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 №8. Безопасность системных программ |
| Тема |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  |  |  | А. С. Кузнецов |
|  | |  |  |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ19-17/1б 031939175 | | |  |  |  | А. Д. Непомнящий |
|  | номер группы, зачетной книжки | | |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2021

1. Цель работы

Цель состоит в исследовании вопросов безопасности в GNU/Linux.

1. Задачи

Выполнение работы сводится к следующим задачам.

1. Ознакомиться с теоретическими сведениями по проблеме обеспечения информационной безопасности в ОС GNU/Linux.
2. Модифицировать результаты выполнения ЛР 7 добавлением использования программных средств обеспечения информационной безопасности в ОС GNU/Linux. Обеспечить сборку программы с использованием скрипта configure и утилиты GNU make. Серверная часть должна запускаться только привилегированным пользователем, а клиентская часть для передачи сообщений должна пройти процедуру аутентификации.
3. Используя изученные механизмы, разработать и отладить серверную и клиентскую части приложения.
4. Написать отчет и представить его к защите с исходными текстами программ, предварительно загрузив код и отчет в электронный курс в виде единственного архива формата \*.tar.gz. Исходные тексты программ должны содержать комментарии в стиле системы doxygen.

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.

Обе части программы должны быть устойчивыми к некорректному пользовательскому вводу, а сообщения об ошибках должны быть информативными. Исходный код пользовательского интерфейса должен быть логически и физически отделен от кода, реализующего операции по обмену сообщениями. Должна быть обеспечена сборка программы с использованием скрипта configure и утилиты GNU make. Для обработки аргументов командной строки рекомендовано использование функций семейства getopt.

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, отсылает 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.

1. 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 <errno.h>

#include "task14.h"

#include "logOutput.h"

#include "timer.h"

#include "serverFunctions.h"

#include "securityFunctions.h"

char g\_logPath[INPUT\_SIZE];

int g\_idleTime = 10;

/\*! \brief SIGALRM 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

}

}

/\*! \brief SIGINT signal handler for server

\*/

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);

}

}

/\*! \brief SIGTERM signal handler for server

\*/

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),

Продолжение листинга 3

0,

(struct sockaddr\*) &clientName,

&clientNameLength);

if (-1 == recvResult)

{

perror("recvfrom");

}

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))

{

fprintf(stderr, "Streams redirection is prohibited\n");

exit(EXIT\_FAILURE);

}

if (!isatty(STDOUT\_FILENO))

{

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, "hp: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;

Продолжение листинга 3

}

break;

case 'l':

strcpy(g\_logPath, optarg);

logPathParsed = true;

break;

case 't':

g\_idleTime = atoi(optarg);

if (g\_idleTime > 0)

{

idleTimeParsed = true;

}

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;

}

}

CheckRoot();

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);

}

ServerCheckRunning(portNumber);

if (checkDisks)

{

WriteLogEntry(g\_logPath, "");

if (errno == ENOSPC)

{

fprintf(stderr, "No free space\n");

if (access(PATH\_TO\_CHECK\_FILE, F\_OK) == 0)

{

remove(PATH\_TO\_CHECK\_FILE);

}

exit(EXIT\_FAILURE);

Окончание листинга 3

}

}

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);

}

ServerTask(socketFileDescriptor);

close(socketFileDescriptor);

}

Листинг 4 – Код в файле serverFunctions.h

/\*! \file securityFunctions.h

\* \brief Header for security functions

\*/

#ifndef SECURITYFUNCTIONS\_H

#define SECURITYFUNCTIONS\_H

#endif //SECURITYFUNCTIONS\_H

/\*!

\* \brief Checks if process is running by root user,

\* exits if not

\*/

void CheckRoot();

/\*!

\* \brief Authenticates user, exits on failure

\*/

void CheckAuthentication();

Листинг 5 – Код в файле serverFunctions.c

/\*! \file serverFunctions.c

\* \brief Implements function from ServerFunction.h

\*/

#include <unistd.h>

Окончание листинга 5

#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);

}

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

Окончание листинга 6

#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

\*/

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()

{

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));

Окончание листинга 7

sa.sa\_handler = TimerHandler;

return sa;

}

Листинг 8 – код в файле client.c

/\*! \file client.c

\* \brief Code of client executable

\*/

#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 "securityFunctions.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)

Продолжение листинга 8

{

if (signum == SIGINT)

{

WriteLogEntry(g\_logPath, "Client terminated by Ctrl+C\n");

exit(0);

