

Операционные системы. Лекция 4

■ Синхронизация параллельных процессов

1. Назначение синхронизации

2. Типичные задачи синхронизации

- задача взаимного исключения

- задача производители-потребители

- задача «читатели-писатели»

- задача кругового распределения ресурсов

3. Механизмы синхронизации

- аппаратная реализация взаимоисключений

- программная реализация взаимоисключений

Назначение синхронизации процессов

- **упорядочение развития процессов во времени** в зависимости от типа отношения между процессами
(отношение предшествования, отношение приоритетности, отношение взаимного исключения)
- **взаимодействие между процессами**, выражающееся в передаче информации между ними
(отношение «производитель-потребитель», отношение «читатель-писатель»)

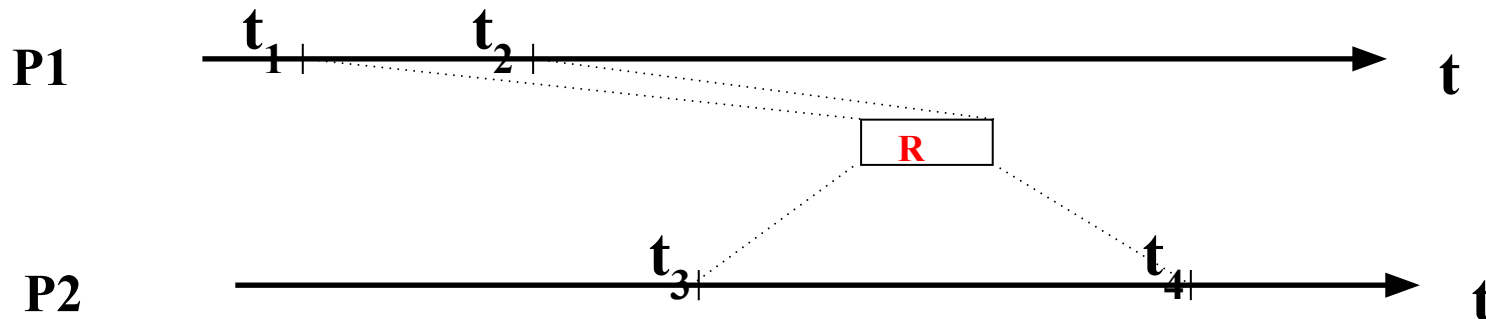
Отношения между процессами, влияющие на синхронизацию процессов

| | |
|--|---|
| Отношение предшествования | Для двух процессов это означает, что первый процесс должен переходить в состояние выполнения всегда раньше второго |
| Отношение приоритетности | Процесс с приоритетом P может быть переведен в состояние выполнения только при соблюдении двух условий: – в состоянии готовности к рассматриваемому процессору нет процессов с большим приоритетом; – процессор либо свободен, либо используется процессом с меньшим, чем P приоритетом |
| Отношение взаимного исключения | Устанавливается для процессов, использующих общий ресурс. При этом совокупность действий над этим ресурсом в составе одного процесса называют <i>критическим интервалом</i> . <i>Критический интервал одного процесса не должен выполняться одновременно с критическим интервалом над этим же ресурсом в составе другого процесса</i> |
| Отношение «производитель-потребитель» | Устанавливается для двух процессов с жестко распределенными между ними функциями. Один процесс вырабатывает сообщения, предназначенные для восприятия и обработки другим. |
| Отношение «читатели-писатели» | – <i>процессы-писатели</i> могут записывать информацию в область памяти <i>процессы-читатели</i> считывают информацию из области памяти |

Задача взаимного исключения

Необходимо согласовать работу параллельных процессов при использовании критического ресурса, чтобы удовлетворить следующим требованиям:

- Одновременно внутри критического интервала должно находиться не более одного процесса
- Освобождение критического ресурса и выход из критического интервала должно быть произведено за конечное время



R – критический ресурс (воспроизводимый, последовательно-используемый)

