Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп'ютерних технологій, автоматики та метрології Спеціальність «Комп’ютерна Інженерія»

Кафедра СКС



**Звіт**

з лабораторної роботи №1

з дисципліни «Системне програмне забезпечення»

на тему «Керування процесами та потоками»

**Варіант №3**

Виконав:

студент групи KI-307

Ващишин І.І.

Прийняв:  
Олексів М.В.

ЛЬВІВ 2025

**Мета роботи:** навчитися керувати процесами і потоками в середовищі операційної системи, розробляти програми керування процесами і потоками.

**Теоретична частина**

***Складові елементи процесу:***

1. Адресний простір процесу складається з набору адрес віртуальної пам'яті, які він може використати. Адресний простір процесу недоступний іншим процесам.

2. Процес володіє системними ресурсами, такими як файли, мережні з'єднання, пристрої введення-виведення, об'єкти синхронізації тощо.

3. Процес містить деяку стартову інформацію для потоків, які в ньому створюватимуться.

4. Процес має містити хоча б один потік, який система скеровує на виконання. Без потоків у Windows наявність процесів неможлива.

***Структури даних процесу***

1. Для виконавчої системи Windows кожний процес зображується об'єктом-процесом виконавчої системи (executive process object); його також називають керуючим блоком процесу (executive process block, EPROCESS).

2. Для ядра системи процес зображується об'єктом-процесом ядра (kernel process object), його також називають блоком процесу ядра (process kernel block, KPROCESS).

3. У режимі користувача доступним є блок оточення процесу (process environment block, РЕВ), що перебуває в адресному просторі цього процесу.

Керуючий блок процесу містить такі основні елементи:

1. блок процесу ядра (KPROCESS);

2. ідентифікаційну інформацію;

3. інформацію про адресний простір процесу;

4. інформацію про ресурси, доступні процесу, та обмеження на використання цих ресурсів;

5. блок оточення процесу (РЕВ);

6. інформацію для підсистеми безпеки.

До ідентифікаційної інформації належать:

1. ідентифікатор процесу (pid);

2. ідентифікатор процесу, що створив цей процес;

3. ім'я завантаженого програмного файлу.

Блок процесу ядра містить усю інформацію, що належить до потоків цього процесу:

1. покажчик на ланцюжок блоків потоків ядра, де кожний блок відповідає потоку;

2. базову інформацію, необхідну ядру системи для планування потоків

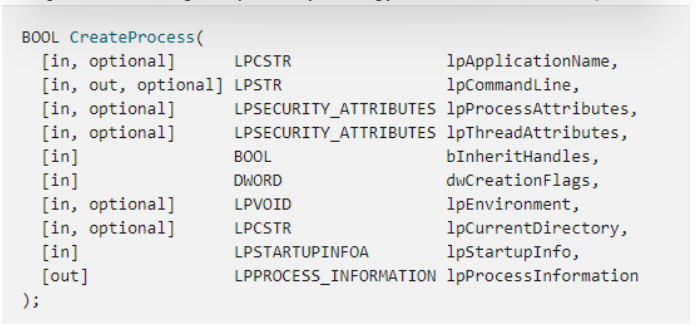
Блок оточення процесу містить інформацію про процес, яка призначена для доступу з режиму користувача:

1. початкову адресу ділянки пам'яті, куди завантажився програмний файл;

