МІНІСТЕРСТВО ОСВІТИ І НАУКИ, НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Інститут післядипломної освіти

**Розрахунково-графічна робота**

з дисципліни «Операційні системи»

Варіант №2

Виконав:

Студент групи 7.05013001

Пікалюк Р.М.

Перевірив:

Київ 2016

Зміст

[2.1. Завдання. 3](#_Toc443008176)

[2.2. Теоретичні відомості, опис використаних функцій та методів 4](#_Toc443008177)

[2.3. Текст програми з коментарями 5](#_Toc443008178)

[2.4. Результати роботи. 10](#_Toc443008179)

[2.5. Висновки. 14](#_Toc443008180)

# 2.1. Завдання.

Завдання 1.

Shell-програма переглядає каталог, ім'я якого вказане параметром Shell-програми і виводить імена каталогів, що зустрілися. Потім здійснює перехід в батьківський каталог, який стає поточним, і повторюються вказані дії доти, доки поточним каталогом не стане кореневий каталог. Форма виведення результату:

|  |  |
| --- | --- |
| **каталог <ім’я каталогу >** | ← **початковий каталог** |
| **каталог <ім’я >** |  |
| **каталог <ім’я >** | **каталоги в поточному каталозі** |
| **……………….** |  |
| **каталог <ім’я каталогу >** | ← **батьківський підкаталог** |
| **каталог <ім’я >** |  |
| **каталог <ім’я >** | **каталоги в поточному каталозі** |
| **……………….** |  |

**і т.д.**

Завдання 2.

Розробити програму, яка переглядає поточний каталог і виводить на екран імена усіх файлів, що зустрілися в ньому, із заданим розширенням. Потім здійснюється перехід в батьківський каталог, який потім стає поточним, і вказані вище дії повторюються доти, доки поточним каталогом не стане кореневий каталог.

Завдання 3.

Написати програму, яка шифрує зміст заданого файлу методом Цезаря (ключ задавати через командний рядок).

# 2.2. Теоретичні відомості, опис використаних функцій та методів

4 основні принципи командного рядка Unix:

1. Користувач взаємодіє з системою через командний рядок.
2. Послідовність операцій може бути автоматизована через внесення операцій в файл.
3. Командний рядок Unix – повноцінна мова програмування зі змінними, умовами і можливістю запускати програми.
4. Командний рядок Unix дозволяє писати програми, які можуть бути не тільки додатками до інших програм, а й самостійними програмами, що мають усі права, що й інша програма в операційній системі Unix.

Так як існує більше одного командного рядка Unix, важливо розуміти різницю між ними. Введення команд в один командний рядок, в той час як використовується інша, може викликати неочікувану поведінку системи.

В даній роботі використовується Bourne Shell, який є одним з найпопулярніших командний рядків для Unix-подібних операційних систем.

# 2.3. Текст програми з коментарями

**Завдання 1.**

Файл 2variant1task.sh:

#!/bin/bash

FULLPATH=$1

cd $FULLPATH

while true

do

#виводить на екран ім’я поточного каталогу (без повного шляху)

echo `basename "$FULLPATH"`

#виводить на екран імена підкаталогів

for element in ${FULLPATH}/\*; do

if [ -d ${element} ]; then

echo " `basename ${element}`"

fi

done

if [ "${FULLPATH}" = "/" ]; then

break

else

cd ..

FULLPATH=`pwd`

fi

done

**Завдання 2.**

Файл 2variant2task.c:

#include <stdio.h>

#include <dirent.h>

#include <string.h>

#include <unistd.h>

#include <libgen.h>

//Task 2. Variant 2.

