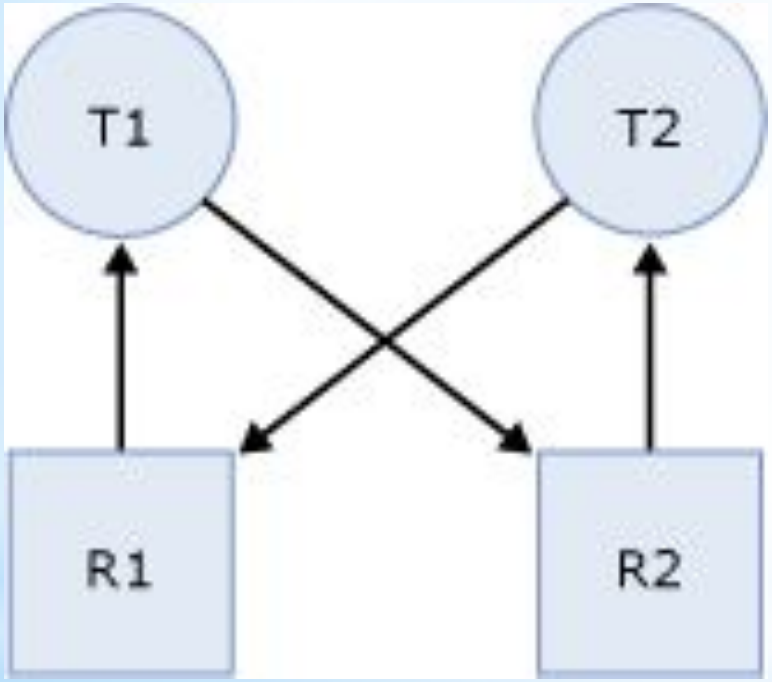
Тема 4.3 Механизмы синхронизации и взаимоблокировка ресурсов в многозадачных системах

* **ВОПРОСЫ:**

  1. Решение задачи взаимоблокировки ресурсов.
  2. Механизмы синхронизации: семафоры, мьютексы, мониторы, сообщения.

1. Решение задачи взаимоблокировки ресурсов.

Взаимоблокировка возникает, когда две и более задач постоянно блокируют друг друга из-за того, что задача каждой из сторон блокирует ресурс, необходимый другой стороне.



Задача T1 блокирует ресурс R1 (изображается в виде стрелки от R1 к T1) и запросила блокировку ресурса R2 (изображается в виде стрелки от T1 к R2).

Задача T2 блокирует ресурс R2 (изображается в виде стрелки от R2 к T2) и запросила блокировку ресурса R1 (изображается в виде стрелки от T2 к R1).

Так как ни одна из задач не может продолжиться до тех пор, пока не освободится ресурс, а ни один из ресурсов не может быть освобожден до тех пор, пока не продолжится задание, существует состояние взаимоблокировки.

1. Решение задачи взаимоблокировки ресурсов.

С рассматриваемой здесь точки зрения ресурсы могут быть разделены на два типа: **выгружаемые и невыгружаемые**.

Выгружаемый ресурс можно безболезненно забрать у владеющего им процесса. Образцом такого ресурса является память.

Невыгружаемый ресурс, в противоположность выгружаемому, - это такой ресурс, который нельзя забрать у текущего процесса без уничтожения результатов вычислений.

1. Решение задачи взаимоблокировки ресурсов.

Невыгружаемый ресурс

Пример

Если в момент записи компакт-диска внезапно забрать у процесса устройство для записи и передать его другому процессу, в результате получим испорченный компакт-диск.

Взаимоблокировки касаются невыгружаемых ресурсов. Потенциальные тупиковые ситуации, в которые вовлечены выгружаемые ресурсы, обычно можно разрешить с помощью перераспределения ресурсов от одного процесса другому.

1. Решение задачи взаимоблокировки ресурсов.

Коффман и другие исследователи доказали, что для возникновения тупиковой ситуации должны выполняться четыре условия одновременно.

