**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №7

**на тему: «**Статичні та динамічні бібліотеки. WINDOWS та LINUX»

**з дисципліни: «**Операційні системи»

**Лектор:**

доц. каф. ПЗ

Грицай О. Д.

**Виконав:**

ст. гр. ПЗ-21

Качур А. М.

**Прийняв:**

доц. каф. ПЗ

Грицай О. Д.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2024

**Тема роботи** Управління файловою системою.

**Мета роботи:** Ознайомитися з файловими системами операційних систем Windows та Linux.

**Теоретичні відомості**

Файлова система – це система правил, згідно яких операційна система забезпечує зберігання даних на диску. Ці правила визначають формат зберігання даних, їх розміщення на диску, адресацію і ідентифікацію.

Кожен комп'ютерний файл зберігається на цифровому носії із певною місткістю. Накопичувач можна уявити як лінійний простір для читання і запису цифрової інформації. Кожен байт даних на ньому має своє зміщення від початку сховища, відомий як адреса, яка використовується для посилання на цей файл. Накопичувачі для посилання на байт інформації можуть використовувати адресацію у вигляді пари - сектор і зміщення.

Загалом, файлова система це структуроване представлення даних та набір метаданих, що описують ці дані. Вона записується до сховища під час операції форматування. Зазвичай ця структура оперує блоками, а не секторами. Блоки ФС – це групи секторів, за допомогою яких оптимізується адресація сховища. Файли, як правило, зберігаються на початку блоку та займають цілі блоки.

Крім даних користувача, файлова система також містить власні параметри (наприклад, розмір блоку), файлові дескриптори (такі як його розмір, розташування, фрагменти і т.д.), імена та ієрархію каталогів. Вона також може зберігати інформацію про безпеку, розширені атрибути та інші параметри.

Щоб відповідати різноманітним вимогам користувачів, таким як продуктивність сховища, стабільність і надійність, розробляється безліч типів (або форматів) файлових систем, здатних більш ефективно виконувати певні функції.

**Файлові системи Windows**

Microsoft Windows в основному використовує дві файлові системи: NTFS – основний формат за замовчуванням більшості сучасних версій цієї ОС, а також FAT – файлова система, успадкована від старої DOS, і exFAT, її подальше розширення. ReFS була також представлена Microsoft як файлова система нового покоління для серверних комп'ютерів, починаючи з Windows Server 2012. Файлову систему HPFS, розроблену Microsoft спільно з IBM, можна знайти тільки на дуже старих пристроях під управлінням Windows NT до версії 3.5.

**Файлові системи macOS**

Apple macOS застосовує два типи ФС: HFS+, розширення застарілої файлової системи HFS та APFS, формат, який використовується сучасними комп'ютерами Mac під управлінням macOS 10.14 і пізніших версій. Просунуті серверні продукти також використовують файлову систему Apple Xsan, кластерну файлову систему від StorNext і CentraVision.

**Файлові системи Linux**

Linux як операційна система з відкритим кодом спрямована на запровадження, тестування та використання різних типів файлових систем. Найпопулярніші формати для Linux охоплюють:

**Завдання**

1. Створити за допомогою API функцій файл для запису результатів виконання лабораторної роботи №3 (відповідно до свого варіанту).

2. Створити файл для запису результатів виконання лабораторної роботи №4 (відповідно до свого варіанту).

3. Реалізувати зміну прав доступу користувача до файлу.

4. Результати виконання роботи відобразити у звіті.

**Результати**

**Windows**

Код програми:

**main.cpp**

#include <Windows.h>

#include <algorithm>

#include <iostream>

#include <string>

#include <vector>

#include <chrono>

#include <AclAPI.h>

HANDLE file;

void tabulate(double a, double b, int n, int maxProcesses) {

if (a > b) {

std::cerr << "Invalid input" << std::endl;

return;

}

std::wstring commandLine = L"C:\\Users\\USER\\source\\repos\\university\\sem3\\os\\lab3\_tab\\x64\\Debug\\lab3\_tab.exe ";

if (SetFilePointer(file, 0, NULL, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

std::cerr << "Failed to move file pointer. Error: " << GetLastError() << std::endl;

return;

