**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **ПЗ**

### ЗВІТ

до лабораторної роботи № 3

**На тему:** *“Створення та керування процесами засобами API в*

*операційній системі WINDOWS”*

**З дисципліни:** *“Операційні системи”*

**Лектор:**

ст. викладач ПЗ

Грицай О.Д.

**Виконав:**

ст. гр. ПЗ-22

Солтисюк Д.А.

**Прийняв:**

ст. викладач ПЗ

Грицай О.Д.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_ .

Львів – 2022

**Тема роботи:** cтворення та керування процесами засобами API в

операційній системі WINDOWS

**Мета роботи:** ознайомитися з багатопоточністю в ОС Windows. Навчитися

працювати з процесами, використовуючи WinAPI-функції

**Теоретичні відомості**

Процес містить деяку стартову інформацію для потоків, які в ньому

створюватимуться. Процес має містити хоча б один потік, який система

скеровує на виконання.

Для створення нового процесу та його головного потоку використовується

функція CreateProcess().

BOOL CreateProcessA(

LPCSTR lpApplicationName,

LPSTR lpCommandLine,

LPSECURITY\_ATTRIBUTES lpProcessAttributes,

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

BOOL bInheritHandles,

DWORD dwCreationFlags,

LPVOID lpEnvironment,

LPCSTR lpCurrentDirectory,

LPSTARTUPINFOA lpStartupInfo,

LPPROCESS\_INFORMATION lpProcessInformation);

Якщо функція виконалась успішно, то повертається ненульове значення.

Якщо сталась помилка, то повернеться нуль.

Параметри:

1. lpApplicationName

Назва модуля, який потрібно виконати. Це може бути як повний шлях та

ім’я, так і скорочене ім’я. У випадку скороченого іменні, розглядається

поточний каталог. Обов'язково вказувати розширення. Параметр може мати

значення NULL. У такому випадку ім'ям програми важатиметься перше

ім'я, що стоїть в стрічці lpCommandLine.

2. lpCommandLine

Командний рядок, який потрібно виконати. Unicode-версія цієї функції

CreateProcessW може змінювати вміст цього рядка. Цей параметр не може

бути вказівником на пам'ять - читання (наприклад, змінну const). Якщо цей

параметр є константним рядком, функція може викликати порушення

доступу. Параметр lpCommandLine може бути NULL. У цьому випадку

функція використовує рядок, на який вказує lpApplicationName, як

командний рядок.

3. lpProcessAttributes

Вказівник на структуру SECURITY\_ATTRIBUTES, що визначає чи

повернений дескриптор об’єкту нового процесу може бути успадкований

дочірнім процесом. Якщо NULL, то дескриптор не успадковується.

4. lpThreadAttributes

Вказівник на структуру SECURITY\_ATTRIBUTES, що визначає чи

повернений дескриптор об’єкту нового процесу може бути успадкований

дочірнім процесом. Якщо NULL, то дескриптор не успадковується.

5. bInheritHandles

Відповідає за успадкування дескрипторів. Якщо TRUE, то кожен

дескриптор, що можна успадкувати у процесі що викликається,

успадковується новим процесом. Успадковані дескриптори мають теж

значення і права доступу як і оригінальні дескриптори.

6. dwCreationFlags

Прапорці, які керують класом пріоритетності та створенням процесу. Цей

параметр dwCreationFlags також керує новим класом пріоритетності

процесу, який використовується для визначення пріоритетів планування

потоків процесу. Якщо жоден із прапорів класу пріоритетів не вказаний,

клас пріоритету за замовчуванням до NORMAL\_PRIORITY\_CLASS, якщо

класом пріоритету процесу створення не є IDLE\_PRIORITY\_CLASS або

BELOW\_NORMAL\_PRIORITY\_CLASS. У цьому випадку дочірній процес

отримує типовий клас пріоритетності процесу виклику.

7. lpEnvironment

Вказівник на блок оточення для нового процесу. Якщо цей параметр

NULL, новий процес використовує середовище процесу виклику.

8. lpCurrentDirectory

Повний шлях до поточного каталогу для процесу. Якщо цей параметр

NULL, новий процес матиме той самий поточний диск та каталог, що і

процес виклику.

9. lpStartupInfo

Вказівник на структуру STARTUPINFO або STARTUPINFOEX.