}

}

/\*! \brief SIGTERM signal handler for client

\*/

void ClientTerminateHandler(int signum)

{

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;

Продолжение листинга 8

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);

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");

Продолжение листинга 8

exit(EXIT\_SUCCESS);

}

}

CheckAuthentication();

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 (!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;

Продолжение листинга 8

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(EXIT\_FAILURE);

}

name.sin\_port = htons((u\_short) portNumber);

socketFileDescriptor = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (socketFileDescriptor < 0)

{

perror("socket");

exit(EXIT\_FAILURE);

}

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;

Окончание листинга 8

int resSend;

resSend = (int) sendto(socketFileDescriptor, data, sizeof(taskData),

0,

(struct sockaddr\*) &name, sizeof(name));

if (0 > resSend)

{

perror("sendto");

free(data);

exit(EXIT\_FAILURE);

}

WriteLogEntry(g\_logPath, "Sent task\n");

free(data);

}

close(socketFileDescriptor);

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.с

/\*! \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"

#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

\*

Окончание листинга 10

\* \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);

write(\*fd, "", 0);

if (\*fd < 0)

{

return -1;

}

return 0;

}

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);

}

Листинг 11 – код в файле securityFunctions.h

/\*! \file securityFunctions.h

\* \brief Header for security functions

\*/

#ifndef SECURITYFUNCTIONS\_H

Окончание листинга 11

#define SECURITYFUNCTIONS\_H

#endif //SECURITYFUNCTIONS\_H

/\*!

\* \brief Checks if process is running by root user,

\* exits if not

\*/

void CheckRoot();

/\*!

\* \brief Authenticates user, exits on failure

\*/

void CheckAuthentication();

Листинг 12 – код в файле securityFunctions.h

/\*! \file securityFunctions.c

\* \brief Implements functions from securityFunctions.c

\*/

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <security/pam\_appl.h>

#include <security/pam\_misc.h>

#include "securityFunctions.h"

#include "serverFunctions.h"

void CheckRoot()

{

if (geteuid() != 0)

{

fprintf(stderr, "Server can only be started by root\n");

exit(EXIT\_FAILURE);

}

if (access(PATH\_TO\_CHECK\_FILE, F\_OK) == 0)

{

remove(PATH\_TO\_CHECK\_FILE);

}

Окончание листинга 12

}

void CheckAuthentication()

{

pam\_handle\_t\* pamh;

struct pam\_conv pamc;

pamc.conv = &misc\_conv;

pamc.appdata\_ptr = NULL;

pam\_start("su", getenv("USER"), &pamc, &pamh);

if (pam\_authenticate(pamh, 0) != PAM\_SUCCESS)

{

fprintf(stderr,"Authentication fail\n");

exit(EXIT\_FAILURE);

}

pam\_end(pamh, 0);

}

1. 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

1. Тестовые примеры работы программ

Далее на рисунках приведены тестовые примеры работы программы.

