Семафоры

Замок – это здорово, но замок работает только в определенных ситуациях. Зачастую бывают ситуации сложнее. Туалеты бывают не одноместные – бывают туалеты и двухместные, и трехместные, n-местные. И если, в ситуации с туалетом, можно просто на каждую кабинку поставить по замку, в программировании тоже можно на каждую такую задачу поставить по блокировке, по локу. Есть чуть более продвинутое решение под названием – семафор. Семафор очень похож на замок, но у замка есть только два состояния, а **у семафора есть целый счетчик** – это целочисленный счетчик (то есть там целые числа: 0, 1, 2, 3 и –10, –20, –17), **ограничивающий количество процессов, которые могут войти в определенный участок кода**, и у него **есть три**, что немаловажно, **атомарные операции**:

- **Создание или инициализация** этого семафора, где мы говорим: «Семафор, создайся вот с таким вот счетчиком», – и говорим ему число.
- Может быть **увеличение (signal)**
- и **уменьшение (wait)** или ожидание.

При уменьшении, если счетчик равен нулю, то процесс блокируется.

Простой пример (рисунок 5).

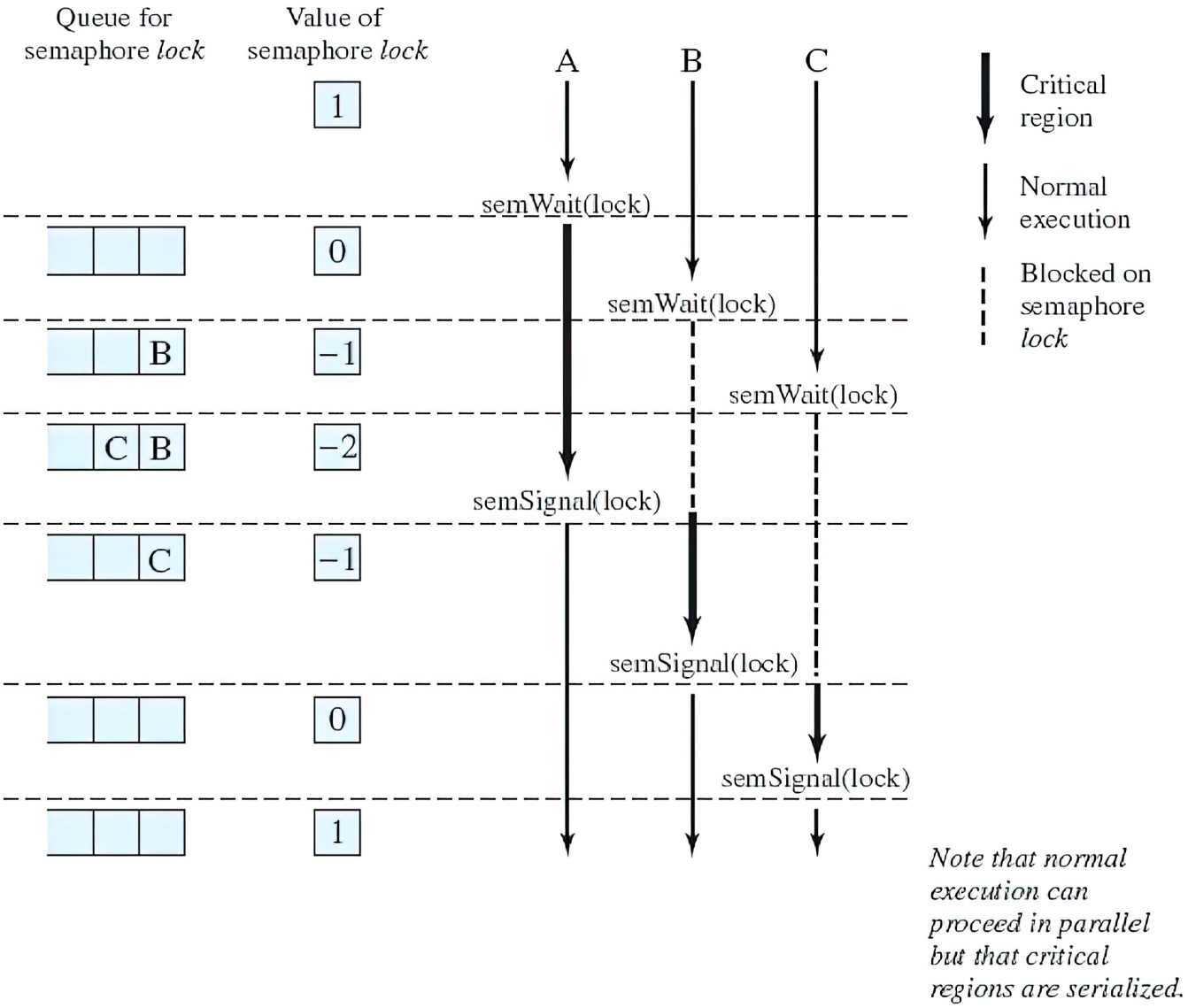

Рисунок 5. Доступ процессов к общим данным, защищенных семафором

Но тут очень простой пример, потому что мы создаем семафор с счётчиком «1», и это по сути замок. У нас есть три процесса: A, B, C. Тонкой линией обозначается обычное выполнение программы не в критической секции, то есть три тонкие стрелки могут одновременно исполняться и им ничего не мешает. Толстая стрелка – это критическая секция и мы здесь допускаем, что критические секции у программ A, B, C – это одни и те же критические секции, это общие для них секции, то есть они не могут одновременно быть в своих критических секциях. А пунктирная линия – это ожидание. Если программе нужно попасть в критическую секцию, а другая программа в ней уже находится, то она будет ждать.

И мы создаем семафор со счетчиком «1», запускаем процессы A, B и C, и процесс A с самого начала пытается войти в свою критическую секцию. На семафоре у нас «1», процесс A попадает в свою критическую секцию, занимает этот виртуальный замок-семафор. Процессам B и C пока ничего не нужно, они продолжают выполнять свои параллельные дела. Счетчик семафора становится «0», когда процесс A воспользовался им. На следующем шаге процесс A продолжает свою работу в критической секции, но процесс B тоже