Практическое задание 7. Знакомство с POSIX потоками

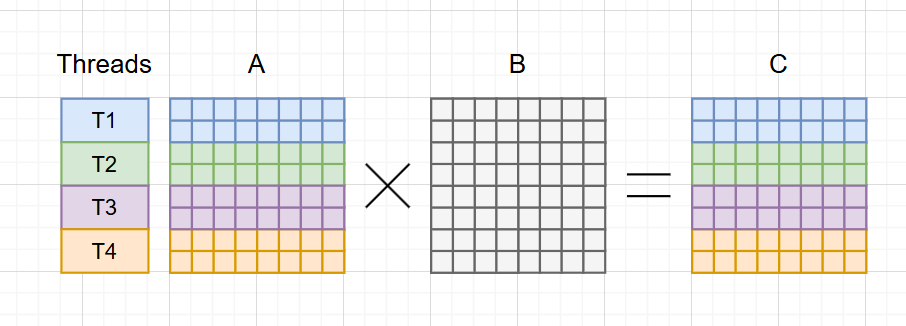
**Требование:** Оформить небольшой отчет в текстовом файлике laba7.md с форматом md (md = markdown), в котором будут приложены скриншоты/графики/пояснения с описанием выполненной работы. Загрузить исходники и laba7.md в GitHub

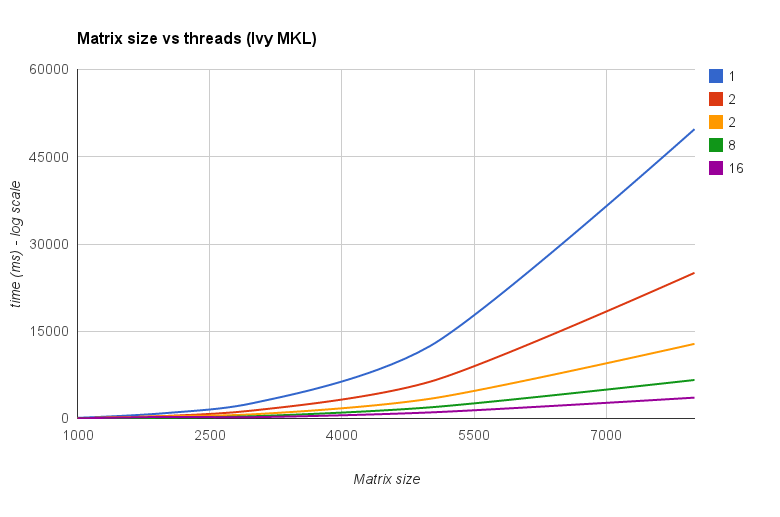
**Оценка 3. Знакомство с pthread:**

1. **Создать поток**Написать программу, которая создает поток с помощью **pthread\_create().** Использовать атрибуты по умолчанию. Родительский и дочерний потоки должны вывести на экран по **5** строк текста.
2. **Ожидание потока**  
   Модифицировать упр.1 так, что родительский поток выводит текст после завершения дочернего потока. Подсказка: **pthread\_join()**
3. **Параметры потока**Модифицировать упр.2 так, что основной поток создает 4 потока, исполняющих одну и ту же функцию. Эта функция должна распечатать последовательность текстовых строк, переданных как параметр. Каждый из созданных потоков должен распечатать различные последовательности строк.
4. **Завершение нити без ожидания**Добавить сон с помощью sleep() в функцию потоков между выводами строк. Спустя две секунды после создания дочерних потоков основной поток должен прервать работу всех дочерних потоков с помощью **pthread\_cancel()**.
5. **Обработать завершение потока**Модифицировать упр. 4 так, чтобы дочерний поток перед завершение распечатывал сообщение об этом. Использовать pthread\_cleanup\_push()
6. **Реализовать простой Sleepsort**Реализовать прикольный алгоритм сортировки **Sleepsort** с асимптотикой O(N) (по времени). Суть алгоритма: на вход подается массив, **пусть будет не более 50 элементов и пусть будет состоять из целочисленных значений**. Для каждого элемента массива создается отдельный поток, в который в качестве аргумента передается значение элемента. Сам поток должен уйти в сон с помощью sleep() или usleep() с параметром равным аргументу потока (значение элемента массива), а после вывести на экран значение.

