Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Навчально-науковий комплекс «Інститут прикладного системного аналізу»

Кафедра системного проектування

Лабораторна робота № 1

з курсу «Операційні системи»

Виконала студентка

Групи ДА-02

Лесечко О. Р

Київ 2022

**Завдання:**

Розробити вирішення класичної задачі синхронізації про «філософів, що обідають» за допомогою взаємо виключення (WIN32).

**Опис завдання:**

За круглим столом сидить п'ять філософів, які не спілкуються один з одним. Перед кожним стоїть одна тарілка спагеті, а також на столі лежить п'ять виделок, по одній зліва і по одній справа від тарілки. Кожен філософ в певний момент часу може або їсти, або думати. Щоб поїсти дуже довгі макарони, філософу під час їжі потрібно мати дві виделки (альтернативний варіант задачі описує рис і китайські палички).

Коли філософ їсть, він бере дві виделки зі столу, коли поїв — кладе виделки на стіл і починає думати. Через деякий час процес повторюється.

В результаті цього за столом може виникнути ситуація, при якій філософи почали їсти і кожен взяв тільки одну виделку. Оскільки всі філософи тримають одну виделку, і чекають на іншу, це призводить до взаємного блокування. Навіть якщо кожен філософ буде класти на стіл взяту раніше одну виделку, після неможливості взяти іншу виделку через деякий час, а потім знову пробуватиме взяти спочатку одну, а потім іншу, це може також не допомогти їм пообідати, якщо вони повторно братимуть виделки одночасно.

**Рішення:**

Пронумеруємо філософів від 1 до 5.

Після цього, введемо правило, що філософ має право брати спочатку виделку з меншим номером, і тільки після цього може брати виделку з більшим.

Таким чином, якщо чотири філософи візьмуть виделки з меншим номером, на столі залишиться вільною тільки виделка 5. Філософ що знаходиться між 5 та 1 не зможе її взяти, бо він мусить спочатку взяти 1, яка вже зайнята, і тому її зможе взяти філософ що знаходиться між 5 та 4, і таким чином наїстись.

Цей розв'язок був запропонований Едсгером Дейкстрою.

**Код програми:**

#define \_CRT\_SECURE\_NO\_WARNINGS // для работы localtime

#include <iostream>

#include <chrono>

#include <thread>

#include <mutex>

#include <semaphore>

#include <random>

// Количество философов

const int N = 5;

// Номер левого соседа id

#define LEFT (id + N - 1) % N

// Номер правого соседа id

#define RIGHT (id + 1) % N

// класс философа, реализация всех активностей философа

class Philosopher {

public:

enum {

THINKING = 0, // философ думает

EATING = 2 // философ кушает

};

/\* это циклический процесс, и каждый раз,

проходя свой цикл, он производит определенную порцию информации,

которую должен обработать философ \*/

void philosophicalProcess(int id) {

while (true) {

think(id); // думать

takeForks(id); // взять вилки

eat(id); // кушать

put\_forks(id); // положить вилки

}

}

// заполнить массив семафоров значениями

void populateArraySemaphores() {

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

s[i] = new std::binary\_semaphore{ 0 };

}

}

private:

std::mutex m\_;

std::mutex mutex\_;

// время

tm\* time\_;

// массив состояний семафоров

std::binary\_semaphore\* s[N];

// массив состояний философов

int state[5] = { 0, 0, 0, 0, 0 };

// вектор имен философов

std::vector<std::string> names\_{ "ARISTOTLE", "SOCRATES", "PLATO", "CONFUCIUS", "DEVTEROV" };

/\* генерация случайного числа задержки

@param min минимальное число с какого рандомится число

@param max максимальное число до какого рандомится число

@returns случайное число задержки \*/

int generationRandomSleep(int min, int max) {

static std::mt19937 rnd(std::time(nullptr));

return std::uniform\_int\_distribution<>(min, max)(rnd);

}

/\* фунція для знаходження точного часу \*/

tm\* nowTime() {

time\_t now = time(NULL);

struct tm\* tm\_struct = localtime(&now);

return tm\_struct;