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

if (argc < 2) {

printf("The program expects an argument that you can use to specify an extension that only files that have it will be shown.\n");

return 0;

}

//a variable for the name of the current folder

char currFolderName[1024];

while (strcmp(currFolderName, "/") != 0) {

//reads the path to the current folder to a variable

getcwd(currFolderName, sizeof(currFolderName));

//name without the full path

char \*simpleName = basename(currFolderName);

printf("%s\n", simpleName);

printFilesWithExtension(argv[1]);

//goes to the parent directory

chdir("..");

}

return 0;

}

int printFilesWithExtension(char \*extension) {

int extensionLenth = strlen(extension);

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir("./")) != NULL) {

while ((ent = readdir(dir)) != NULL) {

//type 4 means a directory. 8 means a plain file.

if (ent->d\_type == 8) {

int fileNameLength = strlen(ent->d\_name);

//if the extension is longer than the file name, then skip this file.

if (extensionLenth < fileNameLength) {

//the last n characters. n is the length of the extension

const char \*extension\_place = &ent->d\_name[fileNameLength - extensionLenth];

if (strcmp(extension, extension\_place) == 0) {

printf(" %s\n", ent->d\_name);

}

}

}

}

closedir(dir);

} else {

perror("could not open directory");

return 1;

}

return 0;

}

**Завдання 3.**

Файл 2variant3task.c:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define CODELIMIT 256

//This program encrypt a specified file with a specified shift, using a Caesar cipher.

//The parameters are set in command line.

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

if (argc < 3) {

printf("The first argument is a path to a file to encrypt. The second argument is a key for the Caesar cipher. The second argument must be a number.\n");

return 1;

}

int shift;

{

//This variable will contain the next character in argv[2] after the numerical value

char \*firstNoNum;

shift = strtol(argv[2], &firstNoNum, 10);

int lengthNoNum = strlen(firstNoNum);

if (lengthNoNum != 0) {

printf("The second argument must be a number which will be used as a shift for the Caesar cipher.\n");

return 1;

}

}

//read the file

FILE \*file = fopen(argv[1], "r");

if (file == NULL) {

printf("Unable to open the specified file.\n");

return 1;

}

int c;

while ((c = fgetc(file)) != EOF) {

int newCode;

//if the character's code is bigger than the limit, then do nothing with this character - leave it as it is.

if (c > CODELIMIT) {

newCode = c;

} else {

//shifting the character

int shiftedRaw = c + shift;

int RANGE = CODELIMIT + 1;

if (CODELIMIT < shiftedRaw) {

newCode = shiftedRaw - RANGE;

} else if (0 > shiftedRaw) {

newCode = shiftedRaw + RANGE;

} else {

//if the new code is not out of the range, then there is no needs to transform it.

newCode = shiftedRaw;

}

}

char newChar = (char) newCode;

printf("%c", newChar);

}

//close the file after using

fclose(file);

}

# 2.4. Результати роботи.

Програми було протестовано наступними сценаріями:

**Завдання 1.**

1. Створено у домашньому каталозі каталог folder1, в ньому файл file1.txt і каталог folder2, командами:

$ mkdir ~/folder1

$ touch ~/folder1/file1.txt

$ mkdir ~/folder1/folder2

2. Запуск програми 2variant1task.sh з параметром ~/folder1 вивів на екран наступний результат:

$ ./2variant1task.sh ~/folder1

folder1

folder2

studentnau1

Desktop

development

folder1

images

home

studentnau1

/

bin

boot

dev

disks

etc

home

lib

lib32

lib64

lost+found

media

mnt

opt

proc

root

run

sbin

sys

tmp

usr

var

Як ми бачимо, програма вивела на екран ім’я каталогу folder2, але не вивела ім’я файлу file1.txt, що означає, що програма виводить лише імена каталогів.