**Оценка 4. Перемножение матриц:**

**Сделать на оценку 3,** а затем..

1. **Синхронизированный вывод**  
   Модифицируйте программу упр. 5 так, чтобы вывод родительского и дочернего потока был синхронизован: сначала родительский поток выводить первую строку, затем дочерний, затем родительский вторую строку и т.д. Использовать mutex.
2. **Перемножение квадратных матриц NxN**
   1. Написать функцию произведения двух квадратных матриц A и B размером NxN (на выходе получим матрицу C). Исходные матрицы A и B заполнить единицами в **основном потоке с функцией main**. Для матриц размером меньше 5 в **основном потоке** вывести на экран матрицы A, B и C (для проверки правильности работы функции).
   2. С командной строки считать размер матрицы и количество потоков. Распараллелить перемножение матриц разбив матрицу на равные части между потоками в главной функции по следующему принципу: N / threads, например если размер матрицы N = 8, а потоков 4, то каждый из потоков заберет вычисление N/4 = 2 строки и столбца, наглядно: ****
   3. ***Дополнительно*** (не обязательно) Учесть, что размер матриц может быть не кратен числу потоков, например, N = 18, а число потоков 4, тогда необходимо каждому из потоков выдать по 4x4, а оставшиеся 2x2 досчитать в основном потоке.
3. **Время выполнения**

Замерить время выполнения с момента создания потоков (до цикла с pthread\_create) и до завершения работы потоков (после цикла pthread\_join). Позапускать с разным числом потоков и размером матрицы. **Построить график в любой программе** (excel, python, matlab и т.д.) или онлайн, который покажет зависимость времени выполнения от размера матрицы и количества потоков. Количество потоков от 1 до 128 с любым разумным шагом. Размер матрицы не более 2500, шаг размера матрицы на свое усмотрение. Пример графика (цветами кол-во потоков, по оси х размер матрицы, по y время в мс):  


**Уметь пояснить за полученную картинку**

**Оценка 5. Простой односторонний чат:**

**Сделать на оценку и 3 и 4,** а затем..

1. **Производитель-потребитель**Реализовать очередь сообщений (FIFO), которая будет использоваться для обмена сообщениями между потоками «сервера» (потребитель) и «клиента» (производитель).   
   int msgSend(queue \*, char \* msg); // ф-ия производителя  
   int msgRecv(queue \*, char \* buf, size\_t bufsize); // ф-ия потребителя  
     
   msgSend принимает в качестве параметра ASCII строку символов, обрезает ее до 128 символов (если необходимо) и помещает ее в очередь. Если очередь содержит более 10 записей, то msgSend блокируется. Функция возвращает количество переданных символов. msgRecv извлекает полностью первую запись из очереди, обрезая ее до размера bufsize (если необходимо). Если очередь пуста, то функция блокируется. Функция возвращает количество прочитанных символов.   
     
   Сервера постоянно принимают сообщения от клиентов и выводят их на экран в формате **[имя клиента] сообщение**, а затем засыпают на случайное время (usleep и sleep в помощь). Клиенты постоянно пытаются отправить сообщения серверам через случайное время (usleep и sleep снова).   
     
   Необходимо продемонстрировать работу очереди с несколькими клиентами и несколькими серверами.  
     
   **Дополнительно (не обязательно)**  
   Реализовать функцию msgDrop(), которая будет приводить к разблокировке ожидающих функций msgRecv() и msgSend().

**Под Звездочкой (+ балл)**

Реализовать **все дополнительно** и решить задачку с обедающими философами:

Возьмите за основу программу https://github.com/kruffka/C-Programming/blob/master/2024-2025/threads/src/din\_phil.c

Эта программа симулирует известную задачу про обедающих философов. Пять философов сидят за круглым столом и едят спагетти. Спагетти едят при помощи двух вилок. Каждые двое философов, сидящих рядом, пользуются общей вилкой. Философ некоторое время размышляет, потом пытается взять вилки и принимается за еду. Съев некоторое количество спагетти, философ освобождает вилки и снова начинает размышлять. Еще через некоторое время он снова принимается за еду, и т.д., пока спагетти не кончатся. Если одну из вилок взять не получается, философ ждет, пока она освободится. В программе din\_phil.c философы симулируются при помощи нитей, периоды размышлений и еды – при помощи usleep(), а вилки – при помощи mutex. Философы всегда берут сначала левую вилку, а потом правую. При некоторых обстоятельствах это может приводить к мертвой блокировке. Измените протокол взаимодействия философов с вилками таким образом, чтобы мертвых блокировок не происходило.