytes_read;
    int line_number = 1;

    while ((bytes_read = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
    char *ptr = buf;
    while (ptr < buf + bytes_read) {
        char *endptr = strchr(ptr, '\n');
        if (endptr == NULL) {
            endptr = buf + bytes_read;
        } else {
            *endptr = '\0';
            endptr++;
        }

        if (strlen(ptr) == 0) {
            const char msg[] = "Empty input detected, exiting parent process...\n";
            write(STDOUT_FILENO, msg, sizeof(msg) - 1);
            return;
        }
```

```
        int target_channel = (line_number % 2 == 1) ? channel_1[1] : channel_2[1];
        if (write(target_channel, ptr, endptr - ptr) == -1) {
            const char msg[] = "error: failed to write to pipe\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
        }

        ptr = endptr;
        line_number++;
    }
}


    close(channel_1[1]);
    close(channel_2[1]);

    waitpid(child_1, NULL, 0);
    waitpid(child_2, NULL, 0);
}
```

Child.c

```
#include "../inc/child.h"

int main(int argc, char **argv) {
    char buf[4096];
    ssize_t bytes;

    pid_t pid = getpid();

    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 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 (bytes == 1 && buf[0] == '\n') {
    const char msg[] = "Empty input detected, exiting client...\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    break;
}

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

char filtered_buf[256];
ssize_t filtered_len = 0;

for (ssize_t i = 0; i < bytes; i++) {
    if (!strchr("aeiouAEIOU", buf[i])) {
        filtered_buf[filtered_len++] = buf[i];
    }
}

int32_t written = write(file, filtered_buf, filtered_len);
if (written != filtered_len) {
    const char msg[] = "error: failed to write to file\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

for (int i = 0; i < filtered_len; i++)
    write(STDOUT_FILENO, &filtered_buf[i], 1);

write(STDOUT_FILENO, "\n", 1);

{
    char msg[32];
    const int32_t length = snprintf(msg, sizeof(msg), "Print new word\n");
    write(STDOUT_FILENO, msg, length);
}
```

```
}
const char term = '\0';

    ssize_t eof_written = write(file, &term, sizeof(term));
    if (eof_written != sizeof(term))
    {
        const char msg[] = "error: failed to write EOF to file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    close(file);
}
```

Child.h

```
#ifndef __CHILD_H__
#define __CHILD_H__

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

#endif
```

Parent.h

```
#ifndef __PARENT_H__
#define __PARENT_H__

#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

#define CLIENT_PROGRAM_NAME "child"

void get_program_path(char *progpath, size_t size);
void create_pipe(int channel[2]);
void handle_child_process(int channel[2], char *progpath, char *filename);
void handle_parent_process(int channel_1[2], int channel_2[2], pid_t child_1, pid_t child_2);

#endif
```

Протокол работы программы

```
leoelena@DESKTOP-HJEL67G:/mnt/c/Users/Елена/Desktop/LAB_L$ ./bin/parent
Print first file name:
input.txt
Print second file name:
output.txt
318189: I'm a parent, my children have PID: 318502 and 318503
318503: I'm a child
318502: I'm a child
hello
hll
Print new word
world
wrld
Print new word
first
frst
Print new word
second
scnd
Print new word

Empty input detected, exiting parent process...
leoelena@DESKTOP-HJEL67G:/mnt/c/Users/Елена/Desktop/LAB_L$ strace -f ./bin/parent
execve("./bin/parent", ["./bin/parent"], 0x7fff9124a028 /* 36 vars */) = 0
brk(NULL)                               = 0x555fe1d61000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f50dd285000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=20115, ...}) = 0
mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f50dd280000
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\220\243\2\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
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f50dd06e000
mmap(0x7f50dd096000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7f50dd096000
mmap(0x7f50dd21e000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) =
0x7f50dd21e000
mmap(0x7f50dd26d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1fe000) = 0x7f50dd26d000
mmap(0x7f50dd273000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0x7f50dd273000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f50dd06b000
arch_prctl(ARCH_SET_FS, 0x7f50dd06b740) = 0
set_tid_address(0x7f50dd06ba10)         = 321357
set_robust_list(0x7f50dd06ba20, 24)     = 0
rseq(0x7f50dd06c060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f50dd26d000, 16384, PROT_READ) = 0
mprotect(0x555fe0333000, 4096, PROT_READ) = 0
mprotect(0x7f50dd2bd000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f50dd280000, 20115)           = 0
readlink("/proc/self/exe",
"/mnt/c/Users/\320\225\320\273\320\265\320\275\320\260/Desktop/"..., 4095) = 48
write(1, "Print first file name:\n", 23Print first file name:
) = 23
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}) = 0
getrandom("\xe1\x3a\x31\x7c\xc1\x07\x83\x14", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x555fe1d61000
brk(0x555fe1d82000)                     = 0x555fe1d82000
read(0, input.txt
"input.txt\n", 1024)            = 10
write(1, "Print second file name:\n", 24Print second file name:
) = 24
read(0, output.txt
"output.txt\n", 1024)           = 11
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
322160 attached
, child_tidptr=0x7f50dd06ba10) = 322160
[pid 322160] set_robust_list(0x7f50dd06ba20, 24 <unfinished ...>
[pid 321357] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD
<unfinished ...>
[pid 322160] <... set_robust_list resumed>) = 0
strace: Process 322161 attached
```

