**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ» (МАИ)**

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

**Лабораторная работа №1**

по дисциплине

**«Операционные системы»**

На тему:

**«Архиватор»**

Выполнили студенты группы М30-325Бк-18

Назарчук Г. В.,

Черноморец М. С.

Проверил:

доцент кафедры 304, к.т.н. Охотников Д. А.

Москва 2021

**Задание**

1. Написать программу архиватор, на вход которой передаётся папка с файлами, результатом работы программы является файл, объединяющий все файлы из входной директории, а также содержащий заголовок с информацией об именах файлов и их размерах.

2. Написать программу разархиватор, разделяющий разархивированный файл обратно на файлы.

**Описание программы**

Архиватор и разархиватор объединены в одну программу. Управление производится через аргументы при вызове программы:

* флаг -p означает упаковку в архив, после него перечисляется произвольное количество файлов и директорий для упаковки;
* флаг -u означает разархивирование, после него один аргумент – имя архива, который нужно разархивировать.

Формат файла архива:

Первым идёт число файлов и директорий в архиве. Затем подряд записаны структуры дескрипторов для каждого файла или папки. Формат структуры дескриптора:

* флаг, указывающий, что описывает структура: файл или директорию;
* id этого дескриптора (нумерация начинается с 0);
* размер файла, который описывает этот дескриптор (если директория, то 0);
* имя файла или директории;
* id дескриптора родительской директории, если есть; -1, если нет.

После дескрипторов в том же порядке подряд записаны байты файлов.

Программа поддерживает вложенные директории. Архивация и разархивация для вложенных директорий происходит рекурсивно.

Код программы доступен в репозитории по адресу: https://github.com/CherMiTT/lab1\_archive\_tool

**Код программы**

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <dirent.h>

#include <stdbool.h>

void pack(char \*file\_names\_arr[], int length, int start\_index);

void unpack(char \*file\_name);

void pack\_fdescr(int arch, char file\_name[80], size\_t file\_size, int \*file\_count, int parent\_id);

void pack\_ddestr(int arch, char file\_name[80], int \*file\_count, int parent\_id);

void pack\_file(int arch, char file\_name[80]);

void unpack\_file(int arch, char file\_name[80],size\_t file\_size, int parent\_id, int \*cur\_id, int nested\_directory\_array[20], int \*index);

void pack\_dir (int arch, char file\_name[80]);

void unpack\_dir(int arch, char file\_name[80], int id, int \*cur\_id, int parent\_id, int nested\_directory\_array[20], int \*index);

//Программа архивирует файлы без сжатия

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

{

//Парсим аргументы

int opt;

while ((opt = getopt(argc, argv, "pu:")) != -1)

{

switch(opt)

{

case 'p': //pack, после него идёт список файлов для архивации

{

//printf("packing, optind = %d\n", optind);

pack(argv, argc, optind);

break;

}

case 'u': //unpack, после него один файл - архив для распаковки

{

//printf("unpacking, optarg = %s\n", optarg);

unpack(optarg);

break;

}

case '?':

{

printf("Invalig input format\n");

}

}

}

return 0;

}

struct file\_descr

{

bool is\_file; //True - файл, false - директория

int id; //Нумерация начинаетя с 0

size\_t file\_size; //размер файла/директории (у директории размер 0)

char file\_name[80]; //имя файла

int parent\_id; //-1 - нет родителя

};

void unpack(char \*file\_name)

{

int arch = open(file\_name, O\_RDONLY);

if (arch == -1)

{

printf("Some error has occured! Can't open archive.\n");

return;

}

int file\_count;

read(arch, &file\_count, sizeof(int));

struct file\_descr descr[file\_count];

int cur\_id = 0;

int nested\_directory\_array[20]; //TODO: redo 20

int index = 0;

for (int i = 0; i< file\_count; i++)

{

if(read(arch, &descr[i],sizeof(struct file\_descr))!=sizeof(struct file\_descr))

{

printf("Incorrect file reading");

}

//printf("File name: %s, file size = %ld, file id = %d, parent's id = %d\n", descr[i].file\_name, descr[i].file\_size, descr[i].id, descr[i].parent\_id);

}

for (int i = 0; i< file\_count; i++)

{

if (descr[i].is\_file)

{

unpack\_file(arch, descr[i].file\_name, descr[i].file\_size, descr[i].parent\_id, &cur\_id, nested\_directory\_array, &index); //TODO: current id

}

else

{

unpack\_dir(arch, descr[i].file\_name, descr[i].id, &cur\_id, descr[i].parent\_id, nested\_directory\_array, &index);

