

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №7. Использование средств управления устройствами и
файловой системой /rroc

Тема

Преподаватель

подпись, дата

А. С. Кузнецов

инициалы, фамилия

Студент

КИ19-17/16 031939175

номер группы, зачетной
книжки

подпись, дата

А. Д. Непомнящий

инициалы, фамилия

Красноярск 2021

1 Цель работы

Цель состоит в изучении особенностей средств управления специфичными объектами файловых систем в GNU/Linux.

2 Задачи

Выполнение работы сводится к следующим задачам.

1. Ознакомление с краткими теоретическими сведениями по управлению устройствами и файловой системой /proc в ОС GNU/Linux.

2. Модификация результатов выполнения ЛР 6 добавлением использования программных средств для работы со специфичными объектами файловой системы. Необходимо запретить передачу аргументов командной строки через перенаправление стандартных потоков ввод-вывода. Один из аргументов командной строки - вывод справки по аргументам командной строки для серверной и клиентской частей программы, соответственно. Справка также должна выводиться в том случае, если не задан ни один из аргументов командной строки. Еще один аргумент - включает и отключает проверку заполнения "дисков", на которых хранятся log-файлы. Обе части должны разворачиваться в виртуальной файловой системе (ВФС). Изначально ВФС отсутствует.

3. Написание настоящего отчета защита его с исходными текстами и исполняемым модулем программы. Исходные тексты программ должны содержать комментарии в стиле системы doxygen, настоящий отчет должен включать последовательность действий для развертывания приложения в ВФС и его удаления и тестовые примеры работы программы.

Требуется разработать две программы: первая реализует серверную часть, вторая – клиентскую часть. Обмен данными между ними организуется посредством механизма Internet-сокетов и протокола TCP либо UDP. Результат выполнения выводится на терминал/консоль. Должен использоваться интерфейс командной строки (CLI). При реализации обязательно использование изученных в лекционном курсе системных вызовов (ОС Linux), предназначенных для работы с сокетами. Программный код, относящийся к пользовательскому

интерфейсу, должен быть физически отделен от кода, реализующего межпроцессное взаимодействие, и оба они, в свою очередь, отделены от кода реализации основной логики, например, вычислений.

Вариант 14. Клиент принимает от пользователя беззнаковое целое число N – основание системы счисления (диапазон $(1..20]$) и последовательность цифр в соответствии с заданной системой счисления, отправляет серверу. Сервер принимает основание системы счисления и число в этой системе, выводит число на экран, переводит его в десятичную систему, выводит на экран, осуществляет его реверс (меняет порядок следования знаков на обратный), выводит на экран значение измененной последовательности, переводит ее в десятичную систему и выводит его на экран.

3 Исходные тексты программы

Далее приведено содержимое файлов с исходным ходом программы.

Листинг 1 – Код в файле task14.h

```
/*! \file    task14.h
 * \brief    Header file of functions with numeral systems
 *           essential for task 14
 */

#include <stdbool.h>
#include "input.h"
#include <inttypes.h>

#ifndef LAB1_TASK14_H
#define LAB1_TASK14_H

/*! \struct taskData
 * \brief Struct for PerformTask() function
 *
 * \details Keeps data that is used as PerformTask() argument
 */
typedef struct
{
    /*!
     * Number in required numeric system
     */
    char number[INPUT_SIZE];
    /*!
     * Radix of numeric system
     */
    int8_t radix;
} taskData;

/*! \brief Performs task14 with required output
 *
 * \param data argument for task 14
 */
void PerformTask(taskData* data);

/*! \brief Converts number in any (2-20) numeral system to decimal
 *
 * \param number number to convert.
 * \param radix radix of numeral system.
 * \return Integer conversion result.
 */
int AnyNumeralSystemToDecimal(char* number, int radix);

/*! \brief Checks if number only contains digits, allowed for this numeral
 * system
 *
 * \param numberToCheck number to check.
 * \param radix radix of numeral system.
 * \return true if number only contains digits, allowed for this numeral
```

