A drawing of a building

AI-generated content may be incorrect.

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
 НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проектування**

**Звіт**

**про виконання практичної роботи №3  
з дисципліни «Паралельні обчислення»**

Виконав:  
студент III курсу, групи ДА-21  
Сімков Микита

Прийняв:

асистент Яременко В. С.

Київ – 2025

**ПОСТАНОВКА ЗАДАЧІ**

1. Ознайомитися з визначенням поняття пул потоків (thread pool), використовуючи даний методичний посібник, або ж сторонні джерела. Ознайомитися з примітивами синхронізації та проблемами, пов’язаними з ними.
2. Реалізувати власний пул потоків з характеристиками, зазначеними в обраному варіанті. Обов’язкові для кожного варіанту характеристики: пул потоків повинен бути написаним коректно відносно обраної мови програмування, повинен мати можливість коректного завершення своєї роботи (моментально, з покиданням всіх активних задач, так і з завершенням активних задач), можливість тимчасової зупинки своєї роботи, працювати з використанням умовних змінних)
3. Операції ініціалізації та знищення пулу, додавання та вилучення задач в чергу повинні бути безпечними з точки зору паралелізму.
4. Створити програму, котра буде виконувати задачі за обраним варіантом, використовуючи написаний студентом пул потоків. Код відповідальний за додавання задач в пул потоків, та сам пул потоків повинні знаходитися в різних потоках виконання. Задачі в пул потоків повинні додаватися з декількох потоків.
5. Перевірити та довести коректність роботи програми з використанням системи вводу/виводу інформації в консоль (або іншого засобу – профайлера).
6. Зробити обмежене за часом тестування та розрахувати кількість створених потоків та середній час знаходження потоку в стані очікування. Для задач з необмеженою чергою – визначити середню довжину кожної черги та середній час виконання задач. Для обмежених за розміром черг – визначити максимальний та мінімальний час, поки черга була заповнена, кількість відкинутих задач.
7. В протоколі роботи описати ключові моменти реалізації пулу потоків в цілому, а також ключові моменти при імплементації конкретного варіанту завдання. Описати публічний інтерфейс пулу потоків та механізм його роботи. Описати механізм тестування розробленого рішення. Значення метрик, перечисллений в пункті 6 для різних проміжків часу, обраних в довільному порядку самим студентом.
8. Надати висновок, що повинен містити аналіз отриманих результатів.

Завдання за варіантом (№20)

Пул потоків обслуговується 2-ма робочими потоками й має одну чергу

виконання. Задачі додаються відразу в кінець черги виконання. Черга

задач виконується з інтервалом в 30 секунд (буфер наповнюється

задачами на протязі 30-ти секунд, котрі потім виконуються). Задачі, що

додаються під час виконання черги задач відкидаються. Задача займає

випадковий час від 2 до 8 секунд.

**ХІД РОБОТИ**

Напишемо програму, що вирішить задану задачу. Для генерації тасок використовуватимемо окремі потоки.

Код програми:

#include <iostream>

#include <queue>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <chrono>

#include <functional>

#include <random>

#include <vector>

using namespace std;

class ThreadPool {

private:

queue<function<void()>> tasks;

mutex mtx;

condition\_variable cv;

bool accepting = false;

bool stop = false;

int workerCount;

public:

ThreadPool(int count) : workerCount(count) {

cout << "[POOL] Thread pool created with " << workerCount << " worker threads.\n";

}

bool isEmpty() {

lock\_guard<mutex> lock(mtx);

return tasks.empty();

}

void addTask(function<void()> task) {

lock\_guard<mutex> lock(mtx);

if (!accepting) {

return;

}

tasks.push(task);

cout << "[POOL] Task added to queue. Current queue size: " << tasks.size() << "\n";

}

void workerFunction(int id) {

while (true) {

function<void()> task;

{

unique\_lock<mutex> lock(mtx);

cv.wait(lock, [this] { return stop || !tasks.empty(); });

if (stop) {

cout << "[WORKER " << id << "] Stopping thread.\n";

break;

}

if (!tasks.empty()) {

task = tasks.front();

tasks.pop();

cout << "[WORKER " << id << "] Task taken. Remaining tasks: " << tasks.size() << "\n";