}

}

close(arch);

printf("Unpacked\n");

}

void unpack\_file(int arch, char file\_name[80],size\_t file\_size, int parent\_id, int \*cur\_id, int nested\_directory\_array[20], int \*index)

{

//printf("Unpacking file %s\n. curr\_id = %d, parent\_id = %d\n", file\_name, \*cur\_id, parent\_id);

while(\*cur\_id != parent\_id)

{

chdir("..");

//printf("curr\_id = %d, parent\_id = %d, parent = %d\n", \*cur\_id, parent\_id, nested\_directory\_array[(\*index)]);

\*cur\_id = nested\_directory\_array[(\*index)--];

}

char block[file\_size]; //TODO: разбить на блоки по 1024

if(read(arch, &block,file\_size)!=file\_size)

{

printf("Incorrect file reading");

}

int file = open(file\_name, O\_CREAT | O\_WRONLY | O\_TRUNC, S\_IRUSR|S\_IWUSR);

write(file, &block, file\_size);

close(file);

}

void unpack\_dir(int arch, char file\_name[80], int id, int \*cur\_id, int parent\_id, int nested\_directory\_array[20], int \*index)

{

//printf("Unpacking directory %s\n. curr\_id = %d, parent\_id = %d\n", file\_name, \*cur\_id, parent\_id);

while(\*cur\_id !=parent\_id)

{

chdir("..");

\*cur\_id = nested\_directory\_array[(\*index)--];

}

mkdir(file\_name,O\_CREAT| O\_TRUNC | S\_IRUSR| S\_IWUSR);

chdir(file\_name);

nested\_directory\_array[++(\*index)] = parent\_id;

\*cur\_id = id;

}

void pack(char \*file\_names\_arr[], int length, int start\_index)

{

//printf("In the packing function now.\n");

int file\_count = 0;

//Creating archive file

int arch = open("test\_archive", O\_CREAT | O\_WRONLY | O\_TRUNC, S\_IRUSR|S\_IWUSR); //TODO: change name

if(arch == -1)

{

printf("Couldn't create or open archive!");

return;

}

write(arch, &file\_count, sizeof(int));

//Creating descriptor and filling archive

struct stat file\_stat;

for(int i = start\_index; i < length; i++)

{

int e = stat(file\_names\_arr[i], &file\_stat); //TODO: stat is slow! Change?

if(e == -1)

{

printf("Some error has occured! %s not found!\n", file\_names\_arr[i]);

}

if(S\_ISREG(file\_stat.st\_mode)) //если это файл

{

pack\_fdescr(arch, file\_names\_arr[i], file\_stat.st\_size, &file\_count, 0);

}

else if(S\_ISDIR(file\_stat.st\_mode)) //если это директория

{

pack\_ddestr(arch, file\_names\_arr[i], &file\_count, 0);

}

}

//Writing data into archive

for(int i = start\_index; i < length; i++)

{

int e = stat(file\_names\_arr[i], &file\_stat);

if(e == -1)

{

printf("Some error has occured! %s not found!\n", file\_names\_arr[i]);

}

if(S\_ISREG(file\_stat.st\_mode)) //если это файл

{

pack\_file(arch, file\_names\_arr[i]);

}

else if(S\_ISDIR(file\_stat.st\_mode)) //если это директория

{

pack\_dir(arch, file\_names\_arr[i]);

}

}

//writing true count of files instead of default value in the bgning of the file

lseek(arch,SEEK\_SET, 0);

//printf("%d\n",file\_count);

write(arch, &file\_count, sizeof(int));

close(arch);

printf("Packed\n");

}

void pack\_fdescr(int arch, char file\_name[80], size\_t file\_size, int \*file\_count, int parent\_id) //creates and writes file descriptor in hte archive

{

struct file\_descr descr;

descr.file\_size = file\_size;

strcpy(descr.file\_name, file\_name);

descr.is\_file = true;

descr.id = \*file\_count + 1;

descr.parent\_id = parent\_id;

if(write(arch, &descr, sizeof(struct file\_descr))!=sizeof(struct file\_descr))

{

printf("Couldnt write file descriptor in the archive.\n");

}

(\*file\_count)++;

}

void pack\_ddestr(int arch, char file\_name[80], int \*file\_count, int parent\_id) //creates and writes directory descriptor in the archive

