Оглавление

[1 Постановка задачи 3](#_Toc57067596)

[2 Алгоритм решения задачи 3](#_Toc57067597)

[3 Теоретическая оценка вычислительной и коммуникационной сложности реализованного алгоритма 3](#_Toc57067598)

[3.1 Вычислительная сложность 3](#_Toc57067599)

[3.2 Коммуникационная сложность 3](#_Toc57067600)

[4 Внешние спецификации программного средства вскрытия пароля Unix по словарю 4](#_Toc57067601)

[4.1 Наименование и назначение программного средства 4](#_Toc57067602)

[4.2 Команда сборки программы 4](#_Toc57067603)

[4.3 Формат вызова программы 4](#_Toc57067604)

[4.4 Входные данные 5](#_Toc57067605)

[4.5 Выходные данные 5](#_Toc57067606)

[4.6 Внешние эффекты 5](#_Toc57067607)

[5 Внешние спецификации программного средства шифрования парольной фразы функцией crypt() 6](#_Toc57067608)

[5.1 Наименование и назначение программного средства 6](#_Toc57067609)

[5.2 Команда сборки программы 6](#_Toc57067610)

[5.3 Формат вызова программы 7](#_Toc57067611)

[5.4 Входные данные 7](#_Toc57067612)

[5.5 Выходные данные 7](#_Toc57067613)

[5.6 Внешние эффекты 7](#_Toc57067614)

[6 Результаты вычислительного эксперимента 8](#_Toc57067615)

[Вывод 10](#_Toc57067616)

[Приложение А Код программы pwdcrypter 11](#_Toc57067617)

[Приложение Б Код программы pwdcrypter 12](#_Toc57067618)

# Постановка задачи

Имеется зашифрованный пароль Unix и текстовый файл со списком возможных паролей. Необходимо произвести дешифрование пароля методом перебора всех паролей из файла. Каждая строка файла шифруется с помощью функции crypt(), результат сравнивается с исходным паролем. Если обнаруживается совпадение, пароль считается вскрытым. Если перебраны все строки из файла, и совпадение не обнаружено, то пароль считается невскрытым.

# Алгоритм решения задачи

Объем вычислений определяется числом строк в файле паролей n. Пусть для вычислений выделено m процессов. Файл делится на равные порции, каждый процесс обрабатывает свою порцию. Если граница начала порции попала на середину строки, порция обрабатывается со следующей строки. Если граница конца порции попала на середину строки, эта последняя строка обрабатывается полностью.

При проведении вычислительного эксперимента используется файл без искомого пароля.

# Теоретическая оценка вычислительной и коммуникационной сложности реализованного алгоритма

## Вычислительная сложность

Пусть n – число строк в файле словаря, для вычислений выделено m процессов.

Общее количество входных строк n делится на число процессов m. Каждый процесс обрабатывает свою порцию данных. Теоретическая сложность перебора входных строк в каждом потоке в среднем будет равна .

## Коммуникационная сложность

Пусть n – число строк в файле паролей, для вычислений выделено m процессов. Когда один из процессов находит искомый ключ, он сообщает другим процессам о необходимости завершения работы. Количество сообщений, которые он посылает, всегда меньше m – по одному сообщению на процесс, кроме себя самого.

При этом на каждой итерации цикла, в котором обрабатываются входные строки, осуществляется проверка на необходимость завершения обработки. Таким образом, максимально на одной итерации цикла процесс может проверить необходимость завершения, затем найти пароль и отправить m-1 сообщение. Процесс будет иметь коммуникационную сложность .

# Внешние спецификации программного средства вскрытия пароля Unix по словарю

## Наименование и назначение программного средства

Программное средство pwdfinder используется для дешифрования пароля Unix методом перебора всех строк из файла со списком всех возможных паролей. Каждая строка файла шифруется с помощью функции crypt(), результат сравнивается с исходным паролем. Если обнаруживается совпадение, пароль считается вскрытым. Если перебраны все строки из файла, и совпадение не обнаружено, то пароль считается невскрытым.

## Команда сборки программы

mpic++ -o <pwdfinder> pwdfinder.cpp -lcrypt

Компилятор mpic++ используется, так как программа использует функционал языка программирования С++.