1. Условие взаимного исключения. Каждый ресурс в данный момент или отдан ровно одному процессу, или доступен.
2. Условие удерживания и ожидания. Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.
3. Условие отсутствия принудительной выгрузки ресурсов. У процесса нельзя забрать принудительно ранее полученные ресурсы. Процесс, владеющий ими, должен сам освободить ресурсы.
4. Условие циклического ожидания. Должна существовать круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

1. Решение задачи взаимоблокировки ресурсов.

При столкновении с взаимоблокировками используются четыре стратегии.

1. Пренебрежение проблемой в целом.
2. Обнаружение и восстановление. Позволить взаимоблокировке произойти, обнаружить ее и предпринять какие-либо действия.
3. Избегать тупиковых ситуаций с помощью аккуратного распределения ресурсов.
4. Предотвращать с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки.

1. Решение задачи взаимоблокировки ресурсов.

При столкновении с взаимоблокировками используются четыре стратегии.

1. Пренебрежение проблемой в целом.

Если взаимоблокировки случаются в среднем раз в пять лет, а сбои ОС, компиляторов и неисправности аппаратуры происходят в среднем один раз в неделю, то подходит первая стратегия. К этому надо добавить, что большинство операционных систем потенциально страдают от взаимоблокировок, которые не обнаруживаются, не говоря уже об автоматическом выходе из тупика.

1. Решение задачи взаимоблокировки ресурсов.

При столкновении с взаимоблокировками используются четыре стратегии.

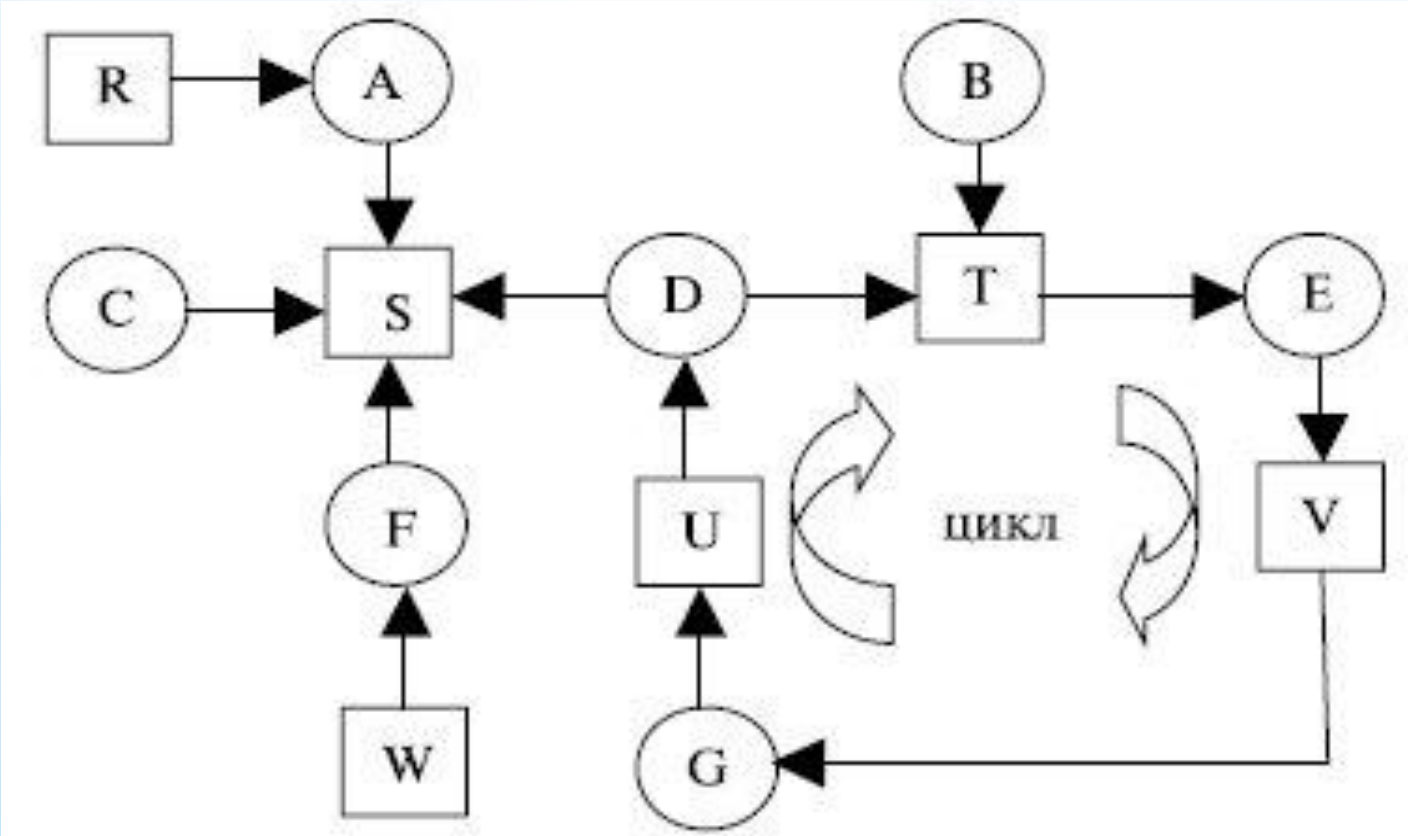
2. Обнаружение и восстановление.

