Тема: Синхронизация

Современные ОС работают в многозадачности и с распределенными вычислениями. В основе этого всего лежат параллельные вычисления. А параллельные вычисления подразумевают под собой использование взаимной блокировки и синхронизации.

Взаимное блокировка – в один момент времени с общим ресурсом работает только один процесс.

Синхронизация – возможность процессов координировать свою работу путем обмена сообщений.

Процесс – некая абстракция, описывающая выполняющуюся программу. Для ОС процесс – запрос ресурсов, единица работы.

Для того, чтобы ОС могла поддерживать взаимную блокировку и синхронизацию она должна:

- уметь работать в многозадачности: чередовать выполнение процессов

- распределять ресурсы между процессами

- поддерживать обмен информации между процессами

Для того, чтобы ОС могла управлять процессами ей нужно:

- знать состояния процессов, в котором они находятся сейчас. Для этого у нее есть подробные структуры данных, описывающих каждый процесс.

- иметь возможность переводить процесс из одного состояния в другой

- предоставлять инструментарий для совместного выполнения процессов и их синхронизации

Под синхронизацией процессов подразумевается синхронизация скоростей путем приостановки одного из процессов до наступления события и активации после наступления этого события.

Цель синхронизации:

Потоки/процессы все асинхронны, поэтому сложно сказать на каком этапе и в какое время будет находиться процесс. А так как мы этого не знаем, то при взаимодействии процессов нам нужно выполнить:

- предотвращение гонок (результат зависит от скорости выполнения потока)

- предотвращения тупиков

Средства синхронизации:

Мы говорили, что потоки должны уметь координировать свои действия при помощи обмена сообщениями. Сообщения – ЖДАТЬ, СИГНАЛИЗИРОВАТЬ

Но стоит отметить, что сообщения могут по-разному вырабатываться и служить разным целям.

Если речь идет об обмене данными между процессами, то тогда

- один СИГНАЛИЗИРУЕТ о нужных ресурсах другому

- и он же ЖДЕТ эти ресурсы

Если речь идет о взаимном использовании ресурсов

- ЖДАТЬ освобождения ресурсов

- СИГНАЛИЗИРОВАТЬ о их освобождении

Критическая секция – часть программы, в которой происходит обращение к общим ресурсам. Нужно гарантировать, что в каждый момент времени доступ к критической секции имеет только один процесс.

Непосредственно о средствах синхронизации

- блокирующие переменные

- алгоритм Деккера

- алгоритм Петерсона

- семафоры

- мьютексы

Отличие семафора от мьютекса

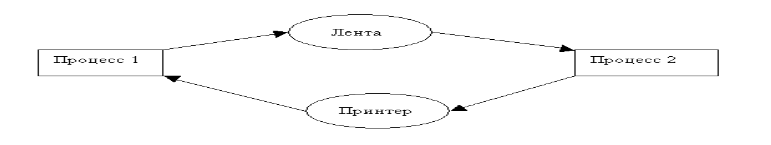
- Мьютекс имеет эсклюзивного владельца, которые и должен его освобождать.

В то время, как семафор никому не принадлежит.

Тупой пример: Возможность посидеть в зале на стуле одному. Семафор – стул есть, вас пропускают, остальных нет и пока вы не уйдете. Мьютекс – ходите со своим стулом. Так как вы сели на стул, условие (посидеть в зале на стуле одному выполнилось) другие не могут. Вы сидите, а потом уходите и стул уносите с собой.

Тема: Тупики. Методы борьбы с ними

Рассмотрим пример тупиковой ситуации:



У нас есть 2 процесса, которые осуществляют вывод с ленты на принтер.

Первый процесс захватил ленту и теперь ожидает, когда будет доступ к принтеру.

Второй процесс пошел другим путем: сначала захватил ресурс принтер, так как он был доступен, а теперь он ожидает доступа к ленте.

В итоге, оба процесса оказались заблокированные, потому что они не могут забрать это ресурс у другого процесса, не могут вдвое использовать ресурс, «разойтись» в разные стороны они не могут.

Как видно, для возникновения тупиковой ситуации должны выполняться 4 условия:

1. наличие монопольного ресурса

2. ожидание ресурсов ( процесс ожидает освобождения ресурса, но при этом свои уже выделенные ресурсы он держит при себе)

3. условие неперераспределяемости (ресурс может быть освобожден только тем процессом, который его захватил)

4. наличие цикла

Если выполняются ВСЕ 4 условия, тогда будет тупик

Идеи для борьбы с тупиками:

1. Игнорировать

Тут идея в том, что а нужно ли тратить много ресурсов компа для того, чтобы решать проблему, которая возникает раз в сто лет.