* pwdfinder указывает имя выходного файла для компиляции;
* lcrypt необходим для того, чтобы компоновщик подключал библиотеку crypt, и функция crypt() работала корректно;

## Формат вызова программы

mpirun -np <p> <pwdfinder> <input\_dictionary> <password\_hash>

* np указывает количество процессов, на которых запускается ПС;
* input\_dictionary – имя файла со списком всех возможных паролей, находящегося в каталоге, из которого запускается программа;
* password\_hash – зашифрованный пароль Unix, который необходимо дешифровать.

## Входные данные

Файл со списком всех возможных паролей должен располагаться в каталоге, из которого запускается программа. Название этого файла передается в качестве первого аргумента командной строки.

Данные в файле имеют текстовый формат. Формат файла: одна строка – один пароль. Максимальная длина строки – 40 символов. Пароль может содержать латинские буквы в верхнем и нижнем регистрах, цифры и специальные символы ASCII (!"#$%&'()\*+,-./:;<=>?@[\]^\_ `{|}~).

## Выходные данные

Текстовый файл «result» с результатом работы ПС создается в процессе работы программы в каталоге, из которого запускается программа.

В случае вскрытия пароля, файл result содержит построчно: пароль, ранг нашедшего его процесса, время поиска в секундах с точностью до десятых долей секунды. Время будет выведено в числовом формате. Дробная часть числа отделена от целой части точкой.

В случае невскрытия пароля, файл result содержит время поиска в секундах с точностью до десятых долей секунды. Время будет выведено в числовом формате. Дробная часть числа отделена от целой части точкой.

## Внешние эффекты

При возникновении ошибок в ходе работы программы, предусмотрен вывод ошибок в стандартный поток ошибок, с указанием кода ошибки и сообщения об ошибке. В случае неверного содержимого входной строки с паролем программа переходит к обработке следующей строки.

Коды ошибок и их описание представлены в таблице 1.

Таблица 1. Внешние эффекты

|  |  |  |
| --- | --- | --- |
| **Код ошибки** | **Сообщение об ошибке** | **Описание** |
| **1** | Код ошибки: 1 «Неверные входные параметры». | Пользователь указал неверные аргументы в командной строке. Использование:  «mpirun [-np <p>] <pwdfinder> <input\_dictionary> <password\_hash>». |
| **2** | Код ошибки: 2 «<pwdfinder>: Невозможно открыть входной файл <input\_dictionary> по причине: <reason>». | Программное средство не может получить доступ к входным данным. Причина, по которой файл не может быть открыт, указана в параметре <reason>. |
| **3** | Код ошибки: 3 «<pwdfinder>: Невозможно создать выходной файл result по причине: <reason>». | Программное средство не может создать файл для выходных данных. Причина, по которой файл не может быть создан, указана в параметре <reason>. |
| **4** | Код ошибки: 4 «Входная строка (строка из файла <input\_dictionary>) пропущена. Неверная длина пароля». | Нужно поверить содержимое входного файла <input\_dictionary>. Длина входной парольной фразы не должна превышать 40 символов. |
| **5** | Код ошибки: 5 «Входная строка (строка из файла <input\_dictionary>) пропущена. Строка содержит символы не из специфицированного набора». | Нужно поверить содержимое входного файла <input\_dictionary>. Входная парольная фраза может содержать только латинские буквы в верхнем и нижнем регистрах, цифры и специальные символы ASCII (!"#$%&'()\*+,-./:;<=>?@[\]^\_ `{|}~). |

# Внешние спецификации программного средства шифрования парольной фразы функцией crypt()

## Наименование и назначение программного средства

Программное средство pwdcrypter используется для шифрования пароля Unix. Парольная фраза шифруется с помощью функции crypt(), результат (зашифрованный пароль) выводится в стандартный поток вывода.

## Команда сборки программы

g++ -o <pwdcrypter> pwdcrypter.cpp -lcrypt

* pwdcrypter указывает имя выходного файла для компиляции;
* lcrypt необходим для того, чтобы компоновщик подключал библиотеку crypt, и функция crypt() работала корректно;

## Формат вызова программы

./<pwdcrypter> <password>

* password – пароль Unix, который необходимо зашифровать.

## Входные данные

Пароль Unix, который необходимо зашифровать. Максимальная длина пароля – 40 символов. Пароль может содержать латинские буквы в верхнем и нижнем регистрах, цифры и специальные символы ASCII (!"#$%&'()\*+,-./:;<=>?@[\]^\_ `{|}~).

