

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 «Сплячий барбер» за допомогою взаємо виключення (UNIX).

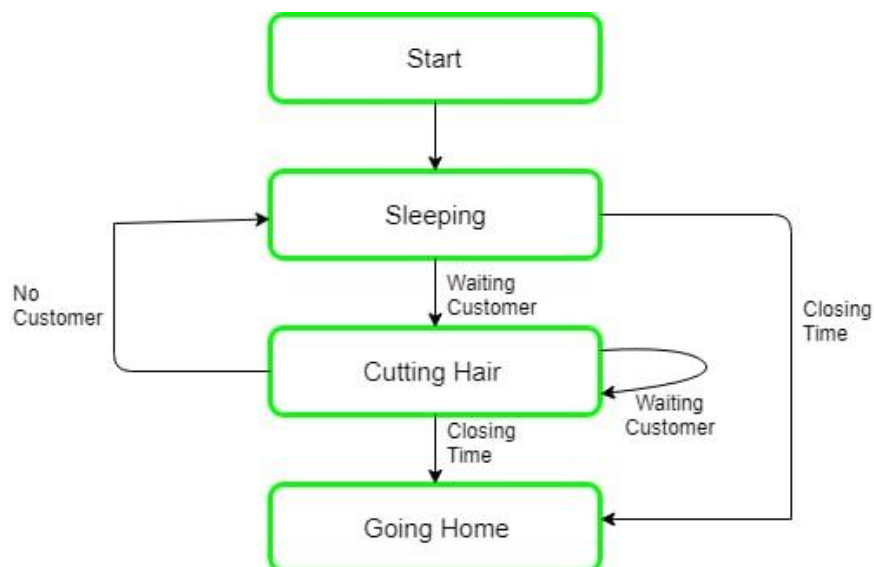
Опис завдання:

Аналогія заснована на гіпотетичній перукарні з одним перукарем. У перукаря є одне робоче місце та приймальня з кількома стільцями. Коли перукар закінчує підстригати клієнта, він відпускає клієнта і потім йде в приймальню, щоб подивитися, чи є клієнти, що чекають там. Якщо вони є, він запрошує одного з них та стриже його. Якщо клієнтів немає, він повертається до свого крісла і спить у ньому.

Кожен клієнт, що приходить, дивиться на те, що робить перукар. Якщо перукар спить, то клієнт будить його і сідає у крісло. Якщо перукар працює, то клієнт йде до приймальні. Якщо в приймальні є вільне стілець, клієнт сідає і чекає своєї черги. Якщо вільного випорожнення немає, то клієнт йде.

Рішення:

Існує кілька можливих рішень цієї проблеми. Основний елемент кожного з рішень – м'ютекс – механізм, який гарантує, що змінити стан (state) у цей час може лише один із учасників. Перукар повинен захопити м'ютекс перед тим, як перевірити клієнтів, і звільнити його, коли він починає або спати, або працювати. Клієнт повинен захопити той же м'ютекс, перш ніж увійти до перукарні, і звільнити його, як тільки він займе місце або в приймальні, або у перукаря