```
[pid 322160] getpid( <unfinished ...>
[pid 321357] <... clone resumed>, child_tidptr=0x7f50dd06ba10) = 322161
[pid 322161] set_robust_list(0x7f50dd06ba20, 24 <unfinished ...>
[pid 322160] <... getpid resumed>)      = 322160
[pid 321357] getpid( <unfinished ...>
[pid 322161] <... set_robust_list resumed>) = 0
[pid 321357] <... getpid resumed>)      = 321357
[pid 322160] openat(AT_FDCWD, "input.txt", O_RDONLY <unfinished ...>
[pid 322161] getpid( <unfinished ...>
[pid 321357] write(1, "321357: I'm a parent, my childre..., 62 <unfinished ...>
[pid 322161] <... getpid resumed>)      = 322161
321357: I'm a parent, my children have PID: 322160 and 322161
[pid 321357] <... write resumed>)       = 62
[pid 322161] openat(AT_FDCWD, "output.txt", O_RDONLY <unfinished ...>
[pid 321357] close(3)                   = 0
[pid 321357] close(5)                   = 0
[pid 321357] read(0,  <unfinished ...>
[pid 322161] <... openat resumed>)      = 7
[pid 322160] <... openat resumed>)      = 7
[pid 322161] dup2(5, 0 <unfinished ...>
[pid 322160] dup2(3, 0 <unfinished ...>
[pid 322161] <... dup2 resumed>)        = 0
[pid 322160] <... dup2 resumed>)        = 0
[pid 322161] close(5 <unfinished ...>
[pid 322160] close(3 <unfinished ...>
[pid 322161] <... close resumed>)       = 0
[pid 322160] <... close resumed>)       = 0
[pid 322161] close(6 <unfinished ...>
[pid 322160] close(4 <unfinished ...>
[pid 322161] <... close resumed>)       = 0
[pid 322160] <... close resumed>)       = 0
[pid 322161] write(1, "322161: I'm a child\n", 20 <unfinished ...>
322161: I'm a child
[pid 322160] write(1, "322160: I'm a child\n", 20 <unfinished ...>
[pid 322161] <... write resumed>)       = 20
322160: I'm a child
[pid 322161] execve("/mnt/c/Users/\320\225\320\273\320\265\320\275\320\260/Desktop/LAB_L/bin/child",
["child", "output.txt"], 0x7ffe6a156fd8 /* 36 vars */ <unfinished ...>
[pid 322160] <... write resumed>)       = 20
[pid 322160] execve("/mnt/c/Users/\320\225\320\273\320\265\320\275\320\260/Desktop/LAB_L/bin/child",
["child", "input.txt"], 0x7ffe6a156fd8 /* 36 vars */ <unfinished ...>
[pid 322161] <... execve resumed>)      = 0
[pid 322160] <... execve resumed>)      = 0
[pid 322161] brk(NULL <unfinished ...>
[pid 322160] brk(NULL <unfinished ...>
[pid 322161] <... brk resumed>)         = 0x55ee8ac11000
[pid 322160] <... brk resumed>)         = 0x555dc48f2000
[pid 322161] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0
<unfinished ...>
```

