## +Синхронизация процессов и потоков: необходимость. Примитивы синхронизации: семафоры, мьютексы, условные переменные, барьеры, мониторы. Классические задачи синхронизации: «обедающие философы», «спящий парикмахер».

//первая часть в предыдущем вопросе

**Задача «Обедающие философы»**

•Если задача «Читатели–Писатели» помогает демонстрировать методы параллельного и исключительного доступа к одному общему ресурсу, то задача «Обедающие философы» позволяет рассмотреть способы доступа нескольких потоков к нескольким разделяемым ресурсам.

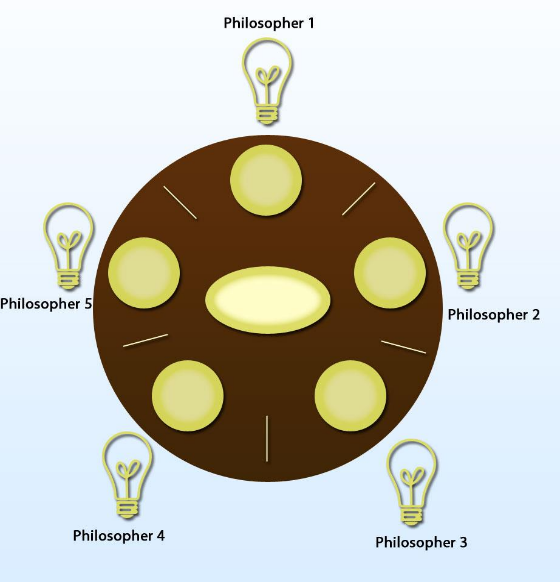
•Исходная формулировка задачи впервые предложена Э. Дейкстрой.

Формулировка задачи

• Представляется ситуация, в которой пять философов располагаются за круглым столом.

• При этом философы либо размышляют, либо кушают.

• Для приема пищи в центре стола большое блюдо с неограниченным количеством спагетти, и тарелки, по одной перед каждым философом.

• Предполагается, что поесть спагетти можно только с использованием двух вилок. Для этого на столе располагается ровно пять вилок – по одной между тарелками философов.

• Для того, чтобы приступить к еде, философ

должен взять вилки слева и справа (если

они не заняты), наложить спагетти из

большого блюда в свою тарелку, поесть, а

затем обязательно положить вилки на

свои места для их повторного

использования (проблема чистоты вилок в

задаче не рассматривается).

• Нетрудно заметить, что в данной задаче

философы представляют собой потоки, а

вилки – общие разделяемые ресурсы.

Изображение выглядит как текст

Автоматически созданное описание

• Можно увидеть, что данное решение может приводить к тупиковым ситуациям – например, когда все философы одновременно проголодаются и каждый из них возьмет свои левые вилки.

• В результате правые вилки для всех философов окажутся занятыми и философы перейдут к бесконечному ожиданию (сложно выявить подобную ситуацию при помощи тестов; кроме того, подобную ошибочную ситуацию сложно повторить при повторных запусках программы).

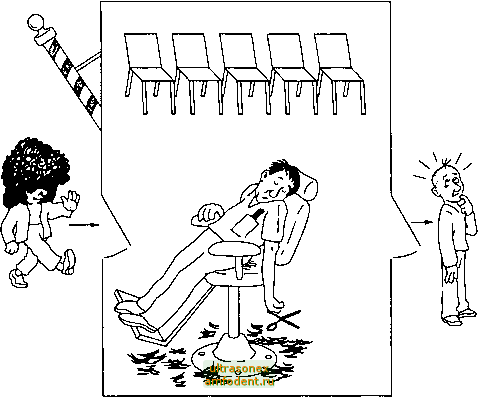
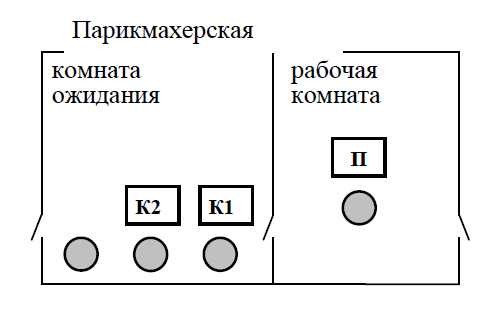
Изображение выглядит как текст

Автоматически созданное описание

**Задача «Спящий парикмахер»**

• Данная задача также в числе широко используемых примеров для демонстрации проблем синхронизации.

• На примере этой задачи можно показать методы последовательного доступа к набору разделяемых ресурсов и рассмотреть организацию вычислений в соответствии со схемой «клиент–сервер».

• В парикмахерской имеется два помещения: комната ожидания, в которой ограниченное количество мест, и рабочая комната с единственным креслом, в котором располагается обслуживаемый клиент.

• Посетители заходят в парикмахерскую – если комната ожидания заполнена, то поворачиваются и уходят; иначе занимают свободные места и засыпают, ожидая своей очереди к парикмахеру.

• Парикмахер, если есть клиенты, приглашает одного из них в рабочую комнату и подстригает его.