## Выходные данные

Результат (зашифрованный пароль) выводится в стандартный поток вывода.

## Внешние эффекты

При возникновении ошибок в ходе работы программы, предусмотрен вывод ошибок в стандартный поток ошибок stderr, с указанием кода ошибки и сообщения об ошибке.

Коды ошибок и их описание представлены в таблице 2.

Таблица 2. Внешние эффекты

|  |  |  |
| --- | --- | --- |
| **Код ошибки** | **Сообщение об ошибке** | **Описание** |
| **1** | Код ошибки: 1 «Неверные входные параметры». | Пользователь указал неверные аргументы в командной строке. Использование: «./<pwdcrypter> <password>». |
| **2** | Код ошибки: 2 «Входная строка <password> имеет неверную длину». | Длина входной парольной фразы не должна превышать 40 символов. |
| **3** | Код ошибки: 3 «Входная строка <password> содержит символы не из специфицированного набора». | Парольная фраза может содержать только латинские буквы в верхнем и нижнем регистрах, цифры и специальные символы ASCII (!"#$%&'()\*+,-./:;<=>?@[\]^\_ `{|}~). |

# Результаты вычислительного эксперимента

Результатом эксперимента являются: зависимость времени выполнения программы от числа процессов **m** и числа строк в входном файле **n.**

Выигрыш в скорости перебора на большом числе процессов наблюдается в таблице 3 и на рисунке 1.

Таблица 3. Зависимость времени выполнения программы от от числа процессов **m (расположены в столбцах)** и числа строк в входном файле **n (расположены в строках)**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **4** | **8** | **16** | **32** |
| 1500000 | 11.95 | 7.00 | 3.83 | 2.37 | 1.36 | 0.90 |
| 3000000 | 24.61 | 12.74 | 7.46 | 5.24 | 2.76 | 1.76 |
| 4500000 | 36.40 | 20.04 | 10.90 | 6.80 | 4.13 | 2.61 |
| 6000000 | 48.44 | 26.71 | 14.48 | 8.84 | 5.35 | 3.62 |
| 7500000 | 59.93 | 31.72 | 18.53 | 11.69 | 6.69 | 4.87 |
| 9000000 | 70.89 | 39.05 | 21.68 | 14.29 | 7.37 | 4.88 |

Рисунок 1 – График зависимости времени выполнения программы от числа процессов

Вывод

При написании долгосрочного домашнего задания были изучены функции MPI для работы с файлами. Эти функции обеспечивают параллельный доступ на чтение и запись к файлам без дополнительных блокировок.

В программном средстве pwdfinder динамически выделялась память на некоторые переменные. При работе с динамической памятью важно правильно выделять размер, во избежание ошибок сегментации, и удалять переменные после их использования.

Было написано программное средство pwdfinder позволяющее выполнить подбор пароля по хэшу, с использованием словаря. Программа хорошо себя показала, при росте количества процессов наблюдается ускорение работы. Перебор входных строк осуществляется достаточно быстро. Возможно использование программы для взлома сложных паролей, при увеличении количества строк во входном словаре.

Приложение А   
Код программы pwdcrypter

/\* Программа шифрует пароль функцией crypt().

\*

\* Использование: ./pwdcrypter <password>

\*

\* Программа принимает аргументы:

\* argv[1] - Пароль, который необходимо зашифровать.

\*/

#include <crypt.h>// Для использования crypt()

#include <string> // Для использования std::string

#include <stdio.h> // Для вывода информации (printf, fprintf, sprintf)

int main(int argc, char \*\*argv) {

//Вывод ошибки и выход при неверном количестве входных аргументов

if (argc != 2) {

fprintf(stderr, "\nUsage: %s password\n\n", argv[0]);

return 1;

}

std::string password = argv[1];

//Вывод ошибки и выход при неверной длине пароля

if ( password.length() > 40) {

fprintf(stderr, "\nWrong password. Max length is 40.\nCharacters allowed: A-Za-z0-9space!\"#$%%&'()\*+,-./:;<=>?@[\\]^\_`{|}~\n\n");

return 2;

}

//Вывод результата работы программы

fprintf( stdout, "\nResult:[%s]\n\n", crypt( argv[1], "any./Sa1t/") );

return 0;

}

Приложение Б  
 Код программы pwdcrypter

/\* Программное средство pwdfinder используется для дешифрования пароля Unix

\* методом перебора всех строк из файла со списком всех возможных паролей.