{

struct file\_descr descr;

descr.file\_size = 0;

strcpy(descr.file\_name, file\_name);

descr.is\_file = false;

descr.id = \*file\_count + 1;

descr.parent\_id = parent\_id;

if(write(arch, &descr, sizeof(struct file\_descr))!=sizeof(struct file\_descr))

{

printf("Couldn't write directory descriptor in the archive.\n");

}

(\*file\_count)++;

DIR \*d;

struct dirent \*entry;

struct stat statbuf;

if ((d = opendir(file\_name)) == NULL)

{

printf("Error occured! Can't open directory.");

}

chdir(file\_name);

//printf("In directory %s\n", file\_name);

while ((entry = readdir(d)) != NULL)

{

lstat(entry->d\_name, &statbuf);

if(S\_ISREG(statbuf.st\_mode))

{

pack\_fdescr(arch, entry->d\_name, statbuf.st\_size, file\_count, descr.id);

}

else if (S\_ISDIR(statbuf.st\_mode))

{

/\* Находит каталог, но игнорирует . и .. \*/

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

continue;

pack\_ddestr(arch, entry->d\_name, file\_count, descr.id);

}

}

chdir("..");

closedir(d);

}

void pack\_dir (int arch, char file\_name[80]) //adds content of directory to the archive

{

DIR \*d;

struct dirent \*entry;

struct stat statbuf;

if ((d = opendir(file\_name)) == NULL)

{

printf("Error occured! Can't open directory.");

}

chdir(file\_name);

while ((entry = readdir(d)) != NULL)

{

lstat(entry->d\_name, &statbuf);

if(S\_ISREG(statbuf.st\_mode))

{

pack\_file(arch, entry->d\_name);

}

else if (S\_ISDIR(statbuf.st\_mode))

{

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

continue;

pack\_dir(arch, entry->d\_name);

}

}

chdir("..");

closedir(d);

}

void pack\_file(int arch, char file\_name[80]) //adds file's data to the archive

{

int nread;

char block[1024];

int file = open(file\_name, O\_RDONLY);

if (file == -1)

{

printf("Some error has occured! Can't open file.\n");

return;

}

while((nread = read(file, block, sizeof(block))) > 0)

{

write(arch, block, nread);

}

close(file);

