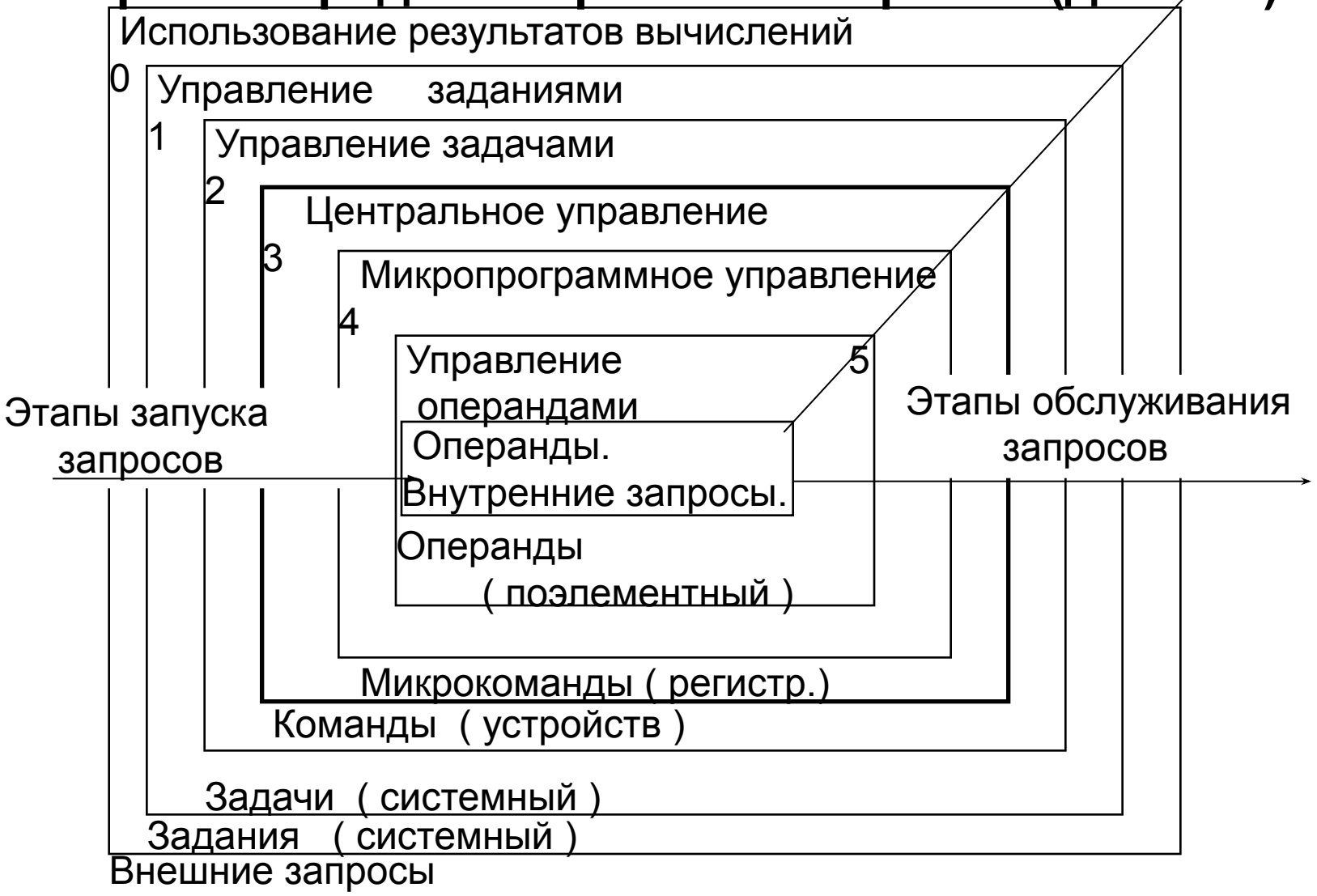
Лек9. Системные механизмы. Синхронизация процессов.

№	Вопрос
1	Проблема синхронизации
2	Состязания.
3	Механизмы синхронизации.
4	Команда ПРОВЕРИТЬ И УСТАНОВИТЬ
5	Механизмы ОЖИДАНИЕ и ОПОВЕЩЕНИЕ.
6	Операции P и V над считающими семафорами.
7	Синхронизация посредством сообщений.
8	Клинчи.
9	Синхронизация в Windows 2000.
10	Синхронизация ядра
11	Синхронизация в исполнительной системе
12	Ожидание на объектах диспетчера ядра
13	Когда объекты переходят в свободное состояние
14	Структуры данных
15	Системные рабочие потоки
16	Глобальные флаги Windows 2000

Уровни средств обработки запросов (данных)



Обслуживание запросов на уровне U(j) является основой реализации запросов на уровне U(j-1). Активизация (инициирование) запросов на уровне U(j) является необходимым условием активизации обработки запросов на уровне U(j+1).

Ор.ЭВМиС Лекция № 7.
Устройство центрального управления.

Назначение, состав и принципы работы центрального управления. Методы реализации устройств управления ВМ. Влияние структуры центрального управления на пропускную способность процессора. Средства управления системой. Динамическое преобразование адресов.

Центральное управление (ЦУ) обеспечивает:

- ***управление потоком команд ЭВМ;***
- ***управление потоком микрокоманд;***
- ***управление потоком фаз синхронизации.***

Уровню (3) ***управления потоком команд ЭВМ*** (детализация ЭВМ на уровне устройств) принадлежат процессы обработки команд, т. е. функции по управлению прохождением команд, реализуемые в процессоре ЭВМ.

Уровню (4) ***управления потоком микрокоманд*** (регистровый уровень детализации ЭВМ) соответствует средства микропрограммного управления обработкой данных в ЭВМ т. е. ОВП на уровне микрокоманд.

Уровень (5) ***управления потоком фаз синхронизации*** (

Основанием разработки структурной *Схема организации* УДИ является универсальная модель процессов взаимодействия: личности; средств обучения; предмета обучения

Связи по управлению, воздействию

Связи по информации

Рис.2. ***Схема организации УДИ***

Работать в коллективе и команде, эффективно общаться с коллегами, руководством, потребителями.


ПРОГРАММА
МОДЕРНИЗАЦИИ
ОБРАЗОВАНИЯ

Главная синхронизация - Всегда держите Единого в поле зрения, и ничто не принесёт вам вреда

Мудрость — это видение Единого везде. Это всеохватывающее единство — Тотальность — и есть Истина. Найдите это единство и постоянно держите его в поле зрения. Если все ваши действия будут основаны на Едином, они станут священными и чистыми. Если вы совершаете все действия во имя Единого, вы будете освящены и наполнены мудростью.

Общие принципы синхронизации процессов.

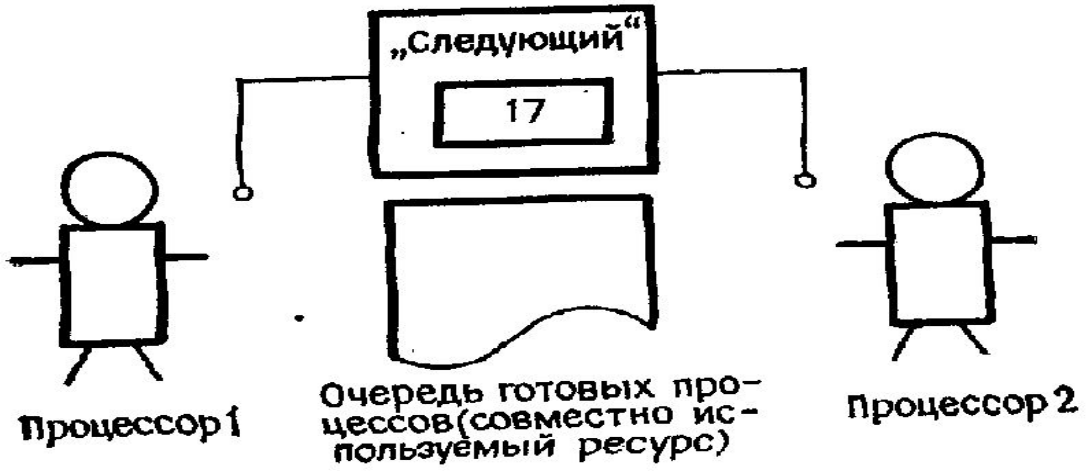
Состязания

Пример условия состязания.

На процедуру извлечения процесса на обслуживание из очереди готовых процессов.