Дескриптори в STARTUPINFO або STARTUPINFOEX повинні закриватися

CloseHandle, коли вони більше не потрібні.

10. lpProcessInformation

Вказівник на структуру PROCESS\_INFORMATION, яка отримує

ідентифікаційну інформацію про новий процес. Дескриптори в

PROCESS\_INFORMATION повинні бути закриті за допомогою

CloseHandle, коли вони більше не потрібні.

Перед викликом функції створення процесу необхідно задати структури

STARTUPINFO та PROCESS\_INFORMATION.

Структура PROCESS\_INF0RMATION містить чотири поля:

1. hProcess — дескриптор створеного процесу;

2. hThread — дескриптор його головного потоку;

3. dwProcessId — ідентифікатор процесу (process id, pid);

4. dwThreadId — ідентифікатор головного потоку (thread id, tid).

**Індивідуальне завдання**

1. Створити окремий процес, і здійснити в ньому розв’язок задачі

згідно варіанту у відповідності до порядкового номера у журнальному

списку (підгрупи).

2. Реалізувати розв’язок задачі у 2-ох, 4-ох, 8-ох процесах. Виміряти

час роботи процесів за допомогою функцій WinAPI. Порівняти результати

роботи в одному і в багатьох процесах.

3. Для кожного процесу реалізувати можливість його запуску,

зупинення, завершення та примусове завершення («вбиття»).

4. Реалізувати можливість зміни пріоритету виконання процесу.

5. Продемонструвати результати виконання роботи, а також кількість

створених процесів у “Диспетчері задач”, або подібних утилітах (н-д,

ProcessExplorer)

Варіант 15:

Табулювати функцію ln x, задану розкладом в ряд Тейлора, в області її

визначення на відрізку від А до В (кількість кроків не менше 100 000

–задається користувачем).

**Протокол роботи**

Окрема програма для виводу розкладу ряду Тейлора на заданому проміжку

#include <cmath>

#include <iostream>

#include <string>

void tabulate\_lnx(double a, double b, double step, double iter\_count) {

for (double n = a; n <= b; n += step) {

double num, m, c, sum = 0;

num = (n - 1) / (n + 1);

for (auto i = 0; i < iter\_count; i++) {

m = (2 \* i) - 1;

c = pow(num, m);

c = c / m;

sum = sum + c;

}

sum = 2 \* sum;

std::cout << "ln(" << n << ") = " << sum << std::endl;

}

}

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

if (argc != 5) {

return 1;

}

double a = atof(argv[1]);

double b = atof(argv[2]);

double step = atof(argv[3]);

int iterations = atoi(argv[4]);

std::cout << "a: " << a << " b " << b << " step " << step << " "

<< " iter count" << iterations << std::endl;

tabulate\_lnx(a, b, step, iterations);

}

Реалізую задання діапазону від А до В, задання кроку (від 100 000), задання точності

(від 10 ітерацій), створення, призупинення, запуск процесу, вивід процесорного часу процеса, а також термінацію процесу

та зміну пріоритету по ідентифікатору процеса.

void MainWindow::createProcesses(int chunksCount, double a, double b,

int stepCount, double iterCount) {

auto distance = b - a;

auto step = distance / stepCount;

auto chunkSize = distance / chunksCount;

iterCount /= chunksCount;

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

createProcess(a + chunkSize \* i, a + chunkSize \* (i + 1), step, iterCount);

}

QString MainWindow::getExecutionTime(PROCESS\_INFORMATION pi) {

long C\_TIME = 0, E\_TIME = 0, K\_TIME = 0, U\_TIME = 0;

GetProcessTimes(pi.hProcess, (FILETIME \*)&C\_TIME, (FILETIME \*)&E\_TIME,

(FILETIME \*)&K\_TIME, (FILETIME \*)&U\_TIME);

return QString::number(U\_TIME \* pow(10.0, -3), 'g', 10) + "ms";

}

void MainWindow::createProcess(const double a, const double b,

const double step, const int iterCount) {

std::string command = "Z:\\3\\program\_7.exe " + std::to\_string(a) + " " +

std::to\_string(b) + " " + std::to\_string(step) + " " +

std::to\_string(iterCount);