2. покажчик на динамічну ділянку пам'яті, доступну процесу

**Створення процесів у Win API**

**CreateProcess()** - функція для створення нового процесу.

***Прототип функції:*** 

***Параметри функції та їх опис:***

***lpApplicationName*** - шлях до виконуваного файлу

***lpCommandLine*** - повний командний рядок для запуску виконуваного файлу

***lpProcessAttributes*, *lpThreadAttributes*** - атрибути безпеки для всього процесу і для головного потоку

***bInheritHandles*** - керує спадкуванням нащадками дескрипторів об'єктів

***dwCreationFlags*** - маска прапорів створенням нового процесу

***lpEnvironment*** - покажчик на пам'ять із новими змінними оточення, які предок може задаватидля нащадка

***lpCurrentDirectory*** - рядок із новим значенням поточного каталогу для нащадка

***lpStartupInfo*** - покажчик на структуру даних типу STARTUPINFO, на базі якої задають параметри для процесу - нащадка

***lpProcessInformation*** - покажчик на структуру даних PR0CESS\_INFORMATION, якузаповнює ОС під час виклику CreateProcess() ***Структура PR0CESS\_INF0RMATI0N містить чотири поля:***

* ***hProcess*** - дескриптор створеного процесу
* ***hThread*** - дескриптор його головного потоку
* ***dwProcessId*** - ідентифікатор процесу (process id, pid)
* ***dwThreadId*** - ідентифікатор головного потоку (thread id, tid)
* **GetCurrentProcess()** - функція для отримання поточного дескриптора процесу.

**Завдання:** Розробити дві програми. Перша шукає найбільше число у динамічному двовимірному масиві цілих чисел розміру n×n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найбільшого оформити як функцію потоку. Друга програма запускає першу як новостворений процес. Обидві програми мають виводити інформацію про усі запущені процеси і потоки (дескриптор та ідентифікатор).

**Індивідуальне завдання:**

Процеси:

5. Розробити дві програми. Перша приймає від користувача два рядки. Далі, якщо обидва рядки містять цілі числа зі знаком, то на екран виводиться сума чисел, в іншому випадку – конкатенація двох введених рядків. Друга програма запускає першу як новостворений процес.

Потоки:  
5.Розробити програму, яка здійснює введення двох рядків, заданих користувачем. Далі, якщо обидва рядки містять цілі числа зі знаком, то на екран виводиться сума чисел, в іншому випадку – конкатенація введенних рядків. Перевірку на відповідність рядка цілому числу, обчислення суми чисел та конкатенації рядків оформити як три різних функції потоку. Введення рядків здійснюється до запуску всіх потоків, а виведення результатів – після їх завершення

**Виконання завдання**

**Код програм для процесів:**

**lab\_one\_executor.cpp**

#include <stdio.h>

#include <windows.h>

int main() {

const char\* lpApplicationName = "./lab\_one.exe";

PROCESS\_INFORMATION pi;

STARTUPINFO si;

int status = 0;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

// Встановлюємо прапорець CREATE\_NEW\_CONSOLE, щоб створити нове консольне

// вікно

si.dwFlags = STARTF\_USESHOWWINDOW;

si.wShowWindow = SW\_SHOWDEFAULT;

if (!CreateProcess(lpApplicationName, NULL, NULL, NULL, FALSE,

CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi)) {

fprintf(stderr, "CreateProcess failed (%lu).\n", GetLastError());

return 1;

}

printf("Child process started...\n");

// Виведення інформації про новий процес та батьківський процес

printf("\nChild process ID: %lu\n", pi.dwProcessId);

printf("Child thread ID: %lu\n", pi.dwThreadId);

printf("Child process Handle: %p\n", pi.hProcess);

printf("Child thread Handle: %p\n", pi.hThread);

// Виведення інформації про батьківський процес

printf("\nParent process ID: %lu\n", GetCurrentProcessId());

printf("Parent thread ID: %lu\n", GetCurrentThreadId());

printf("Parent process Handle: %p\n", GetCurrentProcess());

printf("Parent thread Handle: %p\n", GetCurrentThread());

// Очікування завершення дочірнього процесу

WaitForSingleObject(pi.hProcess, INFINITE);

// Отримання статусу завершення процесу

GetExitCodeProcess(pi.hProcess, (LPDWORD)&status);

printf("\nChild process finished with status %d\n", status);

// Закриття дескрипторів, які більше не потрібні

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return 0;