Потому что любая доп проверка, доп режимы работы процессора – увеличение времени, аппаратных затрат.

В системах жесткого реального времени нельзя игнорировать, потому что это приведет к опасности. А в остальных системах, если не приведет к критической ситуации.

1. Обнаружение тупиков с последующим восстановлением

Тут просто идет проверка на наличие тупиков и определение ресурсов, которые вовлечены в это. Как правило, алгоритмы определения тупика включаются лишь после выполнения 3 условий и проверяют наличие цикла

Один из алгоритмов использует информацию о состоянии системы, содержащуюся в одной из таблиц: текущее распределения запросов, таблица заблокированных ресурсов. После каждой итерации таблицы модифицируются

- принудительный перезапуск

- принудительное завершение процессов

- принудительные последовательное завершение процессов

- перезапуск процессов, находящихся в тупике, с некоторой контрольной точки

- перераспределения ресурсов с последующим последовательным перезапуском процессов, находящихся в тупике

1. Обход тупиков

Алгоритм банкира

Предположим, что у системы в наличии п свободных устройств, например магнитных лент.

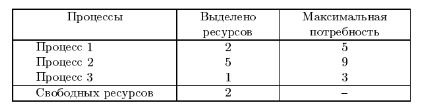
ОС принимает запрос от пользовательского процесса, если его максимальная потребность не превышает п.

Пользователь гарантирует, что если ОС в состоянии удовлетво­рить его запрос, то все устройства будут возвращены системе в те­чение конечного времени.

Текущее состояние системы называется надёжным, если ОС мо­жет обеспечить всем процессам их завершение (выделить необходи­мые ресурсы) в течение конечного времени.

В соответствии с алгоритмом банкира выделение устройств воз­можно, только если состояние системы остаётся надёжным.

Рассмотрим надёжные и ненадёжные состояния на примере. Предположим, что в системе имеется 10 ленточных устройств и ра­ботает 3 процесса. Для завершения работы Процессу 1 потребуется 5 устройств, Процессу 2 — 9 устройств, Процессу 3 — 3 устройства. В какой-то момент времени распределение устройств процессам вы­глядит так, как показано в таблице.



Состояние системы, представленное в таблице, является надёж­ным, поскольку существует такая последовательность выделения ре­сурсов процессам, что с течением времени все потребности процессов в ресурсах будут удовлетворены, и все процессы смогут завершиться в конечное время. Для этого сперва необходимо удовлетворить по­требности Процесса 3, затем (после завершения Процесса 3), Процес­са 1, и лишь затем Процесса 2. При таком порядке удовлетворения запросов на ресурсы система будет проходить только через надёж­ные состояния.

Если же распределять оставшиеся ресурсы в другом порядке, то система перейдёт в ненадёжное состояние.

1. Предотвращения тупиков
2. Нарушение условия взаимоисключения

В общем случае избежать взаимоисключений невозможно. Но в ряде устройств это удается сделать. Например, принтер

Для предо­ставления им такой возможности в современных системах использу­ют механизм спулинга — данные для принтера от каждого процесса помещаются в буферные файлы, а собственно вывод на принтер этих файлов выполняет специальный системный процесс.

1. Нарушение условий ожидания процесса

В этом случае система требует, чтобы каждый процесс запраши­вал все требуемые ему ресурсы сразу, перед началом выполнения.

Если какой-либо ресурс не может быть предоставлен, процесс просто не может начать выполняться, и должен ждать освобожде­ния всех необходимых ему ресурсов, не удерживая при этом за собой никаких ресурсов.

Этот метод требует от программиста (пользователя) точного знания о том, какие ресурсы потребуются процессу во время выполне­ния. А эта информация не всегда доступна до начала выполнения процесса.

Кроме того, применение этого метода ведёт к непроизводитель­ному расходованию ресурсов, так как не все ресурсы, выделенные процессу в начале работы, будут нужны ему в течение всего времени работы.

Проблема: ресурсы должны быть описаны заранее, при помощи декларативных языков. Ну как в sjn, srt задавалось время.

