

**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-22  
Байдюк Антон Олексійович

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.

Варіант 1:

Пул потоків обслуговується 4-ма робочими потоками й має одну чергу

виконання. Задачі додаються одразу в кінець черги виконання. Наявний

механізм отримання результатів виконання задачі (як один з варіантів –

при додаванні задачі отримується її ID, за даним ID можна отримати

результат виконання та статус задачі). Задача береться на виконання з

буферу одразу за наявності вільного робочого потоку. Задача займає

випадковий час від 5 до 10 секунд.

**ХІД РОБОТИ**

В ході лабораторної роботи на мові c++ у середовищі visual studio буде реалізовано пул потоків у форматі роботи “фастфуду”.

Потоків-робітників буде 7:

3 - касири

4 - кухарі

\*усі значення кількості робітників та змінних задаються глобально, тому можна потестувати роботу змінивши їх

**Структура роботи закладу наступна:**

Присутнє “меню” із 8 найменувань. Із випадковим інтервалом часу, замовлення, що складається із ID та назви потрапляє у чергу, звідки його розбирають потоки-касири. Ті в свою чергу оброблюють кожне замовлення випадковий час, якщо вільних касирів немає - замовлення потраляють у чергу обмеженого розміру. Наступні замовлення будуть відхилені, якщо чергу переповнено. Усі замовлення виводяться у консолі із їх ID, статусом (rejected, in progress або done), ім’ям касира та кухаря. В кінці підводиться підсумок роботи усіх працівників, мінімальний а максимальний час переповненого стану черги та кількість відхилених замовлень.