При использовании этого метода система не пытается предотвратить попадания в тупиковые ситуации. Вместо этого она позволяет произойти взаимоблокировке, старается определить, когда это случилось, и затем совершает некоторые действия по возврату системы к состоянию, имевшему место до того, как система попала в тупик.

1. Решение задачи взаимоблокировки ресурсов.

2. Обнаружение и восстановление.

Для обнаружения взаимоблокировки при наличии одного ресурса каждого типа достаточно построить граф ресурсов и процессов. Если в графе нет циклов, система в тупик не попала.



1. Решение задачи взаимоблокировки ресурсов.

2. Обнаружение и восстановление.

Операционной системе необходима реализация формального алгоритма, выявляющего тупики.

Алгоритм обнаружения взаимоблокировок основан на сравнении векторов доступных и требуемых ресурсов.
В исходном состоянии все процессы не маркированы (не отмечены). По мере реализации алгоритма на процессы будет ставиться отметка, служащая признаком того, что они могут закончить свою работу и, следовательно, не находятся в тупике. После завершения алгоритма любой немаркированный процесс находится в тупиковой ситуации.

1. Решение задачи взаимоблокировки ресурсов.

2. Обнаружение и восстановление.

Предположим, обнаружен тупик. Какие методы можно использовать для его ликвидации? Здесь возможно несколько подходов.

Первый – принудительная выгрузка ресурсов: способность забирать ресурс у процесса, отдавать его другому процессу, а затем возвращать назад так, что исходный процесс не замечает того, в значительной мере зависит от свойств ресурса. Выйти из тупика, таким образом, зачастую трудно или невозможно.

1. Решение задачи взаимоблокировки ресурсов.

2. Обнаружение и восстановление.

Предположим, обнаружен тупик. Какие методы можно использовать для его ликвидации? Здесь возможно несколько подходов.

Второй подход – восстановление через откат. В этом способе процессы должны периодически создавать контрольные точки, позволяющие запустить процесс с его предыстории. Когда взаимоблокировка обнаружена, достаточно просто понять, какие ресурсы нужны процессам. 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. Вся работа, выполненная после этой контрольной точки, теряется. Если возобновленный процесс вновь пытается получить данный ресурс, ему придется ждать, когда ресурс станет доступным.

2. Обнаружение и восстановление.

Предположим, обнаружен тупик. Какие методы можно использовать для его ликвидации? Здесь возможно несколько подходов.

Третий подход – восстановление путем уничтожения одного или более процессов. Это грубейший, но простейший выход из тупика. Проблема – решить, какой процесс уничтожать.

1. Решение задачи взаимоблокировки ресурсов.

2. Обнаружение и восстановление.

