Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ДОМАШНЯЯ РАБОТА

«Разработка программного обеспечения под FreeBSD»

ДИСЦИПЛИНА: «Операционные системы»

Выполнил: студент гр. ИУК4-62Б _____ (Карельский М.К.)
(Подпись)

Проверил: _____ (Красавин Е.В.)
(Подпись)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2023

Цель: получение практических навыков по написанию и отладке программ.

Задачи: научиться разрабатывать, компилировать и отлаживать программы под ОС FreeBSD.

Вариант 8

Задание:

Написать комплекс программ для преобразования десятичных чисел в римскую систему счисления. Результат выдать на экран и сохранить в файл. Файл отправляется от клиентского приложения на сервер (протокол TCP).

Блок-схема:

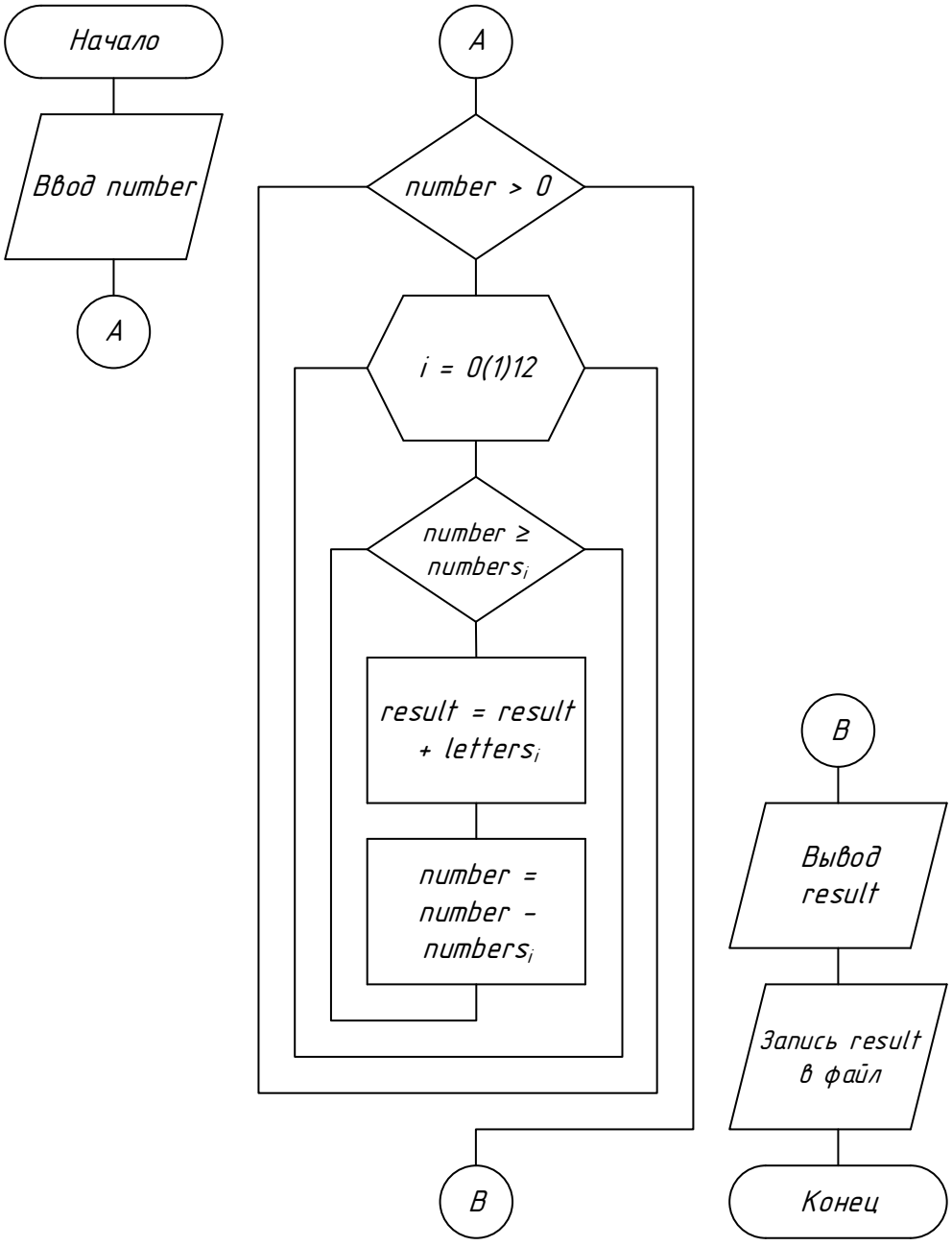

Рис. 1. Блок-схема алгоритма

Листинг:

main.cpp

```
#include <iostream>
#include <fstream>

int main()
{
    const int letterNumber = 13;
    const int numbers[letterNumber] { 1000, 900, 500, 400, 100, 90, 50, 40, 10,
9, 5, 4, 1 };
    const std::string letters[letterNumber] { "M", "CM", "D", "CD", "C", "XC",
"L", "XL", "X", "IX", "V", "IV", "I" };

    int number = 0;
    std::cout << "Input number: ";
    std::cin >> number;

    std::string result = "";
    while (number > 0)
    {
        for (int i = 0; i < letterNumber; ++i)
        {
            while (number >= numbers[i])
            {
                result += letters[i];
                number -= numbers[i];
            }
        }
    }
    std::cout << "Result: " << result << std::endl;

    std::ofstream out;
    out.open("result.txt");
    if (out.is_open())
    {
        out << result;
    }
    out.close();

    return 0;
}
```

client.cpp

```
#include <iostream>
#include <fstream>
#include <streambuf>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/socket.h>
```

```
#include <netdb.h>
#include <arpa/inet.h>
#include <string.h>
#include <string>

using namespace std;

int main()
{
    cout << "Trying to Send File Over Tcp!" << endl;

    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1)
    {
        return 1;
    }

    int port = 27015;
    string ipAddress = "192.168.0.102";
    sockaddr_in hint;
    hint.sin_family = AF_INET;
    hint.sin_port = htons(port);
    inet_pton(AF_INET, ipAddress.c_str(), &hint.sin_addr);

    int connectRes = connect(sock, (sockaddr *)&hint, sizeof(hint));
    if (connectRes == -1)
    {
        return 1;
    }

    ifstream ifs("result.txt");
    string str((istreambuf_iterator<char>(ifs)), istreambuf_iterator<char>());

    int sendRes = send(sock, str.c_str(), str.size() + 1, 0);
    if (sendRes == -1)
    {
        cout << "Could not send to server! Whoops!\r\n";
    }
    else
    {
        cout << "The file \"output.csv\" has been successfully sent to the
server.\r\n";
    }

    // Wait for response
    char buf[4096];
    memset(buf, 0, 4096);
    int bytesReceived = recv(sock, buf, 4096, 0);
    if (bytesReceived == -1)
    {
        cout << "There was an error getting response from server\r\n";
    }
```

```
    else
    {
        // Display response
        cout << "SERVER> Data received:\n"
             << string(buf, bytesReceived) << "\r\n";
    }
    // Close the socket
    close(sock);
    return 0;
}
```

server.cpp

```
#undef UNICODE
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <fstream>

// Need to link with Ws2_32.lib
#pragma comment (lib, "Ws2_32.lib")
// #pragma comment (lib, "Mswsock.lib")

#define DEFAULT_BUFLEN 4096
#define DEFAULT_PORT "27015"

int __cdecl main(void)
{
    WSADATA wsaData;
    int iResult;

    SOCKET ListenSocket = INVALID_SOCKET;
    SOCKET ClientSocket = INVALID_SOCKET;

    struct addrinfo *result = NULL;
    struct addrinfo hints;

    int iSendResult;
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed with error: %d\n", iResult);
        return 1;
    }
```

```
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    // Resolve the server address and port
    iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
    if ( iResult != 0 ) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return 1;
    }

    // Create a SOCKET for the server to listen for client connections.
    ListenSocket = socket(result->ai_family, result->ai_socktype, result-
>ai_protocol);
    if (ListenSocket == INVALID_SOCKET) {
        printf("socket failed with error: %ld\n", WSAGetLastError());
        freeaddrinfo(result);
        WSACleanup();
        return 1;
    }

    // Setup the TCP listening socket
    iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("bind failed with error: %d\n", WSAGetLastError());
        freeaddrinfo(result);
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }

    freeaddrinfo(result);
    iResult = listen(ListenSocket, SOMAXCONN);
    if (iResult == SOCKET_ERROR) {
        printf("listen failed with error: %d\n", WSAGetLastError());
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }

    // Accept a client socket
    ClientSocket = accept(ListenSocket, NULL, NULL);
    if (ClientSocket == INVALID_SOCKET) {
        printf("accept failed with error: %d\n", WSAGetLastError());
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }
```

```
    // No longer need server socket
    closesocket(ListenSocket);
    // Receive until the peer shuts down the connection
    do {

        iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
        if (iResult > 0) {
            printf("Bytes received: %d\n", iResult);

            std::cout << std::string(recvbuf, 0, iResult) << std::endl;
            std::ofstream file("result.txt");
            file << std::string(recvbuf, 0, iResult);
            file.close();

        // Echo the buffer back to the sender
            iSendResult = send( ClientSocket, recvbuf, iResult, 0 );
            if (iSendResult == SOCKET_ERROR) {
                printf("send failed with error: %d\n", WSAGetLastError());
                closesocket(ClientSocket);
                WSACleanup();
                return 1;
            }
            printf("Bytes sent: %d\n", iSendResult);
        }
        else if (iResult == 0)
            printf("Connection closing...\n");
        else  {
            printf("recv failed with error: %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
        }

    } while (iResult > 0);

    // shutdown the connection since we're done
    iResult = shutdown(ClientSocket, SD_SEND);
    if (iResult == SOCKET_ERROR) {
        printf("shutdown failed with error: %d\n", WSAGetLastError());
        closesocket(ClientSocket);
        WSACleanup();
        return 1;
    }

    // cleanup
    closesocket(ClientSocket);
    WSACleanup();

    return 0;
}
```