```
[pid 322160] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4b63000
[pid 322160] <... mmap resumed>)        = 0x7fd83248b000
[pid 322161] access("/etc/ld.so.preload", R_OK <unfinished ...>
[pid 322160] access("/etc/ld.so.preload", R_OK <unfinished ...>
[pid 322161] <... access resumed>)      = -1 ENOENT (No such file or directory)
[pid 322160] <... access resumed>)      = -1 ENOENT (No such file or directory)
[pid 322161] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC <unfinished ...>
[pid 322160] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC <unfinished ...>
[pid 322161] <... openat resumed>)      = 5
[pid 322160] <... openat resumed>)      = 3
[pid 322161] fstat(5,  <unfinished ...>
[pid 322160] fstat(3,  <unfinished ...>
[pid 322161] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=20115, ...}) = 0
[pid 322160] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=20115, ...}) = 0
[pid 322161] mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 5, 0 <unfinished ...>
[pid 322160] mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4b5e000
[pid 322160] <... mmap resumed>)        = 0x7fd832486000
[pid 322161] close(5 <unfinished ...>
[pid 322160] close(3 <unfinished ...>
[pid 322161] <... close resumed>)       = 0
[pid 322160] <... close resumed>)       = 0
[pid 322161] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>
[pid 322160] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>
[pid 322161] <... openat resumed>)      = 5
[pid 322160] <... openat resumed>)      = 3
[pid 322161] read(5,  <unfinished ...>
[pid 322160] read(3,  <unfinished ...>
[pid 322161] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
[pid 322160] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
[pid 322161] pread64(5,  <unfinished ...>
[pid 322160] pread64(3,  <unfinished ...>
[pid 322161] <... 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 322160] <... 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 322161] fstat(5,  <unfinished ...>
[pid 322160] fstat(3,  <unfinished ...>
[pid 322161] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid 322160] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid 322161] pread64(5,  <unfinished ...>
[pid 322160] pread64(3,  <unfinished ...>
[pid 322161] <... 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 322160] <... 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 322161] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 5, 0 <unfinished ...>
[pid 322160] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>
```

```
[pid 322161] <... mmap resumed>)        = 0x7f8ae494c000
[pid 322160] <... mmap resumed>)        = 0x7fd832274000
[pid 322161] mmap(0x7f8ae4974000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 5, 0x28000 <unfinished ...>
[pid 322160] mmap(0x7fd83229c000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4974000
[pid 322160] <... mmap resumed>)        = 0x7fd83229c000
[pid 322161] mmap(0x7f8ae4afc000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 5,
0x1b0000 <unfinished ...>
[pid 322160] mmap(0x7fd832424000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4afc000
[pid 322160] <... mmap resumed>)        = 0x7fd832424000
[pid 322161] mmap(0x7f8ae4b4b000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 5, 0x1fe000 <unfinished ...>
[pid 322160] mmap(0x7fd832473000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4b4b000
[pid 322160] <... mmap resumed>)        = 0x7fd832473000
[pid 322161] mmap(0x7f8ae4b51000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>
[pid 322160] mmap(0x7fd832479000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4b51000
[pid 322160] <... mmap resumed>)        = 0x7fd832479000
[pid 322161] close(5 <unfinished ...>
[pid 322160] close(3 <unfinished ...>
[pid 322161] <... close resumed>)       = 0
[pid 322160] <... close resumed>)       = 0
[pid 322161] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0
<unfinished ...>
[pid 322160] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0
<unfinished ...>
[pid 322161] <... mmap resumed>)        = 0x7f8ae4949000
[pid 322160] <... mmap resumed>)        = 0x7fd832271000
[pid 322161] arch_prctl(ARCH_SET_FS, 0x7f8ae4949740 <unfinished ...>
[pid 322160] arch_prctl(ARCH_SET_FS, 0x7fd832271740 <unfinished ...>
[pid 322161] <... arch_prctl resumed>)  = 0
[pid 322160] <... arch_prctl resumed>)  = 0
[pid 322161] set_tid_address(0x7f8ae4949a10 <unfinished ...>
[pid 322160] set_tid_address(0x7fd832271a10 <unfinished ...>
[pid 322161] <... set_tid_address resumed>) = 322161
[pid 322160] <... set_tid_address resumed>) = 322160
[pid 322161] set_robust_list(0x7f8ae4949a20, 24 <unfinished ...>
[pid 322160] set_robust_list(0x7fd832271a20, 24 <unfinished ...>
[pid 322161] <... set_robust_list resumed>) = 0
[pid 322160] <... set_robust_list resumed>) = 0
[pid 322161] rseq(0x7f8ae494a060, 0x20, 0, 0x53053053 <unfinished ...>
[pid 322160] rseq(0x7fd832272060, 0x20, 0, 0x53053053 <unfinished ...>
[pid 322161] <... rseq resumed>)        = 0
[pid 322160] <... rseq resumed>)        = 0
[pid 322161] mprotect(0x7f8ae4b4b000, 16384, PROT_READ <unfinished ...>
[pid 322160] mprotect(0x7fd832473000, 16384, PROT_READ <unfinished ...>
[pid 322161] <... mprotect resumed>)    = 0
```