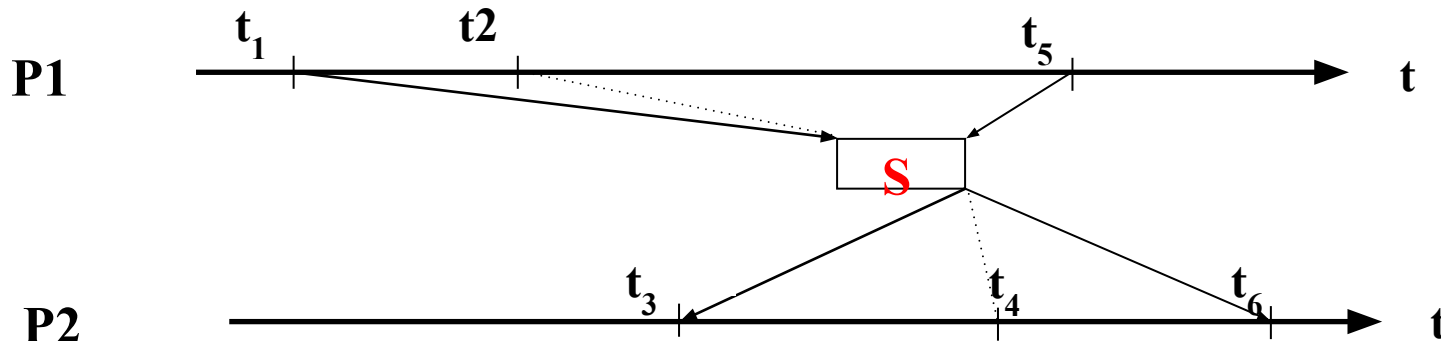
$t_1 - t_2$ критический интервал процесса P1

$t_3 - t_4$ критический интервал процесса P2

Задача производителя-потребителя

Необходимо согласовать выполнение параллельных процессов при обмене сообщениями таким образом, чтобы удовлетворить следующим требованиям:

- Выполнять требования задачи взаимного исключения по отношению к критическому ресурсу – общей области памяти для хранения сообщения
- Учитывать состояние общей области памяти, характеризующее возможность или невозможность посылки(принятия) очередного сообщения