Ну и еще одна проблема – долгое время ожидания программ с большим количеством требуемых ресурсов

1. Нарушение принципа перераспределения

Можно попробовать отобирать ресурсы у процесса до момента их завершения, но возможна потеря результатов работы, проделанных до настоящего момента. Это происходит из-за того, что не все состояния мы можем сохранить.

1. Нарушение цикла

Один из них действовать в соответствии с правилом, согласно которому каждый процесс может иметь только один ресурс в каждый момент времени. Если нужен второй ресурс - освободи первый. Очевидно, что для многих процессов это не приемлемо, например, для тех, которые распечатывают большие файлы с ленты на принтер.

Другой способ - присвоить всем ресурсам уникальные номера и потребовать, чтобы процессы запрашивали ресурсы в порядке возрастания номеров. Тогда круговое ожидание возникнуть не может.

Небольшой вариацией этого алгоритма будет нумерация в возрастающем порядке не ресурсов, а запросов процесса. После последнего запроса и освобождения всех ресурсов можно разрешить процессу опять осуществит первый запрос

**Тема: Планирование в многопроцессорных системах**

В многопроцессорных системах планировщик выполняет те же функции, что и в однопроцессорных. Но помимо этого добавляется еще одна функция: определение на каких процессорах будет выполняться процесс.

Алгоритмы планирования в многопроцессорных системах должны гарантировать, что процессоры не будут простаивать, если в очереди есть процессы.

Когда нужно определить, на каком процессоре будет выполняться процесс, алгоритм должен взять во внимание стратегию планирования

Стратегия планирования

|  |  |
| --- | --- |
| Временное распределение | Пространственное |
| -процессы задачи выполняются одновременно на разных процессорах | - процессы одной задачи размещаются на одном процессоре |
| Плюсы:  - увеличение производительности  - эффективно организована работа  Минусы:  - много обращений к памяти | Плюсы:  - уменьшение числа обращений к памяти  - увеличение скорости  Минусы:  - уменьшение производительности |
| Мягкая привязка | Жесткая привязка |

Во многих алгоритмах планирования в многопроцессорных системах процессы организуются в глобальную очередь выполнения.

Глобальные очереди выполнения можно использовать для сортировки процессов по их приоритетам, по задачам, к которым они относятся, или по тому, какой процесс выполнялся последним

Задачно-независимые алгоритмы

- пытаются разместить процесс на первом доступной процессоре

FCFS

Все процессы становятся в общую очередь. Как только освобождается процессор первый из очереди процесс встает на выполнение. Выполняться он будет до тех пор, пока не закончит или не выдаст ошибку.

Плюсы:

- прост в реализации

- все процессы будут выполнены

Минусы:

- не походит для интерактивной системы:

Процесс с наибольшем приоритетом должен ждать окончания работы процесса с меньшим приоритетом;

- неэффективное распределение ресурсов

Долгий процесс заставляет ждать медленный процесс

Круговой алгоритм

Все процессы становятся в одну очередь. При освобождении процессора на выполнение становится первый из очереди и выполняется на протяжении кванта времени. При истечении кванта времени снимается с процессора и ставится в конец очереди.

Плюсы:

- предотвращает бесконечное откладывание процесса

- немного решена проблема с долгим выполнением, как у FCFS

Минусы:

- нет привязки к процессорам

- нет высокой степени параллелизма

SPF

Процессы становятся в очередь, где сортируются по времени выполнения. В первую очередь выполняются те, которые требуют меньше времени.

Плюсы:

- перераспределение ресурсов:

Самый короткий – самый первый. Он выполнится и освободит ресурсы

Минусы:

- самый долгий процесс входит в «бесконечное» откладывание

Задачно – зависимые:

Пытаются достичь максимального распараллеливания

SNPF – наименьшее количество процессов первым

Задачи становятся в глобальную очередь выполнения. Там они сортируются по уменьшению количества процессов в этой задаче.

Плюсы:

- начинает проглядывать параллелизм: как правило процессы одной задачи будут выполнятся параллельно

Минусы:

- Все еще не параллелизм: не все процессы одной задачи выполняются параллельно

- Откладывание задачи с большим кол-вом процессов

Круговой алгоритм планирования

Все готовые к выполнению задачи ставятся в глобальную очередь. Из этой очереди каждая задача назначается группе процессоров (не обязательно, что одной и той же).

