Лабораторная работа

По курсу Методы программирования 3 курс

1 задача

1.1 Задача «Быстрая сортировка»

Постановка задачи

Реализовать параллельную версию алгоритма быстрая сортировка. Схема разбиения должна определяться с помощью `SchemePolicy` классов. Нижеприведенные сигнатуры функций `pquicksort` фиксированы и не подлежат изменению.

```
//pquicksort.hpp
template <class RandomIt, class SchemePolicy>
void pquicksort(RandomIt first, RandomIt last, SchemePolicy&& partition);


template <class RandomIt, class Compare, class SchemePolicy>
void pquicksort(RandomIt first, RandomIt last, Compare comp, SchemePolicy&& partition
);
```

Параллельная версия алгоритма работает лучше, чем последовательная реализация только если размер диапазона превышает определенный порог, который может варьироваться в зависимости от параметров компиляции, платформы или оборудования. Поэкспериментируйте с различными порогами и размерами диапазонов, чтобы увидеть, как изменяется время выполнения. И установите порог на минимальное количество элементов для выполнения многопоточной реализации.

Требования

Сигнатуры функций `pquicksort` фиксированы и не подлежат изменению.

Названия header и source файлов следующие: `pquicksort.hpp`.

Написать unit test'ы для реализованных функций.

Провести замеры времени выполнения функций для сравнения параллельной версии и последовательной.

1.2 Задача «Сортировка пузырьком»

Постановка задачи

Реализовать параллельную версию алгоритма Сортировка пузырьком (метод чет-нечетной перестановки).

Параллельная версия алгоритма работает лучше, чем последовательная реализация только если размер диапазона превышает определенный порог, который может варьироваться в зависимости от параметров компиляции, платформы или оборудования. Поэкспериментируйте с различными порогами и размерами диапазонов, чтобы увидеть, как изменяется время выполнения. И установите порог на минимальное количество элементов для выполнения многопоточной реализации.

Требования

Написать unit test'ы для реализованных функций.

Провести замеры времени выполнения функций для сравнения параллельной версии и последовательной.

1.3 Задача «Сортировка алгоритмом Шелла»

Постановка задачи

Реализовать параллельную версию алгоритма Шелла.

Параллельная версия алгоритма работает лучше, чем последовательная реализация только если размер диапазона превышает определенный порог, который может варьироваться в зависимости от параметров компиляции, платформы или оборудования. Поэкспериментируйте с различными порогами и размерами диапазонов, чтобы увидеть, как изменяется время выполнения. И установите порог на минимальное количество элементов для выполнения многопоточной реализации.

Требования

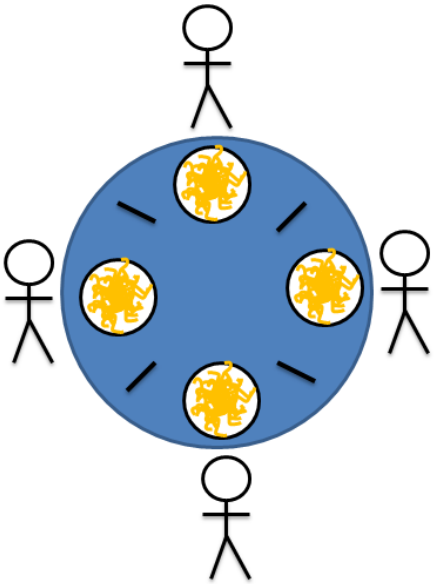
1. Написать unit test'ы для реализованных функций.
2. Провести замеры времени выполнения функций для сравнения параллельной версии и последовательной.

2 задача

2.1 Задача об обедающих философах

Задача об обедающих философах —классический пример, используемый в информатике для иллюстрации проблем синхронизации при разработке параллельных алгоритмов и техник решения этих проблем.