S - потребляемый ресурс

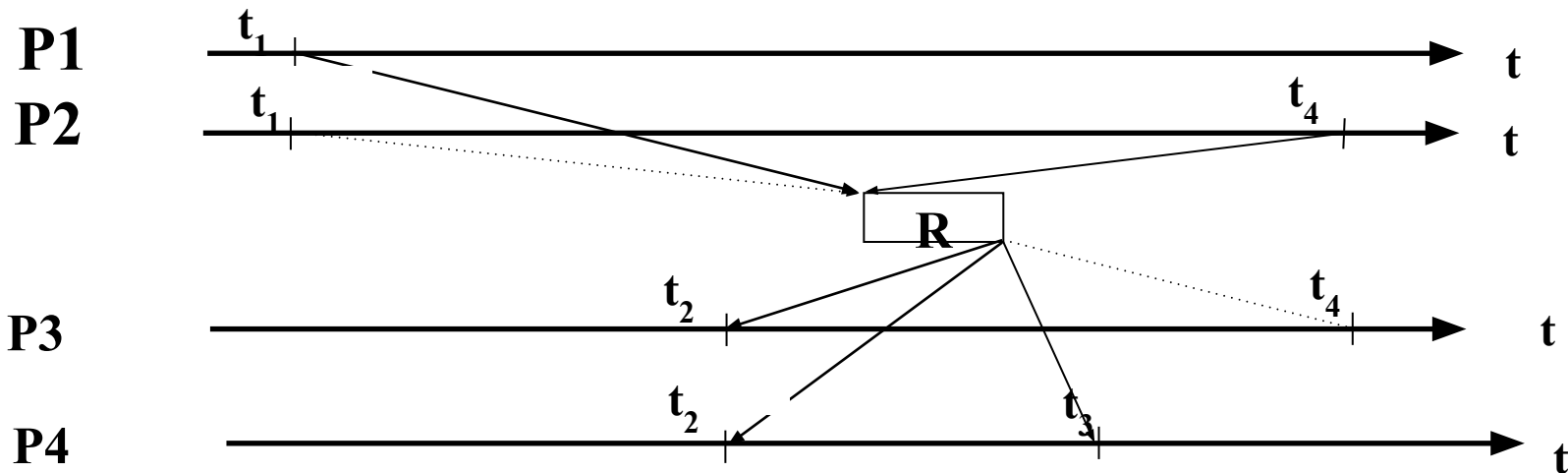
P1 – процесс-производитель

P2 – процесс-потребитель

Задача писатели-читатели

Необходимо согласовать работу процессов-писателей и процессов читателей:

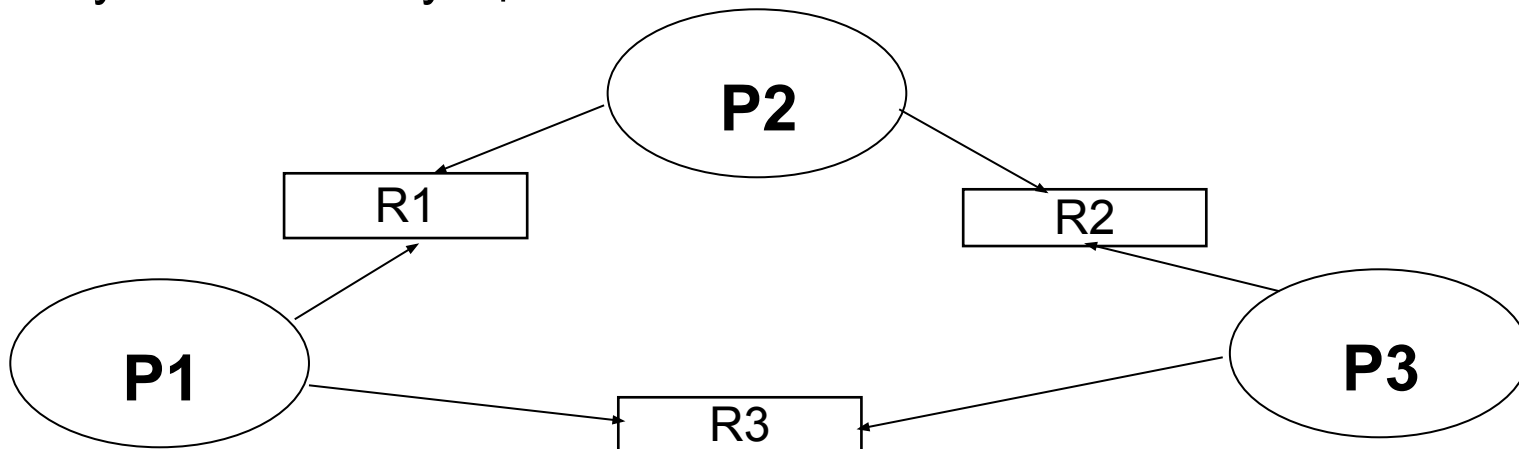
- Выполняя требования задачи взаимного исключения по отношению к критическому ресурсу – общей области памяти для хранения информации
- Учитывая приоритетность использования общей области памяти различными типами процессов



S – критический ресурс P1, P2 – процессы-писатели P3, P4 – процессы-читатели

Задача кругового распределения ресурсов

Необходимо обеспечить максимально параллельное и правильное развитие процессов при круговом распределении ресурсов, упорядочивая действия процессов по захвату ресурсов во избежание возможных блокировок одними процессами других и тупиковых ситуаций



P1, P2, P3 - параллельные процессы

R1,R2, R3 - последовательно-используемые ресурсы

Процессу **P1** требуются ресурсы

R1,R3

Процессу **P2** требуются ресурсы

R1,R2

Процессу **P3** требуются ресурсы

R2,R3

Аппаратная реализация задачи взаимного исключения

1)блокировка памяти.

