Министерство образования и науки РФ

ФГБОУ ВО ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

ОТЧЕТ

о лабораторной работе №7

МНОГОПОТОЧНЫЕ ВЫЧИСЛЕНИЯ В ПРОГРАММАХ НА ЯЗЫКЕ СИ++

Дисциплина: Языки программирования

Группа: 18ПИ1

Выполнил: Нестеров И.С.

Количество баллов:

Дата сдачи:

Принял: к.т.н., доцент Лупанов М.Ю.

Пенза 2019

1 Цель работы

1.1 Освоить компоненты стандартной библиотеки Си++ для создания многопоточных приложений .

2 Задание к лабораторной работе

2.1 В одном университете студенты-гуманитарии использовали пароли, получаемые перестановкой символов в строке 123456789. А студенты из соседнего колледжа написали функцию для подбора таких паролей, если известен хэш от пароля. Код функции приведен в приложении Е. Варианты хэшей паролей студентов-гуманитариев приведены в приложении Ж. Используя функцию из приложения Е напишите программу (однопоточную), которая может последовательно находить пароли для любого количества хэшей от 1 до 8 включительно. Программа не должна интерактивно взаимодействовать с пользователем. Хэши должны либо передаваться через параметры командной строки, либо читаться из файла.

2.2 Модифицируйте функцию из приложения Е таким образом, чтобы она могла быть использована в многопоточной программе. Реализуйте с использованием этой модифицированной функции многопоточную программу, которая может параллельно находить пароли для любого количества хэшей от 1 до 8 включительно. Программа не должна интерактивно взаимодействовать с пользователем. Хэши должны либо передаваться через параметры командной строки, либо читаться из файла.

2.3 Выполните сравнение скорости подбора паролей для программ, разработанных в первом и втором задании. Для этого воспользуйтесь утилитой time. Формат запуска: time ./myprog, где myprog — ваша программа. Сравнение проводить для поиска одного, двух, четырех и восьми паролей на одних и тех же выборках хэшей. Выборку хэшей сделать самостоятельно. Результаты

эксперимента свести в таблицу и проанализировать. Использовать таблицу, аналогичную таблице И.1 приложения И.

2.4 Задание повышенной сложности. Модифицируйте функцию из приложения Е таким образом, чтобы она выполняла не полный перебор всего диапазона паролей, а частичный, от одного значения пароля до другого. Реализуйте программу, делящую весь диапазон паролей на 2, 4 или 8 частей и выполняющую подбор одного пароля по частям в параллельных потоках. Программа 48 также должна позволять подбирать пароль в одном потоке, без деления диапазона на части.

2.5 Задание повышенной сложности. Выполните сравнение скорости подбора пароля для первого значения хэш-функции из таблицы Ж.1 приложения Ж для случаев без деления на диапазоны, деления на два, четыре и восемь диапазонов с параллельным подбором в каждом диапазоне с помощью программы time. В качестве границ диапазонов использовать значения 231456789, 341256789, 451236789, 561234789, 671234589, 781234569, 891234567. Результаты эксперимента свести в таблицу и проанализировать. Использовать таблицу, аналогичную таблице И.2 приложения И.

3 Результаты работы

3.1 Работа программы представлена на Рисунке 1. Полный текст программы представлен в приложении А.