У каждой задачи есть своя очередь процессов.

На задачу выделяется p\*q времени, где p-кол-во процессоров q – квант времени. Так как в задаче не всегда кол-во процессов равно кол-ву процессоров, то алгоритм отправляет процессы на выполнение пока не израсходуется в общей сложности p\*q времени, пока не выполнится до конца или пока все процессы не будут заблокированы.

Можно еще p\*q разделить поровну между всеми процессами, позволяя каждому процессу выполняться, пока он не израсходует свой квант времени.

Альтернатива: если в задаче больше, чем р процессов, то алгоритм может выбрать р процессов для выполнения в течение кванта времени длиной q.

Плюсы:

Копланирование

Выполнять задачи параллельно, без привязки к процессорам.

Есть «окно» = кол-ву процессоров. В течении кванта времени все процессы, попавшие в это окно, выполняются. При истечении кванта времени окно сдвигается и начинает выполнение другая группа процессов. Если какой-то процесс не может выполниться в этой группе, то окно расширяется.

Динамическое распределение

Плюсы:

- уменьшает кол-во промахов в кэше: планировщик поровну распределяет процессоры между задачами

Тема: синхронизация в распределенных системах

Распределенные системы – автономные пк, которые работают совместно, как единое целое (так видно пользователю)

Преимущество:

- упрощают интеграцию приложений, работающих на разных пк, в единую систему

Недостатки:

- усложнение ПО

В процессе работы могут возникнуть конфликтные ситуации:

- невозможность одновременно получить доступ к совместно используемым ресурсам

Например, принтер. Процессы должны взаимодействовать, позволяя друг другу получать эксклюзивный доступ к ресурсу

- соглашение в порядке отправке сообщений

Например, сообщение m от процесса P должно отправиться раньше, чем n от Q

Синхронизация, используемая в однопроцессорных, не подходит, потому что нет общей памяти.

Когда мы синхронизируем 2 процесса в однопроцессорной, мы используем семафоры. Оба эти должны иметь доступ к этому семафору. Если процессы выполняются на одном пк, то они могут иметь совместный доступ к семафору, хранящемуся в ядре, делая системные вызовы. Но если они на разных пк, то этот метод не применим.

Существуют разные классы алгоритмов синхронизации:

- синхронизация часов

Проблема распределенных систем и вообще компьютерных сетей состоит в том, что для них не существует понятия единых совместно используемых часов.

Если есть машина с WWV, то синхронизироваться нужно с ней, если нет, то по возможности синхронизировать машины между собой.

Все эти алгоритмы основаны на обмене информацией. Следовательно, надо учитывать задержки на посылку и получение сообщений.

Алгоритм Кристиана

Тут есть несколько вариацией, несколько рассуждений, как считать время, но рассмотрим самый последний, который был использован в установке:

Каждую половину такта каждая машина делает запрос времени к серверу. При получении ответа вычисляется погрешность: (текущее время – время запроса – время обработки прерывания )/ 2. Потом эта поправка прибавляется ко времени сервера. Получается текущее правильно время. Далее из правильного времени вычитается текущее время в пк. Если положительно, то добавить эту поправку, если отрицательно, то приостановить

Алгоритм Беркли

Сервер (демон времени) время от времени опрашивает все машины, сколько времени у них на часах. Вычисляет среднее значение и предлагает всем машинам установить его или замедлить, если они спешат.

Применима для машин без WWV. Время демона может периодически выставляться вручную оператором

Логические часы

Тут смысл в том, что не нужно синхронизировать все машины, достаточно только те, что взаимодействуют. Это можно сделать, потому что важно не само время, а порядок выполнения.

- алгоритм голосования

Некоторые алгоритмы требуют того, чтобы кто-то из процессов был координатором. И для того, чтобы их выбрать, есть алгоритмы голосования.

Координатор – процесс, управляющий работой других процессов.

В общем плане, алгоритмы голосования пытаются найти процесс с максимальным номером и назначить его координатором.

Алгоритм забияки

Если один из процессов замечает, что текущий координатор больше не отвечает, то он отправляет сообщение всем процессам с номерами большими, чем у него «голосование».

Если никто не отвечает, то процесс становится координатором. Если отвечают, то процесс остается обычным.

Те, кто ответил, что «ок» продолжают голосование до тех пор, пока не найдут коордиантора.