Окончание листинга 1

```
* system, false - otherwise.
*/
bool CheckRadixMatch(char* numberToCheck, int radix);

/*! \brief Checks if number is not too big to be written to int after
 * conversion
 *
 * \param numberToCheck number to check.
 * \param radix radix of numeral system.
 * \return true if number is not too big to be written to int after
 * conversion, false - otherwise
 */
bool CheckIntOverflow(char* numberToCheck, int radix);

/*! \brief Checks if number can be numeral system radix for task 14
 *
 * \param intToCheck number to check.
 * \return true if number can be numeral system radix for task 14
 * false - otherwise
 */
bool RadixInputCheck(int intToCheck);

#endif //LAB1_TASK14_H
```

Листинг 2 – Код в файле task14.c

```
/*! \file input.c
 * \brief Implements functions of task14.h
 */

#include "task14.h"
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>

/*! \enum
 * \brief Essential constants for task 14
 */
enum NumeralSystemsConstants
{
    VIGESIMAL_A = 'A', /** Digit next to 9 */
    MIN_RADIX = 2, /** Minimal numeral system radix */
    MAX_RADIX = 20 /** Maximal numeral system radix for task */
};

void PerformTask(taskData* data)
{
    char reversedNumber[INPUT_SIZE];
    char* number = data->number;
    int8_t radix = data->radix;

    for (int i = (int) strlen(number) - 1; i >= 0; i--)
    {
        reversedNumber[strlen(number) - (i + 1)] = number[i];
    }
}
```

Продолжение листинга 2

```
    reversedNumber[strlen(number)] = '\\0';

    while (reversedNumber[strlen(reversedNumber) - 1] == '0')
    {
        reversedNumber[strlen(reversedNumber) - 1] = '\\0';
    }

    printf("Original: %s\\n", number);
    printf("To decimal: %d\\n",
        AnyNumeralSystemToDecimal(number, radix));
    printf("Reversed: %s\\n", reversedNumber);
    if (CheckIntOverflow(reversedNumber, radix))
    {
        printf("Reversed to decimal: %d\\n",
            AnyNumeralSystemToDecimal(reversedNumber, radix));
    }
    else
    {
        printf("Reversed number is too big");
    }
}

int AnyNumeralSystemToDecimal(char* number, int radix)
{
    int result = 0;
    int multiplier = 1;
    int currentDigit;
    for (int i = (int) strlen(number) - 1; i >= 0; i--)
    {
        if (number[i] >= VIGESIMAL_A)
        {
            currentDigit = 10 + number[i] - VIGESIMAL_A;
        }
        else
        {
            currentDigit = number[i] - '0';
        }
        result += currentDigit * multiplier;
        multiplier *= radix;
    }
    return result;
}

bool CheckRadixMatch(char* numberToCheck, int radix)
{
    int currentDigit;
    for (int i = 0; i < strlen(numberToCheck); i++)
    {
        if (numberToCheck[i] >= VIGESIMAL_A)
        {
            currentDigit = 10 + numberToCheck[i] - VIGESIMAL_A;
        }
        else
        {
            currentDigit = numberToCheck[i] - '0';
        }
        if (currentDigit >= radix || currentDigit < 0)
        {

```

Окончание листинга 2

```
        return false;
    }
}
return true;
}

bool CheckIntOverflow(char* numberToCheck, int radix)
{
    return (double) strlen(numberToCheck) <
        (log((double) __INT_MAX__) / log((double) radix) - 1);
}

bool RadixInputCheck(int intToCheck)
{
    if (intToCheck < MIN_RADIX || intToCheck > MAX_RADIX)
    {
        return false;
    }
    return true;
}
```

Листинг 3 – Код в файле server.c

```
/*! \file    server.c
 * \brief    Code of server executable and server's task
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <signal.h>
#include "task14.h"
#include "logOutput.h"
#include "timer.h"
#include "ServerFunctions.h"

char g_logPath[INPUT_SIZE];
int g_idleTime = 10;

/*! \brief Signal handler for server
 */
void ServerTimerHandler(int signum)
{
    if (signum == SIGALRM)
    {
        if (access(PATH_TO_CHECK_FILE, F_OK) == 0)
        {
            remove(PATH_TO_CHECK_FILE);
        }
        WriteLogEntry(g_logPath, "Server terminated by timer\n");
        exit(0);
    }
}
```