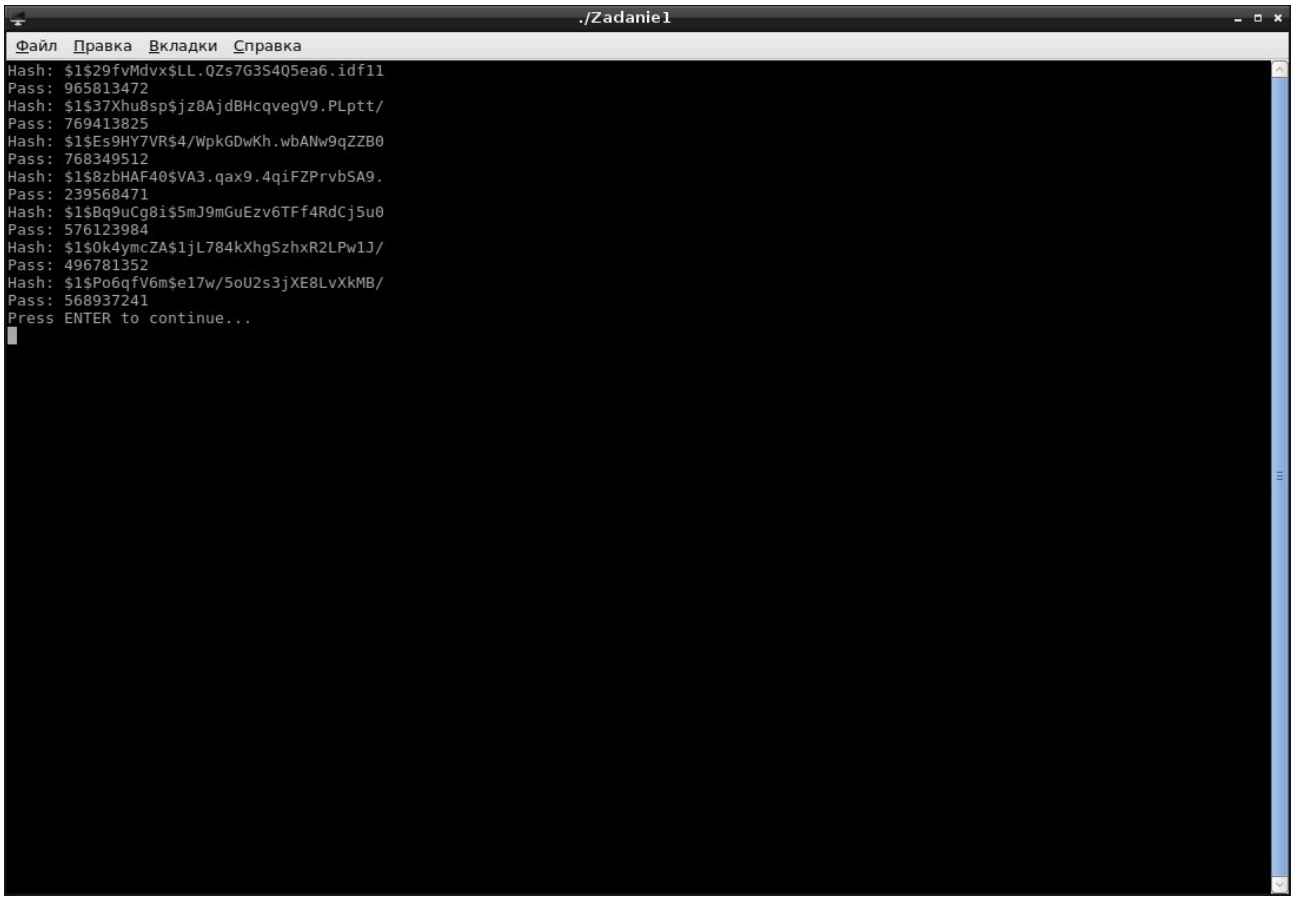

Рисунок 1 - Работа программы

3.2 Работа программы представлена на Рисунке 2. Полный текст программы представлен в приложении Б.