Результат:

```
root@blackline:~/hw # g++ -o main main.cpp
root@blackline:~/hw # g++ -o client client.cpp
root@blackline:~/hw # ls
client          client.cpp      main            main.cpp
root@blackline:~/hw # main
main: Command not found.
root@blackline:~/hw # ./main
Input number: 123
Result: CXXIII
root@blackline:~/hw # ls
client          client.cpp      main            main.cpp        result.txt
root@blackline:~/hw # ./client
Trying to Send File Over Tcp!
The file "result.txt" has been successfully sent to the server.
SERVER> Data received:
CXXIII
root@blackline:~/hw # ee result.txt
```

Рис. 2. Сторона клиента

```
^[ (escape) menu ^y search prompt ^k delete line   ^p prev li     ^g prev page
^o ascii code    ^x search        ^l undelete line ^n next li     ^v next page
^u end of file   ^a begin of line ^w delete word   ^b back 1 char ^z next word
^t top of text   ^e end of line   ^r restore word  ^f forward char
^c command       ^d delete char   ^j undelete char              ESC-Enter: exit
=====line 1 col 6 lines from top 1 ============================================
CXXIII
```

Рис. 3. Файл результата на стороне клиента

```
PS D:\Files\Other\Study\MultinamesArchive\3 курс\2 семестр\Операционные системы\ДР> g++ -o .\server.exe .\server.cpp -lws2_32
PS D:\Files\Other\Study\MultinamesArchive\3 курс\2 семестр\Операционные системы\ДР> .\server.exe
Bytes received: 7
CXXIII
Bytes sent: 7
Connection closing...
```

Рис. 4. Сторона сервера

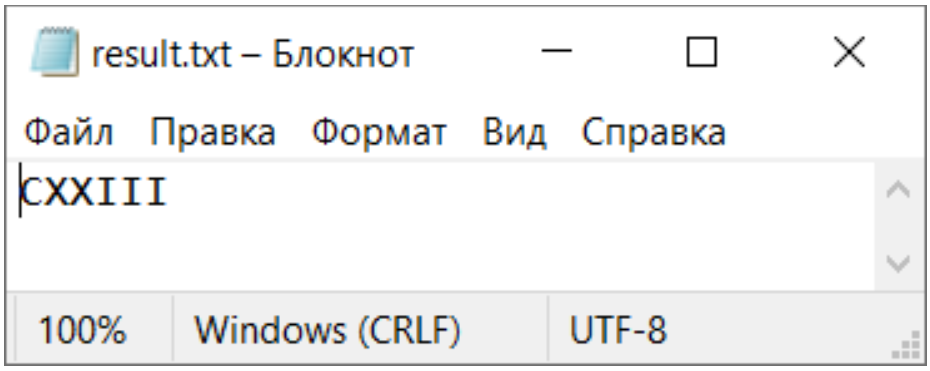

Рис. 5. Файл результата на стороне сервера

Вывод: в ходе выполнения домашней работы были получены практические навыки по написанию и отладке программ под ОС FreeBSD.

Контрольные вопросы:

1. **Назовите особенности ОС FreeBSD.**

FreeBSD основывается на 4.4BSD, стандартной промышленной версии UNIX, компилировать и запускать программы достаточно легко. FreeBSD также включает большую коллекцию пакетов и коллекцию портов, что обеспечивает лёгкость компиляции и установки уже откомпилированного программного обеспечения на вашей рабочей машине или сервере. Имеется также всё увеличивающееся количество коммерческих приложений, написанных для FreeBSD.