Продолжение листинга 3

```
void ServerInterruptHandler(int signum)
{
    if (signum == SIGINT)
    {
        if (access(PATH_TO_CHECK_FILE, F_OK) == 0)
        {
            remove(PATH_TO_CHECK_FILE);
        }
        WriteLogEntry(g_logPath, "Server terminated by Ctrl+C\n");
        exit(0);
    }
}

void ServerTerminateHandler(int signum)
{
    if (signum == SIGTERM)
    {
        if (access(PATH_TO_CHECK_FILE, F_OK) == 0)
        {
            remove(PATH_TO_CHECK_FILE);
        }
        WriteLogEntry(g_logPath, "Server terminated by kill signal\n");
        exit(0);
    }
}

/*! \brief Reads data from socket and calls PerformTask()
 *
 * \param serverSocket descriptor of socket to listen
 */
int ServerTask(int serverSocket)
{
    struct sockaddr_in clientName;
    socklen_t clientNameLength = sizeof(clientName);
    taskData* data;
    data = (taskData*) malloc(sizeof(taskData));

    struct sigaction saTimer = CreateSAHandler(ServerTimerHandler);
    sigaction(SIGALRM, &saTimer, NULL);
    struct sigaction saInterrupt = CreateSAHandler(ServerInterruptHandler);
    sigaction(SIGINT, &saInterrupt, NULL);
    struct sigaction saTerminate = CreateSAHandler(ServerTerminateHandler);
    sigaction(SIGTERM, &saTerminate, NULL);

    struct itimerval timer = InitTimer(g_idleTime, 0);
    setitimer(ITIMER_REAL, &timer, NULL);

    while (true)
    {
        int recvResult = (int) recvfrom(serverSocket, data, sizeof(taskData),
                                         0,
                                         (struct sockaddr*) &clientName,
                                         &clientNameLength);

        if (-1 == recvResult)
        {
            perror("recvfrom");
        }
    }
}
```


Продолжение листинга 3

```
    }
    if (recvResult > 0)
    {
        WriteLogEntry(g_logPath, "Got new task\n");
        PerformTask(data);
        RollbackTimer(&timer, g_idleTime, 0);
    }
}
if (access(PATH_TO_CHECK_FILE, F_OK) == 0)
{
    remove(PATH_TO_CHECK_FILE);
}
free(data);
return 0;
}

/*! \brief main function of server
*/
int main(int argc, char** argv)
{
    if (isatty(STDIN_FILENO) == 0)
    {
        fprintf(stderr, "Streams redirection is prohibited\n");
        exit(EXIT_FAILURE);
    }

    int portNumber;
    bool helpRequested = false;
    bool portNumberParsed = false;
    bool logPathParsed = false;
    bool idleTimeParsed = false;
    bool checkDisks = false;
    int result;
    while ((result = getopt(argc, argv, "h:p:l:t:c")) != -1)
    {
        switch (result)
        {
            case 'h':
                helpRequested = true;
                break;

            case 'p':
                portNumber = atoi(optarg);
                if (MIN_PORT <= portNumber && portNumber <= MAX_PORT)
                {
                    portNumberParsed = true;
                }
                break;
            case 'l':
                strcpy(g_logPath, optarg);
                logPathParsed = true;
                break;
            case 't':
                g_idleTime = atoi(optarg);
                if (g_idleTime > 0)
                {
                    idleTimeParsed = true;
                }
        }
    }
}
```