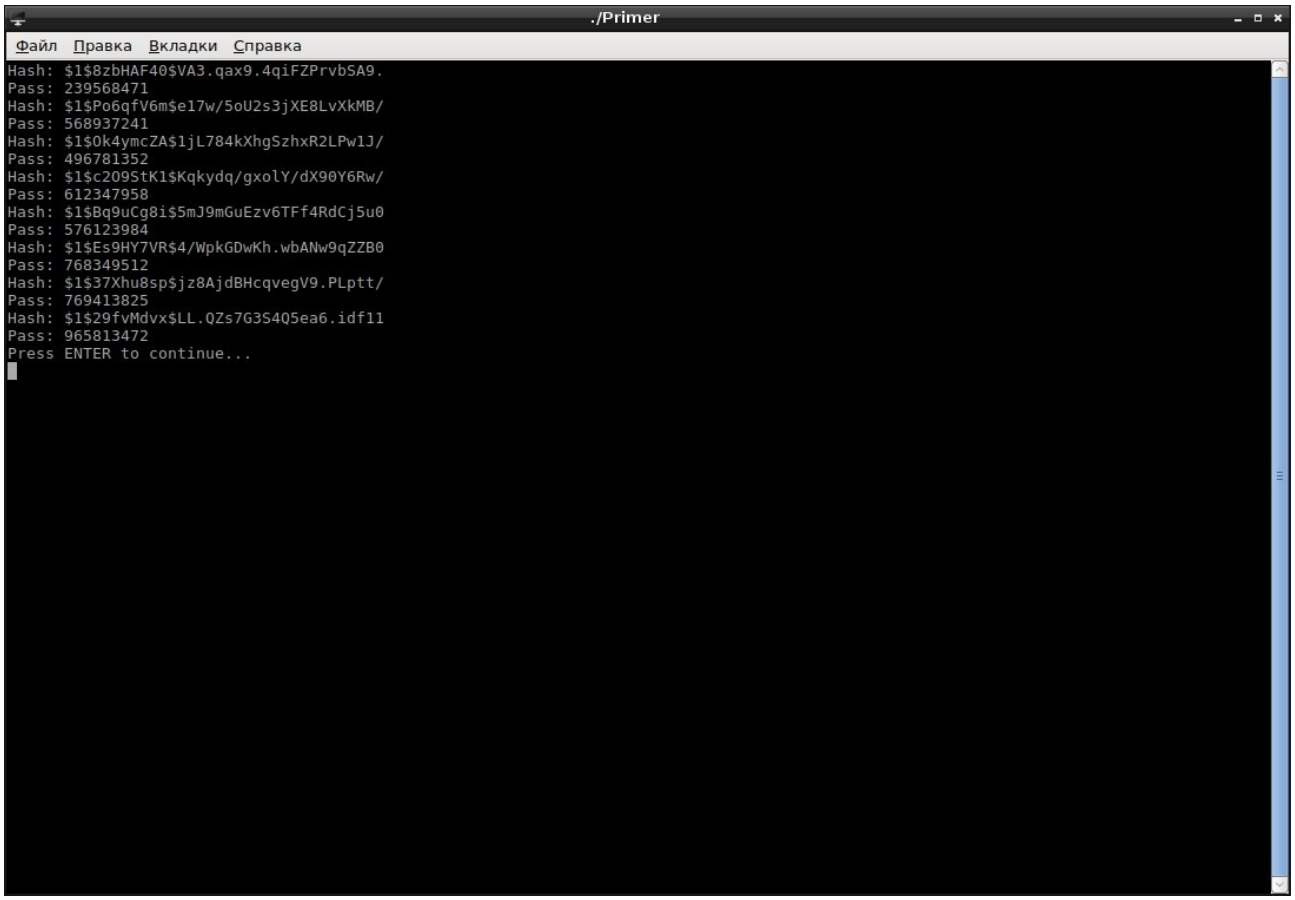

Рисунок 2 - Работа программы

3.3 Время работ и результаты сравнения программ представлено на Таблице 1.

Таблица 1 - Результаты сравнения времени подбора для одно- и многопоточных приложений

Количество хэшей	Однопоточная программа		Многопоточная программа	
	User, c	Real, c	User, c	Real, c
1	1m15,659 s	1m16, 654s	1m5, 180s	1m4, 232s
2	2m16, 534s	1m14, 822s	1m55, 523s	1m50,434s
4	2m55, 534s	2m57, 227s	2m48, 643s	1m44, 142s
8	5m19, 534s	5m16, 165s	5m14, 452s	1m28, 564s

Как видно из таблицы, многопоточная программа затрачивает меньше времени с увеличением количества потоков и хэшей, чем однопоточная, за исключением восьми потоков, так как тестирование проводилось на четырех ядерном процессоре.

3.4 Работа программы представлена на Рисунке 3. Полный текст программы представлен в приложении В.