}

if (!SetEndOfFile(file)) {

std::cerr << "Failed to truncate file. Error: " << GetLastError() << std::endl;

return;

}

std::string output;

std::vector<double> timeResults;

std::vector<double> speedups;

for (int currentProcessesCount = 1; currentProcessesCount <= maxProcesses; currentProcessesCount++) {

std::cout << "------------------------------\n" << currentProcessesCount << " processes:\n";

output += "------------------------------\n" + std::to\_string(currentProcessesCount) + " processes:\n";

std::vector<STARTUPINFO> sis(currentProcessesCount);

std::vector<PROCESS\_INFORMATION> pis(currentProcessesCount);

std::vector<double> start(currentProcessesCount);

start.push\_back(a);

double division = (b - a) / currentProcessesCount;

int nDivided = n / currentProcessesCount;

int remainder = n % currentProcessesCount;

std::vector<double> end(currentProcessesCount);

for (int i = 0; i < currentProcessesCount; i++) {

start[i] = a + i \* division;

end[i] = start[i] + division;

}

end[currentProcessesCount - 1] += division / nDivided;

for (int i = 0; i < currentProcessesCount; i++) {

int currentDivisions = nDivided + (i == currentProcessesCount - 1 ? remainder : 0);

std::wstring newCommandLine = commandLine + std::to\_wstring(start[i])

+ L" " + std::to\_wstring(end[i]) + L" " + std::to\_wstring(currentDivisions);

ZeroMemory(&sis[i], sizeof(sis[i]));

sis[i].cb = sizeof(sis[i]);

ZeroMemory(&pis[i], sizeof(pis[i]));

if (!CreateProcess(

NULL,

&newCommandLine[0],

NULL,

NULL,

FALSE,

CREATE\_NO\_WINDOW,

NULL,

NULL,

&sis[i],

&pis[i])) {

std::cerr << "CreateProcess failed for process " << i << " (" << GetLastError() << ").\n";

}

}

std::vector<HANDLE> handles(currentProcessesCount);

auto startTime = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < currentProcessesCount; i++) {

handles[i] = pis[i].hProcess;

}

WaitForMultipleObjects(currentProcessesCount, handles.data(), TRUE, INFINITE);

for (int i = 0; i < currentProcessesCount; i++) {

CloseHandle(pis[i].hProcess);

CloseHandle(pis[i].hThread);

}

auto endTime = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> duration = endTime - startTime;

std::cout << "Launched program has finished. Time: " << duration.count() \* 1000 << std::endl;

output += "Launched program has finished. Time: " + std::to\_string(duration.count() \* 1000) + '\n';

timeResults.push\_back(duration.count() \* 1000);

}

for (int i = 0; i < timeResults.size(); i++) {

speedups.push\_back(timeResults[0] / timeResults[i]);

}

auto maxit = std::max\_element(speedups.begin(), speedups.end());

int index = std::distance(speedups.begin(), maxit);

std::cout << "\nMax speedup: " << \*maxit << " is reached using " << index + 1 << " processes.\n";

output += "\nMax speedup: " + std::to\_string(\*maxit) + " is reached using " + std::to\_string(index + 1) + " processes.\n";

DWORD bytesWritten;

BOOL result = WriteFile(

file,

output.c\_str(),

output.length(),

&bytesWritten,

NULL

);

}

void SetCustomSecurityInfoForFile() {

LPCSTR fileName = "results.txt";

std::string userName;

std::string permissionChoice;

DWORD permissions = 0;

std::string exitChoice = "";

while (true) {

if (exitChoice == "y") {

break;

}

std::cout << "Enter username:\n";

std::cin >> userName;

std::cout << "Select permissions for user " << userName << ":\n";

std::cout << "r - Read\n";

std::cout << "w - Write\n";

std::cout << "x - Execute\n";

std::cout << "Enter combination of permissions (e.g., RW for Read and Write):\n";

std::cin >> permissionChoice;

permissions = 0;

for (char c : permissionChoice) {

switch (tolower(c)) {

case 'r':

permissions |= GENERIC\_READ;

break;

case 'w':

permissions |= GENERIC\_WRITE;

break;

case 'x':

permissions |= GENERIC\_EXECUTE;

break;

default:

std::cerr << "Invalid permission choice: " << c << std::endl;

continue;