}

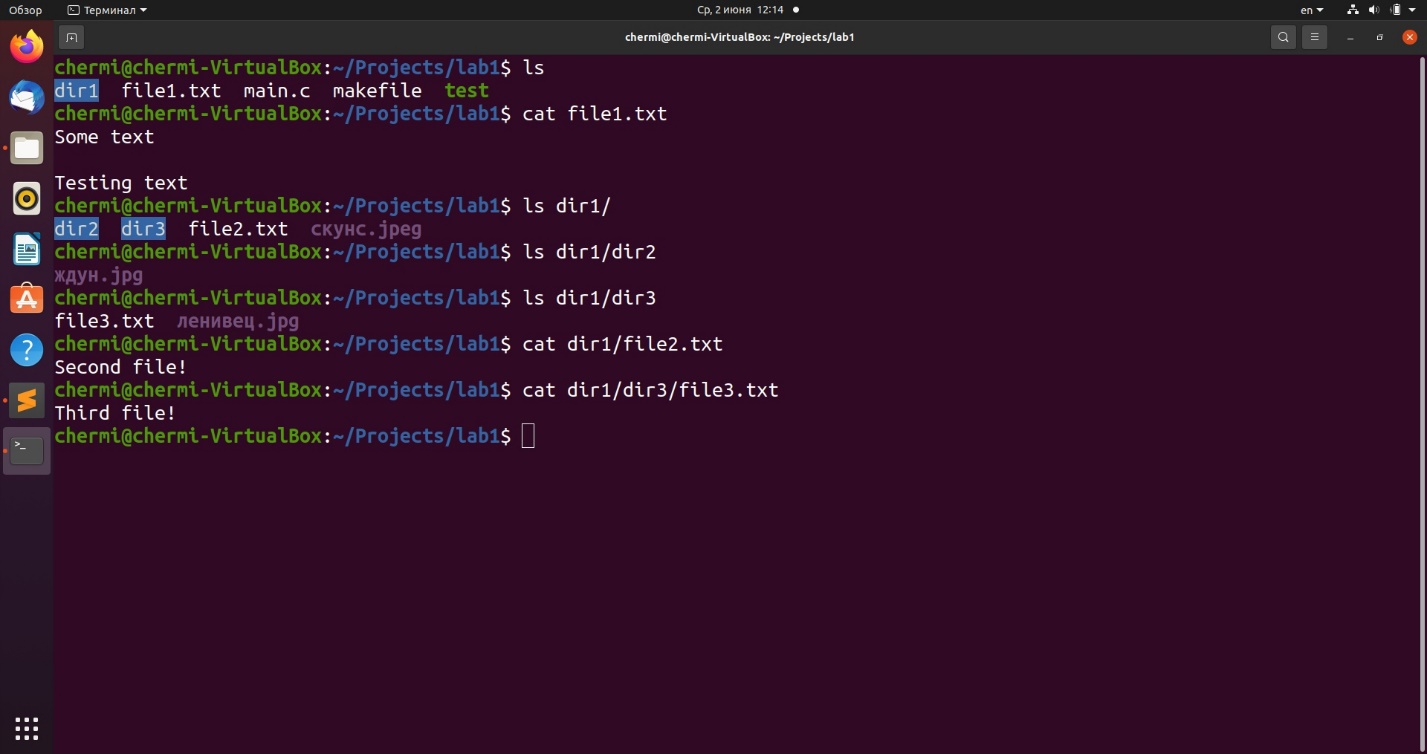
**Использованный makefile**

test:

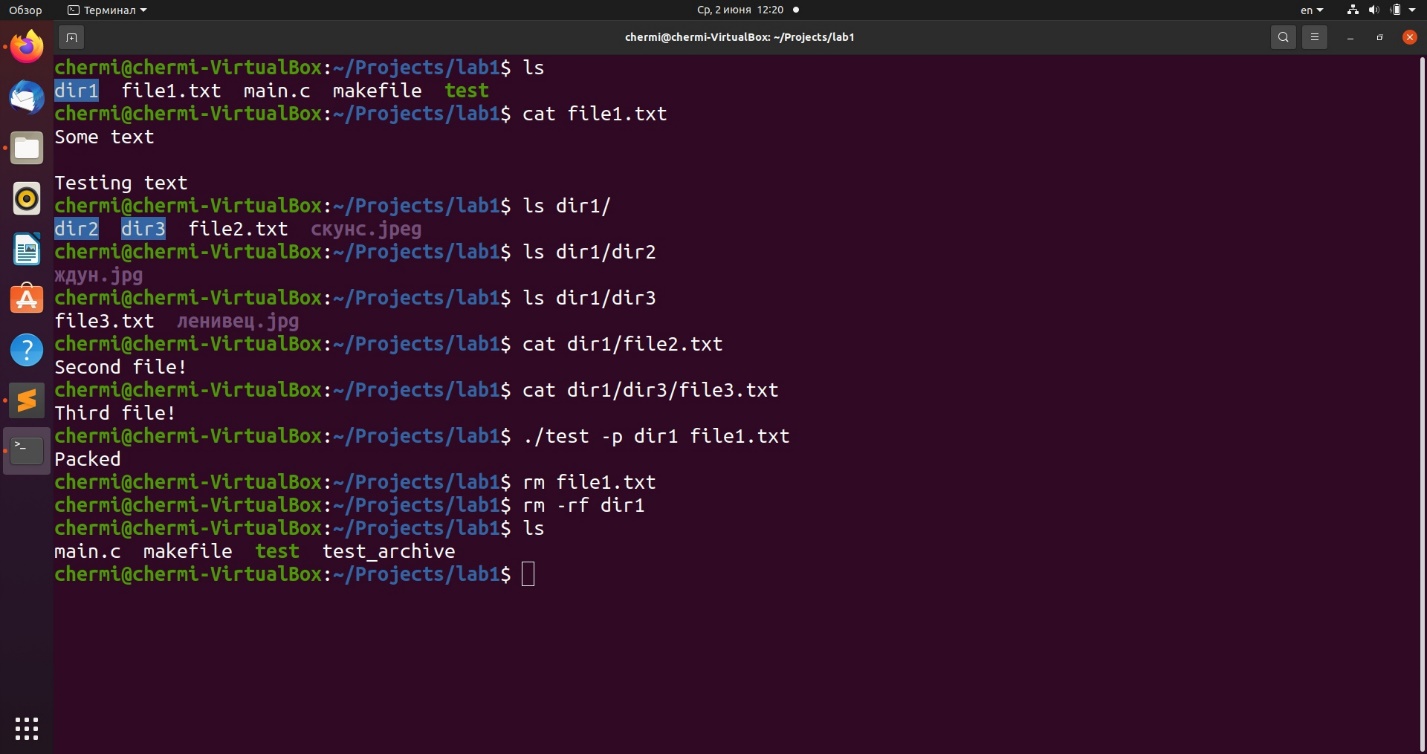
gcc main.c -o test -Wall -fsanitize=address