}

**lab\_one.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

// Структура для передачі аргументів у функцію потоку

struct ThreadArgs {

int \*array; // Динамічний масив

int array\_size; // Розмір масиву

int max;

};

// Функція потоку для обертання масиву

DWORD WINAPI reverseArray(LPVOID lpParam) {

struct ThreadArgs \*threadArgs = (struct ThreadArgs \*)lpParam;

int \*array = threadArgs->array;

int size = threadArgs->array\_size;

int max\_index = -1;

if (size > 0)

max\_index = 0;

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

if (array[i] > array[max\_index])

max\_index = i;

}

threadArgs->max = array[max\_index];

return 0;

}

int main() {

int n;

printf("Enter the size of the array: ");

scanf\_s("%d", &n);

// Генерування динамічного масиву з випадковими значеннями

int \*dynamicArray = (int \*)malloc(n \* sizeof(int));

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

dynamicArray[i] = rand() % 100; // Випадкові значення від 0 до 99

}

// Виведення початкового масиву

printf("\nOriginal array:\n");

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

printf("%d ", dynamicArray[i]);

}

printf("\n");

// Створення структури з аргументами для функції потоку

struct ThreadArgs threadArgs;

threadArgs.array = dynamicArray;

threadArgs.array\_size = n;

HANDLE hThread;

DWORD dwThreadId;

// Створення потоку для обертання масиву

hThread = CreateThread(NULL, // default security attributes

0, // use default stack size

reverseArray, // thread function

&threadArgs, // argument to thread function

0, // use default creation flags

&dwThreadId); // returns the thread identifier

if (hThread == NULL) {

fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());

return 1;

}

// Чекаємо завершення потоку

WaitForSingleObject(hThread, INFINITE);

// Виведення обернутого масиву

printf("\Max elemnet is:%d\n", threadArgs.max);

printf("\n");

// Виведення інформації про потік та основний процес

printf("\nThread ID: %lu\n", dwThreadId);

printf("Thread Handle: %p\n\n", hThread);

printf("Main process ID: %lu\n", GetCurrentProcessId());

printf("Main thread ID: %lu\n", GetCurrentThreadId());

printf("Main process Handle: %p\n", GetCurrentProcess());

printf("Main thread Handle: %p\n", GetCurrentThread());

// Закриття дескрипторів, які більше не потрібні

CloseHandle(hThread);

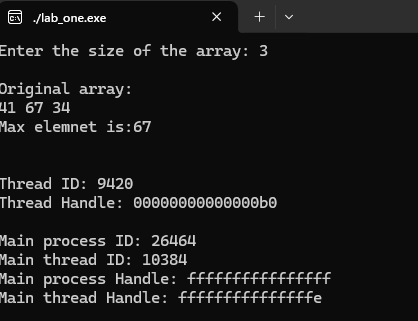
free(dynamicArray);

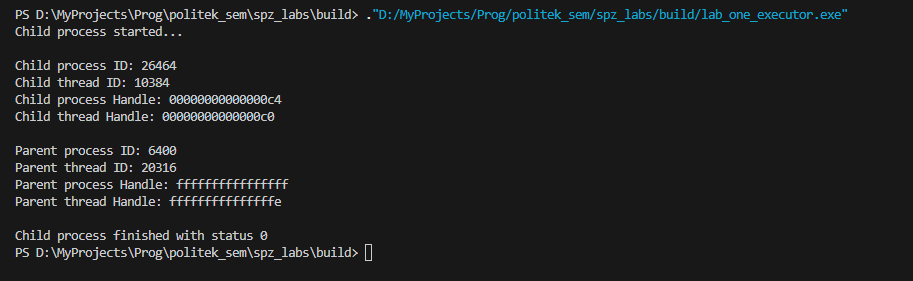
getchar();

getchar();

return 0;

}

**Результат виконання:**

**Результат програми:**

**Висновок:** під час виконання лабораторної роботи я навчився керувати процесами і потоками в середовищі операційної системи, розробляти програми керування процесами і потоками.