Многозадачность

Многозадачность – способ организации работы системы, при котором в ее памяти одновременно содержатся программы и данные для выполнения нескольких процессов.

Процесс – абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет единицу работы, заявку на потребление системных ресурсов.

Подсистема управления процессов:

- Планирует выполнение процессов
- Создает и уничтожает процессы
- Обеспечивает процессы необходимыми ресурсами
- Поддерживает взаимодействие между процессами

Выделение ресурсов процессу:

- Единоличное пользование
- Совместное с другим процессом пользование

Время выделения:

- При создании
- Динамически по запросу во время выполнения

Приписка ресурсов:

- На все время
- На определенный период

Процесс – это контейнер для набора ресурсов, используемых потоками.

Поток – часть процесса, которая получает процессорное время.

Элементы потока:

- Набор регистров
- Два стека (ядра и пользовательский)
- Локальная область памяти
- Идентификатор

ОС поддерживает таблицы:

- Для памяти
- Для устройств ввода
- Для файлов
- Для процессов

Синхронизирующие примитивы

Семафоры

Для решения проблемы инверсии приоритетов можно допустить временное повышение приоритета процесса, который удерживает семафор до уровня самого высокоприоритетного в очереди. Эта операция называется **наследование приоритета**. Процесс, удерживающий семафор, выполняется с действующим приоритетом, равным

большему из двух значений: исходного приоритета процесса и его унаследованного приоритета.
При разработке семафоров должна быть учтена возможность их параллельного вызова разными процессами. Важным архитектурным вопросом, который следует решить при проектировании системы, является местоположение класса семафора и его связь с подсистемой управления процессами.
При определении операций семафора в случае занятости запрошенного процессом ресурса операционная система заблокирует этот процесс, а при получении семафором сигнала об освобождении ресурса, перевод его в состояние готовности. Причем, в первом случае процесс будет добавлен в очередь семафора, а во втором из нее удален. Таким образом, реализация Wait и Signal может требовать изменение целочисленного значения семафора, его очереди и состояния процесса. Все изменения должны осуществляться в составе одной **атомарной** операции во избежание случайного перевода системы в несогласованной состояние, т. е. должны быть выполнены либо все три действия, либо ни одно из них. В любой однопроцессорной или многопроцессорной реализации без специальных ограничений может одновременно поступать любое количество вызовов методов Wait и Signal одного семафора. Реализация семафора должна отвечать требованию монопольного доступа к его внутренним данным. Методы Wait и Signal содержат критические секции, связанные с внутренней структурой данных семафора, и не могут одновременно выполняться несколькими процессами. Однако семафоры предназначены для решения проблем взаимного исключения и синхронизации, поэтому необходимо обеспечить возможность одновременного обращения к ним.

Мониторы

Монитор – это механизм организации параллелизма, который содержит как данные, так и процедуры, необходимые для реализации динамического распределения конкретного общего ресурса или группы общих ресурсов.
Чтобы обеспечить выделение нужного ему ресурса, процесс должен обратиться к конкретной процедуре монитора. Необходимость входа в монитор в различные моменты времени может возникнуть у многих процессов. Однако, вход в монитор находится под жестким контролем, т. к. здесь осуществляется взаимоисключение процессов, так что в каждый момент времени только одному процессу разрешается войти в монитор. Процессам, которые хотят войти в монитор, когда он уже занят, приходится ждать, причем режимом ожидания автоматически управляет сам монитор.
Поскольку механизм монитора гарантирует взаимоисключение процессов, исключаются серьезные проблемы, связанные с параллельным режимом работы. Внутренние данные монитора могут быть либо глобальными, либо локальными. Ко всем этим данным можно обращаться только изнутри монитора. Процессы, находящиеся вне монитора, просто не могут получить доступ к данным монитора. Если процесс обращается к некоторой процедуре монитора и обнаруживается, что соответствующий ресурс занят, эта процедура монитора выдает команду ожидания Wait. При этом процесс покидает монитор, чтобы не нарушать принцип взаимоисключения, когда в монитор зашел бы другой процесс. Поэтому процесс, переводящийся в режим ожидания, должен вне монитора ждать того момента,

