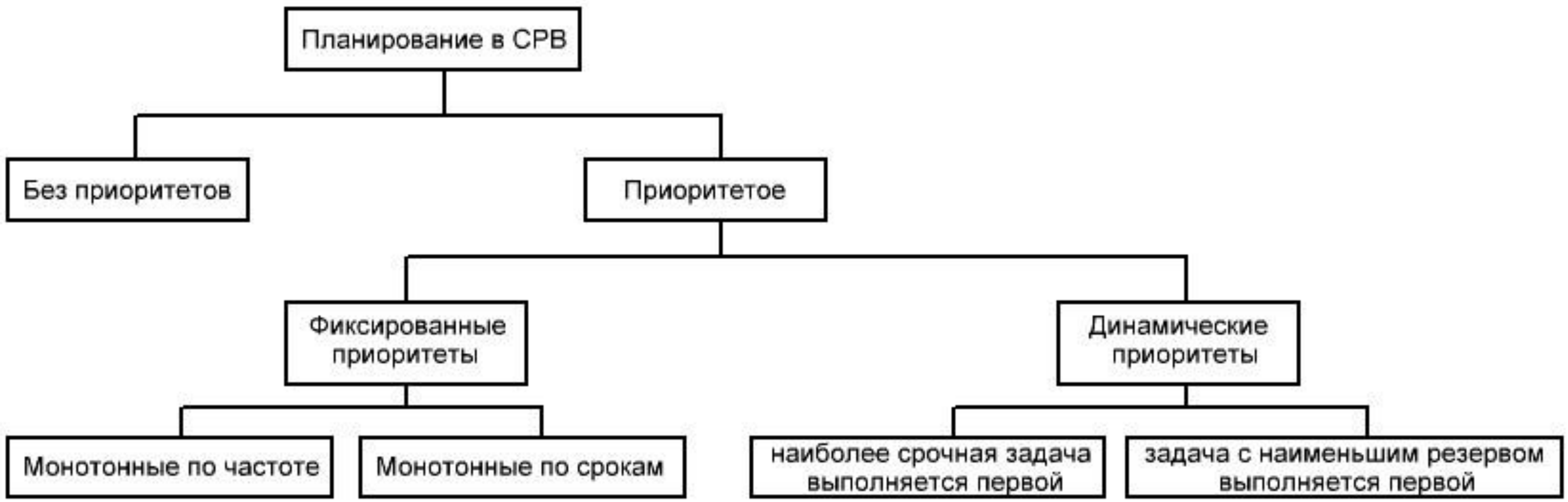
Операционные системы реального времени

Ч2

Алгоритмы планирования

Свойства механизма планирования также определяет используемый алгоритм - критерий, по которому производится упорядочение задач. На рисунке приведена диаграмма существующих алгоритмов планирования.



Классификация алгоритмов планирования

Алгоритмы планирования можно разделить на две категории согласно их поведению после прерываний.

Алгоритмы планирования без переключений (неприоритетное планирование), выбирают процесс и позволяют ему работать вплоть до блокировки либо вплоть до того момента, когда процесс сам не отдаст управление. Процесс не будет прерван, даже если он работает часами. Соответственно, решения планирования не принимаются по прерываниям от таймера. После обработки прерывания управление всегда возвращается приостановленному процессу.

Алгоритмы планирования с переключениями (приоритетное планирование), выбирают процесс и позволяют ему работать некоторое максимально возможное время. Если к концу заданного интервала времени процесс все еще работает, он приостанавливается и управление переходит к другому процессу. Приоритетное планирование требует прерывания по таймеру, происходящего в конце отведенного периода времени (решения планирования могут, например, приниматься при каждом прерывании по таймеру, или при каждом k-ом прерывании), чтобы передать правление планировщику.

Продолжительность выполнения задачи зависит от кванта времени отпущенного для её решения. Продолжительность кванта времени влияет на общую производительность системы.