Планирование в однопроцесс
системе.
Когда текущий процесс завершается
блокируется, необходимо:
1. Найти текущий в очереди готовых
цессов.
2. Пометить его «завершенным» или «
женным».
3. Выбрать номер «следующего» про
(17).
4. Найти процесс 17 в очереди готовых
цессов.
5. Пометить процесс 17 «выполняющим
6. Поместить новый номер «следую
процесса (18).

- **Приведенный алгоритм хорошо работает с одним процессором, но в системах с несколькими процессорами его использование приводит к определенным трудностям, что демонстрируется на рис. 9**



Планирование в мультипроцессорной системе.

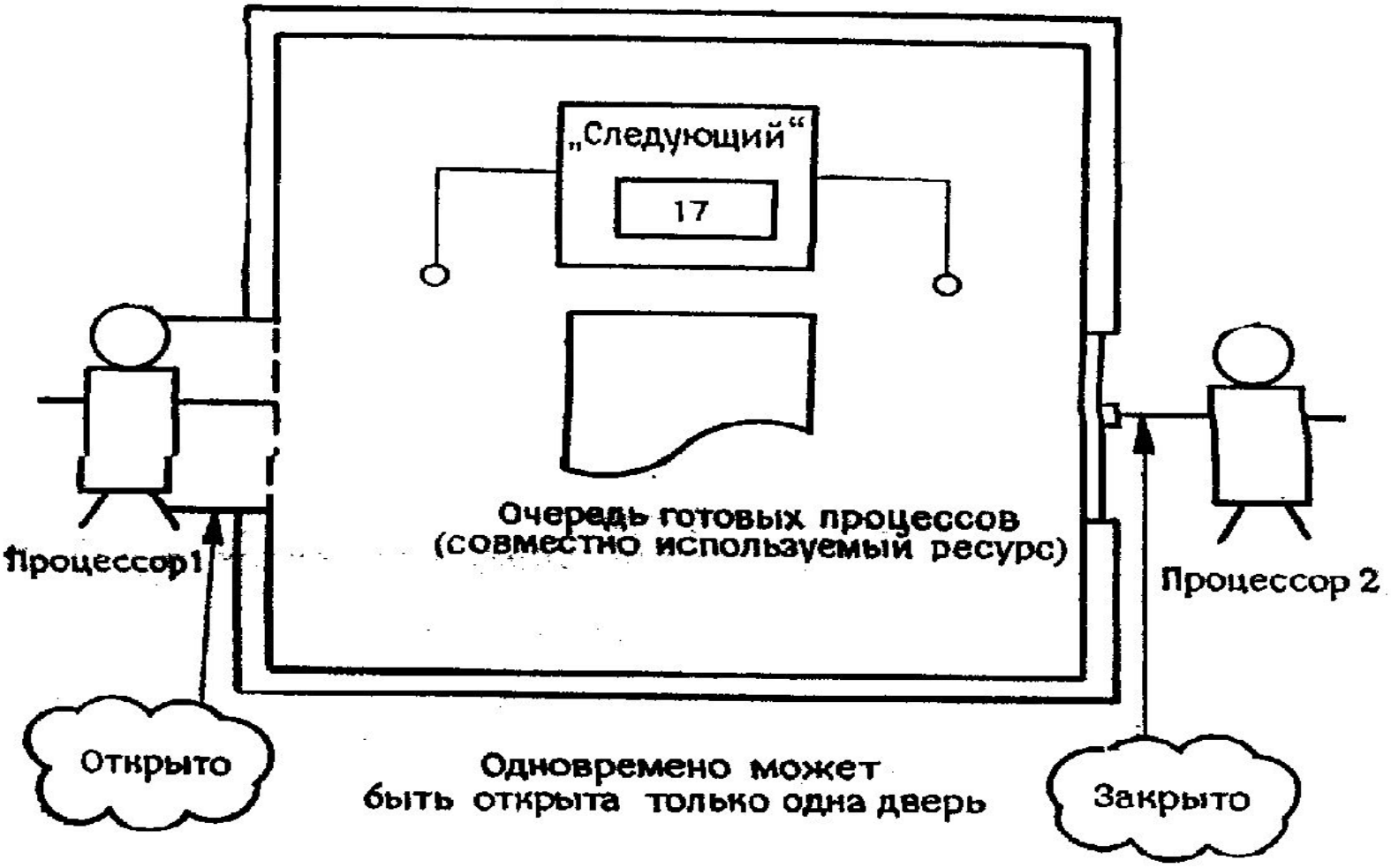
Представим себе, что оба процессора завершили обработку своих т процессов одновременно:

1. Процессор 1 находит свой текущий процесс в очереди готовых
2. Процессор 2 находит свой текущий процесс в очереди готовых.
3. Процессор 1 помечает свой текущий процесс.
4. Процессор 2 помечает свой текущий процесс.
5. Процессор 1 выбирает номер «следующего» (17).
6. Процессор 2 выбирает номер «следующего» (17).
7. Процессор 1 находит процесс 17 в очереди и помечает его.
8. Процессор 2 находит процесс 17 в очереди и помечает его.
9. Процессор 1 помещает номер «следующего» (18).
10. Процессор 2 перемещает номер «следующего» (19).

Теперь оба процессора параллельно обслуживают один и тот же пр (двойная работа), а процесс 18 «потерян».

- Пусть два процессора обслуживают каждый свой процесс и эти процессы, примерно, в одно и то же время блокируются по вводу-выводу. Выполнив процедуру сохранения состояния прерванных процессов, каждый процессор выбирает себе следующий процесс для обслуживания. Поскольку оба процессора применяют при этом один и тот же алгоритм независимо друг от друга, они выберут в качестве следующего один и тот же процесс. Кроме этой ошибки, при обработке очереди готовых процессов возможны и другие ошибки. Например, может быть пропущен процесс, который следовало обслужить, как это показано на рис. 9.

Для разрешения описанной выше проблемы следует ввести синхронизацию независимо работающих процессоров с учетом использования ими общих областей данных. Пример такой синхронизации приведен на рис. 10.