В конце, когда координатор найден, он становится координатором и отправляет сообщение всем, кроме себя, что он координатор

Алгоритм кольцевой

Тут смысл тот же самый, только отправляем сообщения не всем, а по цепочке. Если не отвечает, то возвращаемся на текущую позицию и меняем номер процесса получателя. А еще пишемся в список

- взаимные исключения

Централизованный алгоритм

Все обращаются к одному процессу, который является координатором.

(например, процесс, запущенный на машине с большим сетевым адресом)

Если процесс хочет войти в КС, он делает запрос координатору. Тот смотрит, есть ли там кто-то

Распределенный алгоритм

Алгоритм работает следующим образом. Когда процесс собирается войти в критическую область, он создает сообщение, содержащее имя критической области, свой номер и текущее время. Затем он отсылает это сообщение всем процессам, концептуально включая самого себя. Посылка сообщения, как предполагается, надежная, то есть на каждое письмо приходит подтверждение в получении.

Вместо отдельных сообщений может быть использована доступная надежная групповая связь. Когда процесс получает сообщение с запросом от другого процесса, действие, которое оно производит, зависит от его связи с той критической областью, имя которой указано в сообщении. Можно выделить три варианта:

 если получатель не находится в критической области и не собирается туда входить, он отсылает отправителю сообщение ОК;

 если получатель находится в критической области, он не отвечает, а помещает запрос в очередь;

 если получатель собирается войти в критическую область, но еще не сделал этого, он сравнивает метку времени пришедшего сообщения с меткой времени сообщения, которое он отослал. Выигрывает минимальное. Если пришедшее сообщение имеет меньший номер, получатель отвечает посылкой сообщения ОК. Если его собственное

сообщение имеет меньшую отметку времени, получатель ставит приходящие сообщения в очередь, ничего не посылая при этом. После посылки сообщения-запроса на доступ в критическую область процесс приостанавливается и ожидает, что кто-нибудь даст ему разрешение на доступ. После того как все разрешения получены, он может войти в

критическую область. Когда он покидает критическую область, то отсылает сообщения ОК всем процессам в их очереди и удаляет все сообщения подобного рода из своей очереди.

Алгоритм маркерного кольца

При инициализации кольца процесс 0 получает маркер, или токен

(token). Маркер циркулирует по кольцу. Он передается от процесса k процессу k+1 (это модуль размера кольца) сквозными сообщениями.

Когда процесс получает маркер от своего соседа, он проверяет, не нужно ли ему войти в критическую область. Если это так, он входит в критическую область, выполняет там всю необходимую работу и покидает область. После выхода он передает маркер дальше. Входить в другую критическую область, используя тот же самый маркер, запрещено.

Распределенные транзакции

Закрытое рабочее пространство

Журнал с упреждающей записью

Тема: Многопроцессорные ОС.

Локальная ОС – управляет собственными ресурсами

Все что было до этого относило

Сетевая ОС является основой информационный вычислительной сети, поскольку каждый комп сети работает самостоятельно под сетевой ОС в широком смысле понимается совокупность ОС, отдельных компов, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единим правилам (протоколам). В более узком смысле сетевая ОС – ОС отдельного компа, обеспечивающая его возможность работать в сети. В сетевой ОС отдельного пк можно выделить следующие средства:

- средства управления собственными ресурсами;

Сюда входят функции распределения ОП между процессами, планирование и диспетчеризация процессов, управление переферийными устройствами и тд

- средства предоставления собственных ресурсов в общее пользование;

Это серверная часть ОС. Эта средства обеспечивают ведение справочников имен сетевых ресурсов, обработку и управление очередями запросов удаленного доступа к файловой системе, БД и переферийные устройства

- средства запроса доступа к удаленным ресурсам других компов и их использование;

Это клиентская часть ОС. Эта часть ОС выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей и осуществляет прием ответов от серверов так, что для приложения и пользователя выполнение локальных и удаленных запросов выглядит совершенно одинаково

- средства коммуникации, с помощью которых происходит обмен сообщениями в сети.

Эта часть ОС обеспечивает адресацию и буферизацию сообщений, выбор их маршрута передачи по сети, надежность передачи и другие функции. То есть является средством транспортировки сообщений.

В зависимости от назначения компа и выполняемых им функций в его ОС может отсутствовать клиентская или серверная части. Если есть 2 компа, один из котрых выполняет функции сервера, а второй – функции клиента, то их взаимодействие происходит следующим образом:

Клиентская часть ОС получает все запросы, поступающие от приложений, и анализирует их.