Код програми наведено у лістингу після висновку

**Результат виконання програми:**

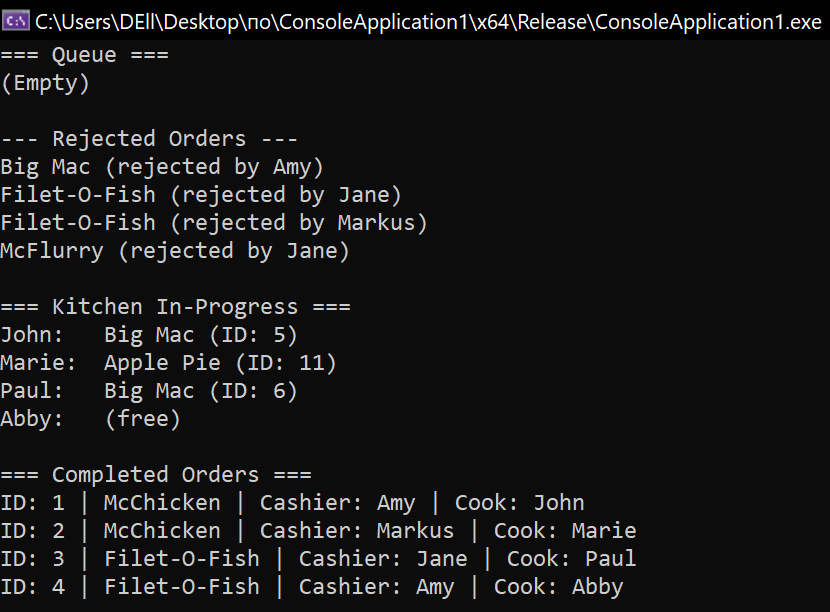


Рис. 1 — Стан консолі під час виконання програми



Рис. 2 — Результат виконаної програма

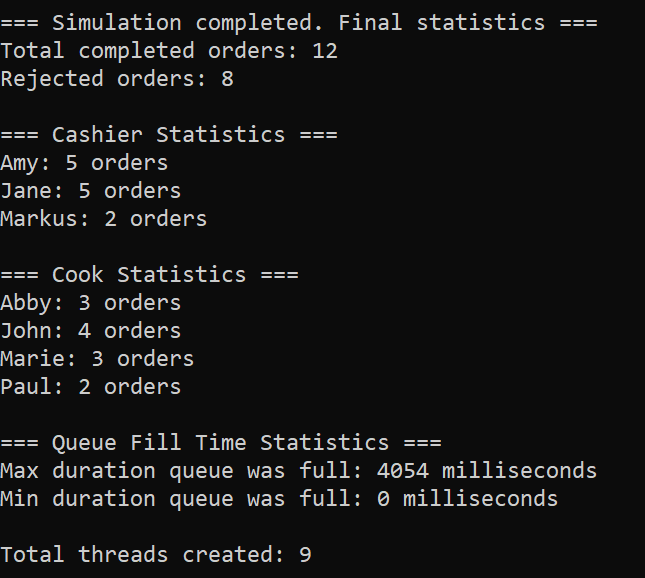


Рис. 3 — Статистика результату програми

**ВИСНОВОК**

В ході виконання лабораторної роботи було створено та протестовано роботи пулу потоків у форматі “фастфуду” із замовленнями, касирами та кухарями. Симуляція виконувалася із випадковими інтервалами затримок 5-10с, із 3-ма касирами та 4-ма кухарями. Також компіляція та запуск програми був здійснений у release x64 моді середовища visual studio 2022. Варто зазначити, що в резульаті роботи програми було зафіксовано 9 створених потоків: із них 7 - робітники та 2 службові

Посилання на репозиторій із матеріалами - [GitHub](https://github.com/anton265463/Labs_Parellel_Computing)

Лістинг коду:

#include <queue>

#include <vector>

#include <thread>

#include <mutex>

#include <random>

#include <condition\_variable>

#include <chrono>

#include <atomic>

#include <map>

#include <iostream>

using namespace std;

using namespace chrono;

const int MAX\_QUEUE\_SIZE = 5;

const int MAX\_ORDER\_ID = 20;

const vector<string> menu = {

"Big Mac", "French Fries", "Cheeseburger", "McChicken",

"Chicken McNuggets", "Filet-O-Fish", "Apple Pie", "McFlurry"

};

const vector<string> cashiers = { "Amy", "Markus", "Jane" };

const vector<string> cooks = { "John", "Marie", "Paul", "Abby" };

struct Order {

int id;

string name;

string cashier;

string cook;

string status;

system\_clock::time\_point created;

};

queue<Order> orderQueue;

vector<Order> rejectedOrders;

vector<Order> doneOrders;

mutex mtxQueue, mtxDone, mtxCout;

condition\_variable cvQueue;

atomic<int> orderId{ 1 };

atomic<bool> simulationDone{ false };

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> menuDist(0, menu.size() - 1);

uniform\_int\_distribution<> cookInterval(5, 10);

uniform\_int\_distribution<> processInterval(5, 10);

uniform\_int\_distribution<> orderInterval{ 1, 5 };

atomic<int> totalThreadsCreated{ 0 };

vector<steady\_clock::time\_point> queueFullTimestamps;

atomic<int> maxQueueDuration{ 0 };

atomic<int> minQueueDuration{ 100 };

void printInterface() {

lock\_guard<mutex> lock(mtxCout);

system("cls");

// перевірка роботи випадкового присвоєння часових інтервалів

// cout << "cookInterval =" << orderInterval(gen) << endl;

// cout << "processInterval =" << processInterval(gen) << endl;

// cout << "orderInterval =" << orderInterval(gen) << endl;

// ------------------------ черга замовлень ----------------------

cout << "=== Queue ===" << endl;

{

lock\_guard<mutex> lock(mtxQueue);

queue<Order> temp = orderQueue;

int i = 1;

while (!temp.empty()) {

const auto& o = temp.front();

cout << i++ << ") " << o.name << " (ID: " << o.id << ", Cashier: " << o.cashier << ")" << endl;

temp.pop();

}

if (orderQueue.empty()) cout << "(Empty)" << endl;

}

// -------------------------------------------------------------

// ------------------------ відкинуті замовлення ---------------

if (!rejectedOrders.empty()) {

cout << "\n--- Rejected Orders ---" << endl;

for (const auto& o : rejectedOrders)

cout << o.name << " (rejected by " << o.cashier << ")" << endl;

}

// -------------------------------------------------------------

// ---------------------- робота на кухні ----------------------

cout << "\n=== Kitchen In-Progress ===" << endl;

for (const auto& cook : cooks) {

cout << cook << ":\t";

bool found = false;

for (const auto& o : doneOrders)

if (o.cook == cook && o.status == "in process") {

cout << o.name << " (ID: " << o.id << ")";

found = true;

break;

}

if (!found) cout << "(free)";

cout << endl;

}

// -------------------------------------------------------------

// ---------------------- виконані замовлення -------------------

cout << "\n=== Completed Orders ===" << endl;

for (const auto& o : doneOrders)

if (o.status == "done")

cout << "ID: " << o.id << " | " << o.name << " | Cashier: " << o.cashier << " | Cook: " << o.cook << endl;