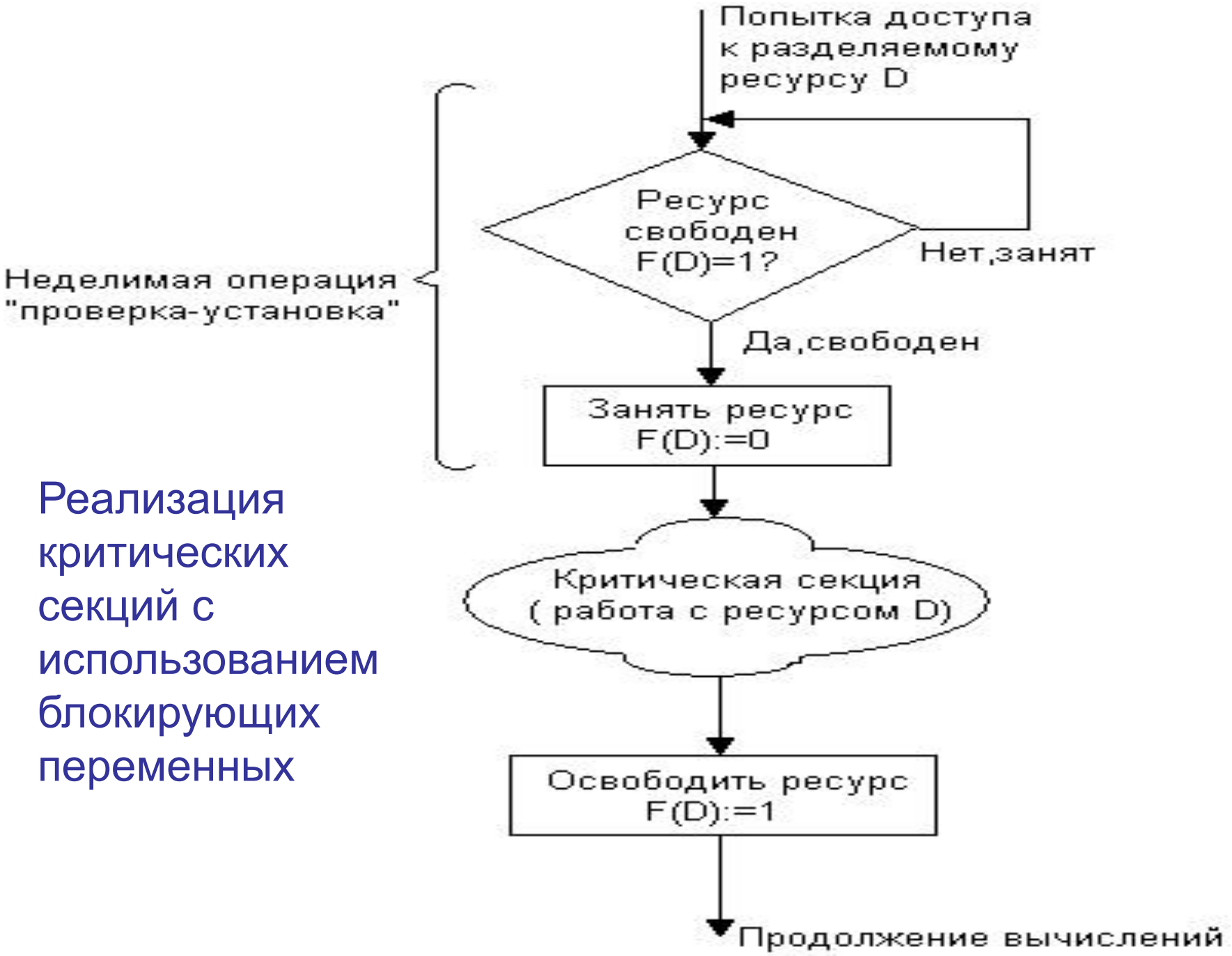


Планирование с блокировкой доступа.

- Перед обращением к очереди готовых процессов каждый процессор должен проверить состояние специального «бита блокировки». Если этот бит не включен, значит, в настоящий момент данные, отображающие состояние процессов в очереди, не используются никаким другим процессором. В этом случае ищущий» процессор включает бит блокировки и приступает к работе с очередью. После завершения обработки процессор должен вернуть бит в состояние «выключен». Если второй процессор во время этой обработки попробует получить доступ к очереди, то он найдет бит блокировки, включенным и будет ожидать снятия блокировки. В таких условиях второй процессор как бы приостанавливается (не может выполнять полезной работы) .

- Эта ситуация называется про*граммной блокировкой процессора. То же самое может случить*ся не только со вторым, но и с третьим, и четвертым процессорами или даже с любым числом процессоров одновременно. Отметим, что в конфликтных ситуациях важна не только координация, но и кооперация процессов. Если процесс «забудет» освободить назначенный ему ресурс, этот ресурс уже никогда не будет доступен для использования никакими другими процессами. На практике реализуют принудительное освобождение операционной системой всех назначенных процессу ресурсов при его завершении.

Реализация критических секций с использованием блокирующих переменных



Команда ПРОВЕРИТЬ И УСТАНОВИТЬ (TEST-AND-SET)

Перед обращением к ресурсу процесс должен выполнить следующие шаги:

- 1. Проверить значение байта блокировки (0 или 1).
- 2. Установить байт блокировки в 1.
- 3. Если первоначальное значение байта блокировки было равно 1, вернуться к шагу 1.
- После завершения использования ресурса процесс должен установить байт блокировки в 0.

Механизмы ОЖИДАНИЕ и ОПОВЕЩЕНИЕ.

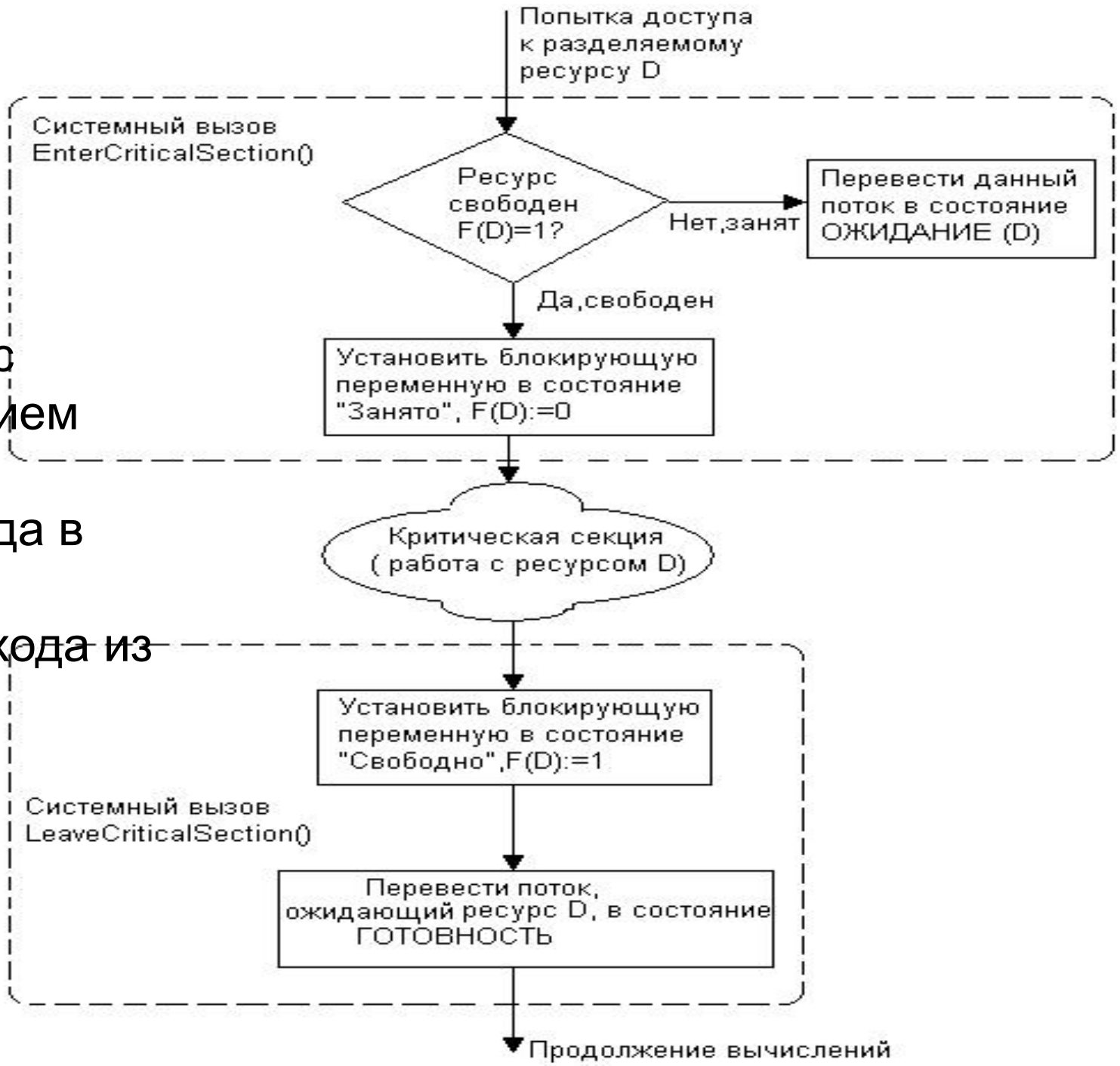
ЗАНЯТЬ (X):

- 1. Проверить значение байта блокировки (0 или 1).
- 2. Установить байт блокировки в 1.
- 3. Если первоначальное значение байта блокировки было
- равно 1, выполнить функцию ОЖИДАНИЕ (X).

ОСВОБОДИТЬ (X):

- 1. Установить байт блокировки в 0.
- 2. Выполнить функцию ОПОВЕЩЕНИЕ (X).

Реализация взаимного исключения с использованием системных функций входа в критическую секцию и выхода из нее



Операции P и V над считающими семафорами.

P (S):

1. Уменьшить значение S на 1 (S: = S - 1).
2. Если S меньше 0, выполнить ОЖИДАНИЕ (S).

V(S):

1. Увеличить значение S на 1 (S:=S+ 1).
2. Если S больше либо равно 0, выполнить ОПОВЕЩЕНИЕ (S).

Читателю следует еще раз убедиться, что синхронизация действительно реализуется. Полезным методом изучения взаимодействия семафоров является рассмотрение последовательности событий, как это показано на рис. 11 для случая n = 2.

