Домашнее задание. Параллельные алгоритмы.

Егорова Алёна

Домашнее задание 1

Задание 1.

Алгоритм Петерсона гарантировать взаимное исключение не будет.

Доказательство:

Для доказательства приведём пример последовательного исполнения в модели чередования инструкций на одном процессоре, в котором вызов $mtx.lock()$ из двух потоков приводит к нарушению взаимного исключения.

Пусть есть поток $A - 0$ и поток $B - 1$.

```
victim = 0;
victim = 1;
want[1] = 1;
while (want[1 - t].load() && victim.load() == t) { } // Цикл для потока B.
// Поток B пройдет его, потому что want[0] = false

want[0] = 1;
while (want[1 - t].load() && victim.load() == t) { } // Цикл для потока A.
// Поток A пройдет его, потому что последним в victim записал поток B
```

Таким образом, оба потока пройдут в критическую секцию.

Задание 2.

1) Воспользуемся подсказкой для данного задания. Поток, вставший на очередь на ticket спинлоке, будет вытеснен планировщиком и будет отправлен в конец очереди. Таким образом, будут пропускаться все потоки, пока очередь не дойдет до того потока, который в приоритете у планировщика. Такое произойдет, если число потоков будет больше, чем максимальное физическое число потоков на машине (вроде бы количество ядер), потому что при меньшем количестве планировщик сможет запустить их на разных ядрах. Это можно проверить $std :: thread :: hardware\_concurrency()$. Написав программу, я убедилась, что при количестве потоков $= 5$ (значение $hardware\_concurrency()$ у меня 4) частота критических секций очень сильно увеличилась.

2) Для $test - and - setspinlock'a$ такой проблемы не возникает, потому что у потоков, помимо очереди планировщика своей очереди не существует. Как только планировщик убирает кого-то в конец очереди, следующий (может быть и тот же самый) поток сразу пытается захватить *spinlock* и в данном случае большая вероятность того, что планировщик впустит просившийся поток.

Задание 4.

Взаимное исключение гарантироваться будет, свобода от взаимной блокировки - нет.

Доказательство:

1) Взаимное исключение: в критической секции находится не более одного потока. Почему это так? Посмотрим на работу мьютекса. Поток входит в критическую секцию при $thread\_count == 0$. Отрицательной данная переменная стать не может, потому что изначально её значение 0, далее, при каждом прохождении цикла *while* значение переменной увеличивается на 1, если мы проходим в критическую секцию, то значение в будущем всё равно уменьшится на 1 в *unlock*. Следующий поток увеличит значение на 1, но, чтобы выйти из цикла ему необходимо будет уменьшить значение переменно. Так, количество увеличений и уменьшений одно. Так пусть некоторый поток находится сейчас в критической секции (значит, у нас есть в запасе одно уменьшение переменной на единицу). Теперь остальные потоки пытаются попасть в критическую секцию, проверяя условие в цикле. Так как один поток уже прошёл в секцию, увеличив значение, то для следующего потока

условие точно ложное, и непрошедший в секцию поток увеличивает значение переменной $thread\_count$. Далее либо он сразу уменьшает ее значение, и для следующего потока ситуация аналогична, либо следующий поток пытается захватить мьютекс, но у него ничего не получается, и он просто опять увеличивает значение переменной, входя внурь цикла. Таким образом, в критическую секцию не смогут пройти несколько потоков одновременно, так как значени переменной $thread\_count$ не становится равной нулю, потому что количество увеличений и уменьшений переменной $thread\_count$ совпадёт, и поток, закончивший критическую секцию, нарушит это равновесие и вскоре пустит поток в критическую секцию.

2) Для доказательства отсутствия свободы от взаимной блокировки смоделируем следующую ситуацию:
1. Поток A захватывает мьютекс. $thread\_count = 1$;
2. Поток B пытается захватить мьютекс, но попадает в цикл. $thread\_count = 2$;
3. Поток A освобождает мьютексю. $thread\_count = 1$;
4. Поток A пытается захватить мьютекс. $thread\_count = 2$;
5. Поток B работает в цикле. $thread\_count = 1$;
6. Поток B пытается захватить мьютекс. $thread\_count = 2$;
7. Поток A работает в цикле. $thread\_count = 1$;
и так далее.

Из данного примера видно, что 2 потока могут крутиться бесконечно, но они так и не попадут в критическую секцию. Таким образом, свободы от взаимной блокировки нет.