Семафоры, мьютексы. Использование семафоров для синхронизации процессов

Вместо двоичных переменных Дийкстра (Dijkstra) предложил использовать переменные, которые могут принимать целые неотрицательные значения.
Такие переменные, используемые для синхронизации вычислительных процессов, получили название семафоров.
Для работы с семафорами вводятся два примитива, традиционно обозначаемых P и V.
Пусть переменная S представляет собой семафор.
Тогда действия V(S) и P(S) определяются следующим образом.
V(S): переменная S увеличивается на 1 единым действием.(Выборка, наращивание и запоминание не могут быть прерваны).
К переменной S нет доступа другим потокам во время выполнения этой операции.
P(S): уменьшение S на 1, если это возможно. Если S=0 и невозможно уменьшить S, оставаясь в области целых неотрицательных значений, то в этом случае поток, вызывающий операцию P, ждет, пока это уменьшение станет возможным.
Успешная проверка и уменьшение также являются неделимой операцией.
Операция P заключает в себе потенциальную возможность перехода потока, который ее выполняет, в состояние ожидания, в то время как операция V может при некоторых обстоятельствах активизировать другой поток, приостановленный операцией P.
В частном случае, когда семафор S может принимать только значения 0 и 1, он превращается в блокирующую переменную, которую по этой причине часто называют двоичным семафором (мьютексом).
Рассмотрим использование семафоров на классическом примере взаимодействия двух выполняющихся в режиме мультипрограммирования потоков, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула.
Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись.
В общем случае поток-писатель и поток-читатель могут иметь различные скорости и обращаться к буферному пулу с переменой интенсивностью
В один период скорость записи может превышать скорость чтения, в другой — наоборот.
Для правильной совместной работы поток-писатель должен приостанавливаться, когда все буферы оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера.
Напротив, поток-читатель должен приостанавливаться, когда все буферы пусты, и активизироваться при появлении хотя бы одной записи.
Введем два семафора: e — число пустых буферов, и f — число заполненных буферов, причем в исходном состоянии e =N, a

f =0. Тогда работа потоков с общим буферным пулом может быть описана следующим образом (рис.).

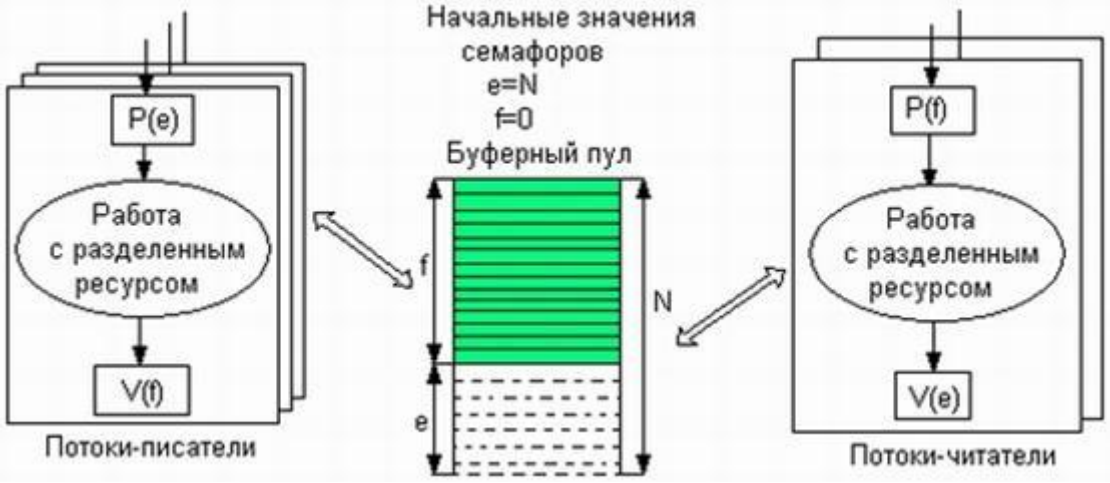


Рис. Использование семафоров для синхронизации потоков

Поток-писатель прежде всего выполняет операцию P(e), с помощью которой он проверяет, имеются ли в буферном пуле незаполненные буферы.

