Министерство образования и науки Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Кафедра №304

Отчет по лабораторной работе №1

«**Работа с файлами и папками в Linux. Создание архиватора**»

по дисциплине «Операционные системы»

Выполнил

Студент группы 3О-310Б

Сомов Д.Н.

Приняли

ассистент каф.304

Балян А.В.

ассистент каф.304

Аляутдин Р.Т.

Москва, 2017

**Задание:**

Ознакомиться с функциями (также называемыми «системные вызовы», стандарт POSIX), использующиеся для создания/открытия файлов и их чтения/записи, и с функциями доступа, создания и перемещения по директориям в языке Си. Используя полученные знания, написать на языке Си программу-«архиватор», которая могла бы паковать всё содержимое какой-либо директории, включая все поддиректории, в один большой файл, а затем распаковывать всё, что содержится в этом файле, в первоначальном виде. Использовать компилятор gcc с проверкой написанной программы на корректность работы с памятью с помощью Memory Sanitizer, контроль coding-style’а с помощью checkpatch.pl и систему контроля версий git.

**Как решалась задача:**

Я реализовал идею записи информации о файле и самого файла по очереди. Так как главная задача архиватора – уметь после паковки всех файлов в один архив распаковать эту кучу байт обратно в неповрежденные файлы, то важно было записывать информацию согласно некоей жесткой структуре, чтобы позже эту информацию можно было восстановить без потерь. Отдельной загвоздкой были пустые поддиректории, т.к. они, по сути, не содержали никаких файлов, однако по-прежнему существовали как некие объекты. Поэтому способ «записать все файлы с их путями, а затем восстанавливать директории, в которых находились эти файлы, переходя по путям этих файлов» не подошел (да и, как оказалось, был бы излишне громоздким в плане восстановления иерархии папок).

Вместо этого было сделано следующее: записывается информация и о файлах, и о директориях, попадающих в архив. Сначала записывается размер пути в байтах (занимающий – как число - фиксированную длину в файле, например, 4 байт, чтобы его всегда можно было считать без потерь); затем пишется сам путь – его легко считать, зная его длину, записанную перед ним. Затем пишется, также фиксированной длины, длина (размер) файла, если мы пишем информацию о файле, или «-1», если пишем информацию о директории. «-1» позволяет «распознать» информацию о директориях, ведь если бы это был размер файла, то он всегда больше или равен 0. Далее – для файла – пишется он сам целиком.

Таким образом:

- Для паковки нужно приведенным выше способом записать все файлы в директории и информацию о самой директории, а для каждой из поддиректорий повторить тот же алгоритм сначала, т.е. рекурсивно пройтись по всей иерархии;

- Для распаковки нужно всего лишь циклически читать архив и обрабатывать получаемую из него информацию (создавать директории, создавать и заполнять файлы их содержимым), пока он не закончится.

P. S. checkpatch для приведенного ниже кода будет выдавать порядка 10 ошибок и 10 предупреждений, которые я сознательно оставил; предупреждения – из-за строк длиннее 80 символов (честно проверил в текстовом редакторе, что они короче 80 символов, но checkpatch продолжает гнуть своё), ошибки – из-за, как он считает, пустых строк, в которых на самом деле у меня написаны комментарии (либо в очень редких случаях строки были оставлены пустыми для разделения кода на блоки, которые выполняют существенно разные функции).

**Исходный код:**

**/\*\*\*\*\*\***

**\* Programming Language – C**

**\* Author – Dmitriy Somov, 3O-310B**

**\*\*\*\*\*\*/**

#include <unistd.h>

#include <dirent.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <sys/stat.h>

#include <stdlib.h>

// Archive file stream.

int larc;

// General-purpose buffer. Useful for changing data through functions.

char \*buf;

// Contains path to packing/unpacking directory

// so it can be cut off or added where necessary.

char \*packingDir;

// Takes full path of the file and truncates name of the file.

void GetFilePath(char \*fullPath)

{

int i = 0;

int n = 0;

while (fullPath[i] != '\0') {

if (fullPath[i] == '/')

n = i;

i++;

}

buf = (char \*)calloc(n + 1, 1);

i = 0;

for (int j = 0; j < n; i++, j++)

buf[i] = fullPath[j];

}

// Corrects path of the directory or file, cutting off the part

// that is in packingDir buffer (to make relative paths out of absolute

// for future unpacking).

void CorrectPackingPath(void)

{

int dif = strlen(buf) - strlen(packingDir);

char \*tbuf = (char \*)calloc(dif + 1, 1);

int offset = strlen(packingDir);

for (int i = 0; i <= dif; i++)

tbuf[i] = buf[i + offset];

free(buf);

buf = (char \*)calloc(1024, 1);

for (int i = 0; i < dif; i++)

buf[i] = tbuf[i];

free(tbuf);

}

// Recursive function which packs information about directories and files

// into one file using strict data arrangement.

void PackIntoArchive(char \*dirToPack)

{

size\_t l;

long int i;

int f;

size\_t len;

struct dirent \*fileinfo;

struct stat filestat;

// Open directory.

printf("packing dir %s\n", dirToPack);

chdir(dirToPack);

DIR \*curDir = opendir(dirToPack);

// Write dir info to restore directories on unpacking.

if (buf != NULL)

free(buf);

buf = (char \*)calloc(1024, 1);

getcwd(buf, 1024);

CorrectPackingPath();

l = strlen(buf);

write(larc, &l, sizeof(l));

write(larc, buf, l);

i = -1;

write(larc, &i, sizeof(i));

if (buf != NULL)

free(buf);

buf = NULL;

// Looking through dir entries.