**Пример работы программы**

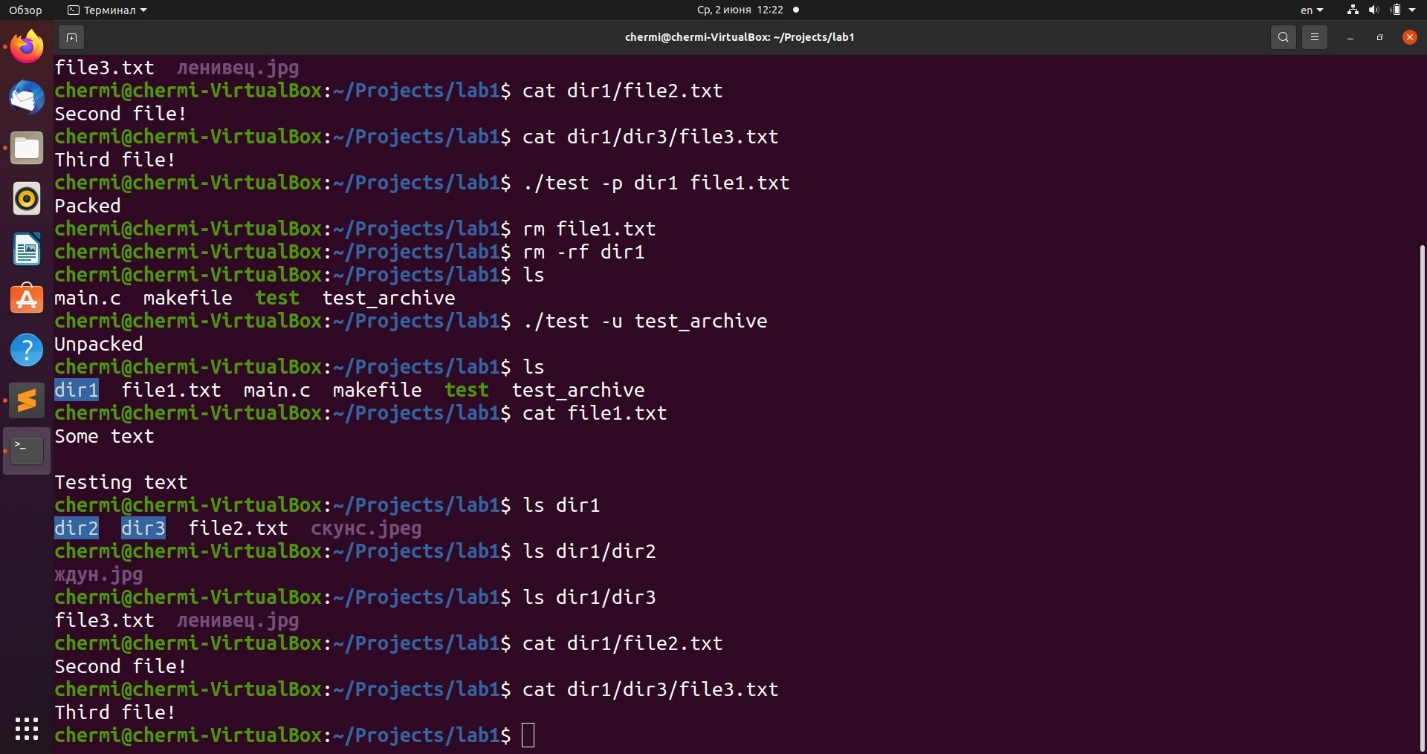
Есть следующая структура файлов и директорий с текстовыми файлами и изображениями:



