Веснин Константин Андреевич БПИ203

Вариант 6

Задача о курильщиках.

Есть три процесса-курильщика и один процесс посредник. Курильщик непрерывно скручивает сигареты и курит их. Чтобы скрутить сигарету, нужны табак, бумага и спички. У одного процесса курильщика есть табак, у второго – бумага, а у третьего – спички. Посредник кладет на стол по два разных случайных компонента. Тот процесс курильщик, у которого есть третий компонент, забирает компоненты со стола, скручивает сигарету и курит. Посредник дожидается, пока курильщик закончит, затем процесс повторяется. Создать многопоточное приложение, моделирующее поведение курильщиков и посредника. При решении задачи использовать семафоры.

В программе был использован рекурсивный параллелизм. Три курильщика создают свои “артефакты” независимо и делают это рекурсивно. Закончил создавать артефакт - принимайся за новый, а на других не смотри.

Но помимо рекурсивного параллелизма в программе также присутствует отношение производителя и потребителя. Брокер производит два артефакта и поставляет их курильщикам, отчасти вмешиваясь в их работу, потому что на время “вмешательства” брокера (то есть в момент, когда он прибавляет к артефактам курильщика свои) нам приходится заморозить работу курильщиков для безопасности данных и корректной работы. Одностороннее обращение говорит именно об этом типе.

Итого, структуру программы можно описать так:

- Курильщики рекурсивно создают каждый свой ресурс: табак, бумагу или спички. Получая третий ресурс от брокера, пробуют покурить.
- Брокер рекурсивно выбирает два рандомных ресурса и вызывает курильщика, который производит третий. Если курильщик сейчас занят, то вызывающий его брокером метод выполнится тогда, когда он освободится. А обратно, если курильщик сейчас на вызове у брокера, он не производит ресурс. Реализовано это с помощью семафор.
- Каждое из происходящих событий выводится в консоль. Доступ в консоль тоже реализован через семафоры. В консоль в каждый момент времени может писать только кто-то один из потоков. Каждый поток пишет в консоль отдельно, однако вывод сообщений согласуется с помощью объекта написанного мною класса ConsoleWriter.
- Программа не предполагает ввода данных. Всё, что требуется - ввести символ и нажать “Enter”, чтобы запустить её, и то же самое, чтобы программа перестала работать.
- Метод, прибавляющий курильщику ресурсы, выданные ему брокером, а также “запускающий” курение, запускается в классе брокера, поэтому брокер не примется за следующего клиента, пока текущий не закончит.

Здесь я читал про виды параллелизма:

http://www.williamspublishing.com/PDF/5-8459-0388-2/part.pdf

Вот так у меня изначально брокер ждал, пока курильщик закончит производство ресурса, а курильщик ждал, пока брокер не передаст ему два не хватающих ресурса. Такая реализация не давала сбоев, когда был использован таймер для плавной работы программы. Но когда я его убрал, то заметил, что либо программа неслабо подвисает, либо брокер отказывается работать. Интерпретировать это можно как непредсказуемое поведение: курильщик может пойти по новой рекурсии раньше, чем брокер даст ему товары.

```
void doInThread() override{
    semaphore->acquire();
    doMoreStaff();
    report("I've done some more resources");
    if (tobaccoCount > 0 && paperCount > 0 && matchesCount > 0)
        smoke();
    semaphore->release();
```

```
    semaphore->acquire();
    paperCount++;
    tobaccoCount++;
    report("I get tobacco and paper from Broker!");
    smoke();
    semaphore->release();
```

Поэтому я исправил программу и сделал semaphore для каждого из ресурсов, а любое изменение ресурса (в + или в -) вынес в отдельный метод. Всё заработало как надо, а также даже если бы прошлый способ работал, этот гораздо эффективнее, так как даёт ограничение на непосредственное изменение одной и той же переменной двумя потоками одновременно.

```
void changeTobacco(int dif) {
    semaphoreTobacco->acquire();
    tobaccoCount += dif;
    semaphoreTobacco->release();
}
void changePaper(int dif) {
    semaphorePaper->acquire();
    paperCount += dif;
    semaphorePaper->release();
}
void changeMatches(int dif) {
    semaphoreMatches->acquire();
    matchesCount += dif;
    semaphoreMatches->release();
}
```

А так реализован класс ConsoleWriter. Один и тот же объект этого класса есть у всех курильщиков и у брокера, через него они и выводят всё, что хотят вывести.

```
class ConsoleWriter {
private:
    std::binary_semaphore* a;
public:
    ConsoleWriter() {
        a = new std::binary_semaphore(1);
    }
    void print(std::string msg) {
        a->acquire();
        std::cout << msg << std::endl;
        a->release();
    }
    ~ConsoleWriter() {}
};
```

Запускаем программу и видим, что данные выводятся слишком быстро и беспорядочно, однако всё работает. Однако в таком виде я программу и оставил, потому что как я убедился, проверить корректную бессбойную работу многопоточности с таймером нельзя. Но я оставил

закомментированные строчки с таймером, раскомментировав которые можно насладиться использованием программы.

Сейчас же инструкция по работе: нажать клавишу Enter и запустить и почти сразу нажать клавишу Enter и выйти из неё. А затем просмотреть окно вывода.

MultiThreading.cpp

```
std::thread smokerTobaccoThread([smokerTobacco]() {smokerTobacco->doInThread(); });
//std::this_thread::sleep_for(std::chrono::seconds(4));
std::thread smokerPaperThread([smokerPaper]() {smokerPaper->doInThread(); });
//std::this_thread::sleep_for(std::chrono::seconds(4));
std::thread smokerMatchesThread([smokerMatches]() {smokerMatches->doInThread(); });
//std::this_thread::sleep_for(std::chrono::seconds(2));
```

Smoker.cpp

```
50          // std::this_thread::sleep_for(std::chrono::seconds(16));
```

Broker.cpp

```
52          // std::this_thread::sleep_for(std::chrono::second(16));
```