// Information about cur entry will be in 'fileinfo'.

while ((fileinfo = readdir(curDir)) != NULL) {

lstat(fileinfo->d\_name, &filestat);

// If it is not a file, but a subdirectory...

if (S\_ISDIR(filestat.st\_mode)) {

// Skip directories-'references'.

if (strcmp(".", fileinfo->d\_name) == 0 ||

strcmp("..", fileinfo->d\_name) == 0)

continue;

// Recursively go to subdirectory.

buf = (char \*)calloc(1024, 1);

getcwd(buf, 1024);

strcat(buf, "/");

strcat(buf, fileinfo->d\_name);

PackIntoArchive(buf);

if (buf != NULL)

free(buf);

buf = NULL;

} else {

f = open(fileinfo->d\_name, O\_RDONLY);

// For file - add to archive:

// - path length

buf = (char \*)calloc(1024, 1);

getcwd(buf, 1024);

CorrectPackingPath();

strcat(buf, "/");

strcat(buf, fileinfo->d\_name);

printf("packing file %s\n", buf);

len = strlen(buf);

write(larc, &len, sizeof(len));

// - relative path of the file with its name

write(larc, buf, len);

// - file length

len = filestat.st\_size;

write(larc, &len, sizeof(len));

// - file itself

free(buf);

buf = (char \*)calloc(len, 1);

read(f, buf, (size\_t)(len));

write(larc, buf, (size\_t)(len));

free(buf);

buf = NULL;

close(f);

}

}

closedir(curDir);

chdir("..");

}

// Simple function which decompiles archive into directories and files

// using the knowledge about strict archive structure.

void UnpackArchive(void)

{

size\_t len;

long int l;

char \*tbuf;

char \*filebuf;

int nf;

read(larc, &len, sizeof(len));

buf = (char \*)calloc(len+1, 1);

while (read(larc, buf, len) == len) {

read(larc, &l, sizeof(l));

// Make a path (suits both for folders and files).

tbuf = (char \*)calloc(strlen(packingDir) + strlen(buf) + 1, 1);

strcat(tbuf, packingDir);

strcat(tbuf, buf);

printf("creating entry %s\n", tbuf);

if (l == -1) {

// It's a dir. Create it if it doesn't exist.

if (chdir(tbuf) == -1)

mkdir(tbuf, O\_RDWR);

} else {

// It's a file. Create it and fill in data.

filebuf = (char \*)calloc(l, 1);

read(larc, filebuf, l);

nf = open(tbuf, O\_WRONLY | O\_CREAT, S\_IRUSR | S\_IWUSR);

write(nf, filebuf, l);

close(nf);

free(filebuf);

}

free(tbuf);

free(buf);

read(larc, &len, sizeof(len));

buf = (char \*)calloc(len+1, 1);

}

}

// Main function, handles input arguments, errors and starts

// packing/unpacking.

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

{

if (argc == 1) {

printf("To use larc, write two arguments:\n");

printf("1) 'pack' or 'unpack' action keyword;\n");

printf(" after completing action, target directory");

printf(" and archive will be in the same folder.\n");

printf("2) path to directory which you want to pack\n\t\tOR\n");

printf(" path to archive you want to unpack.\n");

exit(1);

} else if (argc < 3 || argc > 3) {

printf("Error! Wrong number of arguments.\n");

printf("larc stopped with error.\n");

exit(2);

} else {

if (strcmp(argv[1], "pack") == 0) {

packingDir = (char \*)calloc(1024, 1);

if (chdir(argv[2]) == -1) {

printf("ERROR! Can't pack dir with path = '%s'\n", argv[2]);

printf("Maybe you spell argument '2' wrongly.\n");

printf("larc stopped with error.\n");

exit(3);

}

chdir("..");

getcwd(packingDir, 1024);

// Create path for an archive and the archive itself.

chdir(argv[2]);

buf = (char \*)calloc(1024, 1);

getcwd(buf, 1024);

strcat(buf, ".larc");

larc = open(buf, O\_WRONLY | O\_CREAT, S\_IRUSR | S\_IWUSR);

free(buf);

buf = NULL;

// Start packing.

PackIntoArchive(argv[2]);

close(larc);

free(packingDir);

} else if (strcmp(argv[1], "unpack") == 0) {

GetFilePath(argv[2]);

packingDir = (char \*)calloc(strlen(buf)+1, 1);

strcpy(packingDir, buf);

free(buf);

larc = open(argv[2], O\_RDONLY);

if (larc == -1) {

printf("ERROR! Archive with path = '%s' ", argv[2]);

printf("doesn't exist.\n");

printf("larc stopped with error.\n");

exit(5);

}

UnpackArchive();

close(larc);

free(packingDir);

} else {

printf("ERROR! Cannot understand argument '1' = '%s'\n", argv[1]);

printf("larc stopped with error.\n");

exit(4);

}

}

exit(0);

}