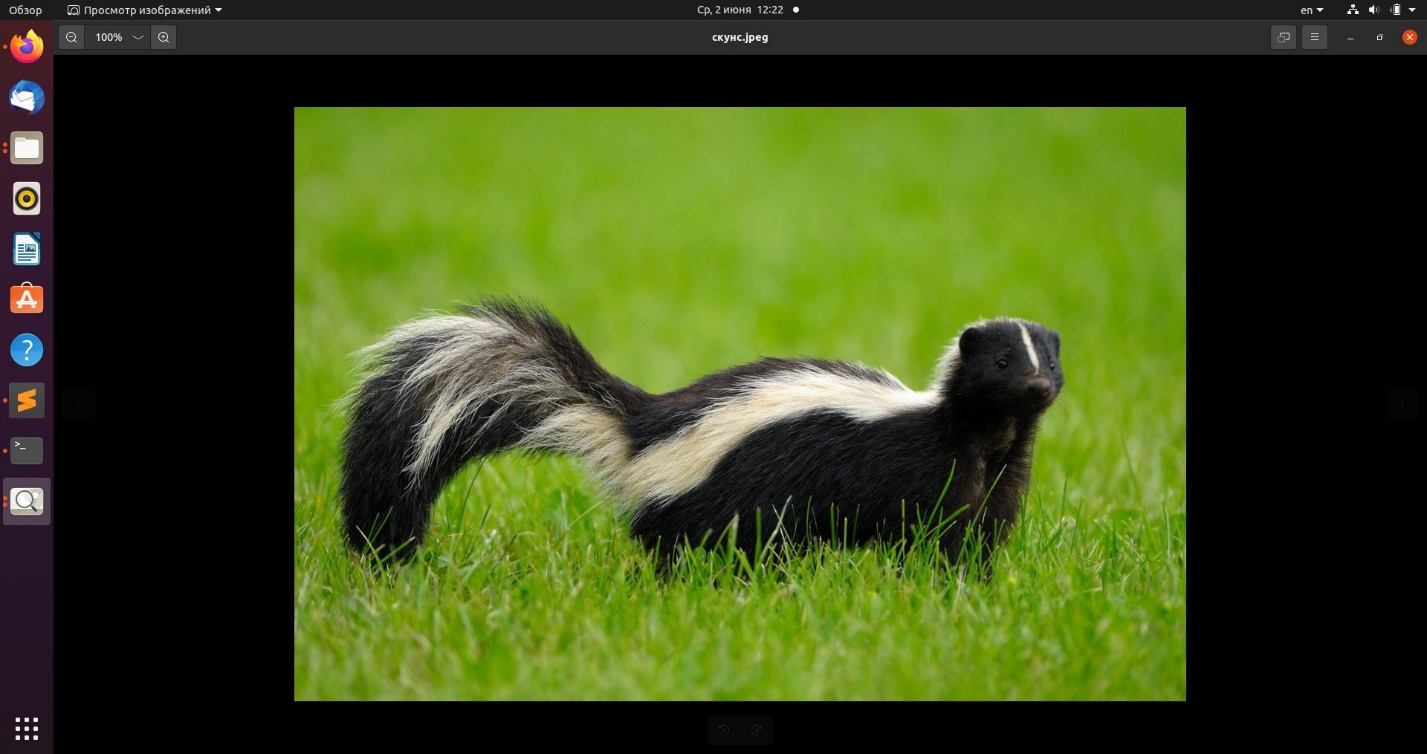
Архивируем и удаляем заархивированные файлы и директории:

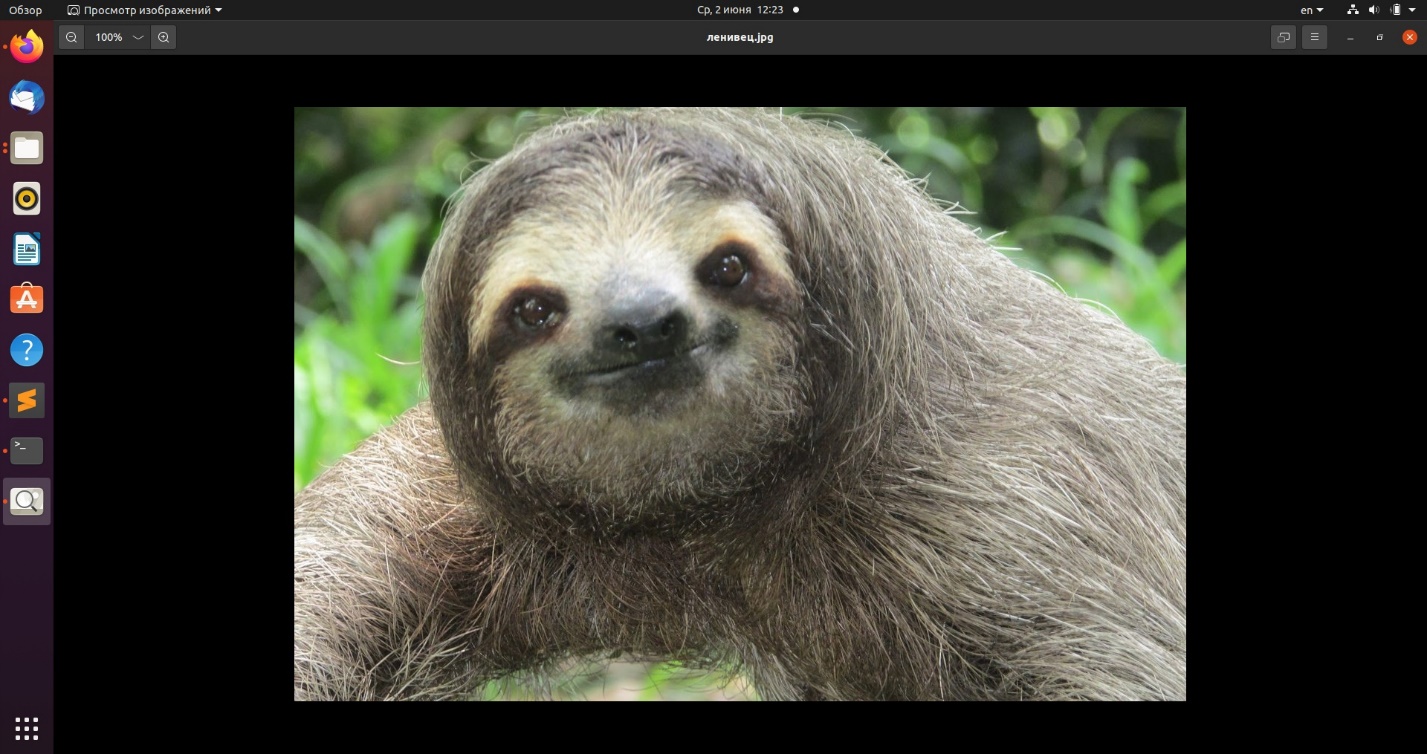


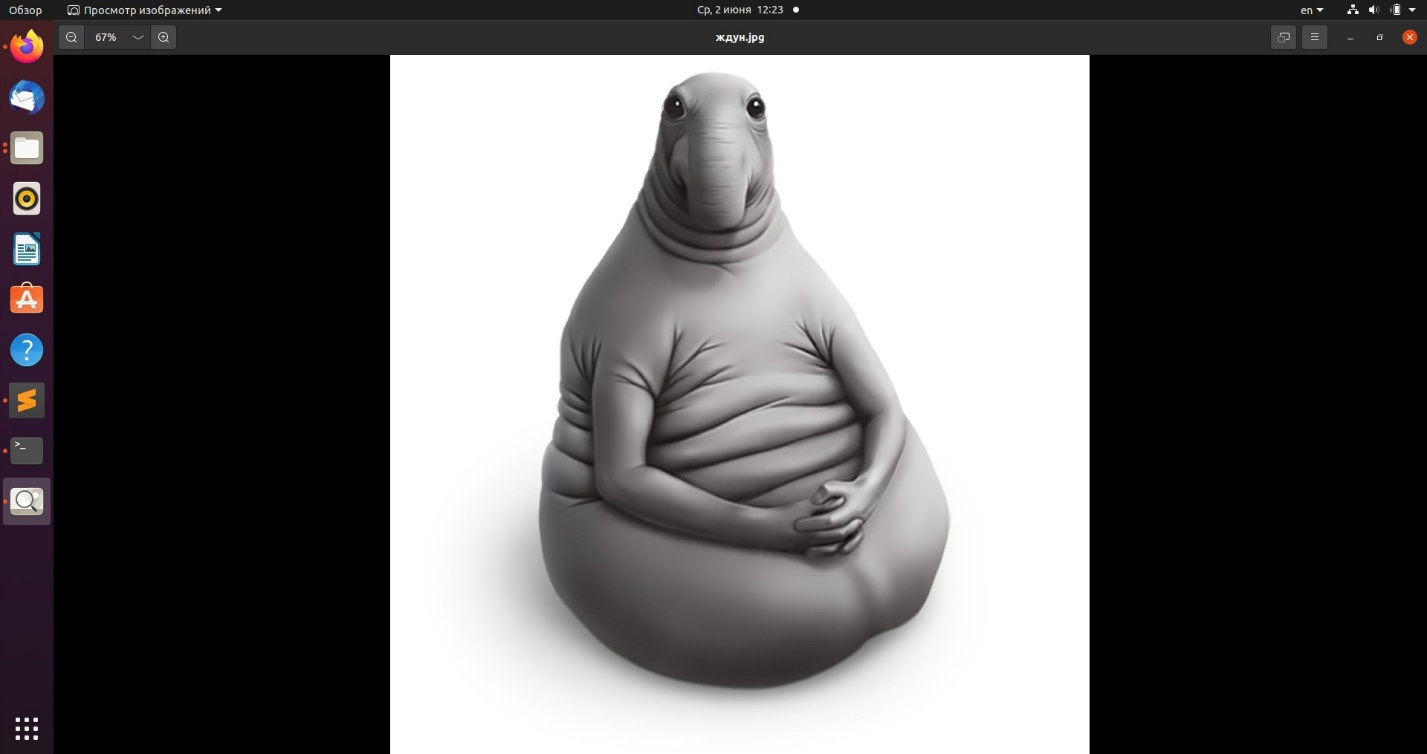
Разархивируем и проверяем:



После разархивации все файлы в директориях успешно открылись. Текстовые файлы видны при проверке выше, результат открытия файлов изображений:

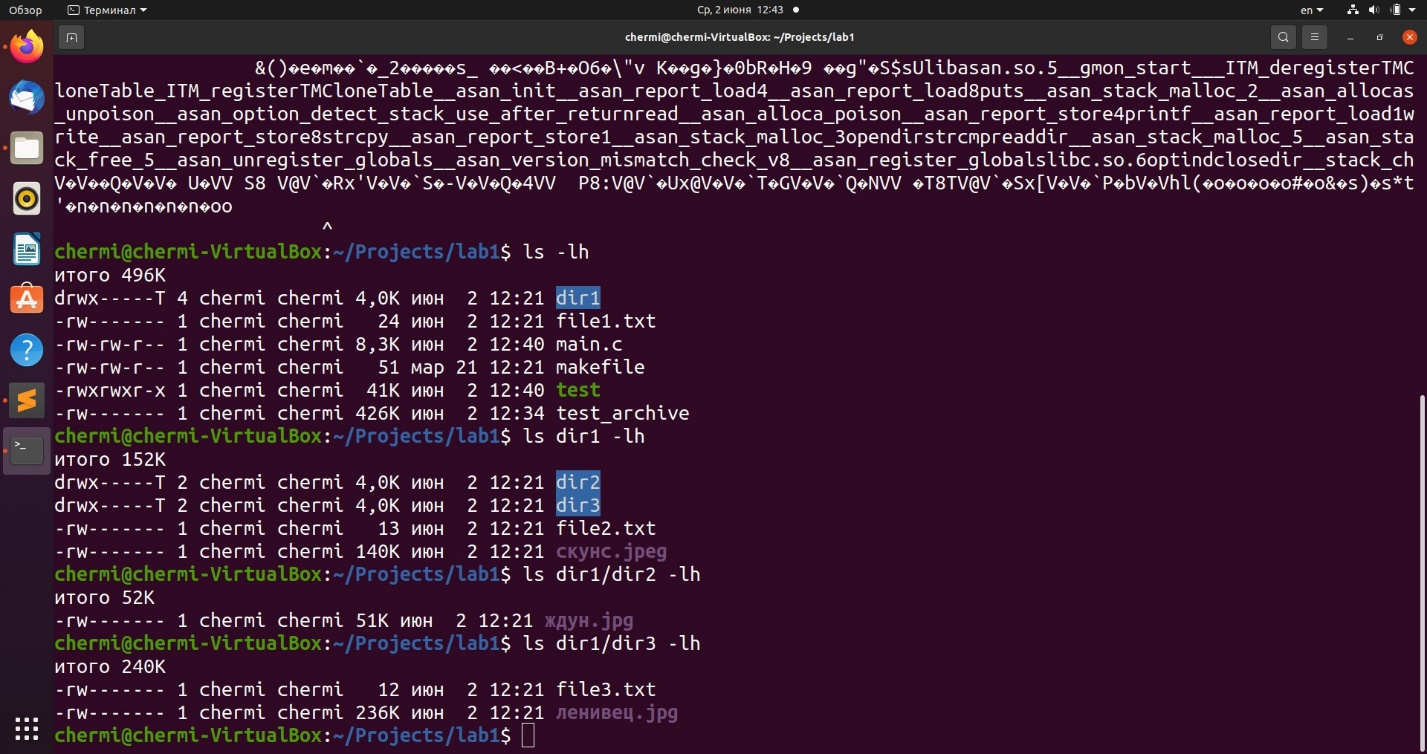






Как видно, все файлы и директории были успешно заархивированы и затем разархивированы.

Проверка размеров архива:



**Список использованной литературы**

1. Основы программирования в Linux: Пер. с англ. / Н. Мэтью, Р. Стоунс. – 4-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2009. – 896 с.: ил.