Продолжение листинга 3

```
        break;
    case 'c':
        checkDisks = true;
        break;
    }
}
if (helpRequested || argc == 1)
{
    {
        fprintf(stdout, "Expected arguments:\n-p - Port number\n"
                       "\n-l - Log file path\n"
                       "\n-t - Idle time\n"
                       "\n-c - Check file system occupancy\n");
        return EXIT_SUCCESS;
    }
}
if (!portNumberParsed)
{
    fprintf(stderr, "Valid port number expected\n");
    exit(EXIT_FAILURE);
}
if (!logPathParsed)
{
    fprintf(stderr, "Log file name expected\n");
    exit(EXIT_FAILURE);
}
if (!idleTimeParsed)
{
    fprintf(stderr, "Valid idle time expected\n");
    exit(EXIT_FAILURE);
}

if (checkDisks)
{
    fprintf(stdout, "Free space on FS: %lu blocks\n",
            fsFreeSize(g_logPath));
}

ServerCheckRunning(portNumber);

int socketFileDescriptor;
struct sockaddr_in name;
socketFileDescriptor = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
int i = 1;
setsockopt(socketFileDescriptor, SOL_SOCKET, SO_REUSEADDR,
           (const char*) &i, sizeof(i)
);
bzero((char*) &name, sizeof(name));
name.sin_family = AF_INET;
name.sin_port = htons((u_short) portNumber);
name.sin_addr.s_addr = INADDR_ANY;
if (-1 == bind(socketFileDescriptor, (const struct sockaddr*) &name,
               sizeof(name)))
{
    perror("bind ");
    close(socketFileDescriptor);
    exit(1);
}
```

Окончание листинга 3

```
    ServerTask(socketFileDescriptor);  
    close(socketFileDescriptor);  
}
```

Листинг 4 – Код в файле serverFunctions.h

```
/*! \file    serverFunctions.h  
 *  \brief   Header for function that checks if server is running and defines  
 *  a few essential constants  
 */  
  
#ifndef SERVERFUNCTIONS_H  
#define SERVERFUNCTIONS_H  
  
#define PATH_TO_CHECK_FILE "/tmp/lab7"  
#define MIN_PORT 1024  
#define MAX_PORT 65635  
#define PORT_LENGTH 5  
  
/*!  
 *  \brief   Checks if server is running, if not - writes information for further  
 *  checks, if running - exits program.  
 *  \details Checks if server is running on ANY port  
 *  \param port Port of server  
 */  
void ServerCheckRunning(int port);  
  
#endif //SERVERFUNCTIONS_H
```

Листинг 5 – Код в файле serverFunctions.c

```
/*! \file    serverFunctions.c  
 *  \brief   Implements function from ServerFunction.h  
 */  
  
#include <unistd.h>  
#include <stdio.h>  
#include <fcntl.h>  
#include <stdlib.h>  
#include "serverFunctions.h"  
  
void ServerCheckRunning(int port)  
{  
    if (access(PATH_TO_CHECK_FILE, F_OK) == 0)  
    {  
        fprintf(stderr, "Server is already running\n");  
        exit(EXIT_FAILURE);  
    }  
}
```

Окончание листинга 5

```
    }
    else
    {
        int fd = open(PATH_TO_CHECK_FILE, O_WRONLY | O_CREAT | O_APPEND, 0660);
        if (fd == -1)
        {
            fprintf(stderr, "Couldn't open or create essential temp file\n");
            exit(EXIT_FAILURE);
        }
        char portStr[PORT_LENGTH];
        sprintf(portStr, "%d", port);
        int writtenBytes = (int) write(fd, portStr, PORT_LENGTH);
        if (writtenBytes <= 0)
        {
            fprintf(stderr, "Couldn't write to essential temp file\n");
            exit(EXIT_FAILURE);
        }
    }
}
```

Листинг 6 – Код в файле timer.h

```
/*! \file    timer.h
 * \brief    Header for functions for working with system time
 *           essential functions for it.
 */

#ifndef LR6_TIMER_UTIL_H
#define LR6_TIMER_UTIL_H

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <sys/time.h>

/*!
 * \brief    Returns string representation of current time
 * \return   String with current time
```

Окончание листинга 6

```
*/
char* GetTimeString();

/*!
 * \brief Initiates timer
 * \param sec    Seconds
 * \param usec   Microseconds
 * \return timer itimerval
 */
struct itimerval InitTimer(int sec, int usec);

/*!
 * \brief Rolls timer back to value
 * \param timer   timer itimerval
 * \param sec     Seconds
 * \param usec    Microseconds
 */
void RollbackTimer(struct itimerval* timer, int sec, int usec);

/*!
 * \brief Creates signal handler
 * \param sa_handler function
 * \return struct sigaction
 */
struct sigaction CreateSAHandler(void* TimerHandler);

#endif
```