1. При коротком кванте **улучшается общее время обслуживания коротких процессов.**
2. Если **величина кванта** сопоставима со **временем**, необходимым для **переключения процессов**, **то большая часть ресурсов процессора будет уходить на планирование и переключение.**
3. Если **величина кванта** слишком **большая**, **увеличивается время ожидания процессов и, соответственно, время реакции.**

Процесс исполняется до тех пор, пока не произойдет одно из следующих событий:

1. **истек** выделенный ему **квант времени;**
2. **процесс заблокирован**, например, ждет завершения операции ввода/вывода;
3. **процесс завершился**;
4. **вытеснен другим процессом**, имеющим более высокий приоритет, например, обработчиком прерываний.

Классификация планировщиков

При возникновении прерывания исполнение текущего процесса приостанавливается и проверяется, должен ли быть прерван текущий процесс и загружен новый.

Принудительная приостановка текущего процесса для передачи управления другому процессу называется **вытеснением**.

Общие принципы планирования задач РВ

1. Целью планирования является выполнение задач за заранее определенный интервал времени. Поэтому алгоритм планирования должен определить последовательность выполнения задач в соответствии с их требованиями к ресурсам и ко времени исполнения.
2. Алгоритмы планирования должны быть вытесняющими.
3. Алгоритмы планирования могут быть статическими и динамическими.

Классификация задач реального времени:

- **периодические** (входят в состояние выполнения периодически);
- **апериодические** (выполняются при возникновении определенных событий, характеризуются мягким реальным временем);
- **спорадические** (выполняются при возникновении определенных событий, характеризуются жестким реальным временем).

В общем случае задачи РВ имеют три параметра:

- период запуска;
- максимальная задержка выдачи сигнала управления;
- наихудшее время выполнения.

Стратегии выбора приоритета процесса

Для определения той или иной **стратегии необходимо принять** во внимание **несколько противоречащих друг другу факторов:**

- **общее время**, необходимое для решения задачи,
- **ограничения на время реакции**,
- **важность задачи и т.д.**

1. **Циклический метод**. Наиболее простая стратегия. Процессы выбираются последовательно один за другим в фиксированном порядке и через равные интервалы времени.
   - **Основное достоинство метода** − **простота реализации**.
   - **Недостаток** – т.к. процессы имеют различные требования к выделяемым ресурсам вычислительной системы, то **некоторые из них обслуживаются неадекватно своим потребностям** (принцип уравниловки).

2. **Метод приоритетов**. Более сложная стратегия. При каждом переключении планировщик передает **управление готовому процессу с наивысшим приоритетом.** Приоритет присваивается процессу в момент его создания и остается постоянным в течение всего времени – **статический приоритет**. **Планирование на основе статических приоритетов может привести к неприятным ситуациям.** **Процесс с наивысшим приоритетом**, если он не находится в состоянии ожидания, **будет всегда выбираться для исполнения** и **практически полностью занимать**

Выбор между процессами с одинаковым приоритетом

Между процессами с одинаковым приоритетом применяется **алгоритм динамического назначения приоритетов.**

Разница в первоначально назначенных приоритетах приводит к тому, что **процессы с более высокими приоритетами будут получать управление чаще, чем другие.** Процессы, обращение к которым происходят более интенсивно и/или время реакции которых ограничено, получают в начальный момент более высокие приоритеты; **менее важным процессам**, для которых допустима отложенная реакция, **присваиваются более низкие приоритеты.**

Планирование процессов, основанное на приоритетах, работает хорошо, только **если разные процессы имеют неодинаковые приоритеты**.

Присвоение наивысших приоритетов всем процессам не повышает скорость исполнения, так как это не увеличивает быстродействие вычислительного процесса – каждый процесс будет ждать в очереди до тех пор, пока все остальные не будут выполнены.