Идеальной была бы такая организация вычислительного процесса, при которой не возникали бы тупики за счет безопасного распределения ресурсов. Подобные алгоритмы базируются на концепции безопасных состояний.

Таблица методов IPC

Метод	Реализуется (операционной системой или другим окружением)
Файл	Все операционные системы.
Сигнал	Большинство операционных систем; некоторые системы, как например, Windows, только реализуют сигналы в библиотеке запуска Си, но не обеспечивают их полноценной поддержки для использования методов IPC.
Сокет	Большинство операционных систем.
Канал	Все системы, соответствующие POSIX.
Именованный канал	Все системы, соответствующие POSIX.
Семафор	Все системы, соответствующие POSIX.
Разделяемая память	Все системы, соответствующие POSIX.
Обмен сообщениями (без разделения)	Используется в парадигме MPI, Java RMI, CORBA и других.
Проецируемый в память файл	Все системы, соответствующие POSIX; несет риск появления состояния гонки в случае использования временного файла. Windows также поддерживает эту технологию, но использует API отличный от POSIX.
Очередь сообщений	Большинство операционных систем.
Почтовый ящик	Некоторые операционные системы.

Семафоры

Семафоры

- Семафоры – примитивы синхронизации предложены Дейкстрой (Dijkstra) в 1968 г. в качестве компонента операционной системы THE
- Семафор - это целочисленная неотрицательная переменная, для которой определены 2 операции: **P** (от датского слова proberen – проверять) и **V** (от verhogen – увеличивать).
- Операции **P** и **V** выполняются **атомарно**

Семафоры

Классическое определение этих операций выглядит следующим образом:

P(S): пока S == 0 процесс блокируется;
S = S - 1;
V(S): S = S + 1;

Эта запись означает следующее: при выполнении операции P над семафором S сначала проверяется его значение. Если оно больше 0, то из S вычитается 1. Если оно меньше или равно 0, то процесс блокируется до тех пор, пока S не станет больше 0, после чего из S вычитается 1. При выполнении операции V над семафором S к его значению просто прибавляется 1.

Виды семафоров

Двоичный семафор

- S может принимать значения 0 и 1, инициализируется значением 1
- обеспечивает эксклюзивный доступ к ресурсу (например, при работе в критической секции)
- одновременно может выполняться только один поток

Виды семафоров

Счетный семафор

- S инициализируется значением N (число доступных единиц ресурса)
- представляет ресурсы, состоящие из нескольких однородных элементов
- позволяет потокам исполняться, пока есть неиспользуемые элементы

Виды семафоров

Мьютексы

Мьютекс – двоичный семафор, обычно используемый для организации согласованного доступа к неделимому общему ресурсу. Мьютекс может принимать значения 1 (свободен) и 0 (занят).

Операции над мьютексами

- acquire(mutex) – уменьшить (занять) мьютекс
- release(mutex) – увеличить (освободить) мьютекс
- tryacquire(mutex) – часто реализуемая неблокирующая операция, выполняющая попытку уменьшить (занять) мьютекс

Виды семафоров

Мьютексы

Мьютексы в конкретных реализациях могут иметь дополнительные свойства

- Запоминание владельца – освободить мьютекс может только поток, захвативший его
- Рекурсивность – поток может многократно захватить мьютекс (вызывать aquire()); для освобождения мьютекса поток должен соответствующее число раз вызвать release()
- Наследование приоритета – поток, захвативший мьютекс, временно наследует максимальный из приоритет потоков, ждущих освобождения данного мьютекса

Семафоры – Итоги

1. С помощью семафоров можно решить любую классическую задачу синхронизации, но по сути, семафоры - это разделяемые глобальные переменные (что является признаком плохой структуры программы)
2. Семафоры используются как для решения задачи взаимного исключения, так и для координации действий
3. Отсутствует связь между семафором и данными, доступом к которым он управляет
4. Нет контроля использования семафоров со стороны компилятора или операционной системы, соответственно - нет гарантий, что семафоры будут использованы правильно

Мониторы

2. Механизмы синхронизации: семафоры, мьютексы, мониторы, сообщения.

