МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №9**

**з дисципліни «Операційні системи»**

**на тему: «Робота з каталогами і індексними дескрипторами.»**

Виконав:

Студент гр. ПЗ2011

Кулик С.В.

Прийняв:

Андрющенко В.О.

Дніпро, 2023

**Тема:** Робота з каталогами і індексними дескрипторами.

**Мета:** - вивчити структуру даних, що забезпечують зберігання, пошук по імені, захист і доступ до файлів;

- набути навичок складання програм, що вимагають обробку інформації, що міститься в індексних дескрипторах і каталогах;

- вивчити елементарні прийоми обробки помилок на мові З в UNIX.

**Короткі теоретичні відомості**

Інтерфейс між призначеною для користувача програмою і зовнішнім пристроєм (або між двома призначеними для користувача програмами) в ОС UNIX здійснюється в рамках єдиної структури даних, званої файлом ОС UNIX. Всякий файл ОС UNIX у відповідність з його типом може бути віднесений до однієї з наступних чотирьох груп: звичайні файли, каталоги, спеціальні файли, канали.

Звичайним файлом є сукупність блоків диска, вхідних до складу файлової системи ОС UNIX. У вказаних блоках може бути довільна інформація.

Каталогами є файли особливого типа, що відрізняються від звичайних перш за все тим, що здійснити запис в них може лише ядро ОС UNIX, тоді як доступ по читанню може отримати будь-який призначений для користувача процес, що має відповідні повноваження. Кожен елемент каталога складається з двох полів: поля імені файлу і поля, що містить покажчик на описувач файлу, де зберігається вся інформація про файл: дата створення, розмір, код захисту, ім'я власника і так далі У будь-якому каталозі міститься, принаймні, два елементи, що містять в полі імені файлу імена "." і "..". Елемент каталога, що містить в полі імені файлу контекст ".", в полі заслання містить заслання на описувач файлу, що описує цей каталог. Елемент каталога, що містить в полі імені файлу контекст "..", у полі заслання містить заслання на описувач файлу, в якому зберігається інформація про батьківський каталог поточного каталога.

Спеціальні файли - це деякі файли, кожному з яких ставиться у відповідність свій зовнішній пристрій, підтримуваний ОС UNIX і що має спеціальну структуру. Його не можна використовувати для зберігання даних, як звичайний файл або каталог. В той же час над спеціальним файлом можна виробляти ті ж операції, що і над звичайним файлом: відкривати, вводити і виводити інформацію і так далі. Результат вживання будь-яка з цих операцій залежить від того, якому конкретному пристрою відповідає оброблюваний спеціальний файл, проте у будь-якому випадку буде здійснена відповідна операція введення-виводу на зовнішній пристрій, якому відповідає вибраний спеціальний файл. Четвертий вигляд файлів - канали, буде розглянутий окремо в подальших лабораторних роботах.

Для здобуття інформації про типа файлу необхідно скористатися системними викликами stat() (fstat()). Формат системних викликів stat() (fstat():

#include <sys/types.h>

#include <sys/stat.h>

int stat(const char \*name, struct stat \*stbuf)

int fstat(int fd, struct stat \*stbuf)

Обидва системні виклики поміщають інформацію про файл (у першому випадку специфікованим ім'ям name, а в другому - дескриптором файлу fd) в структурну змінну, на яку вказує stbuf. Зухвала функція повинна поклопотатися про резервування місця для повертаної інформації; в разі успіху повертається 0, в противному випадку -1 і код помилки в errno. Опис структури stat міститься у файлі <sys/stat.h>. З невеликими модифікаціями вона має вигляд:struct stat

{

dev\_t st\_dev; /\* device file \*/

ino\_t st\_ino; /\* file serial inode \*/

ushort st\_mode; /\* file mode \*/

short st\_nlink; /\* number of links \*/

ushort st\_uid; /\* user ID \*/

ushort st\_gid; /\* group ID \*/

dev\_t st\_rdev; /\* device ident \*/

off\_t st\_size; /\* size of file \*/

time\_t st\_atime; /\* last access time \*/

time\_t st\_mtime; /\* last modify time \*/

time\_t st\_ctime; /\* last status change \*/

}

Поле st\_mode містить, описуючий файл. Прапори несуть наступну інформацію:

S\_IFMT 0170000 - тип файла

S\_IFDIR 0040000 - каталог

S\_IFCHR 0020000 - байт-ориєнтованийй спеціальний файл

S\_IFBLK 0060000 - блок-орієнтований спеціальний файл

S\_IFREG 0100000 – звичайний файл

S\_IFFIFO 0010000 - дисципліна FIFO

S\_ISUID 04000 - ідентифікатор власника

S\_ISGID 02000 – ідентифікатор групи

S\_ISVTX 01000 - зберегти копіюючий текст

S\_ISREAD 00400 - власникові дозволено читання

S\_IWRITE 00200 - власникові дозволений запис

S\_IEXEC 00100 - власникові дозволено виконання.