Система, в которой всем процессам присвоены одинаковые приоритеты, работает по циклическому принципу. **Наилучшие результаты** достигаются в системе реального времени, **если относительные приоритеты тщательно выбраны и сбалансированы.**

Межпроцессное взаимодействие

Межпроцессное взаимодействие – это тот или иной способ передачи информации из одного процесса в другой.

Каждая задача (процесс) в системе, как правило, выполняет какую-либо отдельную функцию. Задач может быть несколько. Поэтому возникает необходимость в согласовании (синхронизации) действий, выполняемых различными задачами.

Синхронизация между задачами необходима, в основном в следующих случаях:

1. Функции, выполняемые различными задачами, связаны друг с другом. Например, если одна задача подготавливает исходные данные для другой, то последняя не выполняется до тех пор, пока не получит от первой задачи соответствующего сообщения. Одна из вариаций в этом случае – это когда задача при определенных условиях порождает одну или несколько новых задач.
2. Необходимо упорядочить доступ нескольких задач к одному разделяемому ресурсу.
3. Необходима синхронизация задачи с внешними событиями. Как правило, для этого используется механизм прерываний.
4. Необходима синхронизация задачи по времени. Диапазон различных вариантов в этом случае достаточно широк, от привязки момента выдачи какого-либо воздействия к точному астрономическому времени до простой задержки выполнения задачи на определенный интервал времени. Для решения этих вопросов в конечном счете используются специальные аппаратные средства, называемые таймером.

Событие – это оповещение процесса со стороны операционной системы о той или иной форме межпроцессного взаимодействия.

Наиболее распространенными формами межпроцессного взаимодействия являются:

1. **Разделяемая память** – два или более процесса могут иметь доступ к одному и тому же блоку памяти. В системах с виртуальной памятью организация такого вида взаимодействия требует поддержки со стороны операционной системы, поскольку необходимо отобразить соответствующие блоки виртуальной памяти на один и тот же блок физической памяти.
2. **Семафоры** – два и более процесса имеют доступ к одной переменной, принимающей значение 0 или 1. Сама переменная часто находится в области данных операционной системы и доступ к ней организуется с помощью специальных функций.
3. **Сигналы** – это сообщения, доставляемые посредством операционной системы процессу. Процесс должен зарегистрировать обработчик этого сообщения у операционной системы, чтобы получить возможность реагировать на него. Часто операционная система извещает процесс сигналом о наступлении какого-либо сбоя, например, делении на 0, или о каком-либо аппаратном прерывании, например, прерывании таймера.
4. **Почтовые ящики** – это очередь сообщений (обычно – тех или иных структур данных), которые помещаются в почтовый ящик процессами и/или операционной системой. Несколько процессов могут ждать поступления сообщения в почтовый ящик и активизироваться по поступлении сообщения. Требует поддержки со стороны операционной системы.

Взаимное согласование задач с помощью сообщений является одним из важнейших принципов операционных систем реального времени.

Сообщения – это любой механизм явной передачи информации от одной задачи к другой (такие объекты, как семафоры, можно отнести к механизму неявной передачи сообщений). Объем информации, передаваемой в сообщении, может меняться от одного бита до всей свободной емкости памяти системы.

Во многих ОСРВ компоненты операционной системы, также как и пользовательские задачи, способны принимать и передавать сообщения.

Сообщения могут быть:

- **асинхронными**. В этом случае доставка сообщений задаче производится после того, как она в плановом порядке получит управление;
- **синхронными**. В этом случае циркуляция сообщений оказывает непосредственное влияние на планирование задач.
    - задача, пославшая какое-либо сообщение, немедленно блокируется, если для продолжения работы ей необходимо дождаться ответа от другого процесса;
    - если низкоприоритетный процесс шлёт высокоприоритетному сообщение, которого последний ожидает;
    - сообщения передаются через почтовый ящик (буфер определенного размера). При этом, как правило, новое сообщение затирает старое, даже если последнее не было обработано.

Наиболее часто используется принцип, когда каждая задача имеет свою очередь сообщений, в конец которой ставится всякое вновь полученное сообщение.