ПРОЦЕСС ПРОИЗВОДСТВА $P_{пр}$	ПРОЦЕСС ПОТРЕБЛЕНИЯ $P_{потр}$
PRODUCE:	CONSUME:
: (изготовить единицу продукции)	P (S_2)
P (S_1)	: (взять из хранилища единицу товара)
: (поместить единицу в хранилище)	V (S_1)
V (S_2)	: (использовать единицу)
GO TO PRODUCE	GO TO CONSUME

	ДЕЙСТВИЯ		ЗНАЧЕНИЕ СЕМАФОРОВ	
События	$P_{пр}$(производство)	$P_{потр}$(потребление)	S_1	S_2
0	---	---	2	0
1	—	P(S_2) блокировка	2	-1
2	P (S_1)	— („пустой	1	-1
3	V (S_2) разблокирует $P_{потр}$	— буфер“)	1	0
4	P (S_1)	—	0	0
5	—	V (S_1)	1	0
6	V (S_2)	—	1	1
7	P (S_1)	—	0	1
8	V (S_2)	—	0	2
9	P (S_1) блокировка	—	-1	2
10	— („полный	P (S_2)	-1	1
11	— буфер“)	V (S_1) разблокирует $P_{пр}$	0	1
и т.д.				

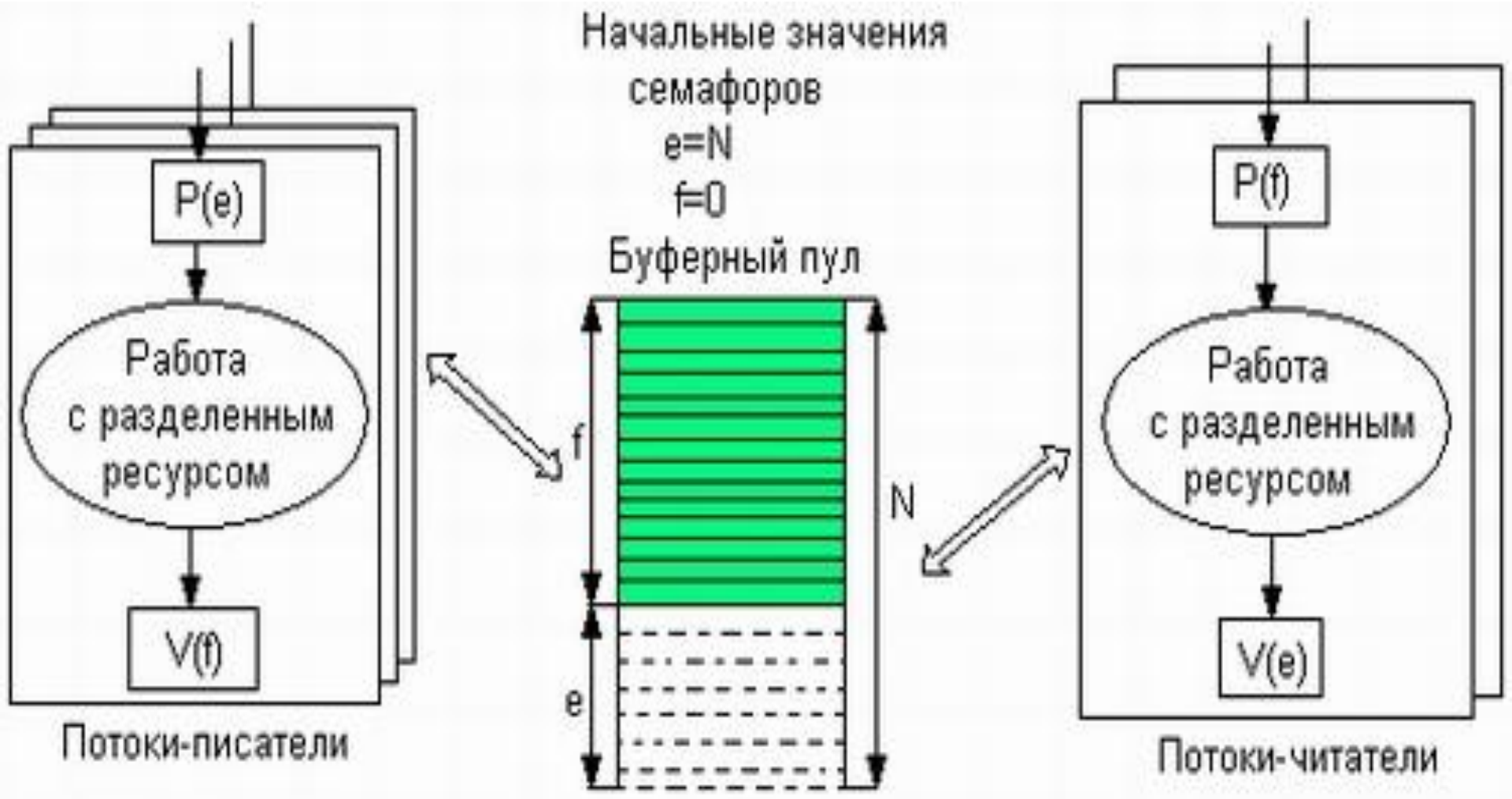
Модель выполнения двух синхронизированных процессов.

Пример

- Рассмотрим использование семафоров на классическом примере взаимодействия двух выполняющихся в режиме мультипрограммирования потоков, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула. Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. В общем случае поток-писатель и поток-читатель могут иметь различные скорости и обращаться к буферному пулу с переменой интенсивностью. В один период скорость записи может превышать скорость чтения, в другой — наоборот. Для правильной совместной работы поток-писатель должен приостанавливаться, когда все буферы оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, поток-читатель должен приостанавливаться, когда все буферы пусты, и активизироваться при появлении хотя бы одной записи.

- Введем два семафора: e — число пустых буферов, и f — число заполненных буферов, причем в исходном состоянии e =N, а f =0. Тогда работа потоков с общим буферным пулом может быть описана следующим образом (рис. 4.20).
- Поток-писатель прежде всего выполняет операцию P(e), с помощью которой он проверяет, имеются ли в буферном пуле незаполненные буферы. В соответствии с семантикой операции P, если семафор e равен 0 (то есть свободных буферов в данный момент нет), то поток-писатель переходит в состояние ожидания. Если же значением e является положительное число, то он уменьшает число свободных буферов, записывает данные в очередной свободный буфер и после этого наращивает число занятых буферов операцией V(f). Поток-читатель действует аналогичным образом, с той разницей, что он начинает работу с проверки наличия заполненных буферов, а после чтения данных наращивает количество свободных буферов.

Использование семафоров для синхронизации потоков (Рис.4.20)



- В данном случае предпочтительнее использовать семафоры вместо блокирующих переменных. Действительно, критическим ресурсом здесь является буферный пул, который может быть представлен как набор идентичных ресурсов — отдельных буферов, а значит, с буферным пулом могут работать сразу несколько потоков, и именно столько, сколько буферов в нем содержится. Использование двоичной переменной не позволяет организовать доступ к критическому ресурсу более чем одному потоку. Семафор же решает задачу синхронизации более гибко, допуская к разделяемому пулу ресурсов заданное количество потоков. Так, в нашем примере с буферным пулом могут работать максимум N потоков, часть из которых может быть «писателями», а часть — «читателями».
- Таким образом, семафоры позволяют эффективно решать задачу синхронизации Доступа к ресурсным пулам, таким, например, как набор идентичных в функциональном назначении внешних устройств (модемов, принтеров, портов), или набор областей памяти одинаковой величины, или информационных структур. Во всех этих и подобных им случаях с помощью семафоров можно организовать доступ к разделяемым ресурсам сразу нескольких потоков.

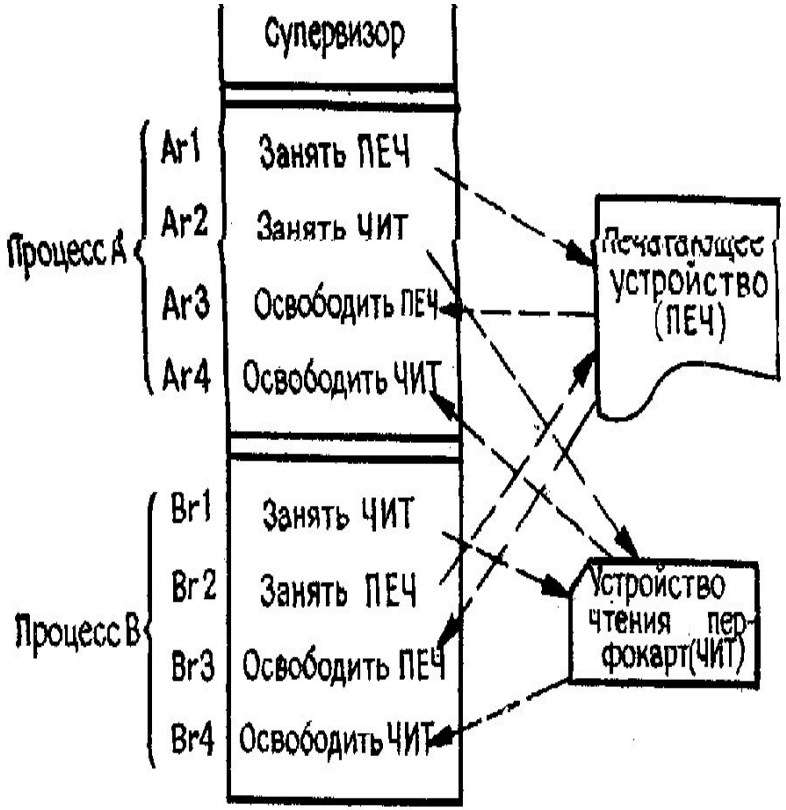
Синхронизация посредством сообщений.