когда необходимый ему ресурс освободится. Со временем процесс, который занимал данный ресурс, обратится к монитору, чтобы возвратить ресурс системе. Соответствующая процедура монитора при этом может просто принять уведомление о возвращении ресурса, а затем ждать, пока не поступит запрос от другого процесса, которому потребуется этот ресурс. Но так как может оказаться, что уже имеются процессы, ожидающие освобождения данного ресурса, монитор выполняет примитив оповещения Signal, чтобы один из ожидающих процессов мог занять данный ресурс и покинуть монитор. Если процесс сигнализирует о возвращении ресурса и нет процессов, ожидающих данного ресурса, то монитор вносит ресурс в список свободных. Процесс, ожидающий освобождения некоторого ресурса, должен находиться вне монитора, чтобы другой процесс имел возможность войти в монитор и возвратить ему этот ресурс. Чтобы гарантировать, что процесс, находящийся в ожидании некоторого ресурса со временем действительно получит этот ресурс, считается, что ожидающий процесс имеет более высокий приоритет, чем новый процесс, пытающийся войти в монитор.

Мьютексы

Отличие от семафора: семафор, если процесс выполнил запрос ресурса и ресурс занят, то процесс переводится в состояние ожидания. Мьютекс блокирует ресурсы. Мьютекс - объект ядра; выполняется с очень высоким приоритетом.

Мьютекс — это объект синхронизации, который устанавливается в особое сигнальное состояние, когда не занят каким-либо потоком. Поскольку только один поток владеет этим объектом, одновременный доступ к общему ресурсу исключается. Чтобы исключить запись двух потоков в общий участок памяти в одно и то же время, каждый поток ожидает, когда освободится мьютекс, становится его владельцем и только потом пишет что-либо в этот участок памяти.

Если процесс знает, что существует мьютекс с каким-то именем, он может сделать этот объект доступным для себя, открыв уже существующий мьютекс. При этом система создает дескриптор объекта, специфичный для данного процесса. Когда система создает мьютекс, она присваивает ему имя; это имя используется при совместном доступе к мьютексу в нескольких процессах.

С любым объектом ядра сопоставляется счетчик, фиксирующий сколько раз данный объект был передавался во владение потоком. Если поток выполнит вызов для уже принадлежащего ему объекта, он сразу получит доступ к защищаемым этим объектом данным, т. к. система определит, что поток уже владеет этим объектом. При этом счетчик числа пользователей увеличится на единице. Чтобы перевести объект в свободное состояние, потоку необходимо соответствующее число раз вызвать Signal (Unlock?).

Объект мьютекса отличается от других синхронизирующих объектов ядра тем, что занявшему его потоку передаются права на владение им. Прочие синхронизирующие объекты могут быть либо свободны, либо заняты, и только мьютексы способны еще и запоминать какому потоку они принадлежат. Отказ от мьютексы происходит, когда ожидавший его поток захватывает этот объект, переводя его в занятое состояние, а потом завершается. В таком случае получается, что мьютекс занят и никакой другой поток не

сможет его освободить. Система не допускает подобных ситуаций и при завершении процесса переводит мьютекс в свободное состояние.