В соответствии с семантикой операции P, если семафор e равен 0 (то есть свободных буферов в данный момент нет), то поток-писатель переходит в состояние ожидания.

Если же значением e является положительное число, то он уменьшает число свободных буферов, записывает данные в очередной свободный буфер после этого наращивает число занятых буферов операцией V(f).

Поток-читатель действует аналогичным образом, с той разницей, что он начинает работу с проверки наличия заполненных буферов, а после чтения данных наращивает количество свободных буферов.

В данном случае предпочтительнее использовать семафоры вместо блокирующих переменных.

Действительно, критическим ресурсом здесь является буферный пул, который может быть представлен как набор идентичных ресурсов — отдельных буферов, а значит, с буферным пулом могут работать сразу несколько потоков, и именно столько, сколько буферов в нем содержится.

Использование двоичной переменной не позволяет организовать доступ к критическому ресурсу более чем одному потоку. Семафор же решает задачу синхронизации более гибко, допуская к разделяемому пулу ресурсов заданное количество потоков. Так, в нашем примере с буферным пулом могут работать максимум N потоков, часть из которых может быть «писателями», а часть — «читателями».

Таким образом, семафоры позволяют эффективно решать задачу синхронизации Доступа к ресурсным пулам, таким, например, как набор идентичных в функциональном назначении внешних устройств (модемов, принтеров, портов), или набор областей памяти одинаковой величины, или информационных структур.

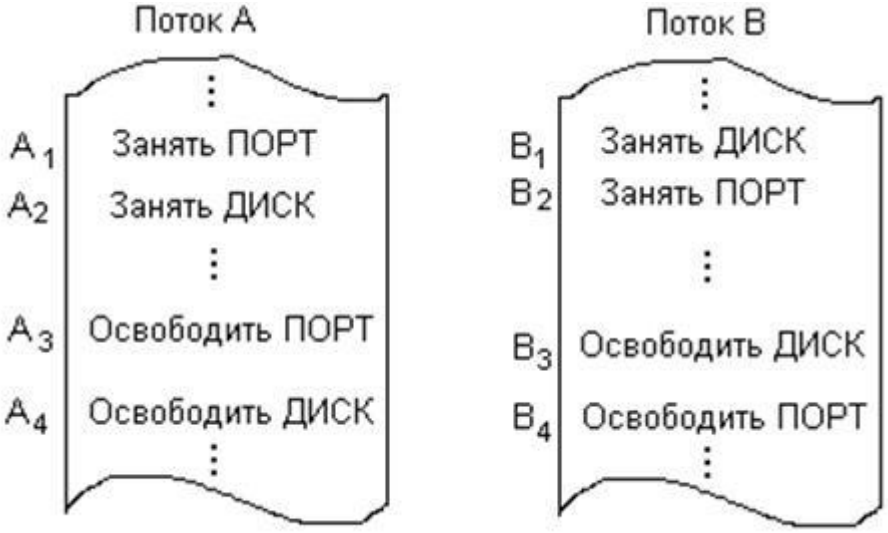
Во всех этих и подобных им случаях с помощью семафоров можно организовать доступ к разделяемым ресурсам сразу нескольких потоков.

Проиллюстрируем еще одну проблему синхронизации — взаимные блокировки, называемые также дедлоками (deadlocks), клинчами (clinch), или тупиками.

Пусть двум потокам, принадлежащим разным процессам и выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например диск и последовательный порт.

На рис. а показаны фрагменты соответствующих программ. Поток А запрашивает сначала порт; а затем диск, а поток В запрашивает устройства в обратном порядке