Все ВС имеют основную форму аппаратной реализации взаимного исключения- **блокирование памяти.**

Блокировка памяти - запрет одновременного исполнения двух и более команд, которые обращаются к одной и той же ячейке памяти.

Если в этой ячейке хранится значение критической переменной, то получить доступ к ней может только один процесс.

Аппаратная реализация задачи взаимного исключения

2) команда проверка и установка (test&set, TS).

Операция «ПРОВЕРКА И УСТАНОВКА» является, как и блокировка памяти, одним из аппаратных средств решения задачи критического интервала.

Команда TS является аппаратно-поддерживаемой составной командой.

Команда TS является неделимой операцией, то есть между ее началом и концом не могут выполняться никакие другие команды.

$F(D)$ – блокирующая переменная ресурса D (0- свободен, 1 – занят)

Выполнение команды:

Перед входом в критический интервал процесс выполняет команду TS:

1. Циклически проверяется $F(D)=0$ (TEST)
2. Если $F(D)=0$ то $F(D)=1$ (SET)
3. Процесс входит в критическую секцию
4. После выполнения критической секции $F(D)=0$

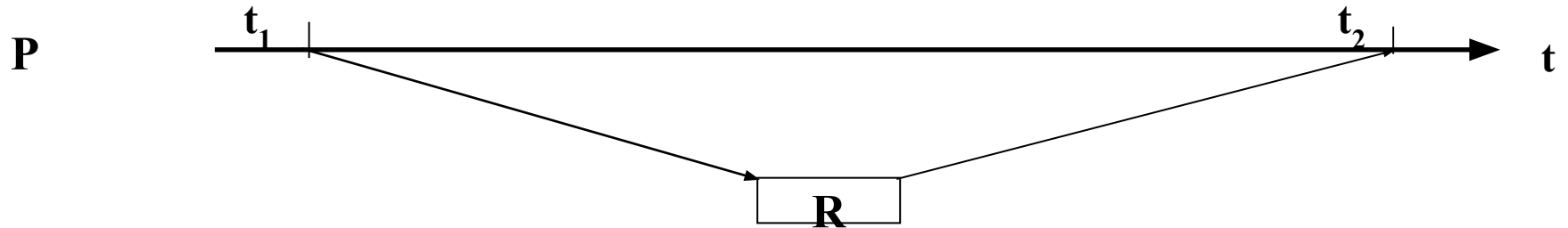
Если все процессы выполняют вышеописанных соглашений, то взаимное исключение гарантируется. При этом процессы могут быть прерваны операционной системой в любой момент и в любом месте, в том числе в критическом интервале.

Нельзя прерывать процесс только между выполнением операций проверки и установки блокирующей переменной.

(например, команды BTC, STR и BTS процессора Pentium),

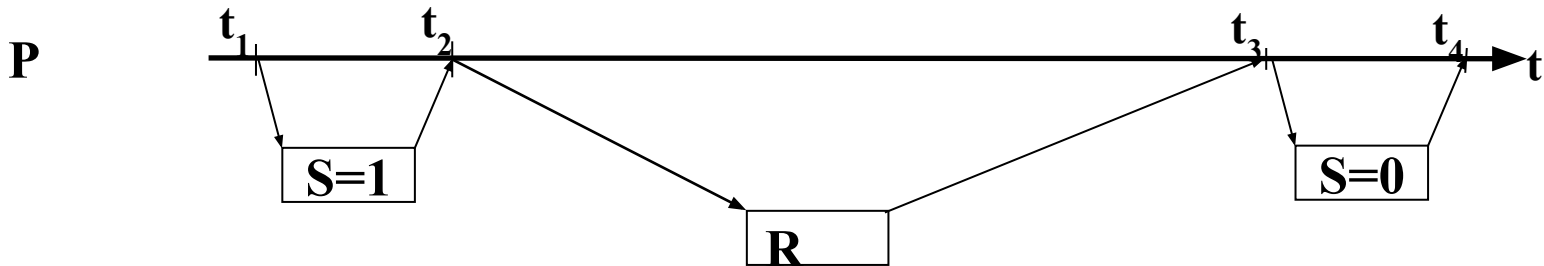
Аппаратная реализация задачи взаимного исключения

3) запрещение обработки прерываний.



t_1 – блокирование обработки прерываний
 t_2 – деблокирование обработки прерываний

2) использование переменной состояния.