Передача сообщений

Метод почтовых ящиков

Требует использования синхронизующих примитивов. Процесс отправитель кладет сообщения в ящик, а процесс получатель забирает их оттуда.
Виды:

1. Однонаправленные
2. Двунаправленные. Необходимо подтверждение получения

Преимущества:

- Управляется ОС

Недостатки:

- Нагрузка на ОС

Классические задачи синхронизации

- Задача "Производитель-потребитель"
- Задача "Обедающие философы"
- Задача "Спящий парикмахер"

Тупик (deadlock) в операционных системах

Группа процессов находится в тупиковой ситуации, если каждый процесс из группы ожидает события, которое может вызвать только другой процесс из той же группы. Так как все процессы находятся в состоянии ожидания, ни один из них не будет причиной какого-либо события, которое могло бы активировать любой процесс в группе, все процессы продолжают ждать до бесконечности.
В большинстве случаев событие, которого ждет каждый процесс, является возвратом какого-либо ресурса, в данный момент занятого другим участником группы. Иначе говоря, каждый участник в группе процессов, зашедший в тупик, ждет доступа к ресурсу, принадлежащему заблокированному процессу. Ни один из процессов не может работать, ни один из них не может освободить какой-либо ресурс и ни один из них не может возобновиться. Кол-во процессов, а также кол-во и вид ресурсов, имеющихся из запрашиваемых здесь не важны. Результат остается тем же самым для любого вида ресурсов, и аппаратных, и программных.
При рассмотрении проблемы тупиков целесообразно понятие ресурсов системы обобщить и разделить их все на два класса:

- Повторно используемые (системные ресурсы), SR
- Расходоумые ресурсы, CR

Повторно используемый ресурс — это конечное множество идентичных единиц со следующими свойствами:

1. Число единиц ресурсов постоянно
2. Каждая единица ресурса или доступна или распределена одному и только одному процессу. Здесь разделение либо отсутствует, либо не принимается во внимание, т. к. не оказывает влияния на распределение ресурсов, а значит и на возникновение тупиковой ситуации
3. Процесс может освободить единицу ресурса, т. е. сделать ее доступной, только если он ранее получил эту единицу. Т. е. никакой процесс не может оказывать какое-либо влияние ни на один ресурс, если он ему не принадлежит

Перечисленные свойства обычных системных ресурсов существенны для изучения проблемы тупиков.

Расходуемый ресурс характеризуется следующими свойствами:

1. Число доступных единиц некоторого ресурса типа CR изменяется по мере того, как приобретаются (расходуются) и освобождаются (производятся) отдельные их элементы выполняющимися процессами. И такое число единиц ресурса является потенциально неограниченным. Процесс-производитель увеличивает число единиц ресурса, освобождая одну или более единиц, которые он создал
2. Процесс-потребитель уменьшает число единиц ресурса, сначала запрашивая, а затем приобретая одну или более единиц. Единицы ресурса, которые приобретены, в общем случае не возвращаются ресурсу, а расходуются

Эти свойства потребляемых ресурсов присущи многим синхронизирующим сигналам, сообщениям и данным, порождаемым как аппаратурой, так и программным обеспечением, и могут рассматриваться как ресурсы типа CR, при изучении тупиков. В их число входят:

- Прерывания от таймеров и устройств ввода/вывода
- Сигналы синхронизации процессов
- Сообщения, содержащие запросы на различные виды обслуживания или данные, а также соответствующие ответы

Для возникновения ситуации взаимоблокировки должны выполняться четыре условия:

1. Условие взаимного исключения, при котором процессы осуществляют монопольный доступ к ресурсам
2. Условие ожидания. Характеризуется тем, что процессы в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы
3. Условия отсутствия перераспределения. У процесса нельзя принудительным образом забрать ранее полученные ресурсы. Процесс, владеющий ими должен сам освободить эти ресурсы.
4. Условие циклического ожидания. В данном случае должна существовать круговая последовательность из двух или более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим процессом последовательности

Для того чтобы произошла взаимоблокировка, должны выполняться все четыре условия. Если хоть одно из них отсутствует, тупиковая ситуация невозможна.

Направления изучения тупиковых ситуаций:

- Метод предотвращения тупиков
- Обходы тупиков
- Обнаружение тупиков

- Восстановление после тупиков

Предотвращение тупиков

Предотвращение тупиков основывается на предположении о чрезвычайно высокой его стоимости. Поэтому лучше потратить доп. ресурсы системы, чтобы исключить вероятность возникновения тупика при любых обстоятельствах. Этот подход используется в наиболее ответственных системах, чаще в realtime-системах. Дисциплина, предотвращающая тупик, должна гарантировать, что какое-либо из 4 условий, необходимых для его наступления, не может возникнуть.