Символьні константи, чотири перші символи яких збігаються з контекстом S\_if, можуть бути використані для визначення типа файлу.

Більшість системних викликів, що працюють з каталогами, оперують структурою dirent, визначеною в заголовному файлі <dirent.h>

struct dirent

{

ino\_t d\_ino; /\* номер індексного дескриптора \*/

char d\_name[DIRSIZ]; /\* ім’я файла \*/

}

Відкриття і закриття каталога виконується системними викликами opendir() і closedir():

#include <sys/types.h>

#include <dirent.h>

DIR \*opendir (char \*dirname);

При успішному відкритті каталога системний виклик повертає покажчик на змінну типа DIR, що є дескриптором каталога, визначену у файлі <dirent.h> і використовувану при читанні і записі в каталог. При невдалому виклику повертається значення NULL.

Читання записів каталога виконується системним викликом readdir():

#include <sys/types.h>

#include <dirent.h>

struct dirent \*readdir (DIR \*dirptr);

Системний виклик readdir() по номеру дескриптора каталога повертає черговий запис з каталога в структуру dirent, або нульовий покажчик досягши кінця каталога. При успішному читанні, покажчик каталога переміщається до наступного запису.

**Завдання**

Програма повинна приймати як параметр командного рядка ім'я каталогу і маску файлу, виробляти пошук вниз по дереву всіх відповідних файлів. Для кожного знайденого файлу друкувати наступну інформацію: ім'я файлу, тип файлу (звичайний або каталог).

**Текст програми**

**Файл lab.cpp**

#include <sys/stat.h>

#include <dirent.h>

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

// Recursive function to explore directory

void explore(const char\* dirName, int minSize, int maxSize)

{

DIR\* dir; // Pointer to open the directory

struct dirent\* content; // Content of directory (entry)

struct stat info; // Information about each entry// 1. Open the directory

dir = opendir(dirName);

if (!dir)

{

cout << "Directory was not found!\n";

return;

}

// 2. Read

while ((content = readdir(dir)) != NULL)

{

if (content->d\_name[0] != '.')

{

string path = string(dirName) + "/" + string(content->d\_name);

stat(path.c\_str(), &info); // Get info of file

if (S\_ISREG(info.st\_mode) && info.st\_size >= minSize && info.st\_size <= maxSize)

{ // Check if file is a regular file and its size falls within the specified range

printf("%s\ttype: regular file\tpermissions: %o\tlinks: %lu\n", content->d\_name, (info.st\_mode & 0777), info.st\_nlink);

}

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

{ // Check if it is a directory

printf("%s\ttype: directory\tpermissions: %o\tlinks: %lu\n", content->d\_name, (info.st\_mode & 0777), info.st\_nlink);

explore(path.c\_str(), minSize, maxSize); // Recursively explore the directory

}

}

}

// 3. Close

closedir(dir);

}

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

{

if (argc != 4)

{

cout << "Usage: " << argv[0] << " directory min\_size max\_size" << endl;

return 1;

}

int minSize = atoi(argv[2]);

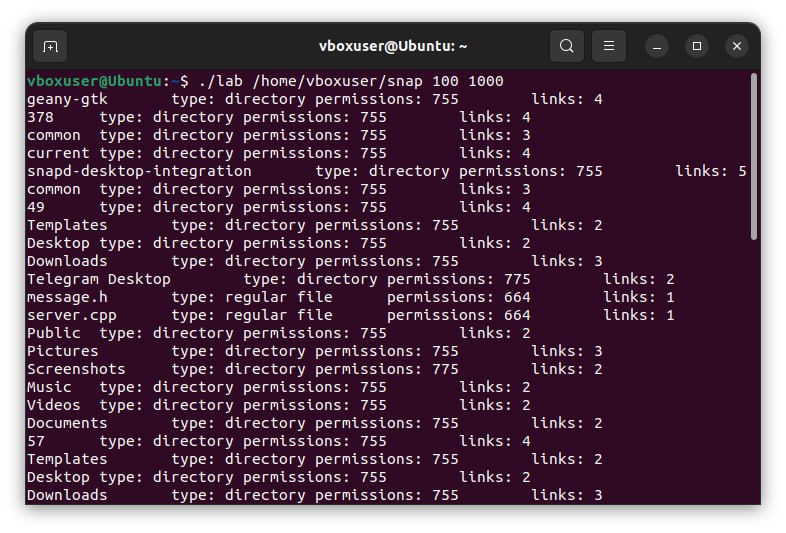
int maxSize = atoi(argv[3]);

explore(argv[1], minSize, maxSize);

return 0;

}

**Результати програми та тестові прилади**



**Висновки**

У результаті виконання лабораторної роботи вивчено структуру даних, що забезпечують зберігання, пошук по імені, захист і доступ до файлів, елементарні прийоми обробки помилок в UNIX. Набуто навичок складання програм, що вимагають обробку інформації, що міститься в індексних дескрипторах і каталогах.