В сложных программах произвести анализ правильности использования семафоров с карандашом в руках становится очень непростым занятием. В то же время обычные способы отладки программ зачастую не дают результата, поскольку возникновение ошибок зависит от interleaving'a атомарных операций, и ошибки могут быть трудно воспроизводимы. Для того чтобы облегчить труд программистов, в 1974 году Хоаром (Hoare) был предложен механизм еще более высокого уровня, чем семафоры, получивший название **мониторов.**

Мониторы

- Мониторы – высокоуровневый механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к неразделяемым ресурсам.
- Предложены Хоаром (Hoare) в 1974 г.
- Пер Бринч Хансен первым, описал и реализовал мониторы, основывая их на идеях Хоара.

Мониторы

Монитор – это конструкция языка программирования, поддерживающая управляемый доступ к разделяемым данным.

Монитор инкапсулирует:

- разделяемые критические данные;
- функции, использующие разделяемые данные;
- синхронизацию выполнения параллельных потоков, вызывающих указанные функции.

Мониторы

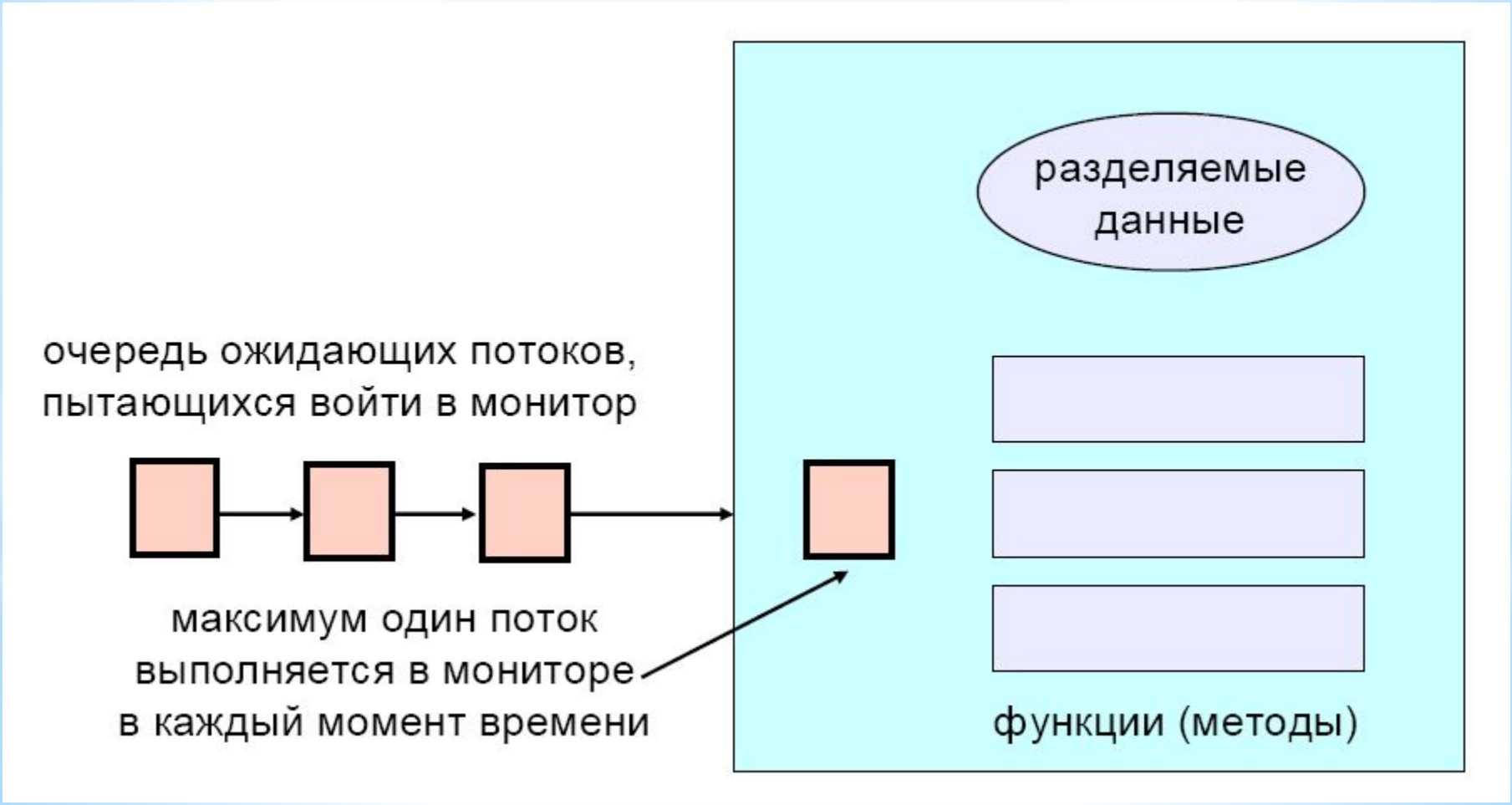
- Доступ к переменным монитора, реализуется **только посредством** вызова предоставленных функций.
- Только **один поток** может находиться в мониторе в любой момент времени, если второй поток пытается вызвать метод монитора, он переходит в состояние ожидания до выхода из монитора первого потока.
- Код синхронизации добавляется **компилятором**.

