Блокировка (lock)

Один из самых очевидных инструментов для решения как проблемы туалета, так и проблемы процессов операционной системы – это замок или блокировка. И замок – это очень простая штука, у него есть **два состояния: занято и свободно**, и, соответственно, **две операции: занять** (перевести в состояние «занято» или ждать освобождения) **и освободить** (перевести в состояние «свободно»). Когда мы занимаем замок, занимаем эту блокировку, то мы или ждем освобождения, если он занят, или, если он свободен, переводим в состояние «занят». Нужно заметить, что это именно просто замок, сам по себе он не связан с системой. То, как мы его используем, по каким правилам мы с ним работаем – это уже задача внутренняя, то есть когда мы покупаете замок в туалет (просто сам этот замок с зеленой и красной этикеткой где видно: занят туалет или не занят) – сам по себе он еще ничего не обещает, он не означает, что туалетом будут правильно пользоваться, не означает, что всё будет хорошо – он означает, что просто есть такой замок с двумя состояниями. И это наша ответственность правильного его поставить, правильно пользоваться, не забывать его запирать, если мы заходим в туалет, и не забывать давать другим людям понимать, что означает зеленый или красный цвет на этом замке.

Использование блокировок

Использование блокировок (или этих замков) – это, опять же, очевидное движение. Нам нужно, как только мы входим в критическую секцию, **занять** какой-то замок, а после освобождения, **после выхода из этой критической секции – освободить** замок. Получается, что если мы попадаем в критическую секцию, то мы можем туда войти только если другая программа не находится в этой же критической секции, потому что другая программа или другой процесс использует тот же замок, ту же блокировку. Чтобы занять замок – нам нужно подождать его освобождения (было одно из правил, что бесконечно занимать этот замок или ресурс нельзя, поэтому в какой-то момент мы дождемся освобождения), тогда мы займем его и войдем в эту критическую секцию, будем делать что-то с этим ресурсом.

Блокировка защищает данные и в этом ее основная задача. Чаще всего нам нужно сделать так, чтобы, когда мы пишем какие-то данные, мы не превращали эти данные в мусор, не превращали эти данные в то состояние, которое не имеет смысла. Поэтому писать по одному адресу данных может только один процесс и это логично (для этого очень очевидно использовать замок). Эти **процессы или потоки*** **могут занимать разные блокировки**. Они одновременно могут занимать разные блокировки, если процесс пишет файл и печатает на принтере. И если для обоих этих ресурсов (файл и принтер) нужна блокировка, то программа может использовать обе блокировки и в разных комбинациях занимать и освобождать. Идея (и вообще всё необходимое, чтобы это всё работало) заключается в том, что одну блокировку, один замок нужно использовать для всех критических секций, использующих одни и те же ресурсы. В общем, если вы обращаетесь в какой-то файл или вы обращаетесь к принтеру, то нужно использовать один и тот же замок в разных процессах, которые обращаются к тому же файлу или к тому же принтеру – тогда это будет работать.

Пример: «счет в банке»

Посмотрим на классический пример. Каждый раз, когда кто-то где-то говорит о транзакциях или о блокировках, или об этой проблеме параллелизации – чаще всего используют пример «счет в банке». Это очень актуально, очень хорошо подумать о том, что же происходит в нашем банковском софте. Допустим, наша задача состоит в том, чтобы **снять сумму со счета**. Происходит это примерно так (рисунок 3):

---

* В этой лекции мы не делаем никакой разницы между процессами и потоками, то есть тредами, потому что в рамках высокоуровневых концепций это не имеет значения, поэтому мы будем везде говорить процессы или потоки – имеется в виду примерно одно и то же – просто какой-то процесс в компьютере.