}

}

PACL pOldDACL = nullptr;

PSECURITY\_DESCRIPTOR pSD = nullptr;

DWORD result = GetNamedSecurityInfoA(

(LPSTR)fileName,

SE\_FILE\_OBJECT,

DACL\_SECURITY\_INFORMATION,

nullptr,

nullptr,

&pOldDACL,

nullptr,

&pSD

);

if (result != ERROR\_SUCCESS) {

std::cerr << "Failed to get existing ACL: " << result << std::endl;

if (pSD) LocalFree(pSD);

continue;

}

EXPLICIT\_ACCESSA ea;

ZeroMemory(&ea, sizeof(ea));

ea.grfAccessPermissions = permissions;

ea.grfAccessMode = SET\_ACCESS;

ea.grfInheritance = NO\_INHERITANCE;

ea.Trustee.TrusteeForm = TRUSTEE\_IS\_NAME;

ea.Trustee.TrusteeType = TRUSTEE\_IS\_USER;

ea.Trustee.ptstrName = (LPSTR)userName.c\_str();

PACL pNewDACL = nullptr;

result = SetEntriesInAclA(1, &ea, pOldDACL, &pNewDACL);

if (result != ERROR\_SUCCESS) {

std::cerr << "Failed to create new ACL: " << result << std::endl;

if (pSD) LocalFree(pSD);

if (pNewDACL) LocalFree(pNewDACL);

continue;

}

result = SetNamedSecurityInfoA(

(LPSTR)fileName,

SE\_FILE\_OBJECT,

DACL\_SECURITY\_INFORMATION,

nullptr,

nullptr,

pNewDACL,

nullptr

);

if (result == ERROR\_SUCCESS) {

std::cout << "Permissions updated successfully for user: " << userName << "\n";

} else {

std::cerr << "Failed to update permissions for user " << userName << ": " << result << std::endl;

}

// Cleanup

if (pSD) LocalFree(pSD);

if (pNewDACL) LocalFree(pNewDACL);

std::cout << "Are you done? y/n\n";

std::cin >> exitChoice;

}

}

int main() {

char func = 0;

double a = 0;

double b = 0;

int n = 0;

int maxProcesses = 0;

while (func != 'e') {

std::cout << "Enter the command: ";

func = getchar();

switch (func) {

case 'i': {

std::cout << "t - start tabulation function and write results to file\n"

<< "c - change file access attributes"

<< "e - exit program\n"

<< "s - set custom security for file\n"

<< "i - info/help\n";

}

case 'c': {

char attribute;

std::cout << "1. n - normal\n2. r - readonly\n3. a - archive\n4. h - hidden\n5. t - temporary\n6. d - directory\n";

std::cin >> attribute;

switch (attribute) {

case 'n':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_NORMAL);

std::cout << "File attributes were changed to normal\n";

break;

case 'r':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_READONLY);

std::cout << "File attributes were changed to readonly\n";

break;

case 'a':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_ARCHIVE);

std::cout << "File attributes were changed to archive\n";

break;

case 'h':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_HIDDEN);

std::cout << "File attributes were changed to hidden\n";

break;

case 't':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_TEMPORARY);

std::cout << "File attributes were changed to temporary\n";

break;

case 'd':

SetFileAttributesA("results.txt", FILE\_ATTRIBUTE\_DIRECTORY);

std::cout << "File attributes were changed to directory\n";

break;

default:

std::cout << "Invalid input\n";

break;

}

break;

}

case 's': {

SetCustomSecurityInfoForFile(); break;

}

case 't': {

file = CreateFileA(

"results.txt",

GENERIC\_READ | GENERIC\_WRITE, // Desired access (write)

NULL, // Share mode (none)

NULL, // Security attributes

CREATE\_ALWAYS, // Creation disposition

FILE\_ATTRIBUTE\_NORMAL, // Flags and attributes

NULL // Template file (not used)

);

if (file == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to create or open file. Error: " << GetLastError() << '\n';

continue;

}

std::cout << "\nEnter the limits: ";

std::cin >> a >> b >> n >> maxProcesses;

tabulate(a, b, n, maxProcesses);

std::cout << '\n';