$t_1 - t_2$ – вспомогательный критический интервал (блокирование обработки прерываний и занятие переменной состояния)

$t_2 - t_3$ – основной критический интервал (прерывания разрешены)

$t_3 - t_4$ – вспомогательный критический интервал (блокирование обработки прерываний и освобождение переменной состояния)

Программная реализация взаимного исключения

- Семафорные примитивы Дейкстры

P(S) (закрытие семафора)

IF S>0 THEN S:=S-1 (занять единицу семафора)

<продолжить текущий процесс>

IF S=0 THEN <остановить процесс и поместить его в очередь ожидания семафора>

V(S) (открытие семафора)

IF S=0 THEN < процесс из очереди ожидания поместить в очередь готовых>

<продолжить текущий процесс>

ELSE S:= S+1(освободить единицу семафора)

Программная реализация взаимного исключения

- **Мьютексы(семафоры взаимного исключения)**
mutex(mutual exclusion semaphore)

Простейшие двоичные семафоры

Отмеченное состояние – мьютекс свободен

Неотмеченное состояние – процесс является владельцем мьютекса

Системные вызовы

Создание мьютекса(**CreateMutex**)

Открытие мьютекса(**OpenMutex**)

Ожидание(**WaitForSingleObject**,
WaitForMultipleObject)

Освобождение(**ReleaseMutex**)

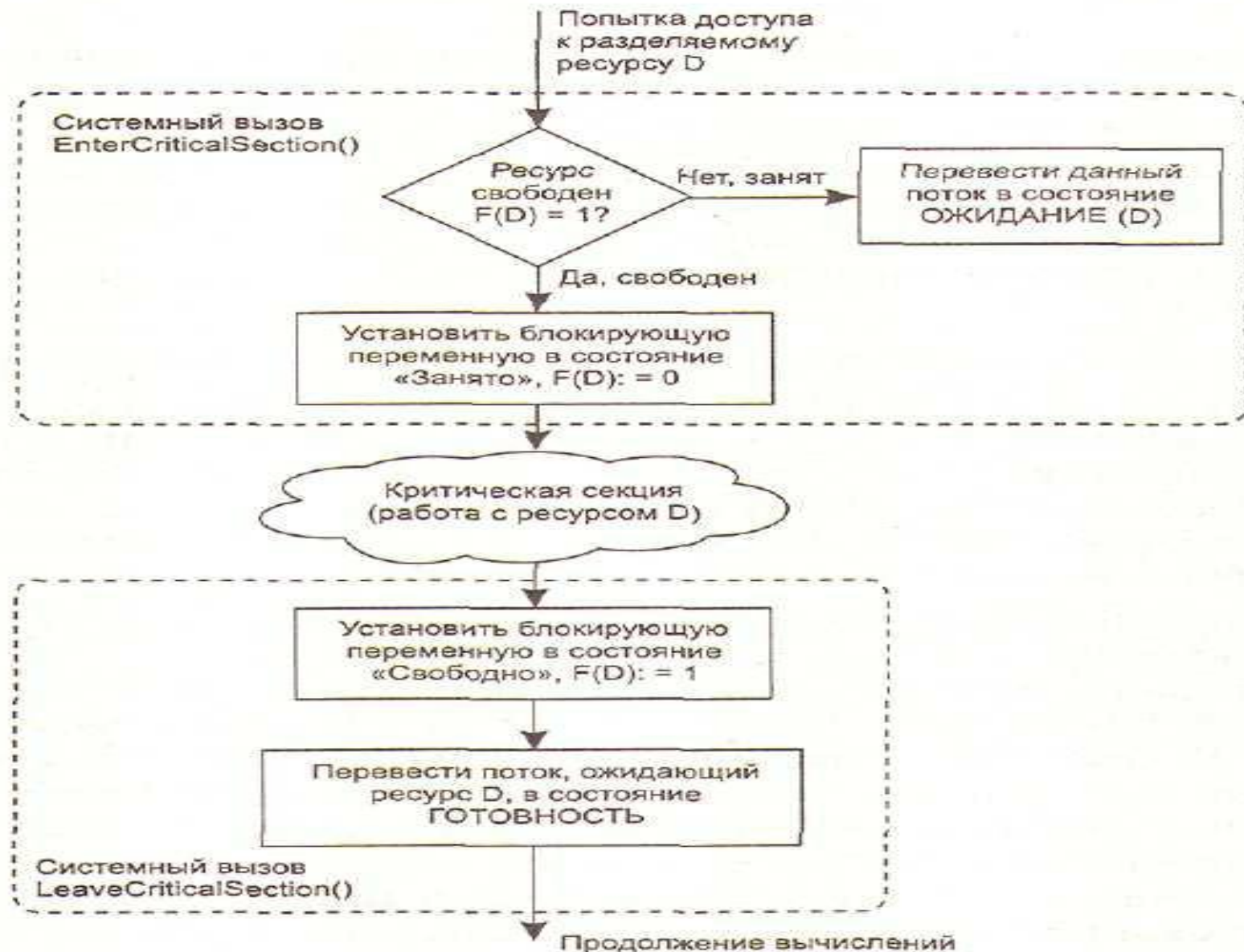
Достоинства семафоров:

- Простота
- Независимость от количества процессов
- Отсутствие «активного ожидания»

Недостатки семафоров:

- Примитивны(семафор не указывает непосредственно на синхронизирующее условие, с которым он связан или на критический ресурс)
- При построении сложных схем синхронизации алгоритмы получаются сложными и ненаглядными

Реализация взаимного исключения в операционной системе Windows



- Перед тем как начать изменение критических данных, поток выполняет системный вызов **EnterCriticalSection()**.
- Выполняется проверка блокирующей переменной, отражающей состояние критического ресурса.
- Если что ресурс занят ($F(D) = 0$), то поток переводится в состояние ожидания (D) и делается отметка о том, что данный поток должен быть активизирован, когда соответствующий ресурс освободится.
- Если ресурс свободен, то поток входит в критическую секцию
- после выхода из критической секции поток должен выполнить системную функцию **LeaveCriticalSection()**, в результате чего блокирующая переменная принимает значение, соответствующее свободному состоянию ресурса ($F(D) = 1$), а операционная система просматривает очередь ожидающих этот ресурс потоков и переводит первый поток из очереди в состояние готовности.

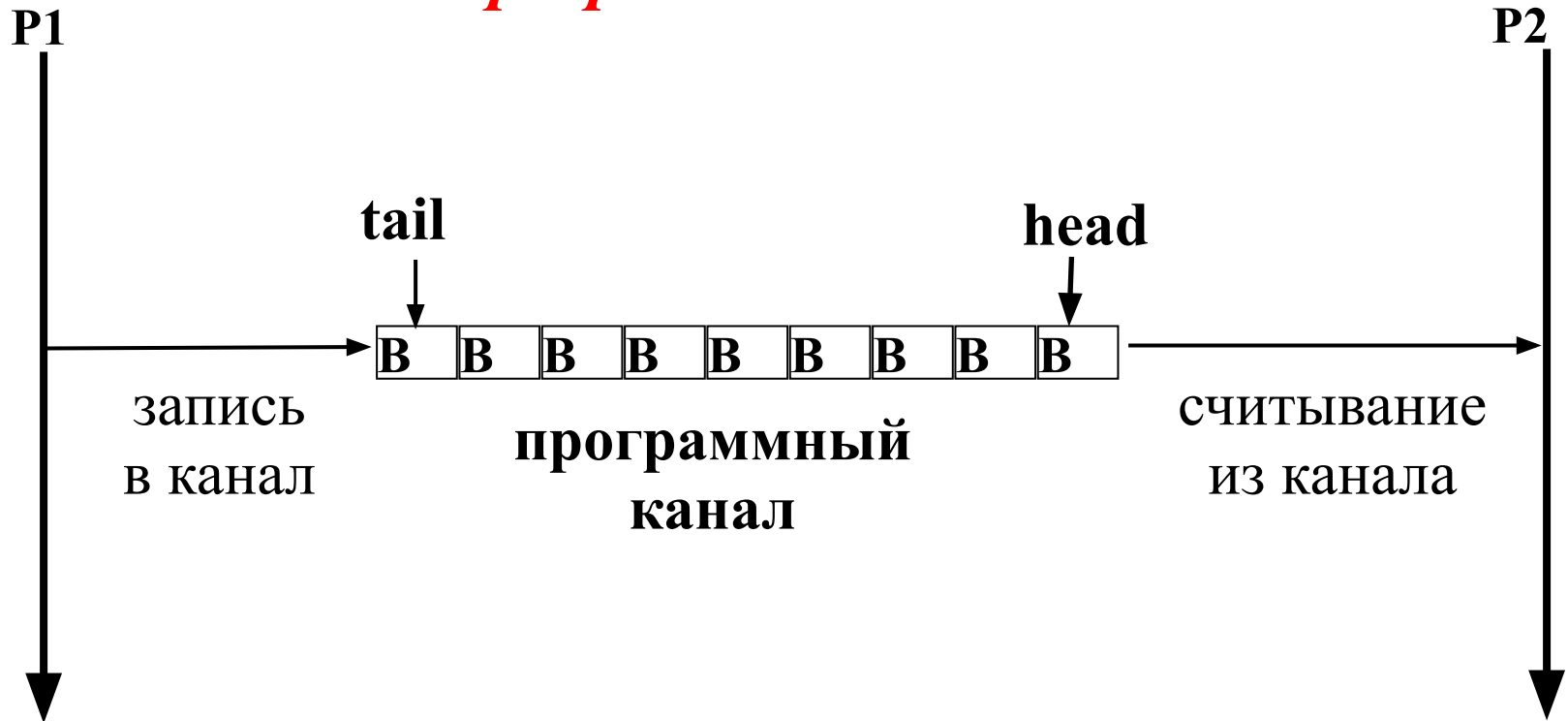
Синхронизация процессов с помощью программного канала

