Домашнее задание. Параллельные алгоритмы.

Егорова Алёна

Домашнее задание 3

Задание 2.

А) Решение с условными переменными. Для решения буду использовать массив флажков. $left$ имеет индекс 0, $right$ - 1. Тогда если в массиве с соответствующим индексом стоит 0, значит, нога с этим индексом шага ещё не сделала, и другая нога засыпает.

Соответствующий код:

```
std::condition_variable not_done;
int flags[2] = {false, true};
std::mutex mtx;

void step(std::string s){

    while (true){
         std::unique_lock<std::mutex> lock(mtx);
         while(!flags[abs((s == "right") - 1)]) {
             not_done.wait(lock);
         }
         std::cout << s << std::endl;
         flags[s == "right"] = true;
         flags[abs((s == "right") - 1)] = false;
         not_done.notify_all();
    }
}

int main(){
    std::thread thread1(step, "left");
    std::thread thread2(step, "right");
    thread1.join();
    thread2.join();
}
```

Б) Решение с семафорами. Заведем 2 семафора. *count* для правой ноги инициализируем единицей. Как только поток входит в функцию он проверяет, шагнула ли другая нога. Если шагнула, то шагает он, если нет, то поток засыпает, пока другая нога не шагнёт.

Соответствующий код:

```
class Semaphore {

public:
    Semaphore(int count = 0): count(count) {}

    void wait() {
        std::unique_lock<std::mutex> lock(mtx);
        while (count.load() == 0) {
            signal_cv.wait(lock);
```

```
        }
        count.fetch_sub(1);
    }

    void signal() {
        std::lock_guard<std::mutex> lock(mtx);
        count.fetch_add(1);
        signal_cv.notify_all();
    }
private:
    std::condition_variable signal_cv;
    std::atomic<int> count;
    std::mutex mtx;
};

Semaphore left;
Semaphore right(1);

void step(std::string s){
    while (true){
        if (s == "left") {
            right.wait();
            std::cout << s << std::endl;
            left.signal();
        }
        else {
            left.wait();
            std::cout << s << std::endl;
            right.signal();
        }
    }
}

int main(){;
    std::thread left_leg(step, "left");
    std::thread right_leg(step, "right");

    left_leg.join();
    right_leg.join();
}
```

Задание 3.

Представим такую ситуацию. Пусть у нас один поток прочитал условие *while*, далее другой поток меняет значение флага на *true*, затем отправляет сигнал. Но только после этого первый поток засыпает в *wait*. Таким образом, сигнал отправленный вторым потоком, теряется, а первый поток не проснется никогда. Значит, код не корректен.