- В случаях, аналогичных задаче производства/потребления, проблема также может быть *решена путем непосредственного обмена сообщениями между* процессами. Д. этого используются операции ПОСЛАТЬ (Pr, M) и ПРИНЯТЬ (Ps, I где Pr , и Ps - идентификаторы процессов, а M - сообщение (некая строка символов**).**

Клинчи.

Существуют следующие способы борьбы с клинчами:

1. Предварительное распределение все совместно используемых ресурсов.
2. Контроль над распределением:
   а) регулируемое распределение;
   б) стандартная последовательность распределения.
3. Обнаружение и восстановление**.**

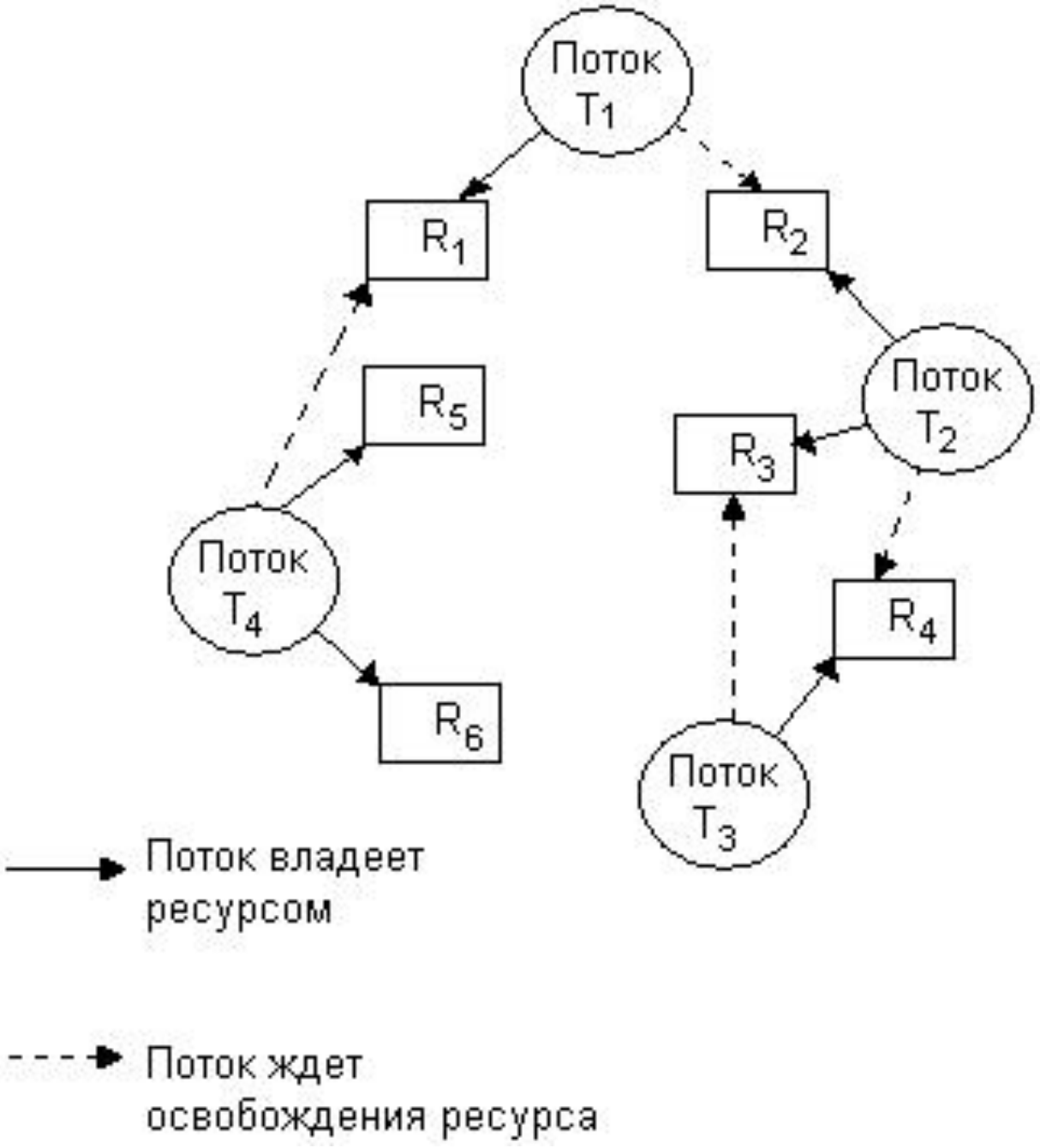


Ситуация клинча.

- В рассмотренных примерах тупик был образован двумя потоками, но взаимно блокировать друг друга может и большее число потоков. На рис. 2.23 показано такое распределение ресурсов Ri между несколькими потоками Tj, которое привело к возникновению взаимных блокировок. Стрелки обозначают потребность потока в ресурсах. Сплошная стрелка означает, что соответствующий ресурс был выделен потоку, а пунктирная стрелка соединяет поток с тем ресурсом, который необходим, но не может быть пока выделен, поскольку занят другим потоком. Например, потоку T1 для выполнения работы необходимы ресурсы R1 и R2, из которых выделен только один — R1, а ресурс R2 удерживается потоком T2. Ни один из четырех показанных на рисунке потоков не может продолжить свою работу, так как не имеет всех необходимых для этого ресурсов.

- Невозможность потоков завершить начатую работу из-за возникновения взаимных блокировок снижает производительность вычислительной системы. Поэтому проблеме предотвращения тупиков уделяется большое внимание. На тот случай, когда взаимная блокировка все же возникает, система должна предоставить администратору-оператору средства, с помощью которых он смог бы распознать тупик, отличить его от обычной блокировки из-за временной недоступности ресурсов. И наконец, если тупик диагностирован, то нужны средства для снятия взаимных блокировок и восстановления нормального вычислительного процесса.

Рис. 4.23. Взаимная блокировка нескольких потоков



Синхронизация в Windows 2000.

Пример потребности в синхронизации

Поскольку второй поток получил значение указателя на конец очереди до того, как первый поток завершил его обновление, второй вставил свои данные в то же место, что и первый. Таким образом, данные первого потока были перезаписаны другими данными, а один участок очереди остался пустым.