Условия взаимного исключения можно подавить путем разрешения неограниченного разделения ресурсов. Это удобно для повторно вводимых программ и ряда драйверов, но совершенно неприемлемо совместно используемым переменным в критических интервалах.

Условие ожидания можно подавить, предварительно выделяя ресурсы. При этом процесс может начинать исполнение только получив все необходимые ресурсы заранее. Предварительное выделение может привести к снижению эффективности работы вычислительной системы в целом.

Условие отсутствия перераспределения можно исключить, позволяя ОС отнимать у процесса ресурсы. Для этого в ОС должен быть предусмотрен механизм, запоминания состояния процесса и ресурсов с целью последующего восстановления.

Условие кругового ожидания можно исключить, предотвращая образование цепи запросов. Это можно обеспечить с помощью принципа иерархического выделения ресурсов. Все ресурсы образуют некоторую иерархию и процесс, затребовавший ресурс на одном уровне может затем потребовать ресурсы на следующем, более высоком уровне. Он может освободить ресурсы на данном уровне только после освобождения всех ресурсов на всех более высоких уровнях. После того, как процесс получил, а потом освободил ресурсы данного уровня, он может запросить ресурсы на том же самом уровне. Иерархическое выделение ресурсов часто не дает никакого выигрыша, если порядок использования ресурсов, определенный в описании процессов, отличается от порядка уровней иерархии. Тогда ресурсы будут использоваться крайне неэффективно.

Обход тупиков (алгоритм банкира, Дейкстра)

Состояние ОС безопасно, если система может обеспечить завершение всех задач в течение какого-то определённого промежутка времени. См. методичку.

Обнаружение тупиков с последующим восстановлением

Это простой алгоритм, который изучает граф и завершается или когда находит цикл, или когда показывает, что циклов в этом графе не существует. Он использует одну структуру данных. Во время работы алгоритма на ребрах графа будет ставиться метка, говорящая о том, что их уже проверяли. Для каждого узла N в графе выполняется 5 шагов, где является начальным узлом:

1. Задаются начальные условия: L - пустой список, все ребра не маркированы

2. Текущий узел добавляется в конец списка L и проверяется кол-во появлений узла в списке. Если узел присутствует в двух местах, значит граф содержит, и работа алгоритма завершается
3. Для заданного узла определяется выходит ли из него хотя бы одно немаркированное ребро. Если да, то выполняется переход к следующему шагу.
4. Случайным образом выбирается любое исходящее немаркированное ребро, которое помечается. Затем по этому ребру выполняется переход к новому текущему узлу и выполняется переход к шагу 2. Если на 4-м шаге не обнаружено ни одного немаркированного ребра, последний узел из списка удаляется и выполняется к предыдущему узлу. Если это первоначальный узел, значит граф не содержит циклов и алгоритм завершается

Алгоритм обнаружения взаимоблокировок при наличии нескольких ресурсов каждого типа основан на сравнении векторов: сравнивается вектор доступных ресурсов и строки матрицы запросов. Алгоритм обнаружения тупиков состоят из следующих шагов:

1. Ищем немаркированный процесс Pi, для которого i-я строка матрицы R меньше или равна вектору A
2. Если такой процесс найден, i-я строка матрицы C прибавляется к вектору A, процесс маркируется и выполняется переход к шагу 1
3. Если таких процессов не существует, работы алгоритма заканчивается. Завершение алгоритма означает, что все немаркированные процессы попали в тупик.