```
[pid 322160] <... mprotect resumed>)    = 0
[pid 322161] mprotect(0x55ee8a2ee000, 4096, PROT_READ <unfinished ...>
[pid 322160] mprotect(0x555dc4853000, 4096, PROT_READ <unfinished ...>
[pid 322161] <... mprotect resumed>)    = 0
[pid 322160] <... mprotect resumed>)    = 0
[pid 322161] mprotect(0x7f8ae4b9b000, 8192, PROT_READ <unfinished ...>
[pid 322160] mprotect(0x7fd8324c3000, 8192, PROT_READ <unfinished ...>
[pid 322161] <... mprotect resumed>)    = 0
[pid 322160] <... mprotect resumed>)    = 0
[pid 322161] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
[pid 322160] prlimit64(0, RLIMIT_STACK, NULL,  <unfinished ...>
[pid 322161] munmap(0x7f8ae4b5e000, 20115 <unfinished ...>
[pid 322160] <... prlimit64 resumed>{rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid 322161] <... munmap resumed>)      = 0
[pid 322160] munmap(0x7fd832486000, 20115 <unfinished ...>
[pid 322161] getpid( <unfinished ...>
[pid 322160] <... munmap resumed>)      = 0
[pid 322161] <... getpid resumed>)      = 322161
[pid 322160] getpid( <unfinished ...>
[pid 322161] openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600 <unfinished ...>
[pid 322160] <... getpid resumed>)      = 322160
[pid 322160] openat(AT_FDCWD, "input.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600 <unfinished ...>
[pid 322161] <... openat resumed>)      = 5
[pid 322160] <... openat resumed>)      = 3
[pid 322161] read(0,  <unfinished ...>
[pid 322160] read(0, hello
 <unfinished ...>
[pid 321357] <... read resumed>"hello\n", 4096) = 6
[pid 321357] write(4, "hello\0", 6)     = 6
[pid 322160] <... read resumed>"hello\0", 4096) = 6
[pid 321357] read(0,  <unfinished ...>
[pid 322160] write(3, "hll", 3)         = 3
[pid 322160] write(1, "h", 1h)           = 1
[pid 322160] write(1, "l", 1l)           = 1
[pid 322160] write(1, "l", 1l)           = 1
[pid 322160] write(1, "\n", 1
)          = 1
[pid 322160] write(1, "Print new word\n", 15Print new word
) = 15
[pid 322160] read(0, world
 <unfinished ...>
[pid 321357] <... read resumed>"world\n", 4096) = 6
[pid 321357] write(6, "world\0", 6)     = 6
[pid 322161] <... read resumed>"world\0", 4096) = 6
[pid 321357] read(0,  <unfinished ...>
[pid 322161] write(5, "wrld", 4)        = 4
[pid 322161] write(1, "w", 1w)           = 1
[pid 322161] write(1, "r", 1r)           = 1
```

```
[pid 322161] write(1, "l", 1l)           = 1
[pid 322161] write(1, "d", 1d)           = 1
[pid 322161] write(1, "\n", 1
)          = 1
[pid 322161] write(1, "Print new word\n", 15Print new word
) = 15
[pid 322161] read(0, first
 <unfinished ...>
[pid 321357] <... read resumed>"first \n", 4096) = 7
[pid 321357] write(4, "first \0", 7)    = 7
[pid 322160] <... read resumed>"first \0", 4096) = 7
[pid 321357] read(0,  <unfinished ...>
[pid 322160] write(3, "frst ", 5)       = 5
[pid 322160] write(1, "f", 1f)           = 1
[pid 322160] write(1, "r", 1r)           = 1
[pid 322160] write(1, "s", 1s)           = 1
[pid 322160] write(1, "t", 1t)           = 1
[pid 322160] write(1, " ", 1 )           = 1
[pid 322160] write(1, "\n", 1
)          = 1
[pid 322160] write(1, "Print new word\n", 15Print new word
) = 15
[pid 322160] read(0, second
 <unfinished ...>
[pid 321357] <... read resumed>"second\n", 4096) = 7
[pid 321357] write(6, "second\0", 7)    = 7
[pid 322161] <... read resumed>"second\0", 4096) = 7
[pid 321357] read(0,  <unfinished ...>
[pid 322161] write(5, "scnd", 4)        = 4
[pid 322161] write(1, "s", 1s)           = 1
[pid 322161] write(1, "c", 1c)           = 1
[pid 322161] write(1, "n", 1n)           = 1
[pid 322161] write(1, "d", 1d)           = 1
[pid 322161] write(1, "\n", 1
)          = 1
[pid 322161] write(1, "Print new word\n", 15Print new word
) = 15
[pid 322161] read(0,
 <unfinished ...>
[pid 321357] <... read resumed>"\n", 4096) = 1
[pid 321357] write(1, "Empty input detected, exiting pa"..., 48Empty input detected, exiting
parent process...
) = 48
[pid 321357] exit_group(0)              = ?
[pid 321357] +++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы я изучила использование системных вызовов в языке Си. Освоила создание программ, работающих с несколькими процессами, и передачу данных между ними через каналы. В процессе отладки программы познакомилась с утилитой strace, которая оказалась эффективным инструментом для анализа работы многопоточных программ. Лабораторная работа была полезной, так как дала практический опыт разработки программ на Си с параллельным выполнением процессов.