Стандартный принцип обработки очереди сообщений осуществляется по принципу FIFO, хотя это может не всегда оптимально соответствать поставленной задаче.

Ресурсы и их характеристики

Ресурс – это общий термин, описывающий объект (физическое устройство, область памяти), который может одновременно использоваться только одной задачей.

По своим характеристикам ресурсы разделяют на:

- **активные** – способны изменить информацию (процессор),
- **пассивные** – способны хранить информацию,
- **локальные** – принадлежат одному процессу; время жизни совпадает с временем жизни процесса ,
- **разделяемые** – могут быть использованы несколькими процессами; существуют, пока есть хоть один процесс, который их использует,
- **постоянные** – используются посредством операций «захватить» и «освободить»,
- **временные** – используются посредством операций «создать» и «удалить».

Разделяемые ресурсы бывают:

- **не критичные** – могут быть использованы одновременно несколькими процессами (например, жесткий диск),
- **критичные** – могут быть использованы только одним процессом, и пока этот процесс не завершит работу с ресурсом, последний не доступен другим процессам (например, разделяемая память, доступная на запись).

По типу взаимодействия различают:

- **сотрудничающие процессы**:
  - процессы, разделяющие только коммуникационный канал, по которому один передает данные, а другой получает их;
  - процессы, осуществляющие взаимную синхронизацию: когда работает один, другой ждет окончания его работы (типично для программ, управляющих рядом технологических процессов)
- **конкурирующие процессы**:
  - процессы, использующие совместно разделяемый ресурс;
  - процессы, использующие критические секции;
  - процессы, использующие взаимные исключения.