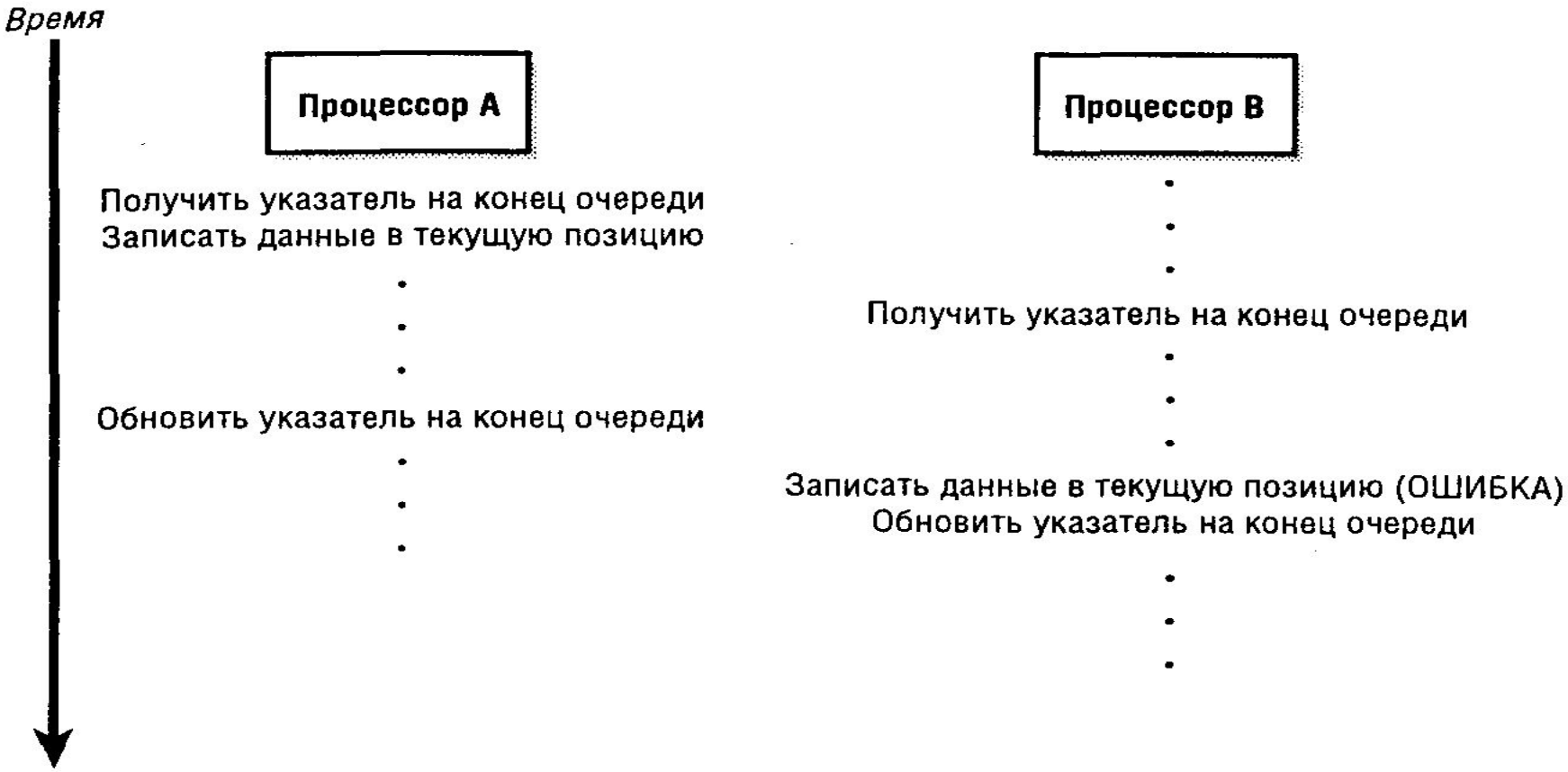


Рис. 3-17. *Некорректное разделение памяти*

Синхронизация ядра

- Механизм, применяемый ядром для взаимоисключения в многопроцессорных системах, называется ***спин-блокировкой*** (spinlock). Спин-блокировка — это блокирующий примитив, сопоставленный с какой-либо глобальной структурой данных вроде очереди DPC (рис. 3-18).
- Во многих архитектурах спин-блокировка реализуется аппаратно поддерживаемой командой test-and-set, которая проверяет значение переменной блокировки и устанавливает блокировку, выполняя всего одну атомарную команду.
- Всем спин-блокировкам режима ядра в Windows 2000 назначен IRQL, всегда соответствующий уровню «DPC/dispatch» или выше. Поэтому, когда поток пытается установить спин-блокировку, все действия на этом или более низком уровне IRQL на данном процессоре прекращаются.

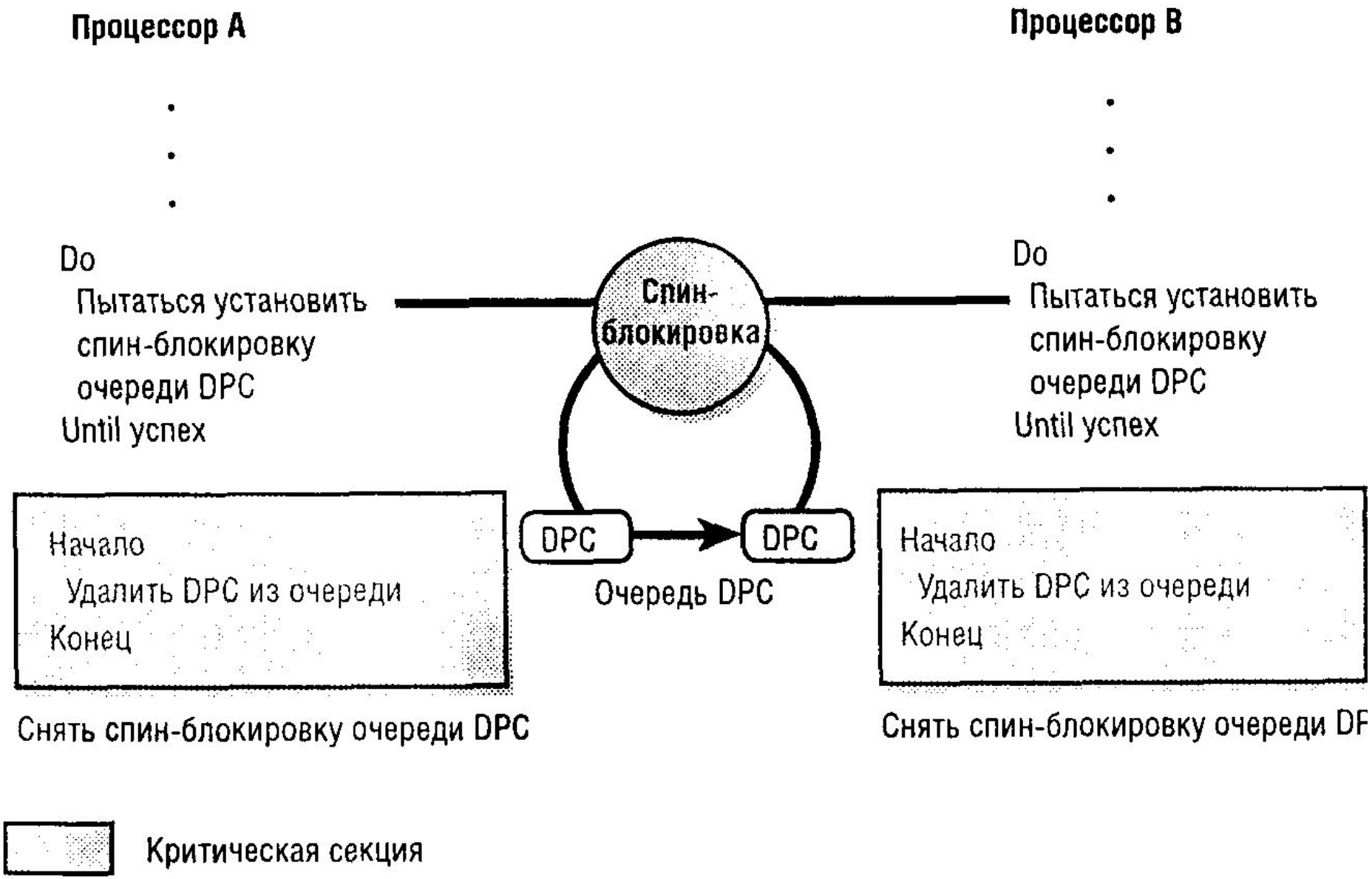


Рис. 3-18. *Применение спин-блокировки*

- Спин-блокировки ядра накладывают ограничения на использующий их код. Как уже отмечалось, их IRQL всегда равен «DPC/dispatch», поэтому установивший спин-блокировку код может привести к краху системы, если попытается заставить планировщик выполнить операцию диспетчеризации или вызовет ошибку страницы.
- В Windows 2000 появился особый тип **спин-блокировки — с очередью** (queued spinlock), применяемый только ядром и недоступный другим компонентам исполнительной системы или драйверам устройств. Тот факт, что спин-блокировка с очередью устанавливает флаги, а не глобальные блокировки, имеет два следствия. Во-первых, уменьшается интенсивный трафик, связанный с межпроцессорной синхронизацией. Во-вторых, вместо случайного выбора процессора из группы ожидающих спин-блокировку, реализуется четкий порядок спин-блокировки по типу FIFO («первым вошел, первым вышел»). Такой порядок позволяет достичь более согласованной работы процессоров, использующих одну и ту же блокировку.

Синхронизация в исполнительной системе

Поскольку спин-блокировка означает фактическую остановку процессора, она применяется только при двух условиях:

- требуется непродолжительное обращение к защищенным ресурсам без сложного взаимодействия с другим кодом;
- код критической секции нельзя выгрузить в страничный файл, он не ссылается на данные в подкачиваемой памяти, не вызывает внешние процедуры (включая системные сервисы) и не генерирует прерывания или исключения.