CloseHandle(file);

break;

}

case 'e': break;

default: std::cout << "\nThe command is not recognized (for more info type i). Try again.\n";

}

getchar();

}

std::cout << "\nEnd of program.\n";

}

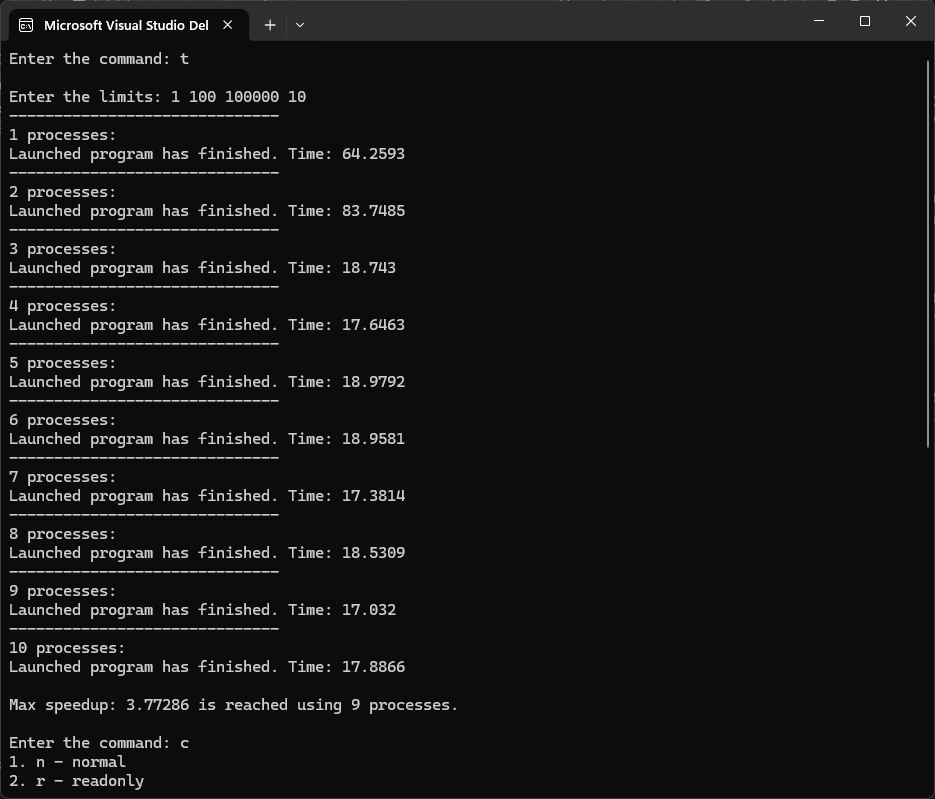


Рис. 1. Результат роботи програми у командному рядку (Частина 1)