На первом шаге алгоритм ищет процесс, который может доработать до конца. Такой процесс характерен тем, что все требуемые для него ресурсы должны находиться среди доступных в данный момент ресурсов. Тогда выбранный процесс проработает до своего завершения и после этого вернет ресурсы, которые он занимал. Затем процесс маркируется как законченный. Если окажется, что все процессы могут работать, то взаимоблокировки нет.

После обнаружения взаимоблокировок необходимо применить меры для восстановления системы:

1. Восстановление при помощи принудительной выгрузки ресурсов. Всегда можно временно отобрать ресурс у владельца и отдать его другому процессу. Способность забирать ресурс у процесса, отдавать его другому процессу, а затем возвращать назад, так, что исходный процесс этого не замечает, в значительной мере зависит от свойств ресурса
2. Восстановление через откат. Если есть вероятность появления взаимоблокировок, можно организовать работу таким образом, чтобы процессы периодически создавали контрольный точки. Это означает, что состояние процесса записывается в файл и в последствии он может быть возобновлен из этого файла. Контрольные точки содержат не только образ памяти, но и состояние ресурсов, т. е. информацию о том, какие ресурсы в данный момент предоставлены процессу. При обнаружении взаимоблокировки достаточно понять какие ресурсы нужны процессам, и чтобы выйти из тупика процесс, занимающий необходимый ресурс, откатывается к тому моменту времени, перед которым он получил данный ресурс, для чего запускается одна из его контрольных точек
3. Восстановление путем уничтожения процессов.

Двухфазное блокирование

Для предотвращения блокировок во многих системах баз данных, когда часто выполняются операции блокирования нескольких записей, используется подход двухфазного блокирования. В первой фазе процесс пытается заблокировать все требуемые записи по одной. Если операция успешна, процесс переходит ко второму этапу, выполняя обновление и освобождение блокировок. Никакой работы не совершается. Если во время выполнения первой фазы какая-либо необходимая запись оказывается заблокированной, процесс просто освобождает все свои блокировки и начинает первую фазу заново.

Синхронизация процессов в распределённых системах

Обычно децентрализованные алгоритмы имеют следующие свойства:

1. Относящаяся к делу информацию распределена среди всех компьютеров;
2. Процессы принимают решения на основе только локальной информации;
3. Не должно быть единственной критической точки, выход из строя которой приводил бы к краху системы;
4. Не существует общих часов или другого источника точного глобального времени.

В распределенных системах сложно достигнуть согласия относительно времени, что может привести к невозможности зафиксировать глобальное состояние для анализа ситуации, например, для обнаружения взаимоблокировок или для организации промежуточного запоминания.

Тупики в распределённых системах

Тупики в РПС подобны тупикам в централизованных системах, но их сложнее обнаружить и предотвратить. Иногда выделяют особый вид тупиков в РПС - коммуникационные тупики. При борьбе с тупиками в РПС используется несколько стратегий:

1. Полное игнорирование проблемы
2. Обнаружение тупиков

При обнаружении тупиков используются два основных метода:

1. Централизованное обнаружение тупиков. Заключается в распространении идеи графа размещения ресурсов на РПС. Координатор строит единый граф для всех узлов системы и на нем выполняет редукцию.
2. Распределенное обнаружение тупиков. Здесь инициирование задачи обнаружения тупика начинает процесс, который подозревает, что система находится в тупике. Он посылает тем процессам, которые держат нужный ему ресурс сообщение из трех информационных полей. Первое - номер процесса, инициировавшего поиск, и оно не меняется при дальнейшей пересылке. Второе и третье поля содержат соответственно номер процесса, который зависит от ресурса и номер процесса, который держит этот ресурс. Если в конце концов сообщение вернется к тому процессу, который инициировал поиск, то он приходит к выводу, что система зациклена и он должен предпринять действия по ликвидации тупика.

Предотвращение тупиков в РПС базируется на идеи упорядочивания процессов по их временным меткам. Допустимой является ситуацией, когда старший процесс ждет ресурсы, которые захватил младший процесс. В противном случае младший процесс должен отказаться от ожидания.