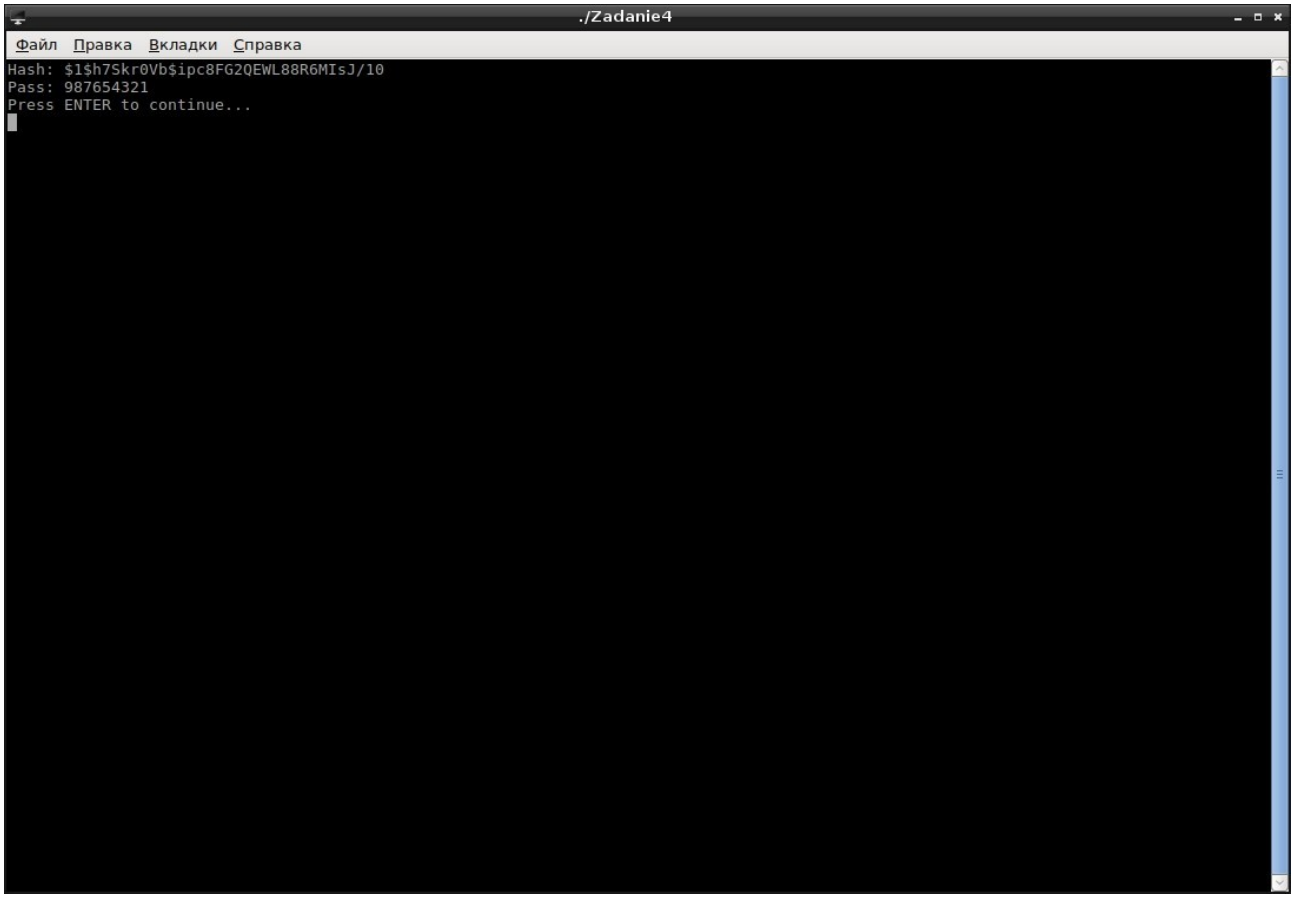

Рисунок 3 - Работа программы

3.5 Время работ и результаты сравнения программ представлено на Таблице 2.

Таблица 2 - Результаты сравнения времени подбора одного пароля при распараллеливании задачи

Количество диапазонов	Время подбора одного пароля	
	User, c	Real, c
1	1m5, 324s	1m5,121s
2	1m5, 523s	0m34, 543s
4	1m6, 325s	0m29, 432s
8	1m6, 421s	0m18, 432s

Как видно из таблицы, с увеличением потоков, время работы программы уменьшается.

4 Вывод

В результате выполнения работы были изучены потоки языка Си++, а также были написаны и модифицированы функции, которые подбирали пароли, и получены практические навыки в написании многопоточных программ на с++.

Приложение А

Текст программы однопоточной программы

```
#include <iostream>
#include <thread>
#include <chrono>
#include <algorithm>
#include <crypt.h>
#include <iomanip>
#include <fstream>
using namespace std;

void findPass(string pass, const string& hash);

int main(int argc, char* argv[])
{
    int i = 0, j = 0;
    ifstream f("/root/Laba8/Zadanie1/hashs");
     f.seekg(0, ios::end);
     int fsize = f.tellg();
     f.seekg(0, ios::beg);
     char *buf = new char [fsize];
     char *save = new char[60];
     f.read(buf, fsize);
     for (i = 0; i < fsize; i++)
     {
          if (buf[i] == '\n' || i == (fsize-1))
          {
               save[j] = '\0';
            findPass("123456789", save);
               delete[] save;
               save = new char[60];
               j = 0;
          }
```

```
            else
            {
                save[j] = buf[i];
                ++j;
            }
     }
    return 0;
}

void findPass(string pass, const string& hash)
{
    crypt_data *pCryptData = new crypt_data;
    size_t pos = hash.find_last_of('$');

    string hashHead = hash.substr(0,pos);
    do {
                                                                string
newHash(crypt_r(pass.data(),hashHead.data(),pCryptData));
        if (newHash == hash) {
            cout<<"Hash: "<<hash<<endl<<"Pass: "<<pass<<endl;
            break;
        }
    } while ( next_permutation( pass.begin(), pass.end() ) );
    delete pCryptData;
}
```