}

}

if (task) {

cout << "[WORKER " << id << "] Executing task...\n";

task();

cout << "[WORKER " << id << "] Task execution finished.\n";

}

}

}

void startAcceptingTasks() {

lock\_guard<mutex> lock(mtx);

accepting = true;

cout << "[POOL] Now accepting new tasks.\n";

}

void stopAcceptingAndExecute() {

{

lock\_guard<mutex> lock(mtx);

accepting = false;

}

cout << "[POOL] 30 seconds elapsed. Stopping acceptance of new tasks and starting execution.\n";

cv.notify\_all();

}

void stopWorkers() {

{

lock\_guard<mutex> lock(mtx);

stop = true;

}

cv.notify\_all();

}

};

int main() {

ThreadPool pool(2);

vector<thread> workers;

for (int i = 0; i < 2; i++) {

workers.push\_back(thread(&ThreadPool::workerFunction, &pool, i));

}

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dist(2, 8);

thread generator([&pool, &gen, &dist]() {

cout << "[GENERATOR] Task generator started.\n";

while (true) {

int taskTime = dist(gen);

function<void()> task = [taskTime]() {

cout << "[TASK] Task executing for " << taskTime << " seconds.\n";

this\_thread::sleep\_for(chrono::seconds(taskTime));

cout << "[TASK] Task completed.\n";

};

pool.addTask(task);

this\_thread::sleep\_for(chrono::seconds(3));

}

});

while (true) {

pool.startAcceptingTasks();

this\_thread::sleep\_for(chrono::seconds(30));

pool.stopAcceptingAndExecute();

// Execution phase (30 seconds)

this\_thread::sleep\_for(chrono::seconds(30));

}

pool.stopWorkers();

for (auto& w : workers) {

if (w.joinable())

w.join();

}

generator.join();

return 0;

}

Подивимося на результат виконання програми:

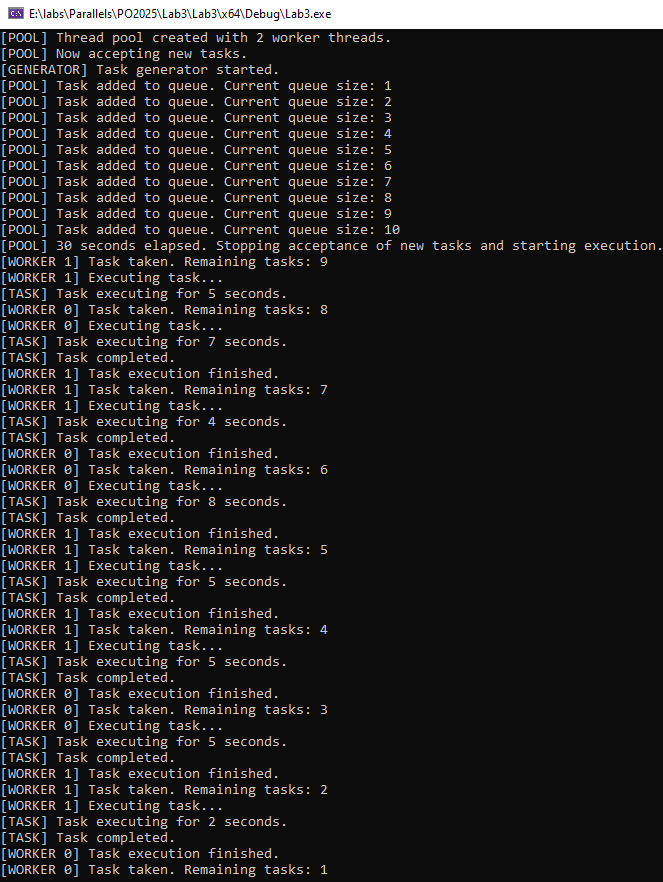


Рисунок 1 — Логи створення та виконання ,,задач,, потоками.

**ВИСНОВКИ**

Було реалізовано thread pool, що працює за заданим алгоритмом. 1 потік генерує завдання, а 2 інших паралельно їх виконують. Виконання та генерація завдань відбуваєтсья у вічному циклі.