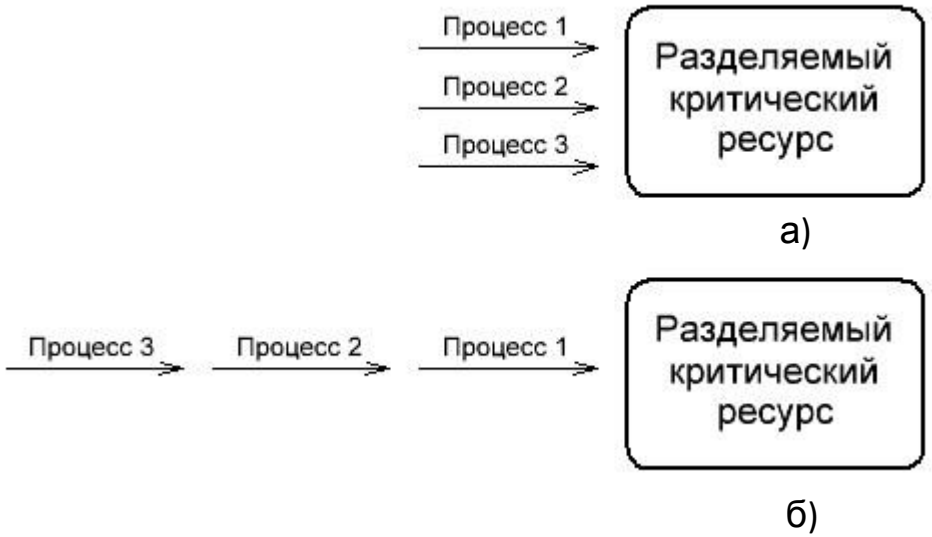


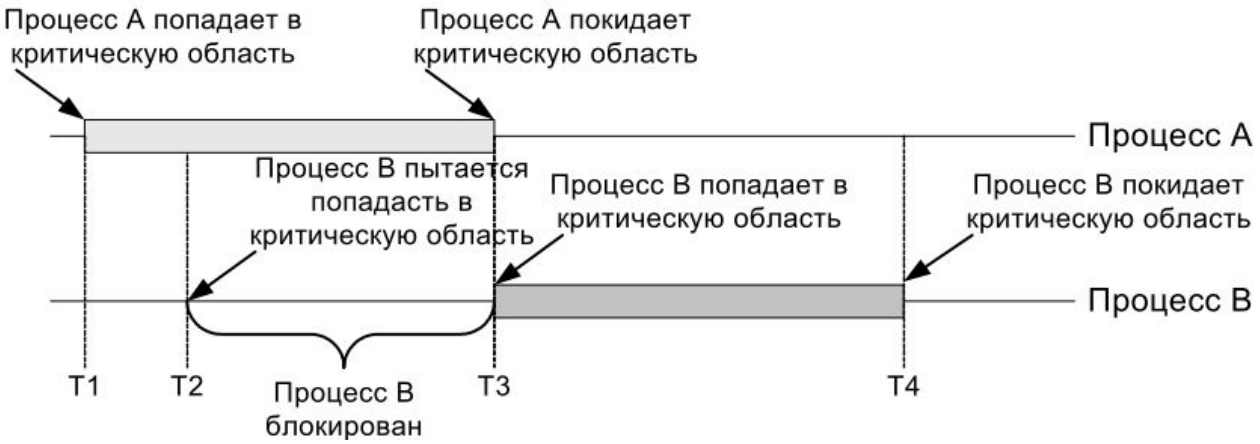
Рисунок 1 – Виды разделяемых ресурсов. А) сотрудничающие процессы, б) конкурирующие процессы

Проблемы взаимодействия процессов

Состязания – это ситуация, в которой два процесса считывают или записывают данные одновременно и конечный результат зависит от того, какой процесс был первым.
Основным способом предотвращения состояния состязания является запрет использования совместно используемых данных одновременно нескольким процессам.

Критическая секция (области) – это участок программы, в котором есть обращение к совместно используемым данным. Критическая секция может в один момент времени принадлежать только одному потоку. На этом участке запрещается переключение задач для обеспечения исключительного использования ресурсов процессом. Все ОСРВ предоставляют вызовы «войти в критическую секцию» и «выйти из критической секции».

Взаимное исключение (mutual exclusion, mutex) – это способ синхронизации параллельно работающих процессов, использующих разделяемый постоянный критичный ресурс. Если ресурс занят, то системный вызов «захватить ресурс» переводит процесс из состояния «выполнение» в состояние «ожидание». Когда ресурс будет освобожден посредством системного вызова «освободить ресурс», то этот процесс вернется в состояние «выполнение» и продолжит работу. Ресурс при этом перейдет в состояние «занят».



При взаимодействии процессов возможны следующие проблемы:

- **Зацикливание** (зависание, тупик (Deadlock)). Причина этого может быть достаточно проста – «задачи не поделили ресурсы». Пусть, например, Задача A захватила ресурс клавиатуры и ждет, когда освободится ресурс дисплея, а в это время Задача B, успев захватить ресурс дисплея, ждет теперь, когда освободится клавиатура. В таких случаях рекомендуется придерживаться тактики «или все, или ничего». Иначе, если задача не смогла получить все необходимые для дальнейшей работы ресурсы, она должна освободить все, что уже захвачено, и повторить попытку только через определенное время.
- **Инверсия приоритетов** (Priority inversion). Пусть имеются: высокоприоритетная Задача A, среднеприоритетная Задача B и низкоприоритетная Задача C.

  Пусть в начальный момент времени Задачи A и B блокированы в ожидании какого-либо внешнего события. Допустим, получившая в результате этого управление Задача C захватила Ресурс A, но не успела его отдать, как была прервана Задачей A. В свою очередь, Задача A при попытке захватить Ресурс A будет блокирована, так как этот ресурс уже захвачен Задачей C. Если к этому времени Задача B находится в состоянии готовности, то управление после этого получит именно она, как имеющая более высокий, чем у Задачи C, приоритет. Теперь Задача B может занимать процессорное время, пока ей не надоест, а получается ситуация, когда высокоприоритетная Задача A не может функционировать из-за того, что необходимый ей ресурс занят низкоприоритетной Задачей C.
- **Блокировка** (Lockout). Процесс ожидает ресурс, который никогда не освободится.
- **Голодовка** (Starvation). Процесс монополизировал процессор.