Листинг 7 – Код в файле timer.c

```
/*! \file   timer.c
 * \brief   Implements functions from timer.h
 */

#include <time.h>
#include "timer.h"
#include "input.h"

char* GetTimeString()
{
```

Окончание листинга 7

```
    struct tm* localTime;
    const time_t timer = time(NULL);
    localTime = localtime(&timer);
    char* result;
    result = (char*) malloc(INPUT_SIZE);
    strftime(result, INPUT_SIZE, "%Y-%m-%d %H:%M:%S", localTime);
    return (result);
}

struct itimerval InitTimer(int sec, int usec)
{
    struct itimerval timer;
    timer.it_value.tv_sec = sec;
    timer.it_value.tv_usec = usec;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    return timer;
}

void RollbackTimer(struct itimerval* timer, int sec, int usec)
{
    timer->it_value.tv_sec = sec;
    timer->it_value.tv_usec = usec;
    setitimer(ITIMER_REAL, timer, NULL);
}

struct sigaction CreateSAHandler(void* TimerHandler)
{
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = TimerHandler;
    return sa;
}
/*! \file    timer.h
```

Листинг 8 – код в файле client.c

```
/*! \file    client.c
 * \brief    Code of client executable
 */

#include <stdio.h>
#include <stdlib.h>
```

Продолжение листинга 8

```
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include "signal.h"
#include "task14.h"
#include "logOutput.h"
#include "timer.h"

#define MIN_PORT 1024
#define MAX_PORT 65635

char g_logPath[INPUT_SIZE];
int g_idleTime;

/*! \brief SIGALRM signal handler for client
 */
void ClientTimerHandler(int signum)
{
    if (signum == SIGALRM)
    {
        WriteLogEntry(g_logPath, "Client terminated by timer\n");
        exit(0);
    }
}

/*! \brief SIGINT signal handler for client
 */
void ClientInterruptHandler(int signum)
{
    if (signum == SIGINT)
    {
        WriteLogEntry(g_logPath, "Client terminated by Ctrl+C\n");
        exit(0);
    }
}

/*! \brief SIGTERM signal handler for client
 */
void ClientTerminateHandler(int signum)
{

```

Продолжение листинга 8

```
        if (signum == SIGTERM)
        {
            WriteLogEntry(g_logPath, "Client terminated by kill signal\n");
            exit(0);
        }
    }

    /*! \brief Catches ctrl+C signal, closes socket and terminates server
     * \details Parses CL arguments, checks them and sends to the server
     */
    int main(int argc, char** argv)
    {
        if (!isatty(STDIN_FILENO))
        {
            fprintf(stderr, "Streams redirection is prohibited\n");
            exit(EXIT_FAILURE);
        }
        if (!isatty(STDOUT_FILENO))
        {
            fprintf(stderr, "Streams redirection is prohibited\n");
            exit(EXIT_FAILURE);
        }

        char ip[INPUT_SIZE];
        int portNumber;
        bool helpRequested = false;
        bool ipParsed = false;
        bool portNumberParsed = false;
        bool logPathParsed = false;
        bool idleTimeParsed = false;
        int result;
        while ((result = getopt(argc, argv, "ha:p:l:t:")) != -1)
        {
            switch (result)
            {
                case 'h':
                    helpRequested = true;
                    break;

                case 'p':
                    portNumber = atoi(optarg);
```