\* Каждая строка файла шифруется с помощью функции crypt(), результат сравнивается с исходным паролем.

\* Если обнаруживается совпадение, пароль считается вскрытым.

\* Если перебраны все строки из файла, и совпадение не обнаружено, то пароль считается невскрытым.

\*

\* Использование: mpirun -np <p> <pwdfinder> <input\_dictionary> <password\_hash>

\*

\* Программа принимает аргументы:

\* argv[1] - Имя входного файла с паролями для перебора

\* argv[2] - Хэш, который необходимо дешифровать

\*/

#include <stdio.h>// Для вывода информации (printf, fprintf, sprintf)

#include <mpi.h>// Для работы функционала MPI

#include <ctype.h> // Для функций isalpha, isdigit, isspace

#include <string.h> // Для использования strlen

#include <string> // Для использования std::string

#include <crypt.h> // Для использования crypt()

#include <errno.h>// Для использования глобальной переменной errno

/\* Функция разбивает входной файл на части (chunks).

\* Затем каждый процесс параллельно считывает свою порцию данных.

\* При этом к каждой строке входных данных применяется функция crypt.

\*

\* Функция принимает параметры:

\* MPI\_File \*in - Указатель на входной файл для параллельного чтения

\* const int rank - Ранг для идентификации процесса

\* const int proc\_cnt - Общее количество процессов

\* const int delta - Размер "пересечения" (для корректного разделения по строкам)

\* const char \*pwd\_hash - Хэш от пароля, который необходимо вскрыть

\*/

std::string process\_in\_chunks(MPI\_File \*in, const int rank, const int proc\_cnt, const int delta, char \*pwd\_hash) {

MPI\_Request request;

std::string word = "";

MPI\_Offset globalstart;

MPI\_Offset globalend;

MPI\_Offset filesize;

MPI\_Offset chunk\_size;

int flag\_msg = 0;

int flag\_len\_err = 0;

int flag\_sym\_err = 0;

char \*cmp\_res = new char[13];

char c;

//Массив разрешенных символов

const char allowed\_sym[] = {'!','"','#','$','%',' ','&','\'','(',')','\*','+','-','.','/',':',';', \

'<','=','>','?','@','[','\\',']','^','\_','`','{','|','}','~'};

MPI\_File\_get\_size(\*in, &filesize);

//Файл делится на части, основываясь на рангах

chunk\_size = filesize / proc\_cnt;

globalstart = rank \* chunk\_size;

globalend = globalstart + chunk\_size - 1;

if (rank == proc\_cnt-1) globalend = filesize-1;

//Добавить дополнительный интервал, чтобы не обрезать строку с паролем

if (rank != proc\_cnt-1) globalend += delta;

chunk\_size = globalend - globalstart + 1;

char \*chunk = new char[chunk\_size + 1];

//Чтение данных в переменную chunk

MPI\_File\_read\_at\_all(\*in, globalstart, chunk, chunk\_size, MPI\_CHAR, MPI\_STATUS\_IGNORE);

chunk[chunk\_size] = '\0';

int locstart=0, locend=chunk\_size-1;

//Обрезаем лишние данные, порция данных начинается с \n и заканчивается \n (кроме начала и конца файла)

if (rank != 0) {

while(chunk[locstart] != '\n') locstart++;

locstart++;

}

if (rank != proc\_cnt-1) {

locend-=delta;

while(chunk[locend] != '\n') locend++;

}

chunk\_size = locend-locstart+1;

int i=locstart;

//Посимвольный перебор порции данных

while(i<=locend) {

//Сообщение о необходимости завершения работы (пароль найден)

MPI\_Iprobe( MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD, &flag\_msg, MPI\_STATUS\_IGNORE);

if (flag\_msg == 1) break;

c = chunk[i];

int iter = 0;

//В цикле выделяются слова

while(!isspace(c) || chunk[i] != '\n') {

word += c;

i++;

iter++;

//Проверка на превышение длины

if (iter >= 40) flag\_len\_err = 1;