захотел в критическую секцию. Процесс B не можешь туда попасть, потому что на счетчике у нас было «1», а стало «0», а если в момент входа у нас «0», то нам нужно ждать, поэтому процесс B помещается в специальную очередь, а счетчик становится «–1» и всё продолжается: процесс A продолжает работу в своей критической секции, процесс C еще работает сам по себе, процесс B в это время ждет, он ничего не делать. На следующем шаге процесс A всё ещё продолжает свое движение в критической секции и в этот момент процесс C тоже захотел в критическую секцию. В этом случае происходит то же самое: процесс C попадает в очередь, счетчик уменьшается еще на один, и, наконец-то, в какой-то момент процесс A завершает работу своей критической секции, счётчик увеличивается на один и в критическую секцию может войти следующий процесс в очереди – процесс B. Он попадает в свою критическую секцию, начинает работу и теперь процесс C ждет процесс B. Счетчик указывать на один процесс, который там находится. Потом процесс B завершается, начинается процесс C. Очередь пуста, счетчик «0» и наконец в конце, когда C освободит этот семафор – он снова становится единицей. Теперь у нас ни одного процесса нет в критической секции и нам удалось сделать это взаимное исключение аж для целых 3 процессов.

Иногда этого недостаточно:

- Дело в том, что **процесс может ожидать какого-то события,** не обязательно связанного с блокировкой, это может быть просто какое-то событие, которое ему нужно удовлетворить. Возможно оно не связано с критической секцией, это просто какое-то ожидание.
- Во-вторых, **семафоры** – это тоже здорово, также, как и локи, но они **усложняют программирование**. Это инструмент, дополнительный инструмент, который программистам нужно использовать, им нужно помнить, какие данные относятся к каким блокировкам, какие семафоры отвечают за какие ресурсы, им всё это нужно вручную прописывать, думать об этом – всё это усложняет программирование, усложняет отладку и так далее.
- И очень часто **приходится использовать несколько семафоров в одной задаче,** потому что задача, как правило, использует несколько ресурсов и если эти все ресурсы общие, но разных типов, то для каждой из них нужно создавать семафор, думать о них и так далее.

Естественно программисты подумали и сделали для себя несколько решений. Одно из решений – это использовать так называемый монитор.