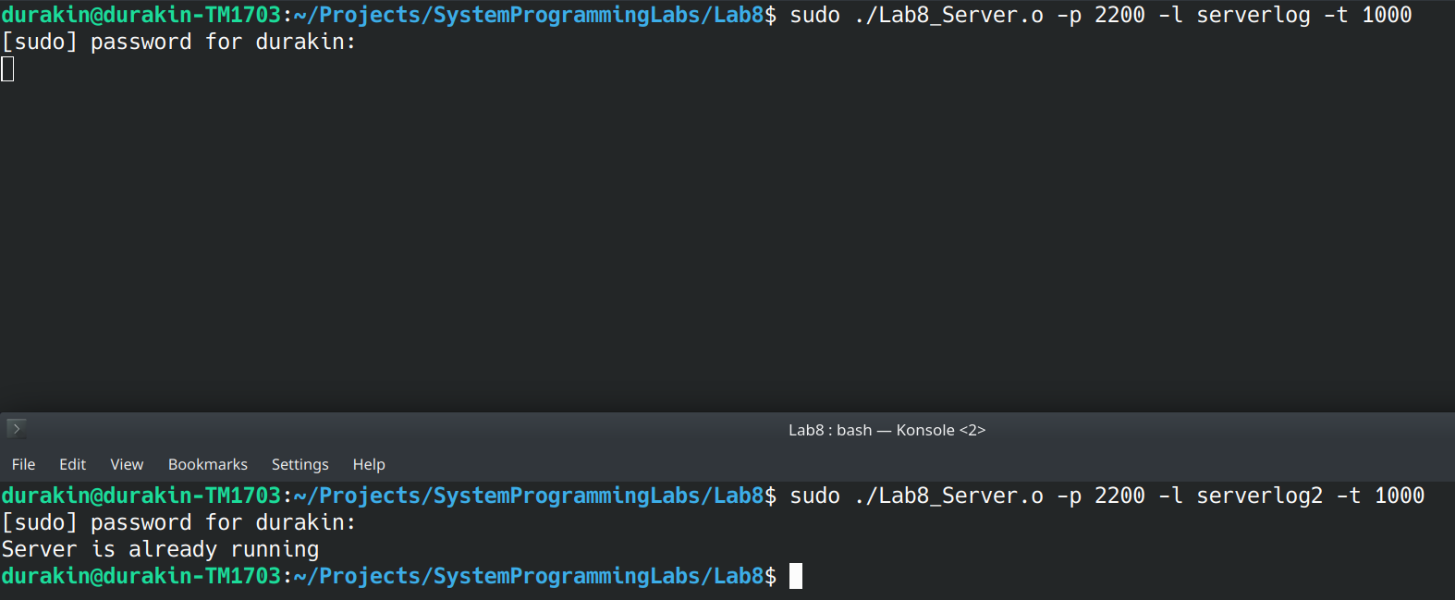


Рисунок 1 – Запуск сервера и попытка повторного запуска процесса сервера

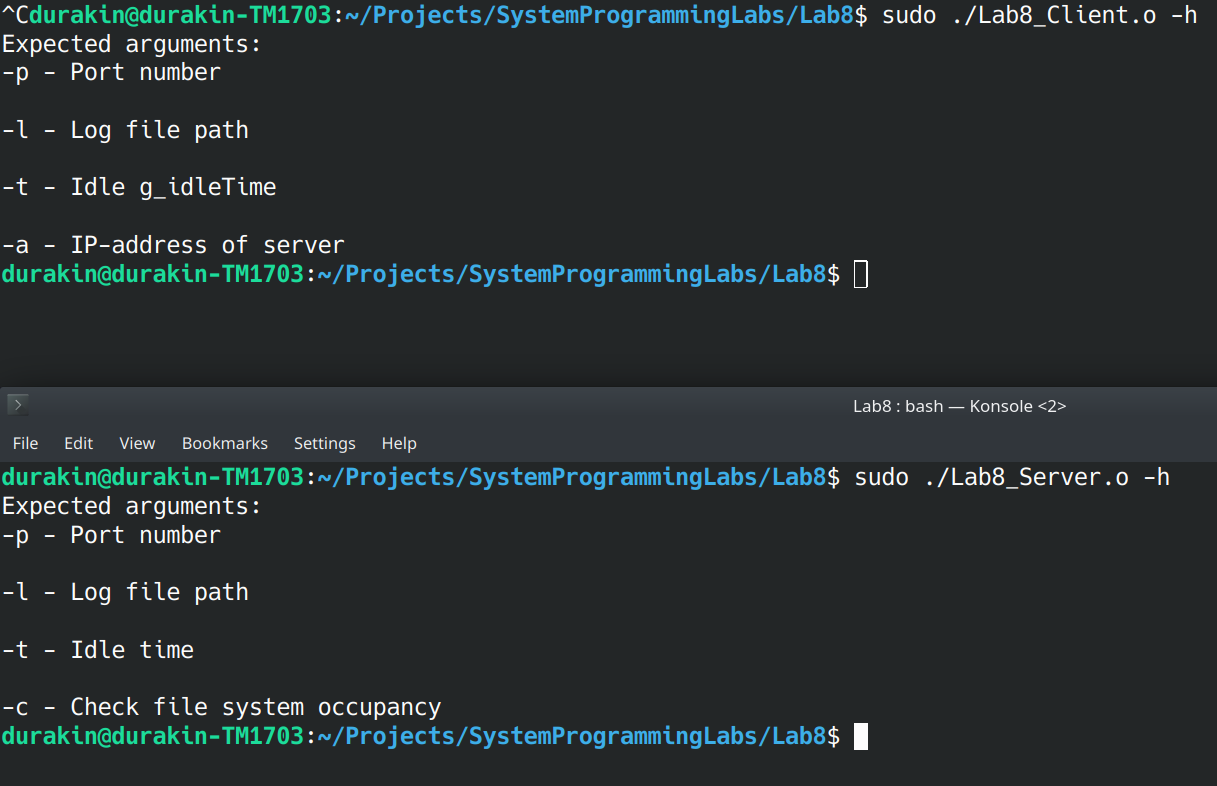


Рисунок 2 – Вывод справки

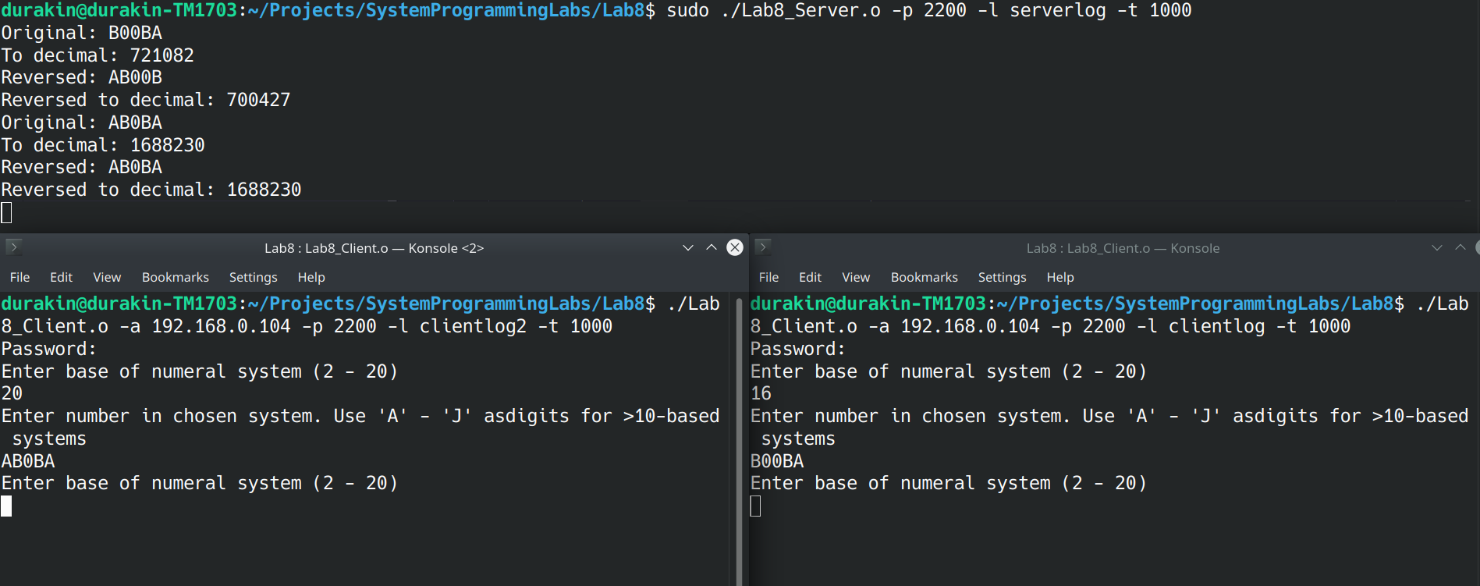


Рисунок 3 – Одновременное подключение двух клиентов (для запуска требуется ввод пароля)