}

// функция проверки соседей едят они или нет

void check(int id) {

if (state[LEFT] != EATING && state[RIGHT] != EATING) {

state[id] = EATING;

s[id]->release();

}

}

// осуществление мыслительного процесса философа

void think(int id) {

{

std::lock\_guard<std::mutex> lk(m\_);

time\_ = nowTime(); // время в данный момент

std::cout << "> id thread = " << std::this\_thread::get\_id() << "\n";

std::cout << " Time: " << time\_->tm\_hour << ":" << time\_->tm\_min

<< ":" << time\_->tm\_sec << "\n";

std::cout << " The " << names\_[id] << " (" << id + 1<< ") begins to think\n\n";

}

// задержка или время работы барбера

int duration = generationRandomSleep(5000, 10000);

std::this\_thread::sleep\_for(std::chrono::milliseconds(duration));

time\_ = nowTime(); // время в данный момент

std::cout << "> id thread = " << std::this\_thread::get\_id() << "\n";

std::cout << " Time: " << time\_->tm\_hour << ":" << time\_->tm\_min

<< ":" << time\_->tm\_sec << "\n";

std::cout << " The " << names\_[id] << " (" << id + 1 << ") ends to think\n\n";

}

// процесс взять вилки

void takeForks(int id) {

mutex\_.lock();

check(id); // проверить философа, на возможнось приступить кушать

mutex\_.unlock();

s[id]->acquire();

}

// функция обеда

void eat(int id) {

{

std::lock\_guard<std::mutex> lk(m\_);

time\_ = nowTime(); // время в данный момент

std::cout << "> id thread = " << std::this\_thread::get\_id() << "\n";

std::cout << " Time: " << time\_->tm\_hour << ":" << time\_->tm\_min

<< ":" << time\_->tm\_sec << "\n";

std::cout << " The " << names\_[id] << " (" << id + 1 << ") begins to eat\n\n";

}

// задержка или время работы барбера

int duration = generationRandomSleep(5000, 10000);

std::this\_thread::sleep\_for(std::chrono::milliseconds(duration));

time\_ = nowTime(); // время в данный момент

std::cout << "> id thread = " << std::this\_thread::get\_id() << "\n";

std::cout << " Time: " << time\_->tm\_hour << ":" << time\_->tm\_min

<< ":" << time\_->tm\_sec << "\n";

std::cout << " The " << names\_[id] << " (" << id + 1 << ") ends to eat\n\n";

}

// процесс положить вилки

void put\_forks(int id) {

mutex\_.lock();

state[id] = THINKING; // статус философа поменять на думать

check(LEFT); // проверить левого сосоеда, на возможнось приступить кушать

check(RIGHT); // проверить левого сосоеда, на возможнось приступить кушать

mutex\_.unlock();

}

};

int main() {

Philosopher philosopher;

philosopher.populateArraySemaphores();

std::thread t1(&Philosopher::philosophicalProcess, &philosopher, 0);

std::thread t2(&Philosopher::philosophicalProcess, &philosopher, 1);

std::thread t3(&Philosopher::philosophicalProcess, &philosopher, 2);

std::thread t4(&Philosopher::philosophicalProcess, &philosopher, 3);

std::thread t5(&Philosopher::philosophicalProcess, &philosopher, 4);

t1.join();

t2.join();

t3.join();

t4.join();

t5.join();

return 0;

}

**Результат роботи:**

На скріншоті ми бачимо роботу програми.

* В 22:14:41 ми бачимо, що одразу 5 філософів почали думати.
* В 22:14:45 перший філософ закінчує думати і починає їсти.
* В 22:14:47 четвертий філософ закінчує думати і починає їсти, також закінчує думати другий філософ, але одна з виделок зайнята тому він чекає (голодує).