// -------------------------------------------------------------

cout << endl;

}

void cashierThread(const string& cashierName) {

totalThreadsCreated++;

while (!simulationDone) {

this\_thread::sleep\_for(seconds(orderInterval(gen)));

Order order;

order.name = menu[menuDist(gen)];

order.cashier = cashierName;

order.created = system\_clock::now();

int newId = orderId++;

if (newId > MAX\_ORDER\_ID) {

simulationDone = true;

cvQueue.notify\_all();

break;

}

order.id = newId;

order.status = "pending";

{

lock\_guard<mutex> lock(mtxQueue);

if (orderQueue.size() >= MAX\_QUEUE\_SIZE) {

rejectedOrders.push\_back(order);

auto now = steady\_clock::now();

queueFullTimestamps.push\_back(now);

if (queueFullTimestamps.size() > 1) {

int duration = duration\_cast<milliseconds>(now - queueFullTimestamps[queueFullTimestamps.size() - 2]).count();

maxQueueDuration = max(maxQueueDuration.load(), duration);

minQueueDuration = min(minQueueDuration.load(), duration);

}

}

else {

orderQueue.push(order);

cvQueue.notify\_one();

}

}

printInterface();

}

}

void cookThread(const string& cookName) {

totalThreadsCreated++;

while (true) {

unique\_lock<mutex> lock(mtxQueue);

cvQueue.wait(lock, [] { return !orderQueue.empty() || simulationDone; });

if (simulationDone && orderQueue.empty())

return;

Order order = orderQueue.front();

orderQueue.pop();

order.cook = cookName;

order.status = "in process";

{

lock\_guard<mutex> doneLock(mtxDone);

doneOrders.push\_back(order);

}

lock.unlock();

printInterface();

this\_thread::sleep\_for(seconds(cookInterval(gen)));

{

lock\_guard<mutex> doneLock(mtxDone);

for (auto& o : doneOrders)

if (o.id == order.id)

o.status = "done";

}

printInterface();

}

}

int main() {

srand(time(nullptr));

int cashiersNum = 3;

int cookNum = 4;

thread cashierThreads[3], cookThreads[4]; // створення масву потоків робітників

// ------------------------- Статистика опрацьованих засовлень робітниками ------------------

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

cashierThreads[i] = thread(cashierThread, cashiers[i]); // створення потоків касирів

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

cookThreads[i] = thread(cookThread, cooks[i]); // створення потоків кухарів

for (auto& t : cashierThreads) t.join(); // очікування завершення роботи касирів

for (auto& t : cookThreads) t.join(); // очікування завершення роботи кухарів

cout << "\n=== Simulation completed. Final statistics ===" << endl;

int totalDone = count\_if(doneOrders.begin(), doneOrders.end(), [](const Order& o) {

return o.status == "done";

});

cout << "Total completed orders: " << totalDone << endl;

cout << "Rejected orders: " << rejectedOrders.size() << endl;

// ------------------------- Статистика опрацьованих засовлень робітниками ------------------

cout << "\n=== Cashier Statistics ===" << endl;

map<string, int> cashierStats;

for (const auto& o : doneOrders) // перебір усіх завершених замовлень

if (o.status == "done")

cashierStats[o.cashier]++; // збільшуємо лічильник замовлень для відповідного касира

for (const auto& p : cashierStats) // виводимо на екран ім’я касира та кількість замовлень

cout << p.first << ": " << p.second << " orders" << endl;

cout << "\n=== Cook Statistics ===" << endl;

map<string, int> cookStats;

for (const auto& o : doneOrders)

if (o.status == "done")

cookStats[o.cook]++;

for (const auto& p : cookStats)

cout << p.first << ": " << p.second << " orders" << endl;

// ---------------------------------------------------------------------------------------

// ---------------------- Статистика заповнення черги ------------------------------------

if (maxQueueDuration == 0) minQueueDuration = 0;

cout << "\n=== Queue Fill Time Statistics ===" << endl;

cout << "Max duration queue was full: " << maxQueueDuration.load() << " milliseconds" << endl;

cout << "Min duration queue was full: " << (minQueueDuration.load() == INT\_MAX ? 0 : minQueueDuration.load()) << " milliseconds" << endl;

// ----------------------------------------------------------------------------------------

cout << "\nTotal threads created: " << totalThreadsCreated.load() +2 << endl;

return 0;

}