Код програми:

```
#define _CRT_SECURE_NO_WARNINGS // для работы localtime

#include <iostream>
#include <chrono>
#include <thread>
#include <mutex>
#include <random>

/* это примитив синхронизации, который можно использовать
   для блокировки потока или нескольких потоков одновременно,
   пока другой поток не изменит общую переменную(условие) и не
   уведомит об этом condition_variable. */
std::condition_variable cv;

/* этот мьютекс используется для условной переменной cv */
std::mutex m_;

/* общая переменная для потока */
bool workBarber_ = true;

/* класс Runnable, служит общим представлением барбера и клиента */
class Runnable {
public:
    /* это циклический процесс, и каждый раз,
       проходя свой цикл, он производит определенную порцию информации,
       которую должен обработать барбер */
    void barber() {
        while(true){
            {
                std::unique_lock<std::mutex> lk(m_);

                if (customerCnt_ == 0) {
                    workBarber_ = false;    // барбер не работает
                    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 << " barber falls asleep\n\n";

                    cv.wait(lk, [] {return workBarber_; }); // спать пока клиент не разбудит
                }
            }
        }

        std::unique_lock<std::mutex> lk(m_);

        customerCnt_--;    // одним ожидающим клиентом меньше
        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 barber begins to work\n\n";

        lk.unlock();      // освободить мьютекс
    }
};
```

```

    /* задержка или время работы барбера */
    int duration = generationRandomSleep(3000, 6000);
    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 barber ends to work\n\n";

}
}
}
/* это циклический процесс, и каждый раз,
   проходя свой цикл, он производит определенную порцию информации,
   которую должен обработать клиент */
void customer() {
    while (true) {
        /* задержка или время прихода барбера */
        int duration = generationRandomSleep(5000, 20000);
        std::this_thread::sleep_for(std::chrono::milliseconds(duration));
        {
            if (customerCnt_ >= numberSeats_) { // проверка на свободные места
                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 << "   customer leaves because seats are full\n\n";
            }
            else {
                customerCnt_++; // одним ожидающим клиентом меньше
                if (!workBarber_) { // проверка, спит ли барбер
                    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 << "   customer woke the barber\n\n";

                    workBarber_ = true; // барбер работает
                    cv.notify_one(); // разбудить барбера
                }
                else {
                    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 client took an empty seat\n\n";
                }
            }
        }
    }
}

```

```

    }
    }
}

private:
    /* время */
    tm* time_;

    /* количество ожидающих клиентов */
    int customerCnt_ = 0;

    /* количество мест */
    int numberSeats_ = 2;

    /* 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);
    }

    /* функция для нахождения точного часу */
    tm* nowTime()
    {
        time_t now = time(NULL);
        struct tm* tm_struct = localtime(&now);
        return tm_struct;
    }
};

int main() {
    Runnable runnable;

    std::thread t1(&Runnable::barber, &runnable);
    std::thread t2(&Runnable::customer, &runnable);
    std::thread t3(&Runnable::customer, &runnable);
    std::thread t4(&Runnable::customer, &runnable);

    t1.join();
    t2.join();
    t3.join();
    t4.join();

    return 0;
}

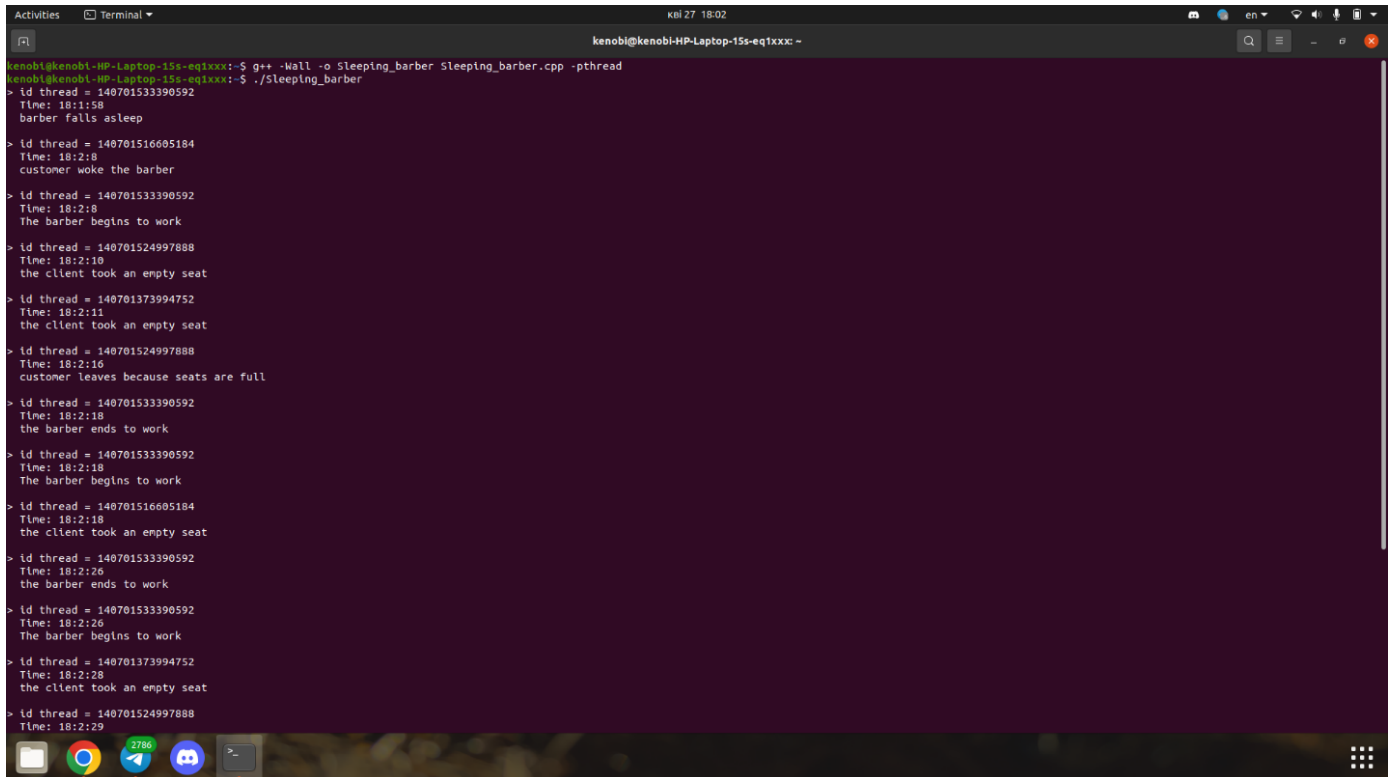
```