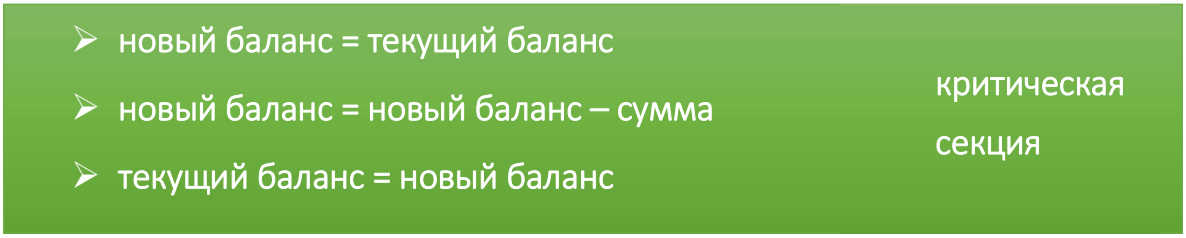

Рисунок 3. Пример снятия суммы со счета в банке

- нам нужно понять, какой сейчас у нас баланс, сколько у нас сейчас денег на счету;
- потом нам нужно обновить этот баланс, потому что мы снимаем со счета деньги и нам нужно этот баланс уменьшить, мы говорим, что новый баланс – это текущий баланс минус сумма. Мы отнимаем ту сумму, которую хотим снять;
- и теперь записываем в текущий баланс наш новый баланс (выдаем деньги или что-то с этими деньгами приходит – суть в том, что со счетом здесь произошло уменьшение).

Допустим, это происходило в 3 этапа. А критическая секция означает, что пока мы в ней находимся, с балансом ничего не должно происходить. Мы работаем с допущением, что пока мы находимся в этой секции, баланс не изменится кем-то другим или чем-то другим в другом конце мира. Поэтому очевидно окружить эту критическую секцию этим самым замком (рисунок 4):

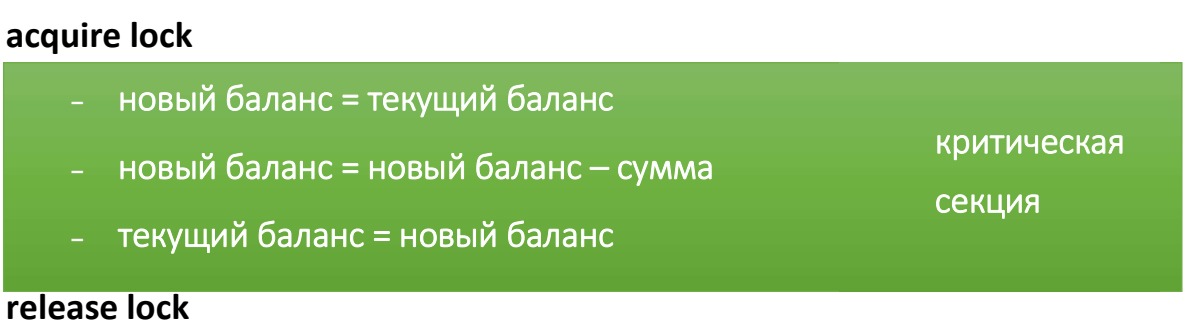

Рисунок 4. Пример снятия суммы со счета в банке

перед входом в критическую секцию мы занимаем замок, делаем все что нам нужно и освобождаем замок. Так как замок занят, пока мы находимся в критической секции никакая другая программа, которая пытается в эту критическую секцию войти (в такую же, но в своей программе) сделать это не может, потому что она ждет освобождения замка. Ну и соответственно мы бы тоже ждали, если кто-то другой в этот момент находился в этой критической секции, мы бы на этом шаге, где мы пытаемся занять замок – ждали бы, пока замок не освободится, потому что мы не можем занять уже занятый замок.

Что нам нужно в критической секции

- Нам нужно **взаимное исключение,** нам нужно, чтобы в этой секции мог находиться только один процесс.
- Нам нужен **прогресс**, нам нужно, чтобы *процесс вне критической секции не мог запретить другому процессу войти в критическую секцию*. Единственное, что должно мешать войти другому процессу в критическую секцию – это замок, или какой-то другой инструмент, который мы используем здесь, чтобы добиться взаимного исключения.
- Нам нужно **конечное ожидание**. Когда мы ждем этот замок, нам нужно знать, что мы когда-нибудь его дождемся.
- Нам нужна **производительность**. Любые эти процессы, любые дополнения: замки и проверки этих замков – естественно не бесплатны. Естественно любое такое действие добавляет времени и работы всей системе, и процессов в частности. Нам очень важно построить эту систему так, чтобы все эти наши надстройки и замки, и новые инструменты, которые мы здесь используем – незначительно добавили нагрузки или чтобы суммарно у нас всё ещё была положительная производительность, чтобы всё у нас ещё было оптимально (или хотя бы в сторону оптимальности).
- Нам нужна **«справедливость»,** нам нужно, чтобы разные процессы, которые имеют одну и ту же критическую секцию в целом имели возможность вообще туда войти, то есть не было большой дискриминации в сторону каких-то процессов.