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 №2

з дисципліни «Системне програмне забезпечення»

на тему «Організація взаємодії між процесами та потоками»

**Варіант №3**

Виконав:

студент групи KI-307

Ващишин І.І.

Прийняв:  
Олексів М.В.

ЛЬВІВ 2025

**Мета:** навчитися планувати процеси і потоки в середовищі операційної системи, розробляти програми планування процесів і потоків.

**Теоретичний вступ**

1 Алгоритми планування

Виконання потоку – це цикл чергування періодів обчислень і періодів очікування введення-виведення. Інтервал часу, коли потік виконує інструкції процесора, називають інтервалом використання процесора (CPU burst), інтервал часу очікування введення-виведення — інтервалом введення-виведення (I/O burst).

1.1 Планування за принципом FIFO Чергу готових потоків організовують за принципом FIFO. Коли в системі створюється новий потік, його керуючий блок додається у хвіст черги. Коли процесор звільняється, його надають потоку з голови черги.

1.2 Кругове планування Кожному потокові виділяють інтервал часу (квант часу) і протягом якого цьому потокові дозволено виконуватися. Коли потік усе ще виконується після вичерпання кванта часу, його переривають і перемикають процесор на виконання інструкцій іншого потоку. Коли він блокується або закінчує своє виконання до вичерпання кванта часу, процесор теж передають іншому потокові. Довжина кванта часу для всієї системи однакова.

1.3 Планування із пріоритетами Кожному потокові надають пріоритет, на виконання ставиться потік із найвищим пріоритетом із черги готових потоків. Пріоритети можуть надаватися потокам статично або динамічно. Розподіл пріоритетів може призвести до того, що потоки процесів із низьким пріоритетом чекатимуть дуже довго. Таку ситуацію називають голодуванням.

1.4 Планування на підставі характеристик подальшого виконання Рішення про вибір потоку для виконання приймають на підставі знання або оцінки характеристик подальшого його виконання. Алгоритм «перший - із найкоротшим часом виконання» (STCF). З кожним потоком пов'язують тривалість наступного інтервалу використання ним процесора і для виконання щоразу вибирають той потік, у якого цей інтервал найкоротший. У результаті потоки, що захоплюють процесор на коротший час, отримують під час планування перевагу і швидше виходять із системи. Алгоритм «перший — із найкоротшим часом виконання, що залишився» (SRTCF). Його відмінність від SCTF полягає в тому, що, коли в чергу готових потоків додають новий, у якого наступний інтервал використання процесора коротший, ніж час, що залишився до завершення виконання поточного потоку, поточний потік переривається, і на його місце стає новий потік.

1.5 Багаторівневі черги зі зворотним зв'язком Є кілька черг готових потоків із різним пріоритетом, потоки черги із нижчим пріоритетом виконуються, тільки коли всі черги верхнього рівня порожні. Потокам дозволено переходити із черги в чергу. Потоки в одній черзі об'єднуються за довжиною інтервалу використання процесора. Потоки із коротшим інтервалом перебувають у черзі з більшим пріоритетом. Усередині всіх черг використовують кругове планування (у найнижчій працює FIFO-алгоритм). Якщо потік вичерпав свій квант часу, він переміщається у хвіст черги із нижчим пріоритетом. У результаті потоки з коротшими інтервалами залишаються з високим пріоритетом, а потоки з довшими інтервалами продовжують свій квант часу. Можна автоматично переміщати потоки, які давно не отримували керування, із черги нижнього рівня на рівень вище.

1.6 Лотерейне планування Ідея лотерейного планування: 1. потік отримує деяку кількість лотерейних квитків, кожен з яких дає право користуватися процесором упродовж часу T; 2. планувальник через проміжок часу Т вибирає випадково один лотерейний квиток; 3. потік, «що виграв», дістає керування до наступного розіграшу.

**Індивідуальне завдання:**

Розробити програму, яка 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. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.

**Виконання завдання**

**Код програми:**

#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);

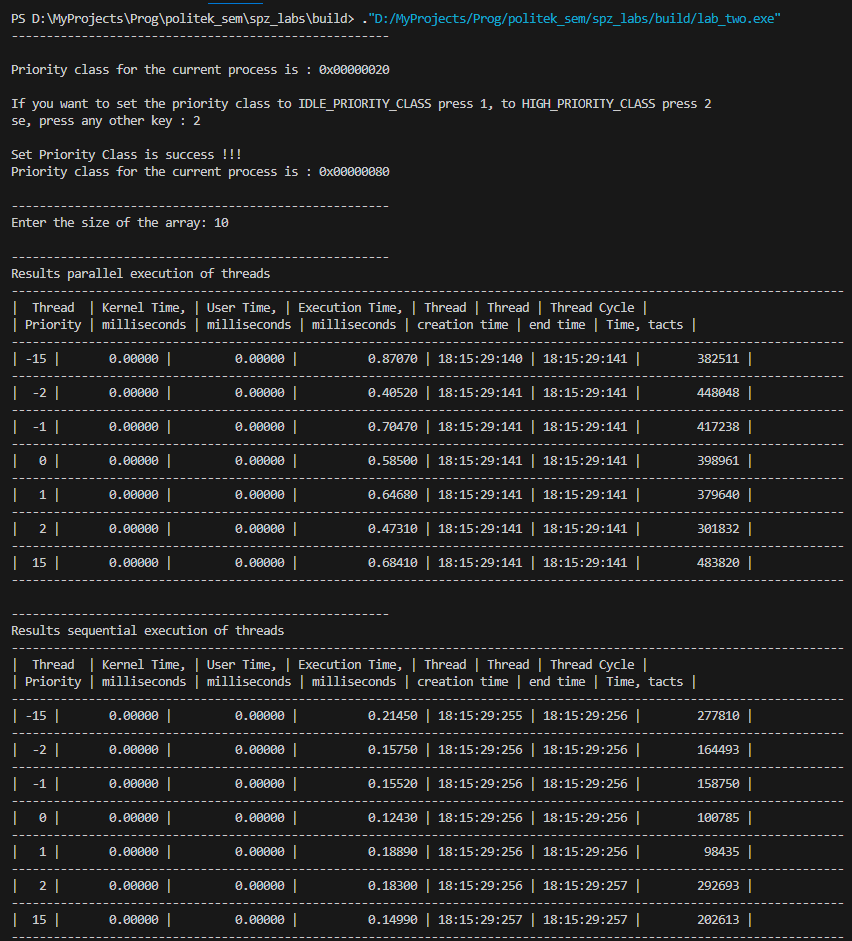
getchar();

getchar();

return 0;

}

**Результат виконання програми**



**Висновок:** під час виконання даної лабораторної роботи, я навчився планувати процеси і потоки в середовищі операційної системи, розробляти програми планування процесів і потоків.