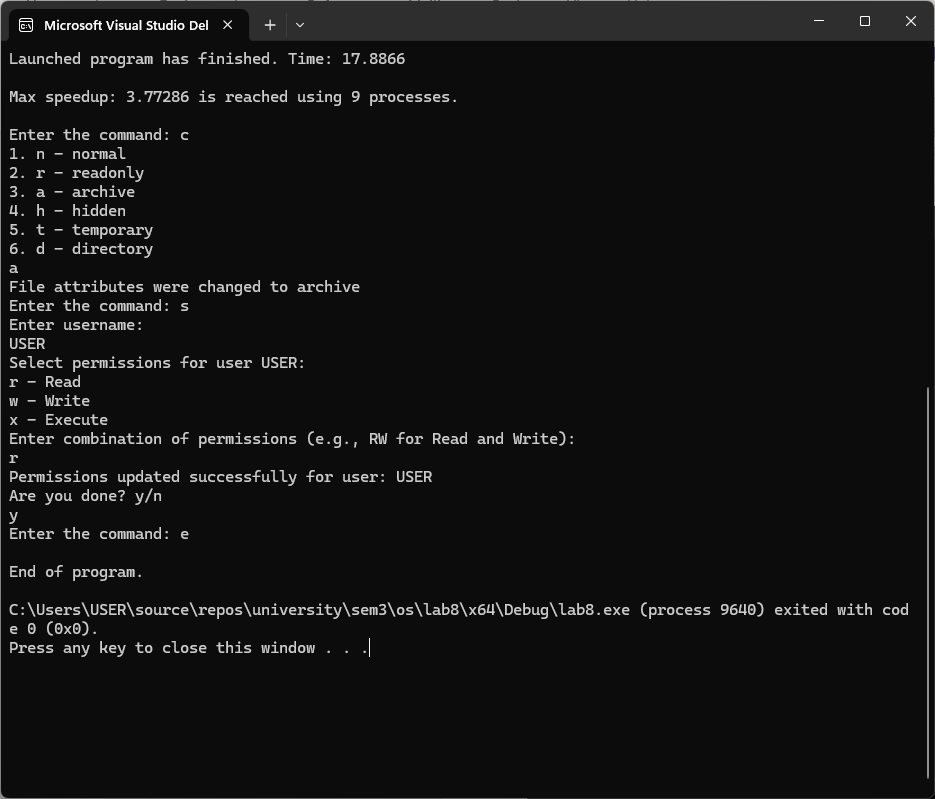


Рис. 2. Результат роботи програми у командному рядку (Частина 2)

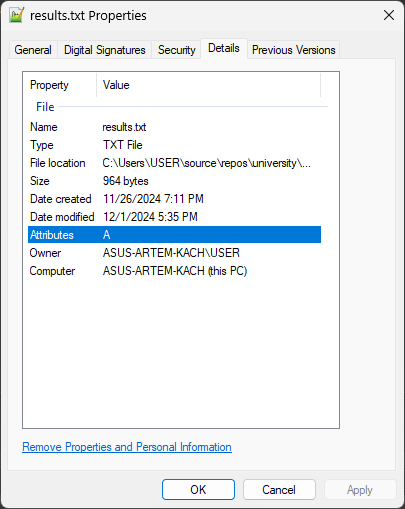


Рис. 3. Результат зміни атрибуту файлу на archive

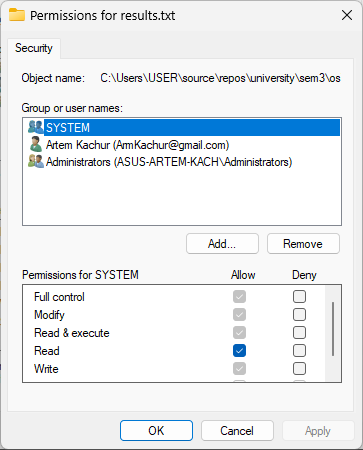


Рис. 4. Результат зміни прав доступу до файлу для користувача USER(Artem Kachur) на Read-only

**Linux**

Код програми:

**main.cpp**

#include <unistd.h>

#include <sys/wait.h>

#include <algorithm>

#include <iostream>

#include <string>

#include <vector>

#include <chrono>

#include <sys/stat.h>

#include <fcntl.h>

int file;

void tabulate(double a, double b, int n, int maxProcesses) {

if (a > b) {

std::cerr << "Invalid input" << std::endl;

return;

}

std::string output = "";

std::string commandPath = "/home/artem/OS/Labs/lab8/process\_tab\_part.exe";

std::vector<double> timeResults;

std::vector<double> speedups;

for (int currentProcessesCount = 1; currentProcessesCount <= maxProcesses; currentProcessesCount++) {

std::cout << "------------------------------\n" << currentProcessesCount << " processes:\n";

output += "------------------------------\n" + std::to\_string(currentProcessesCount) + " processes:\n";

std::vector<double> start(currentProcessesCount);

start[0] = a;

double division = (b - a) / currentProcessesCount;

int nDivided = n / currentProcessesCount;

int remainder = n % currentProcessesCount;

std::vector<double> end(currentProcessesCount);

for (int i = 0; i < currentProcessesCount; i++) {

start[i] = a + i \* division;

end[i] = start[i] + division;

}

end[currentProcessesCount - 1] += division / nDivided;

std::vector<pid\_t> pids(currentProcessesCount);

auto startTime = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < currentProcessesCount; i++) {

int currentDivisions = nDivided + (i == currentProcessesCount - 1 ? remainder : 0);

pids[i] = fork();

if (pids[i] < 0) {

std::cerr << "Fork failed for process " << i << std::endl;

return;