Результат роботи:

На скріншоті ми бачимо роботу програми.

- В 18:01:58 ми бачимо, що барбер заснув, тому що клієнтів немає.
- В 18:02:08 приходить клієнт, він бачить що барбер спить, тому будить його, у той же час барбер прокидається і починає працювати.
- В 18:02:10 приходить клієнт, бачить, що барбер працює, тому займає вільне місце.
- В 18:02:11 приходить клієнт, бачить, що барбер працює, тому займає вільне місце.
- В 18:02:16 приходить клієнт, бачить, що вільних місць немає, тому йде звідси.
- В 18:02:18 барбер закінчує працювати, і так як в черзі є клієнти одразу продовжує працювати далі.

...



```
kenobi@kenobi-HP-Laptop-15s-eq1xxx:~$ g++ -Wall -o Sleeping_barber Sleeping_barber.cpp -pthread
kenobi@kenobi-HP-Laptop-15s-eq1xxx:~$ ./Sleeping_barber
> id thread = 140701533390592
Time: 18:1:58
barber falls asleep

> id thread = 140701516605184
Time: 18:2:8
customer woke the barber

> id thread = 140701533390592
Time: 18:2:8
The barber begins to work

> id thread = 140701524997888
Time: 18:2:10
the client took an empty seat

> id thread = 140701373994752
Time: 18:2:11
the client took an empty seat

> id thread = 140701524997888
Time: 18:2:16
customer leaves because seats are full

> id thread = 140701533390592
Time: 18:2:18
the barber ends to work

> id thread = 140701533390592
Time: 18:2:18
The barber begins to work

> id thread = 140701516605184
Time: 18:2:18
the client took an empty seat

> id thread = 140701533390592
Time: 18:2:26
the barber ends to work

> id thread = 140701533390592
Time: 18:2:26
The barber begins to work

> id thread = 140701373994752
Time: 18:2:28
the client took an empty seat

> id thread = 140701524997888
Time: 18:2:29
```

Рис. 1 Результат роботи

Висновки:

У ході виконання даної лабораторної роботи я реалізовувала вирішення проблемної задачі про сплячого барбера, що ставила перед нами завдання про синхронізацію потоків, тобто запобігання одночасному виконанню деякого критичного набору операцій. Головним було використання у контексті задачі «взаємо виключень», тобто м'ютексів. Результат роботи описаний вручну, а до коду додані коментарі.