Продолжение листинга 8

```
        if (MIN_PORT <= portNumber && portNumber <= MAX_PORT)
        {
            portNumberParsed = true;
        }
        break;
    case 'l':
        strcpy(g_logPath, optarg);
        logPathParsed = true;
        break;
    case 't':
        g_idleTime = atoi(optarg);
        if (g_idleTime > 0)
        {
            idleTimeParsed = true;
        }
        break;
    case 'a':
        strcpy(ip, optarg);
        ipParsed = true;
        break;
    }
}

if (helpRequested || argc == 1)
{
    {
        fprintf(stdout, "Expected arguments:\n-p - Port number\n"
                       "\n-l - Log file path\n"
                       "\n-t - Idle g_idleTime\n"
                       "\n-a - IP-address of server\n");
        exit(EXIT_SUCCESS);
    }
}

if (!portNumberParsed)
{
    fprintf(stderr, "Valid port number expected\n");
    exit(EXIT_FAILURE);
}

if (!logPathParsed)
{
    fprintf(stderr, "Log file name expected\n");
    exit(EXIT_FAILURE);
}
```

Продолжение листинга 8

```
}
if (!idleTimeParsed)
{
    fprintf(stderr, "Valid idle time expected\n");
    exit(EXIT_FAILURE);
}
if (!ipParsed)
{
    fprintf(stderr, "Server address expected\n");
    exit(EXIT_FAILURE);
}

struct sigaction saTimer = CreateSAHandler(ClientTimerHandler);
sigaction(SIGALRM, &saTimer, NULL);
struct sigaction saInterrupt = CreateSAHandler(ClientInterruptHandler);
sigaction(SIGINT, &saInterrupt, NULL);
struct sigaction saTerminate = CreateSAHandler(ClientTerminateHandler);
sigaction(SIGTERM, &saTerminate, NULL);

struct itimerval timer = InitTimer(g_idleTime, 0);
setitimer(ITIMER_REAL, &timer, NULL);

int8_t radix;
char number[INPUT_SIZE];

int socketFileDescriptor;
struct sockaddr_in name;
memset((char*) &name, 0, sizeof(name));
name.sin_family = AF_INET;
name.sin_addr.s_addr = inet_addr(ip);
if (INADDR_NONE == name.sin_addr.s_addr)
{
    perror("inet_addr");
    exit(1);
}
name.sin_port = htons((u_short) portNumber);
socketFileDescriptor = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (socketFileDescriptor < 0)
{
    perror("socket");
    exit(1);
}
```

Продолжение листинга 8

```
}

while (true)
{
    printf("Enter base of numeral system (2 - 20)\n");
    radix = (int8_t) CheckedInputInt(RadixInputCheck);
    printf("Enter number in chosen system. Use \'A\' - \'J\' as"
           "digits for >10-based systems\n");

    while (true)
    {
        scanf("%s", number);
        if (CheckIntOverflow(number, radix) &&
            CheckRadixMatch(number, radix))
        {
            break;
        }
        printf("Wrong format or too big number!\n");
    }

    RollbackTimer(&timer, g_idleTime, 0);

    taskData* data;
    data = (taskData*) malloc(sizeof(taskData));
    strcpy(data->number, number);
    data->radix = radix;

    int resSend;
    resSend = (int) sendto(socketFileDescriptor, data, sizeof(taskData),
                           0,
                           (struct sockaddr*) &name, sizeof(name));
    if (0 > resSend)
    {
        perror("sendto");
        free(data);
        exit(1);
    }
    WriteLogEntry(g_logPath, "Sent task\n");
    free(data);
}

close(socketFileDescriptor);
```

Окончание листинга 8

```
    return 0;
}
```

Листинг 9 – код в файле logOutput.h

```
/*! \file    logOutput.h
 * \brief    Header for function to make an entry to the log
 */

#ifndef LOGOUTPUT_H
#define LOGOUTPUT_H

/*! \brief Writes an entry to log by path
 *
 * \param logPath  Path for log file
 * \param info     String that will be written to log
 */
void WriteLogEntry(char* logPath, char* info);

/*! \brief Returns free size of file system
 *
 * \param vfsPath  Path to vfs
 *
 * \return Number of free blocks
 */
unsigned long FsFreeSize(char* vfsPath);

#endif //LOGOUTPUT_H
```

Листинг 10 – код в файле logOutput.c

```
/*! \file    logOutput.c
 * \brief    Implements functions declared in logOutput.h and defines
 *           essential functions for it.
 */

#include "logOutput.h"
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include "input.h"
```