Мониторы

Языки программирования, поддерживающие мониторы:

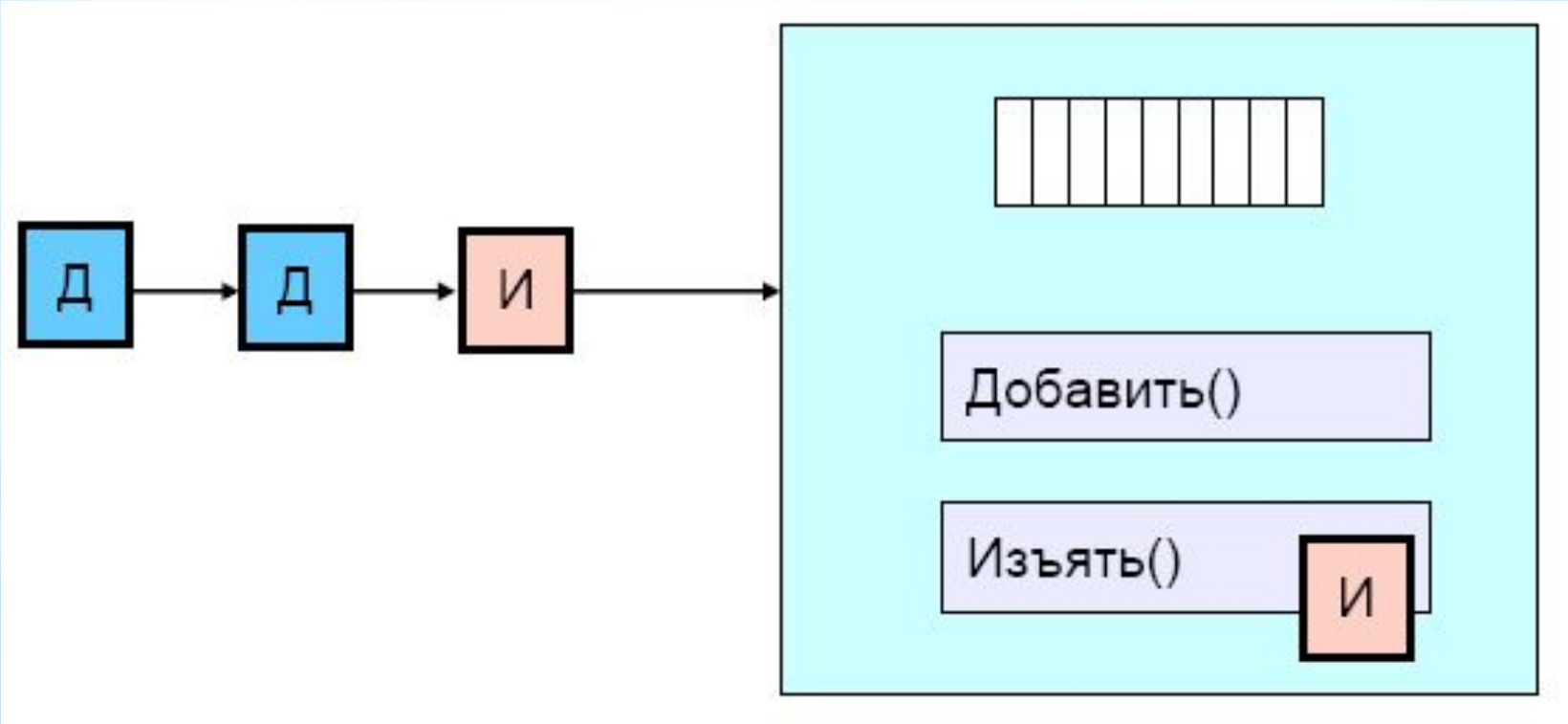
- Ада
- C# (и другие языки, использующие .NET Framework)
- Concurrent Pascal
- D
- Java (с помощью ключевого слова synchronized)
- Mesa
- Модула-3
- Ruby
- Squeak Smalltalk
- uC++
- и др.