Семафор – это объект синхронизации, задающий количество пользователей (задач, процессов), имеющих одновременный доступ к некоторому ресурсу.

С каждым семафором связаны:

- **счетчик** (значение семафора) и
- **очередь ожидания** (процессы ожидающие принятие счетчиком определенного значения).

Различают:

- **двоичные** (булевские) **семафоры** – это механизм взаимного исключения для защиты критичного разделяемого ресурса;
- **счетные семафоры** – это механизм взаимного исключения для защиты ресурса, который может быть использован не более чем ограниченным фиксированным числом задач *n*.

Все операции над семафорами выполняются как единое и неделимое элементарное действие.

Это гарантирует, что после начала операции ни один процесс не получит доступа к семафору до окончания или блокирования операции.

Основными операциями над семафорами являются операции «*взять*» и «*вернуть*».

В начале работы семафор должен быть создан, в конце работы – уничтожен.

Над семафорами определены следующие элементарные операции:

- **взять** k единиц из семафора, т.е. уменьшить счетчик на k. Если в счетчике нет k единиц, то эта операция переводит задачу в состояние «ожидания» наличия как минимум k единиц в семафоре, и добавляет ее в конец очереди ожидания этого семафора. Значение счетчика в таком случае становится равным нулю (отрицательных значений счетчика быть не может).
- **вернуть** k единиц в семафор, т.е. увеличить счетчик на k. Если с семафором связаны один или несколько ожидающих процессов, которые не могут завершить более раннюю операцию «взять», один из них выбирается системой и ему разрешается завершить свою операцию. Таким образом, после операции «вернуть», примененной к семафору, связанному с несколькими ожидающими процессами, значение счетчика так и остается равным 0, но число ожидающих процессов уменьшается. Активизированный процесс в случае более высокого приоритета может вытеснить текущую задачу.
- **попробовать взять** k единиц из семафора. Если в счетчике ≥ k единиц, то взять k единиц из него, иначе вернуть признак занятости семафора без перевода задачи в состояние ожидания.
- **проверить семафор**, т.е. получить значение счетчика.
- **блокировать семафор**, т.е. взять из него столько единиц, сколько в нем есть. При этом иногда бывают две разновидности этой операции:
  - взять столько, сколько есть в данный момент, или
  - взять столько, сколько есть в начальный момент, т.е. максимально возможное количество, поскольку такая задача будет монопольно владеть ресурсом.
- **разблокировать семафор**, т.е. вернуть столько единиц, сколько всего было взято данной задачей по команде «блокировать».

Мьютекс (mutex, взаимное исключение) – это упрощенная версия семафора.

Мьютекс не способен считать, он может лишь управлять взаимным исключением доступа к совместно используемым ресурсам или кодам.

Мьютекс фактически состоит из пары:

- булевского семафора и
- идентификатора задачи – текущего владельца семафора (т.е. той задачи, которая успешно выполнила функцию «взять» и стала владельцем разделяемого ресурса).

Для доступа к объекту типа mutex определены три примитивные операции:

- **блокировать мьютекс** m. Если m уже заблокирован другой задачей, то эта операция переводит задачу в состояние ожидания разблокирования m.
- **разблокировать мьютекс** m. Если m ожидается другой задачей, то она может быть активизирована, удалена из очереди ожидания и может вытеснить текущую задачу, если ее приоритет выше. Если вызвавшая эту операцию задача не является владельцем m, то операция не имеет никакого эффекта.
- **попробовать блокировать мьютекс** m. Если m не блокирован, то эта операция эквивалентна Lock(m), иначе возвращается признак неудачи.

Как и для семафоров, эти операции являются неделимыми.