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

TCHAR cmdLine[1024] = {0};

mbstowcs(cmdLine, command.c\_str(), 1024);

if (!CreateProcess(NULL, cmdLine, NULL, NULL, true, CREATE\_NEW\_CONSOLE, NULL,

NULL, &si, &pi)) {

QMessageBox::warning(

this, "Warning",

"Could not create child process." + QString::number(GetLastError()));

throw 1;

}

procInfos.push\_back(pi);

}

BOOL MainWindow::terminateProcess(const DWORD dwProcessId,

const UINT uExitCode) {

for (auto proc : procInfos) {

if (proc.dwProcessId == dwProcessId) {

BOOL result = TerminateProcess(proc.hProcess, uExitCode);

CloseHandle(proc.hProcess);

return result;

}

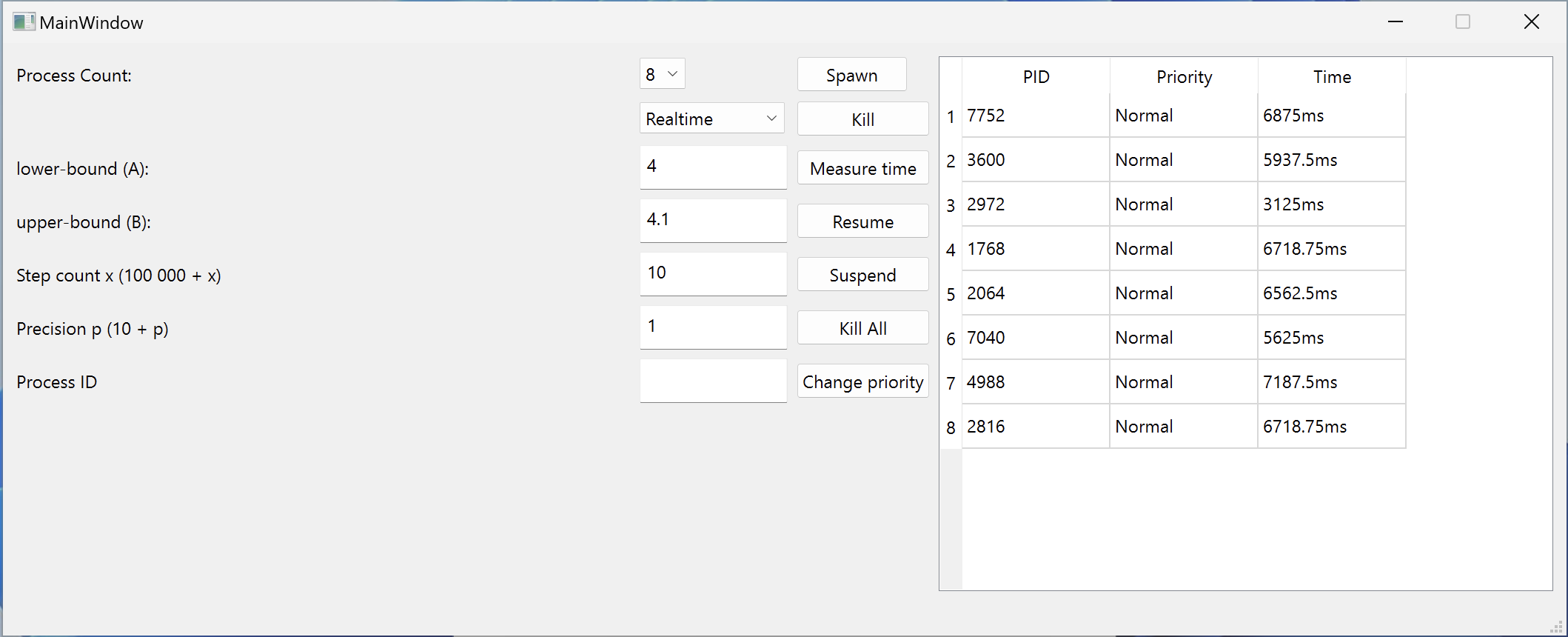
}

QMessageBox::warning(this, "No such child PID found",

"Make sure you entered correct PID");

return false;

}

**Висновки**

Під час виконання даної лабораторної роботи, я вивчив деякі функції з WINAPI та зрозумів як операційна система керує процесами та комунікацією між ними. Я зрозумів, як саме ядро операційної системи може запускати, призупиняти та вбивати процеси, змінювати їхній пріоритет та діставати статистику щодо їхньої роботи.