• После стрижки клиент покидает парикмахерскую, а парикмахер приглашает следующего посетителя и т. д.

• Если клиентов нет (комната ожидания пуста), парикмахер садится в свое рабочее кресло и засыпает. Будит его очередной появляющийся посетитель парикмахерской.

• В данной задаче ресурсами являются места ожидания и рабочее кресло.

• Потоки-клиенты должны получать эти ресурсы строго последовательно: сначала посетитель должен найти место в комнате ожидания и только затем занять очередь к

парикмахеру.

• При этом предоставление рабочего кресла для обслуживания производит специальный процесс - парикмахер.

• В этом плане, парикмахера можно интерпретировать как сервер, предоставляющий требуемый сервис.

**События**

• **Client** – событие, означающее, что есть ожидающие посетители; событие объявляется каждый раз при появлении нового клиента; данное событие пробуждает спящего парикмахера, заснувшего при отсутствии клиентов;

• **Barber** – событие при освобождении парикмахера; по данному событии пробуждается один из ожидающих клиентов, который и переходит в рабочую комнату для обслуживания;

• **Service** – событие при завершении обслуживания очередного клиента; клиент может покинуть парикмахерскую.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

## +Средства распараллеливания языка C++: std::thread, std::mutex. Параллельные алгоритмы.

Начиная со стандарта C++11 в стандартную библиотеку языка C++ была включена библиотека многопоточности, которая содержит потоки (thread), взаимные исключения (mutex) и условные переменные (condition\_variable). Также появилась поддержка асинхронного программирования. В стандартах C++14, C++17 и C++20 возможности многопоточного и асинхронного программирования были расширены.

Перед созданием потоков необходимо подключить заголовочный файл thread

(#include <thread>) и создать объект потока.

std::thread myThread; //создает объект потока, не создавая поток выполнения

Далее запускаем поток на выполнение

myThread = std::thread(foo); //конструктор создает поток и связывает его с потоком выполнения, **foo** – функция, которую выполняет поток.

Если необходимо дождаться завершения «дочернего» потока, то используют функцию **join** (вызов данной функции блокирует текущий поток до завершения «дочернего»). В противном случае необходимо отсоединить созданный поток от «родительского» функцией **detach** (переводит созданный поток «в свободное плавание», «дочерний» поток завершается при завершении «родительского»).

Второй вариант используется редко.

Чтобы обратиться внутри функции к потоку, который её выполняет, можно использовать std::this\_thread.

*Пример 1*. В следующем примере создается один поток, который выводит сообщение о своем старте, затем выводит свой id 10 раз, после каждого вывода засыпает на 1 секунду (имитация сложных вычислений) и выводит сообщение о завершении. Главный поток ждет окончания выполнения дочернего и после этого также завершается.

Изображение выглядит как текст

Автоматически созданное описание

*Пример 2*. Модифицируем программу таким образом, чтобы поток выводил свой id N раз. Число N вводится пользователем в главной функции и может быть передано потоку в качестве параметра.

Изображение выглядит как текст

Автоматически созданное описание

Так как потоки выполняются в одном и том же адресном пространстве (имеют общую память), то возникает необходимость конкурентного доступа к разделяемому ресурсу. Если запустить на выполнение программу из примера 2, то можно заметить, что потоки конкурируют за доступ к объекту потокового вывода. В итоге вывод может выглядеть не так, как предполагал программист.

В данном случае вывод нужно поместить в критическую секцию. В таком случае новый поток не сможет начать вывод до того, как предыдущий выведет сообщение целиком.

Наиболее простой механизм – взаимоблокировка потоков (mutex, от mutual exclusion).

Метод lock блокирует мьютекс, метод try\_lock пытается заблокировать мьютекс. Для разблокировки используется метод unlock.

Вызывающий поток владеет мьютексом со времени успешного вызова lock или try\_lock и до момента вызова unlock. Пока поток владеет мьютексом, все остальные потоки не могут получить доступ к ресурсу (блокируются при вызове lock или получают false при вызове try\_lock). Разблокировать мьютекс может только тот поток, который его захватил. Этим мьютекс отличается от бинарного семафора.

Рекомендуется использовать класс lock\_guard, который реализует принцип RAII (Resource acquisition is initialization – получение ресурса есть инициализация) и является «оберткой» для mutex. При создании объекта lock\_guard захватывается мьютекс, переданный ему в конструкторе. В деструкторе же происходит освобождение мьютекса. Также, lock\_guard содержит дополнительный конструктор, который позволяет инициализировать объект lock\_guard с мьютексом, который уже был захвачен. В случае, если нужно иметь возможность разблокировать мьютекс, используйте объект unique\_lock.

Захват мьютекса изменяет его состояние, поэтому мьютекс нужно передавать в функцию потока по ссылке (ref) или делать глобальной переменной.

*Пример 3*. Добавим в предыдущий пример второй дочерний поток и модифицируем его так, чтобы вывод мог осуществлять только один поток в один момент времени.

Изображение выглядит как текст

Автоматически созданное описание