Ядро предоставляет исполнительной системе дополнительные механизмы синхронизации в форме объектов, в совокупности известных как объекты диспетчера ядра. Синхронизирующие объекты, видимые из пользовательского режима, берут свое начало именно от этих объектов диспетчера ядра. Каждый синхронизирующий объект, видимый из пользовательского режима, инкапсулирует минимум один объект диспетчера ядра.

Еще один тип синхронизирующих объектов исполнительной системы назван (без особой на то причины) *ресурсами исполнительной системы* (executive resources). Ресурсы исполнительной системы являются не объектами диспетчера ядра, а скорее структурами данных, память для которых выделяется прямо из не подкачиваемого пула, имеющего свои специализированные сервисы для инициализации, блокировки, освобождения, запроса и ожидания.

2.3. Ожидание на объектах диспетчера ядра

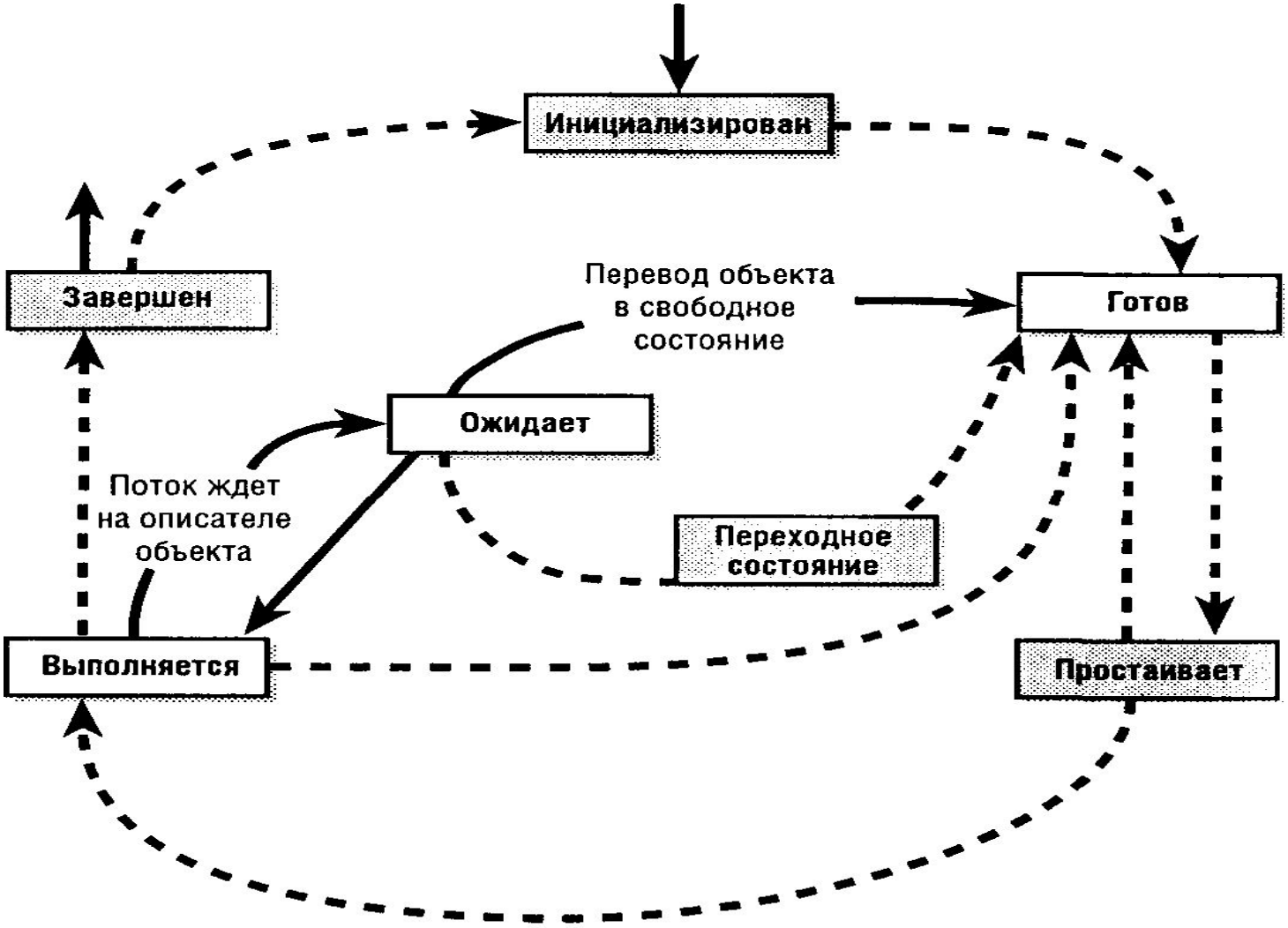


Рис. 3-19. *Ожидание на объекте диспетчера ядра*

Ожидание на объектах диспетчера ядра

Взаимосвязь синхронизации с диспетчеризацией потоков иллюстрирует следующий пример с использованием объекта «событие».

1. Поток пользовательского режима ждет на описателе объекта «событие» (т. е. ждет перехода этого объекта в свободное состояние).
2. Ядро изменяет состояние потока с «готов» на «ожидает» и добавляет его в список потоков, ждущих объект «событие».
3. Другой поток устанавливает объект «событие».
4. Ядро просматривает список потоков, ожидающих этот объект. Если условия ожидания какого-либо потока выполнены, ядро переводит его из состояния «ожидает» в состояние «готов». Если это поток с динамическим приоритетом, ядро может повысить его приоритет для выполнения.

5. Поскольку новый поток теперь готов к выполнению, происходит перераспределение процессорного времени. Если при этом диспетчер обнаружит, что приоритет выполняемого потока ниже, чем приоритет потока, только что перешедшего в состояние «готов», он вытеснит поток с более низким приоритетом и выдаст программное прерывание для инициации переключения контекста на поток с более высоким приоритетом.

6. Если в данный момент вытеснение невозможно ни на одном из процессоров, диспетчер включает поток в свою очередь потоков, готовых к выполнению.

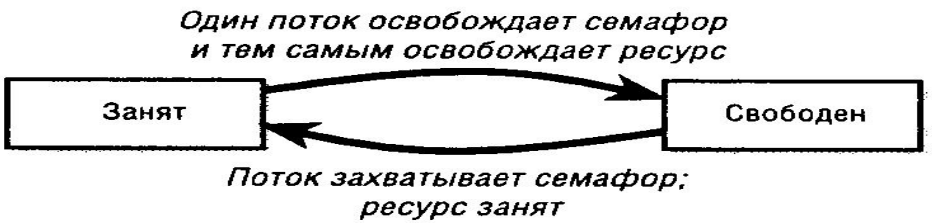
(Некоторые потоки могут ждать более одного объекта, и в таком случае их ожидание продолжается.*)*

Переход синхронизирующих объектов в свободное состояние.

Таблица 3-9. *Условия освобождения различных синхронизирующих объектов*

Тип объекта	Переводится в свободное состояние при:	Действие на ожидающие потоки
Процесс	Завершении последнего потока	Все потоки освобождаются
Поток	Завершении данного потока	Все потоки освобождаются
Файл	Завершении операции ввода-вывода	Все потоки освобождаются
Событие (уведомляющего типа)	Установке события потоком	Все потоки освобождаются
Событие (синхронизирующего типа)	Установке события потоком	Освобождается один поток, и объект «событие» сбрасывается
Семафор	Уменьшении счетчика семафора на 1	Освобождается один поток
Таймер (уведомляющего типа)	Истечении заданного времени или наступлении очередного момента срабатывания	Все потоки освобождаются
Таймер (синхронизирующего типа)	Истечении заданного времени или наступлении очередного момента срабатывания	Освобождается один поток
Мьютекс	Освобождении объекта «мьютекс» потоком	Освобождается один поток
Очередь	Добавлении элемента в очередь	Освобождается один поток

- Когда объект переводится в свободное состояние, ожидающие его потоки обычно немедленно выходят из ждущего состояния. Однако, как показано на рис. 3-20, некоторые объекты диспетчера ядра и системные события ведут себя иначе.
- Например, объект «событие уведомления» — в Win32 API он называется событием со сбросом вручную (manual reset event) — используется для уведомления о каком-либо событии. Когда этот объект переводится в свободное состояние, все потоки, ожидающие его, освобождаются. Исключением является тот поток, который ждет сразу несколько объектов: он может продолжать ожидание, пока не освободятся дополнительные объекты.