}

if (pids[i] == 0) {

// Child process: execute the target program with arguments

std::string startArg = std::to\_string(start[i]);

std::string endArg = std::to\_string(end[i]);

std::string divArg = std::to\_string(currentDivisions);

execl(commandPath.c\_str(), "process\_tab\_part.exe", startArg.c\_str(), endArg.c\_str(), divArg.c\_str(), (char\*)NULL);

perror("execl failed"); // Only called if execl fails

return;

}

}

for (int i = 0; i < currentProcessesCount; i++) {

int status;

waitpid(pids[i], &status, 0);

if (!WIFEXITED(status)) {

std::cerr << "Process " << i << " did not terminate normally." << std::endl;

}

}

auto endTime = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> duration = endTime - startTime;

std::cout << "Launched program has finished. Time: " << duration.count() \* 1000 << " ms" << std::endl;

output += "Launched program has finished. Time: " + std::to\_string(duration.count() \* 1000) + " ms\n";

timeResults.push\_back(duration.count() \* 1000);

}

for (size\_t i = 0; i < timeResults.size(); i++) {

speedups.push\_back(timeResults[0] / timeResults[i]);

}

auto maxit = std::max\_element(speedups.begin(), speedups.end());

int index = std::distance(speedups.begin(), maxit);

std::cout << "\nMax speedup: " << \*maxit << " is reached using " << index + 1 << " processes\n";

output += "\nMax speedup: " + std::to\_string(\*maxit) + " is reached using " + std::to\_string(index + 1) + " processes\n";

write(file, output.c\_str(), output.length());

}

int main() {

char func = 0;

double a = 0;

double b = 0;

int n = 0;

int maxProcesses = 0;

while (func != 'e') {

std::cout << "Enter the command: ";

func = getchar();

switch (func) {

case 'i': {

std::cout << "t - start tabulation function and write results to file\n"

<< "c - change file access attributes"

<< "e - exit program\n"

<< "i - info/help\n";

}

case 'c': {

int mode;

std::cout << "Enter the mode in octal int: ";

std::cin >> std::oct >> mode >> std::dec;

if (chmod("results.txt", mode) == -1) {

std::cerr << "Error setting mode.\n";

} else {

std::cout << "Successfully changed file access rights.\n";

}

break;

}

case 't': {

file = creat("results.txt", S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH);

if (file < 0) {

std::cerr << "Failed to create or open file.";

continue;

}

std::cout << "\nEnter the limits: ";

std::cin >> a >> b >> n >> maxProcesses;

tabulate(a, b, n, maxProcesses);

std::cout << '\n';

close(file);

break;

}

case 'e': break;

default: std::cout << "\nThe command is not recognized (for more info type i). Try again.\n";

}

getchar();

}

std::cout << "\nEnd of program.\n";

}

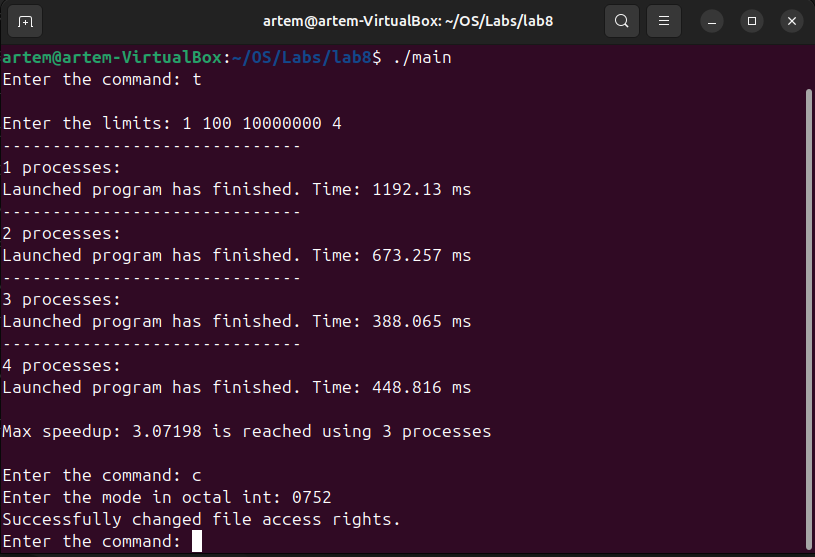


Рис. 5. Приклад виконання програми.