**Завдання 2.**

1. Компіляція файлу, написаного на мові програмування C:

$ gcc 2variant2task.c -o 2variant2task

2. Створено у домашньому каталозі каталог folder3:

$ mkdir ~/folder3

3. Створено у каталозі folder3 порожні файли file3.conf, file4.txt і каталог folder4, командами:

$ touch ~/folder3/file3.conf ~/folder3/file4.txt

$ mkdir ~/folder3/folder4

4. Створено порожні файли file5.txt і file6.conf, в каталозі folder4, командами:

$ touch ~/folder3/folder4/file5.txt ~/folder3/folder4/file6.conf

5. Запуск програми 2variant1task з каталогу folder4 і з параметром conf, і перевірка результатів:

$ cd ~/folder3/folder4

$ ~/nau/rgr/2variant2task.sh conf

folder4

file6.conf

folder3

file3.conf

studentnau1

home

/

Як ми бачимо, програма вивела на екран імена лише тих файлів, що мають розширення .conf.

**Завдання 3.**

1. Компіляція файлу, написаного на мові програмування C:

$ gcc 2variant3task.c -o 2variant3task

1. Створено у домашньому каталозі каталог folder5

2. У каталозі folder5 створено файл 1.txt з наступним вмістом:

Bread of deceit is sweet to a man; but afterwards his mouth shall be filled with gravel.

3. Запущено програму 2variant2task.sh зі шляхом до файлу 1.txt і ключем 5, який використовується для шифрування. Вивід перенаправлено в файл 2.txt. Для цього була виконана наступна команда:

$ ~/nau/rgr/2variant3task ~/folder5/1.txt 5 > ~/folder5/2.txt

4. Перевірено, що файл 2.txt дійсно має зашифрований вміст:

Gwjfi%tk%ijhjny%nx%x|jjy%yt%f%rfs@%gzy%fkyjw|fwix%mnx%rtzym%xmfqq%gj%knqqji%|nym%lwf{jq3

5. Запущено програму 2variant2task.sh зі шляхом до файлу 2.txt і ключем -5. Вивід перенаправлено в файл 3.txt. Для цього була виконана наступна команда:

$ ~/nau/rgr/2variant3task ~/folder5/2.txt -5 > ~/folder5/3.txt

Ключ -5 використовується для того, щоб розшифрувати файл, який був зашифрований ключем 5. Тобто, щоб цією програмою розшифрувати файл, який був зашифрований з якимось ключем, розшифрувати його можна з таким самим ключем, але з протилежним знаком.

5. Перевірено, що файл 3.txt містить розшифровані дані:

Bread of deceit is sweet to a man; but afterwards his mouth shall be filled with gravel.

# 2.5. Висновки.

Borne shell – це не лише інструмент, для уведення команд користувачем Linux, а й повноцінна мова програмування. Вона дозволяє не лише виконувати операції, специфічні для операційної системи, а й виконувати загальні задачі, а й, наприклад, зашифрувати дані, використовуючи шифр Цезаря.

Звісно, для програм з великою кількістю коду мова Bourne shell підходить не найкращим чином, в першу чергу, через те, що вона не підтримує ООП і функціональне програмування.

Мова Bourne Shell є достатньо легкою в вивченні, в ній не багато специфічних конструкцій і операторів. Синтаксис мови достатньо простий і дуже схожий на команди, які потрібно вводити в командному рядку. Короткі скрипти доволі легко підлагоджувати, так як вони не потребують попередньої компіляції і запускаються з командного рядка.

Скрипт – це простий список команд системи, записаний в файл. Якщо файл скрипта починається із послідовності #!, то це вказує системі який інтерпретатор потрібно використовувати для виконання сценарію. Для Bourne shell скрипта перший рядок має бути “#!/bin/bash”.

Мова програмування C надає багато стандартних функцій для роботи з операціями вводу-виводу. Ці функції складають основу заготовочного файлу стандартної бібліотеки мови C <stdio.h>. Функціоальність вводу-виводу є достатньо низькорівневим для наших часів.

Файл відкривається за допомогою функції fopen, котра повертає інформацію потоку вводу-виводу, прикріплену до вказаного файлу чи іншому пристрою, з якого відбувається читання (чи в котру іде запис). У випадку невдачі функція повертає нульовий вказівник.

Після завершення роботи з файлом потрібно викликати функцію fclose. Функція fclose приймає на вхід один аргумент: вказівник на структуру FILE потоку для закриття.