…

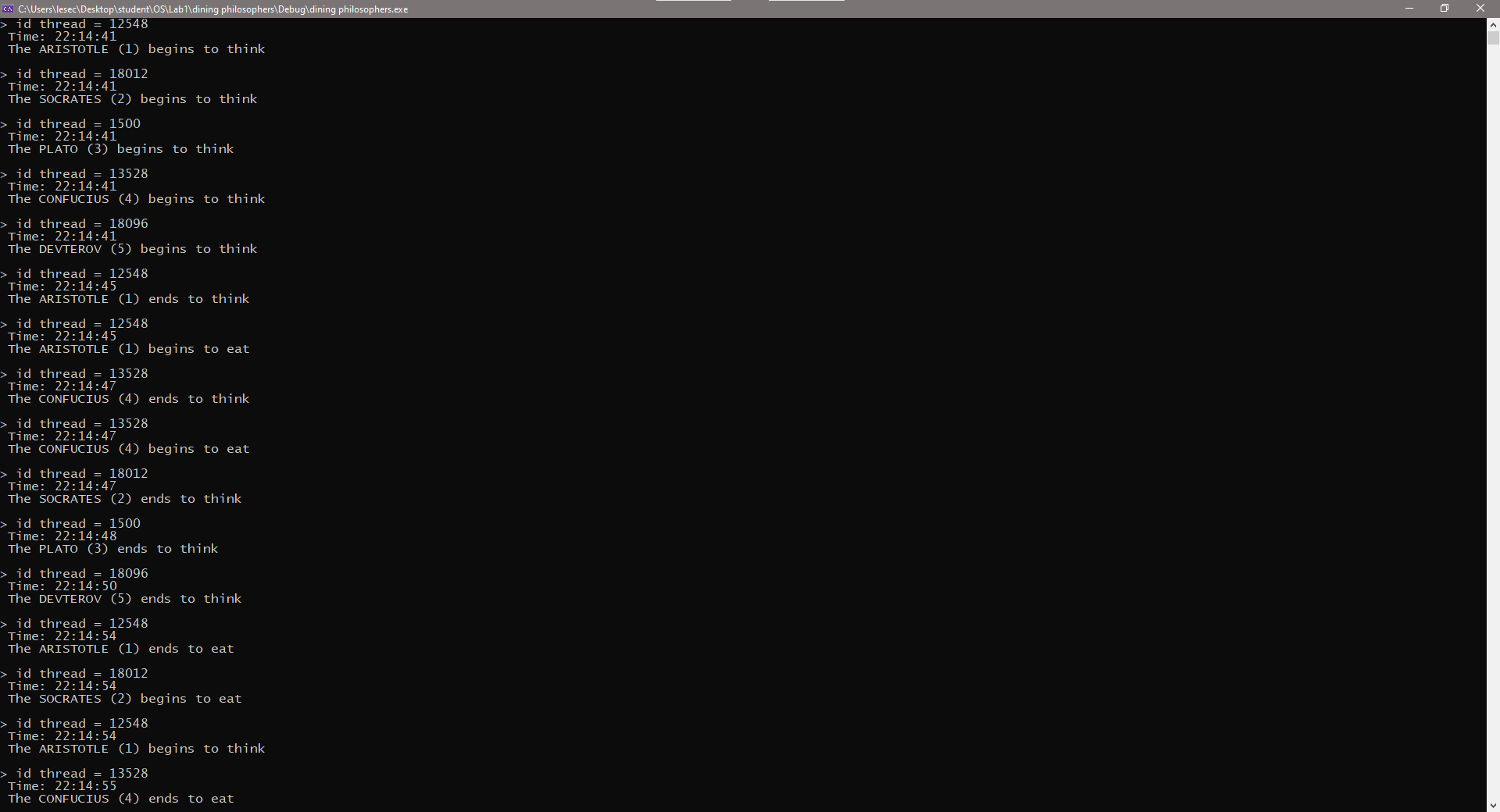


Рис. 1 Результат роботи

**Висновки:**

У ході виконання даної лабораторної роботи я реалізовувала вирішення проблемної задачі про «філософів, що обідають», що ставила перед нами завдання взаємо виключень, при якому підчас звертання одного із процесів до даних які спільно використовуються (розділяються), усім іншим процесам звертання до цих даних заборонено.