а

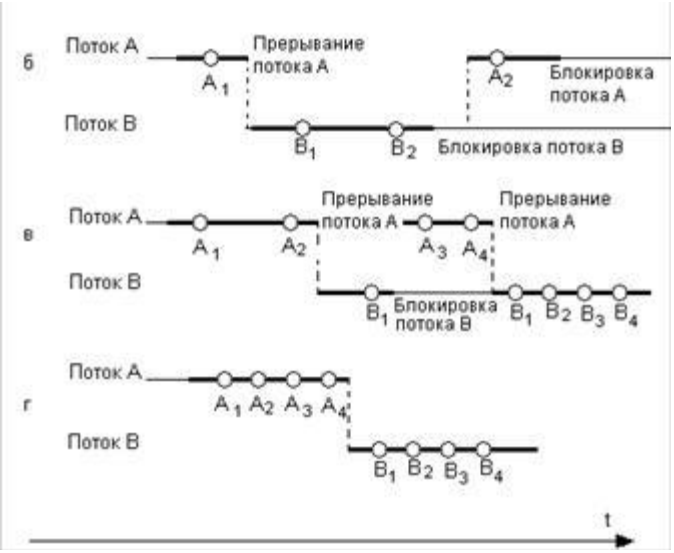


Предположим, что после того, как ОС назначила порт потоку А и установила связанную с этим ресурсом блокирующую переменную, поток А был прерван

б

Управление получил поток В, который сначала выполнил запрос на получение COM- порта, затем при выполнении следующей команды был заблокирован, так как диск оказался уже занятым потоком А. Управление снова получил поток А, который в соответствии со своей программой сделал попытку занять порт и был заблокирован, поскольку порт уже выделен потоку В. В таком положении потоки А и В могут находиться сколь угодно долго.

В зависимости от Соотношения скоростей потоков они могут либо взаимно блокировать друг друга (рис. б), либо образовывать очереди к разделяемым ресурсам (рис. в), либо совершенно независимо использовать разделяемые ресурсы (рис. г).



В рассмотренных примерах тупик был образован двумя потоками, но взаимно блокировать друг друга может и большее число потоков.
Невозможность потоков завершить начатую работу из-за возникновения взаимных блокировок снижает производительность вычислительной системы. Поэтому проблеме предотвращения тупиков уделяется большое внимание. На тот случай, когда взаимная блокировка все же возникает, система должна предоставить администратору-оператору средства, с помощью которых он смог бы распознать тупик, отличить его от обычной блокировки из-за временной недоступности ресурсов. И наконец, если тупик диагностирован, то нужны средства для снятия взаимных блокировок и восстановления нормального вычислительного процесса
Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть при любом соотношении взаимных скоростей потоков

Другой, более гибкий подход к предотвращению тупиков заключается в том, что ОС каждый раз при запуске задач анализирует их потребности в ресурсах и определяет, может ли в данной мультипрограммной смеси возникнуть тупик. Если да, то запуск новой задачи временно откладывается. ОС может также использовать определенные правила при назначении ресурсов потокам, например, ресурсы могут выделяться операционной системой в определенной последовательности, общей для всех потоков.
В тех же случаях, когда тупиковую ситуацию не удалось предотвратить, важно быстро и точно ее распознать, поскольку блокированные потоки не выполняют никакой полезной работы. Если тупиковая ситуация образована множеством потоков, занимающих массу ресурсов, распознавание тупика является нетривиальной задачей. Существуют формальные, программно-реализованные методы распознавания тупиков, основанные на ведении таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ этих таблиц позволяет обнаружить взаимные блокировки.

Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные потоки. Можно снять только часть из них, освободив ресурсы, ожидаемые остальными потоками, можно вернуть некоторые потоки в область подкачки, можно совершить -«откат» некоторых потоков до так называемой контрольной точки, в которой запоминается вся информация, необходимая для восстановления выполнения программы с данного места. Контрольные точки расставляются в программе в тех местах, после которых возможно возникновение тупика.

Механизмы синхронизации, основанные на использовании глобальных переменных процесса, обладают существенным недостатком — они не подходят для синхронизации потоков разных процессов.

В таких случаях операционная система должна предоставлять потокам системные объекты синхронизации, которые были бы видны для всех потоков, даже если они принадлежат разным процессам и работают в разных адресных пространствах.

Примерами таких синхронизирующих объектов ОС являются системные семафоры, мьютексы, события, таймеры и другие — их набор зависит от конкретной ОС, которая создает эти объекты по запросам процессов.

Чтобы процессы могли разделять синхронизирующие объекты, в разных ОС используются разные методы