2. **Перечислите этапы разработки ПО.**

- Спецификация (определение требований к программе): На данном этапе происходит подробное описание исходных данных, осуществляется формулировка требований к 7 получаемому результату, рассматриваются всевозможные поведения программы при возникновении особых случаев (к примеру, если ввели неверные данные), происходит разработка диалоговых окон, которые обеспечат взаимодействие пользователя и самой программы.
- Разработка алгоритма: На этом этапе программист определяет последовательность необходимых действий, которые впоследствии нужно выполнить для получения желаемого результата. Результат данного этапа разработки программы — подробное словесное описание алгоритма программы, либо блок-схема алгоритма.
- Кодирование: После проведения спецификации и составления алгоритма решения, используемый алгоритм в итоге будет записан на необходимом языке программирования (Pascal, Delphi, C++ и др.). Результатом этапа кодирования является готовая программа.
- Отладка: На данном этапе программист занимается отладкой программы, то есть поиском и устранением ошибок. Последние делятся на две группы: алгоритмические и синтаксические (ошибки в тексте исходной программы). Из этих двух групп ошибок наиболее легко устранить синтаксические ошибки, тогда как алгоритмические ошибки определить достаточно трудно. Этап отладки считается законченным лишь тогда, когда исходная программа работает корректно и правильно при одном или двух наборах первичных данных.
- Тестирование: Тестирование программы очень важно, поскольку в большинстве случаев программисты создают программы не для личного применения, а чтоб их программой пользовались другие. На этапе тестирования разработчик проверяет поведение программы при большом числе наборов входных данных, как верных, так и специально подобранных неверных.

3. **Опишите этапы разработки ПО предшествующие написанию кода.**

- Спецификация (определение требований к программе): На данном этапе происходит подробное описание исходных данных, осуществляется формулировка требований к 7 получаемому результату, рассматриваются

всевозможные поведения программы при возникновении особых случаев (к примеру, если ввели неверные данные), происходит разработка диалоговых окон, которые обеспечат взаимодействие пользователя и самой программы.

- Разработка алгоритма: На этом этапе программист определяет последовательность необходимых действий, которые впоследствии нужно выполнить для получения желаемого результата. Результат данного этапа разработки программы — подробное словесное описание алгоритма программы, либо блок-схема алгоритма.

4. **Опишите понятие интерпретатор.**

В случае использования интерпретатора язык представляет собой оболочку, в которой вы набираете команды, а оболочка их выполняет. Для более сложных программ вы можете набрать команды в файле и использовать интерпретатор чтобы загрузить файл и выполнить команды из него. Если что-то идет не так, интерпретатор передает управление отладчику, с помощью которого можно решить проблему.

Плюсом этого подхода является возможность сразу увидеть результаты выполнения команд, а ошибки могут быть быстро исправлены. Недостаток является то, что для запуска программ на другом компьютере у него должен быть точно такой же интерпретатор. С точки зрения производительности, интерпретаторы могут использовать много памяти, и, как правило, генерируемый ими код не так эффективен, как код, генерируемый компиляторами.

5. **Назовите интерпретаторы, встроенные в ОС FreeBSD.**

BASIC, Lisp, Perl, Scheme, Icon, Logo, Python.

6. **Опишите понятие компилятор.**

Компиляторы достаточно сильно отличаются от интерпретаторов. Сначала код записывается в файл (или файлы), с помощью редактора. Затем необходимо запустить компилятор и проверить работает ли программа. Если она не компилируется, необходимо вернутся к редактированию и исправить ошибки; если код компилируется, и программа выполняется не успешно, можно запустить ее в отладчике для пошаговой проверки.

Этот процесс позволяет выполнять множество действий, которые затруднительно или невозможно сделать в интерпретаторе, к примеру, написать код, тесно взаимодействующий с операционной системой или даже написать собственную операционную систему. Это также полезно, если нужно написать очень эффективный код, так как компилятор может затратить время на оптимизацию кода, что может оказаться неудобным для интерпретатора. Кроме того, запустить программу на другом компьютере, написанной для компилятора, обычно проще, чем для интерпретатора, полагая, что используется та же операционная система.

7. Назовите компиляторы, встроенные в ОС FreeBSD.

В число компиляционных языков входят Pascal, C и C++. C и C++ являются более сложными языками со множеством возможностей, и больше подходят для опытных программистов; Pascal, с другой стороны, разрабатывался как язык для обучения, поэтому достаточно прост в освоении. В базовую поставку системы FreeBSD поддержка Pascal не включена, однако в коллекции портов имеется компилятор GNU Pascal Compiler (gpc). В базовую поставку FreeBSD IDE не входит, однако в дереве портов имеется evel/kdevelop, и в этих целях обычно используют Emacs.