Продолжение листинга 10

```
#include "timer.h"
#include <sys/vfs.h>

/*! \brief Writes info from buffer by pointer into file
 *
 * \param fd      File descriptor
 * \param info    Variable with information that will be loaded into file
 * \param size    The number of bytes that will be loaded into file
 *
 * \return 0 on success, -1 otherwise
 */
int WriteInfo(int fd, void* info, size_t size)
{
    int writeReturn = (int) write(fd, info, size);
    if (writeReturn == -1)
    {
        perror("Write into file went wrong.");
        return -1;
    }
    if (writeReturn == 0)
    {
        perror("Nothing was written.");
        return -1;
    }
    return 0;
}

/*! \brief Opens file
 *
 * \details Opens file, changes file descriptor by pointer and
 * reports errors.
 *
 * \param fd      File descriptor pointer
 * \param filename Name of file to open
 *
 * \return 0 on success, -1 otherwise
 */
int OpenFile(int* fd, char* filename)
{
    *fd = open(filename, O_WRONLY | O_CREAT | O_APPEND);
```

Окончание листинга 10

```
    write(*fd, "", 0);
    if (*fd < 0)
    {
        return -1;
    }
    return 0;
}

unsigned long FsFreeSize(char* vfsPath)
{
    struct statfs buf;
    statfs(vfsPath, &buf);
    return buf.f_bfree;
}

void WriteLogEntry(char* logPath, char* info)
{
    int fd;
    if (OpenFile(&fd, logPath) != 0)
    {
        perror("log file");
        return;
    }
    char* timeString = GetTimeString();
    WriteInfo(fd, timeString, strlen(timeString));
    WriteInfo(fd, "\n", sizeof(char));
    WriteInfo(fd, info, INPUT_SIZE);
    WriteInfo(fd, "\n", sizeof(char));
    close(fd);
}
```

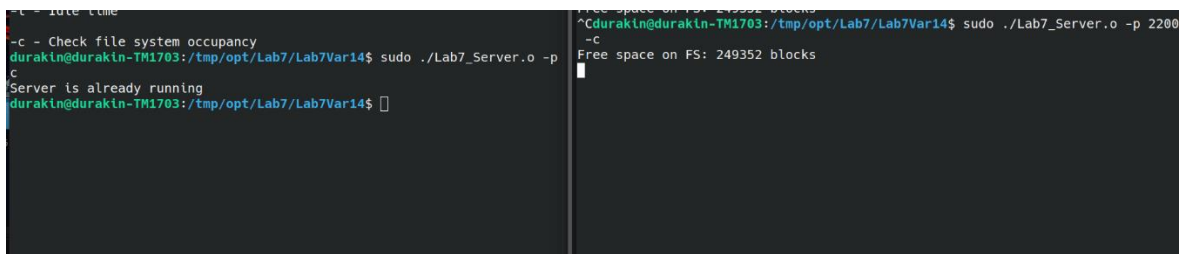
4 Последовательность действий для развертывания приложения в ВФС и его удаления

Листинг 11 – Код скрипта, примененного для развертывания ВФС

```
mkdir /media/durakin/vfs
dd if=/dev/zero of=/media/durakin/vfs/optware.img bs=1024 count=1024k
# Create linux ext3 file system
mkfs.ext3 -F /media/durakin/vfs/optware.img
# Create a temporary mount point
mkdir /tmp/opt
# Mount newly created virtual file system
mount -o loop /media/durakin/vfs/optware.img /tmp/opt
# Umount virtual file system
rm -rf /tmp/opt/*
umount /tmp/opt
rm -rf /tmp/opt
```