Задача была сформулирована в 1965 году Эдсгером Дейкстрой как экзаменационное упражнение для студентов. В качестве примера был взят конкурирующий доступ к ленточному накопителю. Вскоре задача была сформулирована Энтони Хоаром в том виде, в каком она известна сегодня.



N безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. Вилки лежат на столе между каждой парой ближайших философов.

Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством оставшихся спагетти — подразумевается бесконечный запас. Тем не менее, философ может есть только тогда, когда держит две вилки — взятую справа и слева (альтернативная формулировка проблемы подразумевает миски с рисом и палочки для еды вместо тарелок со спагетти и вилок).

Каждый философ может взять ближайшую вилку (если она доступна) или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.

Вопрос задачи заключается в том, чтобы разработать модель поведения (параллельный алгоритм), при котором ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

Требования

Написать unit test'ы для реализованных классов и функций.

При решении задачи должны применяться паттерны проектирования, полиморфизм, stl алгоритмы.

2.2 Система обслуживания клиентов

Постановка задачи

Напишите программу, которая имитирует способ обслуживания клиентов в офисе. В офисе есть N столов, где клиенты могут быть обслужены одновременно. Клиенты могут войти в офис в любое время. Они берут билет с номером из билетной машины и ждут, пока их номер не станет следующим для обслуживания на одном из столов. Клиенты обслуживаются в порядке их поступления в офис или, точнее, в порядке, указанном в их билете. Каждый раз, когда служба поддержки заканчивает обслуживать клиента, обслуживается следующий клиент. Симуляция должна прекратиться после того, как определенное количество клиентов получат билеты и их обслужат.

Вспомогательные классы

Класс ticketing_machine моделирует простую машину, которая выдает инкрементные номера билетов, начиная с начального, указанного пользователем числа.

Класс customer представляет покупателя, который входит в офис и получает билет из билетной машины.

Класс logger потокобезопасный (thread-safety) класс для логирования сообщений в поток (std::ostream). Вся информация (открытие/закрытие стола, появление нового клиента с номером билета, обслуживание клиента за i-ым столом, размер текущей очереди при изменении, завершение обслуживания клиента за i-ым столом) должна логироваться.

Дополнительная информация

Общее количество клиентов Customer Amount, которые могут посетить офис, задается в конфигурационном файле. Новый клиент заходит в офис, например, каждые 200-500 миллисекунд (т.е. в диапазоне от 200 до 500 мс), получает билет и ждет своей очереди. При обслуживании клиента тратится, например, от 2000 до 3000 мс времени.

Офис перестает принимать клиентов, после того как в него зайдут Customer Amount клиентов, и переходит в режим закрытия. Столы обслуживания работают до тех пор, пока офис не перейдет в режим закрытия, но не раньше, чем будут обслужены все клиенты, находящиеся в очереди.

Требования

Написать unit test'ы для реализованных классов и функций.

При решении задачи должны применяться паттерны проектирования, полиморфизм, stl алгоритмы.

2.3 Проблема спящего парикмахера

Аналогия основана на гипотетической парикмахерской с одним парикмахером. У парикмахера есть одно рабочее место и приёмная с несколькими стульями. Когда парикмахер заканчивает подстригать клиента, он отпускает клиента и затем идёт в приёмную, чтобы посмотреть, есть ли там ожидающие клиенты. Если они есть, он приглашает одного из них и стрижёт его. Если ждущих клиентов нет, он возвращается к своему креслу и спит в нём.

Каждый приходящий клиент смотрит на то, что делает парикмахер. Если парикмахер спит, то клиент будит его и садится в кресло. Если парикмахер работает, то клиент идёт в приёмную. Если в приёмной есть свободный стул, клиент садится и ждёт своей очереди. Если свободного стула нет, то клиент уходит.

Проблема заключается в обеспечении того, чтобы парикмахер работал, когда есть клиенты, и отдыхал, когда клиентов нет.

Требования

Написать unit test'ы для реализованных классов и функций.

При решении задачи должны применяться паттерны проектирования, полиморфизм, stl алгоритмы.