8. Перечислите и опишите назначение ключей команды gcc.

Для компиляции потребуется программа gcc (cc, c++, g++).

% gcc foobar.cpp

Имеется огромное количество параметров для gcc, все они описаны на справочной странице. Вот несколько из самых важных, с примерами их использования.

-o

Имя выходного файла. Если вы не используете этот параметр, gcc сгенерирует выполнимый файл с именем a.out.

% gcc foobar.c исполняемый файл называется a.out

% gcc –o foobar foobar.c исполняемяемый файл называется foobar

-c

Выполнить только компиляцию файла, без компоновки. Полезно для программ, когда вы хотите просто проверить синтаксис, или при использовании Makefile.

% gcc –c foobar.c

В результате будет сгенерирован объектный файл (не исполнимый файл) с именем foobar.o. Он может быть скомпонован с другими объектными файлами для получения исполнимого файла.

-g

Создать отладочную версию выполнимого файла. Этот параметр указывает компилятору поместить в выполнимый файл информацию о том, какая строка какого файла с исходным текстом какому вызову функции соответствует.

-O

Создать оптимизированную версию исполняемого файла. Компилятор прибегает к различным ухищрениям для того, чтобы сгенерировать исполняемый файл, обработка которого быстрее, чем обычно.

9. Опишите принцип работы с отладчиком.

Отладчик, поставляемый с FreeBSD, называется gdb (GNU debugger). Он запускается при исполнении команды % gdb progname

Использование отладчика позволяет запустить программу в пошаговом режиме построчно, что позволит изучить значения переменных, изменить их, исполнить программу до определенного места, а затем остановиться, и так далее. Так же существует возможность подключиться к уже работающей

программе или загрузить файл дампа для изучения причины ошибки в программе. Возможно даже отладить ядро, хотя этот процесс является более сложным, чем отладка пользовательских приложений.

В gdb нужно ввести команду break main. Это укажет отладчику пропустить предварительный подготовительный код программы и начать сразу с необходимого участка кода. Теперь необходимо ввести команду run для запуска программы – она начнет выполняться с подготовительного кода и затем будет остановлена отладчиком при вызове main().

Теперь можно выполнять программу построчно по шагам, нажимая n. Если в программе используется вызов функции, то можно перейти в нее при нажатии клавиши s. Для того чтобы вернуться из пошагового выполнения функции необходимо нажать клавишу f. Можно также использовать команды up и down для просмотра вызывающей подпрограммы.

10.Опишите действия, необходимые для запуска программы.

Для запуска программы файлу необходимо установить права на исполнение % chmod +x progname. После этого, программу можно запустить % ./progname.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Вирт, Н. Разработка операционной системы и компилятора. Проект Оберон [Электронный ресурс] / Н. Вирт, Ю. Гуткнехт. — Москва: ДМК Пресс, 2012. 560 с. Режим доступа: https://e.lanbook.com/book/39992.
2. Войтов, Н.М. Основы работы с Linux. Учебный курс [Электронный ресурс]: учебное пособие / Н.М. Войтов. — Москва : ДМК Пресс, 2010. — 216 с. — Режим доступа: URL: https://e.lanbook.com/book/1198
3. Стащук, П.В. Краткое введение в операционные системы [Электронный ресурс] : учебное пособие / П.В. Стащук. — 3-е изд., стер. — Москва : ФЛИНТА, 2019. — 124 с.— URL: https://e.lanbook.com/book/125385

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Войтов, Н.М. Администрирование ОС Red Hat Enterprise Linux. Учебный курс [Электронный ресурс] : учеб. пособие — Москва: ДМК Пресс, 2011. 192 с. Режим доступа: https://e.lanbook.com/book/1081.
5. Стащук П.В. Администрирование и безопасность рабочих станций под управлением Mandriva Linux: лабораторный практикум. [Электронный ресурс]: учебно-методическое пособие / П.В. Стащук. — 2-е изд., стер. - М: Флинта, 2015. https://e.lanbook.com/book/70397

Электронные ресурсы:

1. Научная электронная библиотека http://eLIBRARY.RU.
2. Электронно-библиотечная система http://e.lanbook.com.
3. Электронно-библиотечная система «Университетская библиотека онлайн» http://biblioclub.ru.
4. Электронно-библиотечная система IPRBook http://www.iprbookshop.ru/
5. Losst - Linux Open Source Software Technologies https://losst.ru