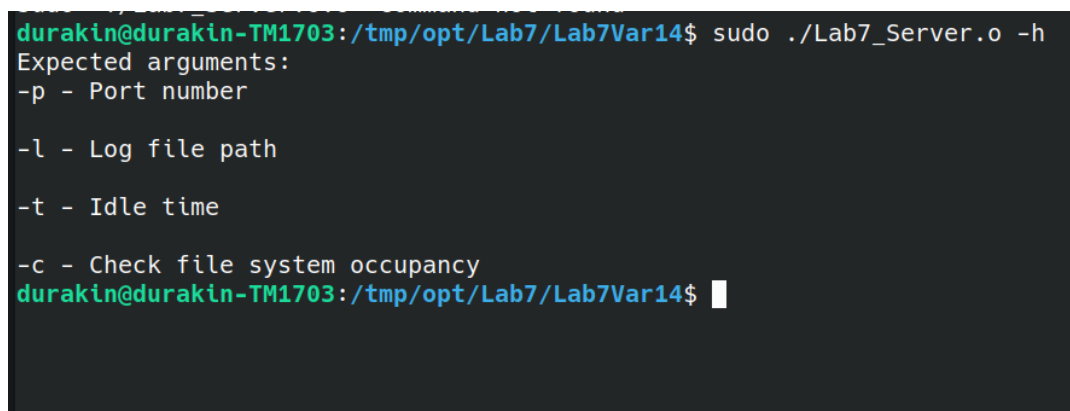
5 Тестовые примеры работы программ

Далее на рисунках приведены тестовые примеры работы программы.



```
-c - Idle time
-c - Check file system occupancy
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ sudo ./Lab7_Server.o -p
c
Server is already running
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$
```

Рисунок 1 – Запуск сервера и попытка повторного запуска процесса сервера



```
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ sudo ./Lab7_Server.o -h
Expected arguments:
-p - Port number

-l - Log file path

-t - Idle time

-c - Check file system occupancy
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$
```

Рисунок 2 – Вывод справки

```
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ sudo ./Lab7_Server.o -p 2210 -l server_log2 -t 600 -c
Free space on FS: 249351 blocks
Original: AB0BA
To decimal: 1688230
Reversed: AB0BA
Reversed to decimal: 1688230
Original: B00BA
To decimal: 721082
Reversed: AB00B
Reversed to decimal: 700427
[]

durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ do ./Lab7_Client.o -a 192.168.0.104 -p 2210 -l Lab7_Client.o -a 192.168.0.104 -p 2210 -t 600
Enter base of numeral system (2 - 20)
20
Enter number in chosen system. Use 'A' - 'J' as its for >10-based systems
AB0BA
Enter base of numeral system (2 - 20)
[]

durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ ./
Lab7_Client.o -a 192.168.0.104 -p 2210 -t 600
Log file name expected
durakin@durakin-TM1703:/tmp/opt/Lab7/Lab7Var14$ su
do ./Lab7_Client.o -a 192.168.0.104 -p 2210 -t 600
-l client_log2
[sudo] password for durakin:
Enter base of numeral system (2 - 20)
16
Enter number in chosen system. Use 'A' - 'J' asdig its for >10-based systems
B00BA
Enter base of numeral system (2 - 20)
[]
```

Рисунок 3 – Одновременное подключение двух клиентов

```
durakin@durakin-TM1703:~/Projects/SystemProgrammingLabs/Lab7/cmake-build-debug$ ./Lab7_Server > output.txt
Streams redirection is prohibited
durakin@durakin-TM1703:~/Projects/SystemProgrammingLabs/Lab7/cmake-build-debug$ cat output.txt
durakin@durakin-TM1703:~/Projects/SystemProgrammingLabs/Lab7/cmake-build-debug$ ./Lab7_Server < output.txt
Streams redirection is prohibited
durakin@durakin-TM1703:~/Projects/SystemProgrammingLabs/Lab7/cmake-build-debug$ []
```

Рисунок 4 – Реакция сервера на попытку перенаправления потоков

ВВОДА-ВЫВОДА

```
durakin@durakin-TM1703:/#$ sudo bash vfsscript
1048576+0 records in
1048576+0 records out
1073741824 bytes (1,1 GB, 1,0 GiB) copied, 2,19069 s, 490 MB/s
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: ae9c23f7-7919-4f0f-89ec-4054c999022b
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

durakin@durakin-TM1703:/#$ []
```

Рисунок 5 – Развертывание ВФС