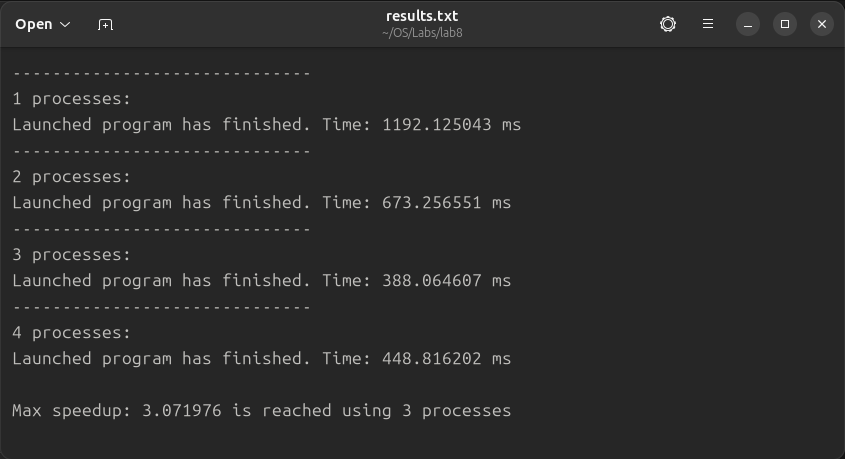


Рис. 6. Результат запису в файл.

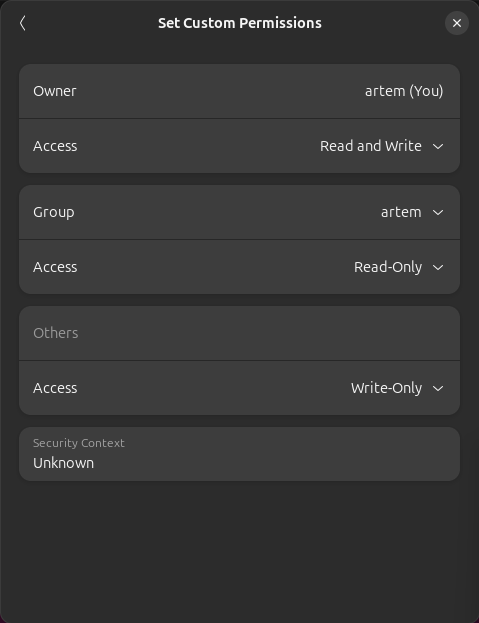


Рис. 7. Результат зміни у властивостях файлу

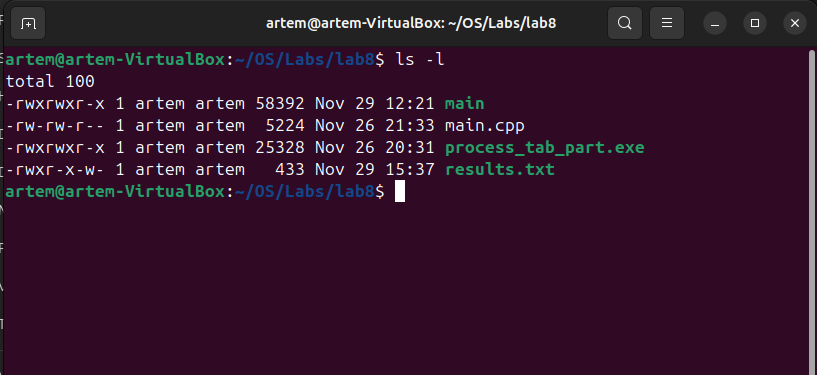


Рис. 8. Результат зміни атрибутів через команду ls -l

**Висновки**

На цій лабораторній роботі я ознайомився з файловими системами операційних систем Windows та Linux. У ході роботи я створював файли за допомогою API-функцій у Windows та відповідних функцій бібліотек <sys/stat.h> та <fcntl.h> і записував у них результати виконання функцій з 3 та 4 лабораторних робіт відповідно (табулювання функції). Також для створених файлів можна змінювати атрибути файлу та безпеки доступу.