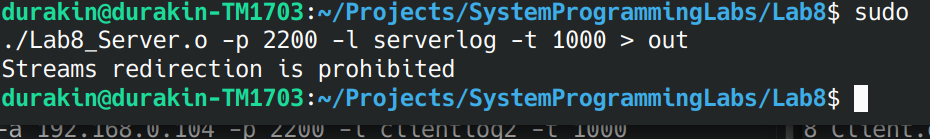


Рисунок 4 – Реакция сервера на попытку перенаправления потоков  
ввода-вывода

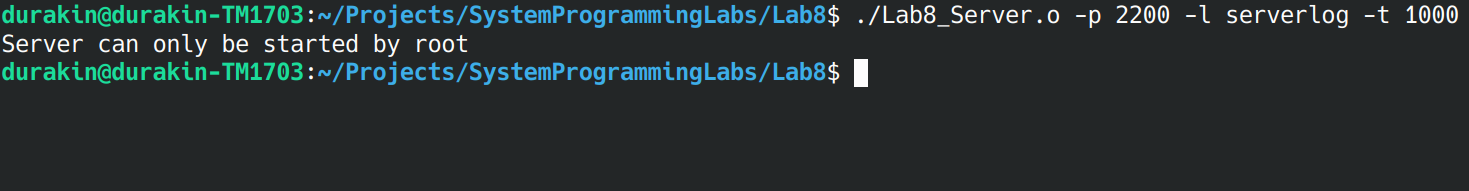


Рисунок 5 – Попытка запуска сервера не в качестве привилегированного пользователя