- Программный канал (pipe) - средство синхронизации и обмена данными между процессами.
- Канал представляет собой поток данных между двумя (или более) процессами
- Операции записи и чтения осуществляются потоком байтов
- Конвейер имеет определенный размер, который не может превышать 64 Кбайт. и работает циклически
- Как информационная структура канал имеет
 - **идентификатор**
 - **размер**
 - **два указателя**

Синхронизация процессов с помощью программного канала

- Каналы представляют собой системный ресурс: чтобы начать работу с конвейером, процесс сначала должен заказать его у операционной системы и получить в свое распоряжение.
- **Системные вызовы для работы с каналами**
 - **Функция создания канала:**
 - описатель для чтения из канала,
 - описатель для записи в канал,
 - размер канала.
 - **Функция чтения из канала:**
 - описатель для чтения из канала,
 - переменная любого типа,
 - размер переменной,
 - количество прочитанных байтов.
 - **Функция записи в канал:**
 - описатель для записи в конвейер,
 - количество записанных байтов.
- **При обращении к полному каналу для записи процесс будет ждать, пока в канале не освободится место.**
- **При обращении к пустому каналу для чтения процесс будет ждать, пока в канале не появится информация для чтения.**

Синхронизация процессов с помощью программного канала



head - указатель ГОЛОВЫ

tail - указатель ХВОСТА