Приложение Б

Текст многопоточной программы

```
#include <iostream>
#include <thread>
#include <chrono>
#include <algorithm>
#include <crypt.h>
#include <iomanip>
#include <fstream>
#include <mutex>
using namespace std;

void findPass(string pass, const string& hash);

int main(int argc, char* argv[])
{
                                                                thread
th_1(findPass,"123456789","$1$29fvMdvx$LL.QZs7G3S4Q5ea6.idf11");
                                                                thread
th_2(findPass,"123456789","$1$37Xhu8sp$jz8AjdBHcqvegV9.PLptt/");
                                                                thread
th_3(findPass,"123456789","$1$Es9HY7VR$4/WpkGDwKh.wbANw9qZZB0");
                                                                thread
th_4(findPass,"123456789","$1$8zbHAF40$VA3.qax9.4qiFZPrvbSA9.");
                                                                thread
th_5(findPass,"123456789","$1$Bq9uCg8i$5mJ9mGuEzv6TFf4RdCj5u0");
                                                                thread
th_6(findPass,"123456789","$1$Ok4ymcZA$1jL784kXhgSzhxR2LPw1J/");
                                                                thread
th_7(findPass,"123456789","$1$Po6qfV6m$e17w/5oU2s3jXE8LvXkMB/");
                                                                thread
th_8(findPass,"123456789","$1$c2O9StK1$Kqkydq/gxolY/dX90Y6Rw/");
    th_1.join();
```

```
    th_2.join();
    th_3.join();
    th_4.join();
    th_5.join();
    th_6.join();
    th_7.join();
    th_8.join();
    return 0;
}

void findPass(string pass, const string& hash)
{
    static mutex mtx;
    crypt_data *pCryptData = new crypt_data;
    size_t pos = hash.find_last_of('$');
    string hashHead = hash.substr(0,pos);
    do {
                                                                string
newHash(crypt_r(pass.data(),hashHead.data(),pCryptData));
        if (newHash == hash) {
            lock_guard<mutex> lock(mtx);
            cout<<"Hash: "<<hash<<endl<<"Pass: "<<pass<<endl;
            break;
        }
    } while ( next_permutation( pass.begin(), pass.end() ) );
    delete pCryptData;
}
```

Приложение В

Текст измененной многопоточной программы

```
#include <iostream>
#include <thread>
#include <chrono>
#include <algorithm>
#include <crypt.h>
#include <iomanip>
#include <fstream>
#include <mutex>
using namespace std;

void findPass(string pass, string pass2, const string& hash);

int main(int argc, char* argv[])
{
                              thread       th_1(findPass,"123456789",
"231456789","$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_2(findPass,"231456789",    "341256789",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_3(findPass,"341256789",    "451236789",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_4(findPass,"341256789",    "451236789",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_5(findPass,"451236789",    "561234789",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_6(findPass,"561234789",    "671234589",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_7(findPass,"671234589",    "891234567",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
                thread    th_8(findPass,"891234567",    "987654321",
"$1$h7Skr0Vb$ipc8FG2QEWL88R6MIsJ/10");
    th_1.join();
```

```
    th_2.join();
    th_3.join();
    th_4.join();
    th_5.join();
    th_6.join();
    th_7.join();
    th_8.join();
    return 0;
}
void findPass(string pass1, string pass2, const string& hash)
{
    static mutex mtx;
    crypt_data *pCryptData = new crypt_data;
    size_t pos = hash.find_last_of('$');
    string hashHead = hash.substr(0,pos);
    do {
                                                                string
newHash(crypt_r(pass1.data(),hashHead.data(),pCryptData));
        if (newHash == hash) {
            lock_guard<mutex> lock(mtx);
            cout<<"Hash: "<<hash<<endl<<"Pass: "<<pass1<<endl;
            break;
        }
        if (pass1 == pass2) break;
    } while ( next_permutation( pass1.begin(), pass1.end() ) );
    delete pCryptData;
}
```