//Проверка на наличие посторонних символов

if (!isalpha(c) & !isdigit(c)) {

if (!flag\_sym\_err)

for (int k = 0; k < 31; k++) {

if (allowed\_sym[k] == c) {

flag\_sym\_err = 0;

break;

}

else flag\_sym\_err = 1;

}

}

//Проверка на конец файла

if (i>locend) break;

c = chunk[i];

}

iter = 0;

if (flag\_len\_err) {

fprintf(stderr, "\nInput string %s is missed (Length error)\n", word.c\_str());

flag\_len\_err = 0;

}

else if (flag\_sym\_err) {

fprintf(stderr, "\nInput string %s is missed (Bad symbols error)\n", word.c\_str());

flag\_sym\_err = 0;

}

else {

//Шифрование слова

strcpy( cmp\_res , crypt(word.c\_str(),pwd\_hash) );

//Сравнение результатов

if (strcmp(pwd\_hash, cmp\_res) == 0) {

//Сигнал к завершению работы другими процессами

for(int i = 0; i < proc\_cnt; i++)

if (i != rank)

MPI\_Isend( &rank, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, &request);

delete chunk;

delete cmp\_res;

return word;

}

}

word = "";

i++;

}

delete chunk;

delete cmp\_res;

return "";

}

int main(int argc, char \*\*argv) {

MPI\_File in, out;

int rank, proc\_cnt;

int ierr;

char \*pwd\_hash = new char[13];

const int delta = 100;

double starttime, endtime;

std::string res;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_cnt);

//Проверка количества входных аргументов

if (argc != 3) {

if (rank == 0) fprintf(stderr, "Usage: %s infilename password\_hash\n", argv[0]);

MPI\_Finalize();

return 1;

}

//Открытие входного файла

ierr = MPI\_File\_open(MPI\_COMM\_WORLD, argv[1], MPI\_MODE\_RDONLY, MPI\_INFO\_NULL, &in);

if (ierr) {

if (rank == 0) fprintf(stderr, "%s: Couldn't open file %s (%s)\n", argv[0], argv[1], strerror(errno));

MPI\_Finalize();

return 2;

}

//Удалить выходной файл, если уже существует

MPI\_File\_open(MPI\_COMM\_WORLD, "result", MPI\_MODE\_CREATE|MPI\_MODE\_DELETE\_ON\_CLOSE|MPI\_MODE\_WRONLY, MPI\_INFO\_NULL, &out);

MPI\_File\_close(&out);

//Открытие выходного файла

ierr = MPI\_File\_open(MPI\_COMM\_WORLD, "result", MPI\_MODE\_CREATE | MPI\_MODE\_WRONLY, MPI\_INFO\_NULL, &out);

if (ierr) {

if (rank == 0) fprintf(stderr, "%s: Couldn't open file 'result' (%s)\n", argv[0], strerror(errno));

MPI\_Finalize();

return 3;

}

//В алгоритме DES функциии crypt выходной хэш имеет фиксированную длину - 13 символов

if ((strlen(argv[2]) != 13)) {

if (rank == 0) fprintf(stderr, "Incorrect hash %s Size should be 13 symbols (%ld given)\n", argv[2], strlen(argv[2]));

MPI\_Finalize();

return 4;

}

//Обработка данных по частям

starttime = MPI\_Wtime();

res = process\_in\_chunks(&in, rank, proc\_cnt, delta, argv[2]);

endtime = MPI\_Wtime();

MPI\_File\_close(&in);

//Пишет процесс, который нашел пароль

if (res != "") {

char out\_buffer[70];

int n = sprintf(out\_buffer,"Password is: %s\nFound on rank: %d\n", res.c\_str(), rank);

MPI\_File\_write(out,out\_buffer,n, MPI\_CHAR,MPI\_STATUS\_IGNORE);

}

//Дождаться, пока данные будут записаны

MPI\_Barrier(MPI\_COMM\_WORLD);

//Дописать время выполнения программы

if (rank == 0) {

char out\_buffer[70];

MPI\_File\_seek( out, 0, MPI\_SEEK\_END );

int n = sprintf(out\_buffer,"Searching time is: %.2f\n", endtime-starttime);

MPI\_File\_write(out,out\_buffer,n, MPI\_CHAR,MPI\_STATUS\_IGNORE);

}

delete pwd\_hash;

MPI\_File\_close(&out);

MPI\_Finalize();

return 0;

}