(в соответствии с моделью osi и происходит передача)

Если выдан запрос к ресурсу данного компа, то он переадресовывается соответствующей подсистеме локальной ОС. Если же это запрос к удаленному ресурсу, то он направляется в сеть. При этом клиентская часть osi(?) преобразует запрос из локальной формы в сетевой формат и передает его транспортной подсистеме, которая отвечает за доставку сообщений указанному серверу. Серверная часть ОС второго компа принимает запрос от транспортной подсистемы, преобразует его и передает для выполнения своей локальной части, которая в свою очередь передает его соответствующему приложению. После того как от приложения получен результат, сервер обращается к транспортной подсистеме и направляет ответ клиенту, выдавшему запрос. Клиентская часть ОС преобразует результат в соответствующий формат и адресует его тому приложению, которое выдало запрос. В зависимости от того, как распределены сетевые функции между компами, сетевые ОС (как и сети) делятся на одноранговые и 2х ранговые или сети с выделенными серверами. Если комп предоставляет свои ресурсы другим пользователям в сети, он выполняет функции сервера, а комп, обращающийся к его ресурсам, является клиентом. Если выполнение серверной функции является основным назначением компа, то он называется выделенным сервером. На выделенных серверах устанавливаются ОС оптимизированные для выполнения необходимых серверных функций. В сетях с выделенными серверами используются сетевые ОС, в состав которых входит несколько вариантов ОС, отличающихся возможностями серверных частей.

**Можно сделать доклад «Особенности организации сетевых ОС»**

Распределенные ОС

- совокупность независимых компов, которая представляется пользователю как единая система. Распределенная ОС имеет единый, глобальный межпроцессный коммуникационный механизм, глобальную схему контроля доступа, одинаковое видение файловой системы и единую очередь процессов, ожидающих выполнение.

Алгоритмы планирования в многопроцессорных системах

Многопроцессорные системы обладают рядом определенных свойств, которые отсутсвуют в однопроцессорных вычислительных системах. Например, в таких системах часто встречается удержание процессом спин блокировки (спин лок).

Спин блокировка – циклическая блокировка или низкоуровневый примитив синхронизации, применяемый в многопроцессорных системах для реализации взаимного исключения исполнения критических участков кода с использованием цикла активного ожидания. При этом другие процессоры, ждущие освобождение спин блокировки просто теряют время в циклах ожидания пока этот процесс не будет запущен снова и не отпустит блокировку. Чтобы решить данную проблему, в некоторых системах применяется так называемые умные планирования. При котором процесс, захватывающий спин блокировку устанавливает флаг, демонстрирующий что он в данный момент обладает спин блокировкой. Когда процесс освобождает спин блокировку, он также очищает и флаг. Таким образом, планировщик не останавливает процесс, удерживающий спин лок, а наоборот дает ему еще немного времени, чтобы тот завершил выполнение критической секции и, например, отпустил мьютекс (в том случае, сели взаимная блокировка выполнялась при помощи мьютекса).

Родственное планирование – планирование по уровням. Нужно чтобы обеспечить уменьшение количества промахов в кэше.

Для увеличения частоты попадания в кэш используют родственное планирование. Основная идея этого метода заключается в том, чтобы процесс был запущен на том же процессоре, что и в прошлый раз. Один из способов реализации этого метода состоит в использовании двухуровневого алгоритма планирования. В момент создания процесс назначается конкретному процессору, например, наименее загруженному в данный момент. Такое назначение процессов процессору представляет собой верхний уровень алгоритма. В результате каждый процессор получает свой набор процессов. Действительное планирование процессов находится на нижнем уровне алгоритма. Оно выполняется отдельно каждым процессором при помощи приоритетов или других средств. В стараниях удерживать процессы на одном и том же процессоре максимизирует родственность кэша. Однако если у кого -либо процессора нет работы у загруженной работой процессора отнимается процесс и отдается менее загруженному (свободному). «х уровневое планирование обладает 3 преимуществами:

1. довольно равномерно распределяет нагрузку среди имеющихся процессоров
2. 2х уровневое планирование по возможности использует преимущества родственности кэша
3. Поскольку у каждого процессора при таком варианте планирования есть свой список процессов минимизируется конкуренция за списки свободных процессов

Еще один подход к планированию в многопроцессорных системах может быть использован, если процессы связаны друг с другом каким-либо способом. Также часто случается, что один процесс создает множество потоков, работающих совместно. Простейший алгоритм работает следующим образом:

Создается целая группа, связанных процессов. В момент их создания планировщик проверяет есть ли свободные процессоры по количеству создаваемых процессов.