Мониторы


разделяемые данные
очередь ожидающих потоков, пытающихся войти в монитор
максимум один поток выполняется в мониторе в каждый момент времени
функции (методы)

Мониторы

Тупиковая ситуация при использовании монитора



Мониторы

Условные переменные

Условная переменная (conditional variable) символизирует ожидание потоком, выполняющим метод монитора, наступления некоторого события.

Мониторы

Операции над условными переменными
wait(cv) – выполняется потоком, который хочет подождать наступления события. Снимает блокировку монитора, после чего другой поток может войти в него; ожидает, пока какой-либо другой поток не подаст условный сигнал.
signal(cv) – выполняется потоком, сигнализирующем о наступлении события. Пробуждает максимум один ожидающий поток; если отсутствуют потоки, ожидающие события, информация о приходе сигнала теряется.
broadcast(cv) – также выполняется потоком, сигнализирующем о наступлении события. Пробуждает все ожидающие события потоки.

Мониторы Хоара и мониторы Меса

Мониторы Хоара обрабатывают вызов signal(cv) следующим образом:

- немедленно запускается поток, ожидавший сигнала;
- поток, пославший сигнал, блокируется и остается блокированным все время, пока выполняется поток, который он вывел из состояния ожидания.

Мониторы Меса обрабатывают вызов signal(cv) несколько другим способом:

- ожидающий поток переводится в состояние "готов к выполнению", а поток, пославший сигнал, продолжает исполнение;
- ожидавший поток запускается при выходе потока, пославшего сигнал, из монитора или его перехода в состояние ожидания.

Мониторы

Реализация ожидания выполнения события

Монитор Хоара:

```
if (not Ready)
wait(c);
```

Монитор Меса:

```
while (not Ready)
wait(c);
```

Сообщения

Сообщения

Сообщения –способ межпроцессного и межпоточного взаимодействия, позволяющий потокам подавать сигналы друг другу и обмениваться данными.

Используется для организации взаимодействия потоков, выполняющихся на различных узлах сети.

Сообщения

Типы доставки сообщений

Асинхронный
поток, посылающий сообщение, инициирует процесс доставки сообщения, после чего продолжает свою работу

Синхронный
поток, пославший сообщение, дожидается подтверждения его получения принимающим потоком

Сообщения

Пример

- Имеются разделяемые данные, над которыми требуется многократно выполнить некоторую операцию.
- Есть два потока, которые выполняют данную операцию над частями разделяемых данных.
- Каждое следующее выполнение операции требует, чтобы предыдущее было полностью завершено обоими потоками.

Сообщения

Адресация

Прямая
send(P, message) – послать сообщение message процессу P
receive(Q, message) – получить сообщение message от процесса Q

Непрямая
send(A, message) – послать сообщение message в буфер A
receive(A, message) – получить сообщение message из буфера A