Взаимоисключение без аппаратной поддержки. Семафоры.

Семафоры являются обобщением блокирующих переменных. Переменные в семафоре могут принимать целые неотрицательные значения и могут использоваться для синхронизации выч. процессов. По сути дела, семафор – объект с операциями Wait и Signal. Для работы с семафорными переменными необходимо иметь операции инициализации самого семафора.

Семафоры можно использовать для взаимного исключения синхронизации взаимодействующих процессов и управлением выделением ресурсов. В зависимости от применения используются разные начальные значения семафора и различные последовательности вызова методов Wait и Signal. С помощью семафора, инициализированного значением «1» можно обеспечить монопольный доступ к общему ресурсу, например, к совместно используемой структуре данных. Семафор также может использоваться для синхронизации двух взаимосвязанных процессов. Например, когда процесс А достигнет определенной точки, он не может продолжать работу, пока процесс Б не выполнит некоторую задачу. Для их синхронизации можно применить инициализированный значением «0» семафор, у которого процесс А должен ждать точки синхронизации, выполнив вызов Wait, пока процесс Б не вызовет метод Signal. Отличие данного способа применения семафора заключается в том, что вызов Wait выполняет один процесс, а вызов Signal другой. Семафор, использующийся таким образом, называется приватным, поскольку ждать у него может только один процесс, а сигнализировать с его помощью о программных событиях могут произвольное количество процессов.

Еще один вариант использования семафора, когда в системе есть несколько единиц одного и того же ресурса, и тогда несколько процессов могут обращаться к ресурсу одновременно, но количество таких ресурсов должно быть ограничено. Семафор такого ресурса инициализируется значением равным количеству экземпляров данного ресурса.

Планирование процессов, ожидающих у семафора может осуществляться с использованием стратегии «Первый вошел – первый обслужен». Но тут существует 2 проблемы:

1) низкоприоритетный процесс, завладевший семафором, может удерживать остальные процессы в состоянии ожидания в независимости от их приоритетов до тех пор, пока он сам не закончит работу с ресурсом и не вызовет метод Signal. В случае приоритетного планирования он может быть вытеснен с процессора до того, как освободит семафор и таким образом увеличится задержка выполнения ожидающих процессов. Эта ситуация называется инверсией приоритетов.

2) Низко приоритетный процесс может стоять в очереди семафора перед процессом с более высоким приоритетом

Схема планирования процессов, ожидающих у семафора, зависит от конкретной реализации и не является частью концепции семафора. Вторую проблему можно решить, если при выполнении одним из процессов вызова Signal освобождать все ожидающие процессы. И тогда в критическую секцию войдет тот из них, который первым снова выполнит вызов Wait. В этом случае выбор процесса будет определяться алгоритмом планирования, используемым в ОС. Еще одна альтернатива заключается в организации планирования для очереди семафора на основе приоритетов процессов, которые в этом случае должны быть известны семафору. Планирование по принципу «Первый вошел – первый обслужен» подходит для систем определенных типов. К ним относятся системы, не требующие ответа в реальном времени и в которых процессы, ожидающие у данного семафора, обычно выполняются с одинаковым приоритетом, а очереди семафоров достаточно короткие. Для решения проблемы инверсии приоритетов нужно допустить временное повышение приоритета процесса, который удерживает семафор, до уровня самого приоритетного в очереди. Эта операция называется **наследованием приоритета**. Процесс, удерживающий семафор, выполняется с действующим приоритетом, равному большему из 2-х значений: исходного приоритета процесса и его унаследованного приоритета.

Мъютекс

Мъютекс – это объект, обеспечивающий исключительный доступ к КС. Для синхронизации потоков разных процессов следует объявить один общедоступный объект класса мъютекс, который будет управлять доступом к списку потоков. Мъютекс предоставляет доступ к объекту любому из потоков, если в данный момент, объект не занят и запоминает текущее состояние объекта. Если объект занят, то мъютекс запрещает доступ. Объект мъютекс необходимо объявить заранее.

При вызове функции открытия мъютекса система скнирует существующие объекты мъютексы, проверяя нет ли среди них объекта с указанным именем. Если таковой обнаружиться, то она создает описатель объекта, специфичный для данного процесса. Имя мъютексу присваивается системой при его создании. Это имя используется при совместном доступе к мъютексу нескольких процессов. С любым объектом ядра сопоставляется счетчик, фиксирующий сколько раз данный объект передавался во владение потока. Объект мъютекс отличается от других синхронизирующих объектов тем, что занявшему его потоку передаются права на владение им. Прочие синхронизирующие объекты могут быть либо свободны, либо заняты, а мъютексы способны еще и запоминать к какому потоку они принадлежат. Отказ от мъютекса происходит, когда ожидавший его поток захватывает этот объект, переводя его в занятое состояние, а поток завершается. В таком случае получается, что мъютекс занят и никогда не освободиться, поскольку другой поток не может этого сделать. Система не допускает подобных ситуаций и заметив, что произошло, автоматически переводит мъютекс в свободное состояние.

В отличая от семафора, который работая в режиме пользователя и блокирует задачи, мъютекс работает в режиме ядра и блокирует сам ресурс. На время выполнения мъютекса задача, которая захватила мъютекс получает максимальный приоритет и соответственно, если задача выполняет какие-либо действия внутри семафора, то более высокоприоритетная задача может прервать его выполнение. Если задача захватила мъютекс, то прекратить ее выполнение, пока она сама не освободит мъютекс, не может никакая другая задача.