\*Тут не все…

**Тема: Миграция процессов**

\*Процессы активные, а не те которые стоят в очереди

Миграция процессов представляет собой перенос процесса с одного процессора на другой. Возможность исполнения процесса на любом процессоре дает множество преимуществ:

1) процессы можно переносить на слабо загруженные процессоры, таким образом уменьшать время реагирования и повышать производительность и пропускную способность.

2) возможность миграции процессов повышает отказоустойчивость. Если есть программа, которая должна без сбоев выполнять вычисления, а узел, на котором она выполняется требуется отключить или он работает неустойчиво и полученные программой результаты могут быть потеряны, миграция процессов позволит программе переместиться на другой узел и продолжить вычисления

3) возможность миграции позволяет улучшить разделение ресурсов, так как в больших системах некоторые ресурсы не продублированы во всех узлах.

4) возможность миграции процессов повышает эффективность взаимодействия: 2 процесса тесно взаимодействующих друг с другом должны выполнять на одном узле или близко расположенных узлах, чтобы уменьшить задержки при взаимодействии. Поскольку каналы связи часто формируются динамически можно воспользоваться миграцией процессов и сделать размещение процессов тоже динамическим

Чтобы миграция была успешна мигрирующие процессы должны обладать рядом свойств:

Мигрирующий процесс должен быть прозрачным, то есть его миграция не должна приводить к отрицательным последствиям. За исключением, возможно, роста времени реагирования. Другими словами, процесс не должен при миграции терять алресованных ему сообщение от других процессов или дискрипторов открытых файлов. Кроме того, система должна быть масштабирована. Так как если остаточная зависимость процесса растет с каждой миграцией система может захлебнуться в потоке сетевых сообщений от процессов, обращающихся к страницам на удаленных узлах. И наконец, достижение в технологиях связи создали потребность в гетерогенной миграции процессов. То есть процессы должны иметь возможность миграции между разными архитектурами в распределенных системах, поэтому информация о состоянии процесса должна храниться в платформенно независимом формате. Все стратегии миграции процессов должны поддерживать равновесие между потерями производительности при переносе больших объемов данных, принадлежащих процессам и выигрышем от минимизации остаточной зависимости процесса.

Стратегии миграции:

- полная миграция

- полная миграция для активных страниц

- миграция с копированием при обращении

- миграция с поздним копированием

- с предварительным копированием

Синхронизация часов

**Тема: Тупики в распределенных ОС**

Алгоритм банкира не используется.

Тупики в распределенных системах подобны тупикам в … только их сложнее обнаруживать и предотвращать. Иногда выделяют особый вид тупиков в распределенных системах – коммуникационные тупики. Для борьбы с тупиками с=в распределенных системах используют следующие стратегии:

1. Игнорирование

2. Обнаружение тупиков. Для этого 2 метода:

- централизованное обнаржуение тупиков заключается в распространении идеи графа размещения ресурсов на распределенную систему. Координатор строит единый граф для всех узлов системы и на нем выполняет редукцию или приведение.

- распределенные обнаружения тупиков. В этом методе инициирование задачи обнаружение тупика начинает процесс, который подозревает, что система находится в тупике. Он посылает тем процессам, которые держат нужный ему ресурс, сообщение состоящие из 3х информационных полей: 1 – номер процесс, инициировавшего поиск, и оно не меняется при дальнейшей пересылке; 2 и 3 = номер процесса, который зависит от ресурса и номер процесса, который держит этот ресурс соответственно. Если в конце концов сообщение вернется тому процессу, который инициировал поиск, то он приходит к выводу, что система зациклена. То есть в ней присутствует взаимоблокировка и он должен предпринять определенные действия по ликвидации тупика.

3. Предотвращение тупиков

Базируется на идее упорядочивания процессов по их временным меркам(?). Допустимой будет ситуация, когда старший процесс, то есть запущенный позже, ждет ресурсы, которые захватил младший процесс (запущенный раньше).В обратном случае младший процесс должен отказаться от ожидания.