Объект диспетчера	Системные события и вызываемое ими изменение состояния	Влияние перехода в свободное состояние на ожидающие потоки
Мьютекс (только для использования в режиме ядра)		**Ядро возобновляет выполнение одного ожидающего потока**
Мьютекс (экспортируемый для пользовательского режима)		**Ядро возобновляет выполнение одного ожидающего потока**
Семафор		**Ядро возобновляет выполнение одного или нескольких ожидающих потоков**
Событие		**Ядро возобновляет выполнение одного или нескольких ожидающих потоков**
Пара событий		**Ядро возобновляет выполнение другого потока**
Таймер		**Ядро возобновляет выполнение всех ожидающих потоков**

Структуры данных

- Учет ожидающих потоков и их объектов ожидания базируется на двух ключевых структурах данных: *заголовках диспетчера* (dispatcher headers) и *блоках ожидания* (wait blocks).
- Заголовок диспетчера содержит тип объекта, информацию о состоянии (занят/свободен) и список потоков, ожидающих этот объект. У каждого ждущего потока есть список блоков ожидания, где перечислены ожидаемые потоком объекты, а у каждого объекта диспетчера ядра — список блоков ожидания, где перечислены ожидающие его потоки. Этот список ведется так, что при освобождении объекта диспетчера ядро может быстро определить, кто ожидает данный объект.
- В блоке ожидания имеются указатели на объект ожидания, ожидающий поток и на следующий

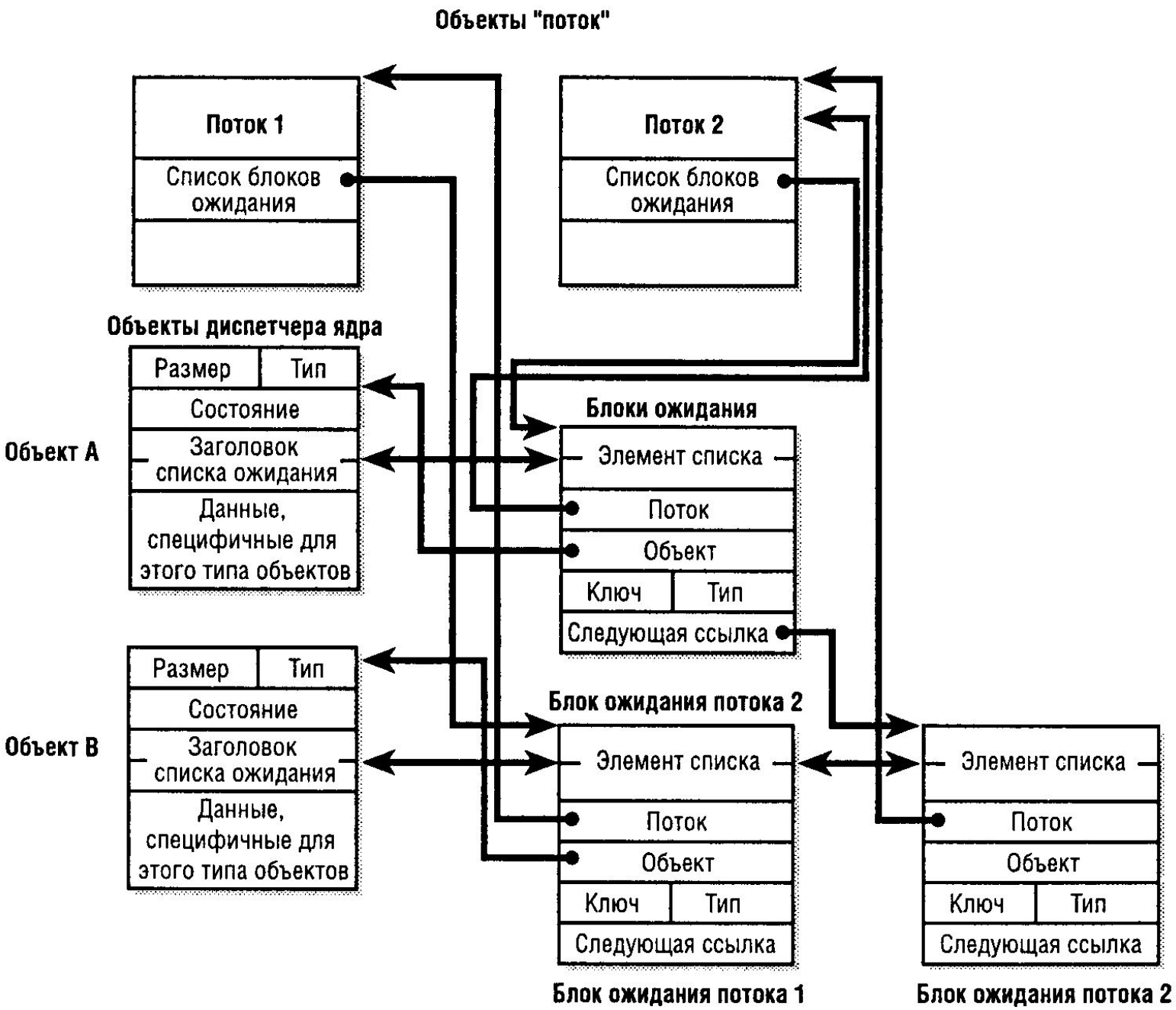
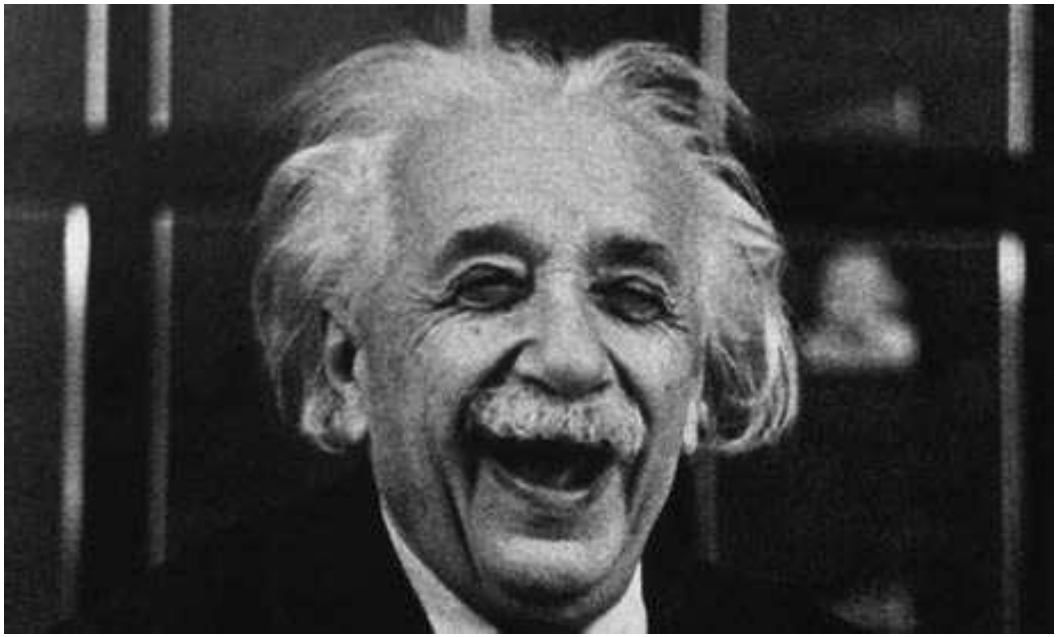


Рис. 3-21. *Структуры данных, необходимые для поддержки ожидания*

- На рис. 3-21 показана связь объектов диспетчера ядра с блоками ожидания потоков. В данном примере поток 1 ждет объект В, а поток 2 — объекты А и В.
- Если объект А освободится, поток 2 не сможет возобновить свое выполнение, так как ядро обнаружит, что он ждет и другой объект. С другой стороны, при освобождении объекта В ядро сразу же подготовит поток 1 к выполнению, поскольку он не ждет никакие другие объекты.

Образование — это то, что остается после того, когда забываешь все, чему учили в школе.



Энштейн

Воспитание имеет приоритет над образованием.

Создает человека воспитание.

Антуан де Сент-Экзюпери

«...самым важным в обучении мы признаем надлежащее воспитание».

«Никто не становится хорошим человеком случайно».

Платон