Раздел 2

Процессы и потоки

Лекция №4

Процессы и потоки

Функции ОС по управлению процессами и потоками:

1. планирование процессов, т.е. распределение процессорного времени между несколькими одновременно выполняющимися в системе процессами
2. создание и уничтожение процессов
3. наделение процессов необходимыми системными ресурсами
4. реализация обмена данными между процессами;
5. синхронизация процессов и потоков

Процесс - программа, находящаяся в стадии выполнения.

Потоки возникли как средство распараллеливания вычислений в рамках одного процесса.

Процесс рассматривается как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени.

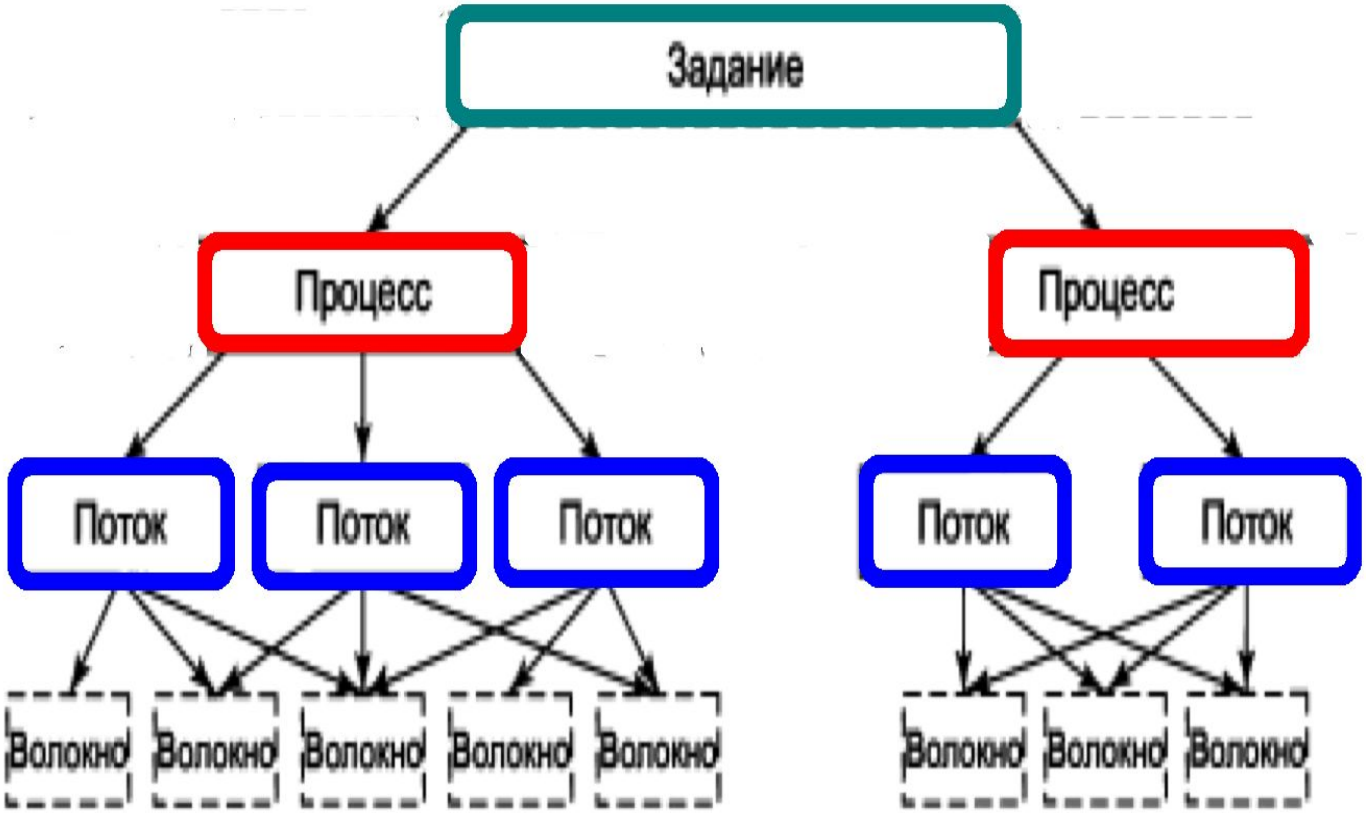
Процессорное время выделяется потокам. В простейшем случае процесс состоит из одного потока.

элементы, совместно используемые всеми потоками процесса	элементы, индивидуальные для каждого потока
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и их обработчики	
Информация об использовании ресурсов	

Преимущества использования потоков:

- \+ Создание потоков требует от ОС меньших накладных расходов, чем при создании процессов
- \+ Быстрота создания потока по сравнению с процессом
- \+ Потоки одного процесса могут взаимодействовать не обращаясь к ОС, а используя общую память
- \+ Повышение производительности программы

Задания и волокна



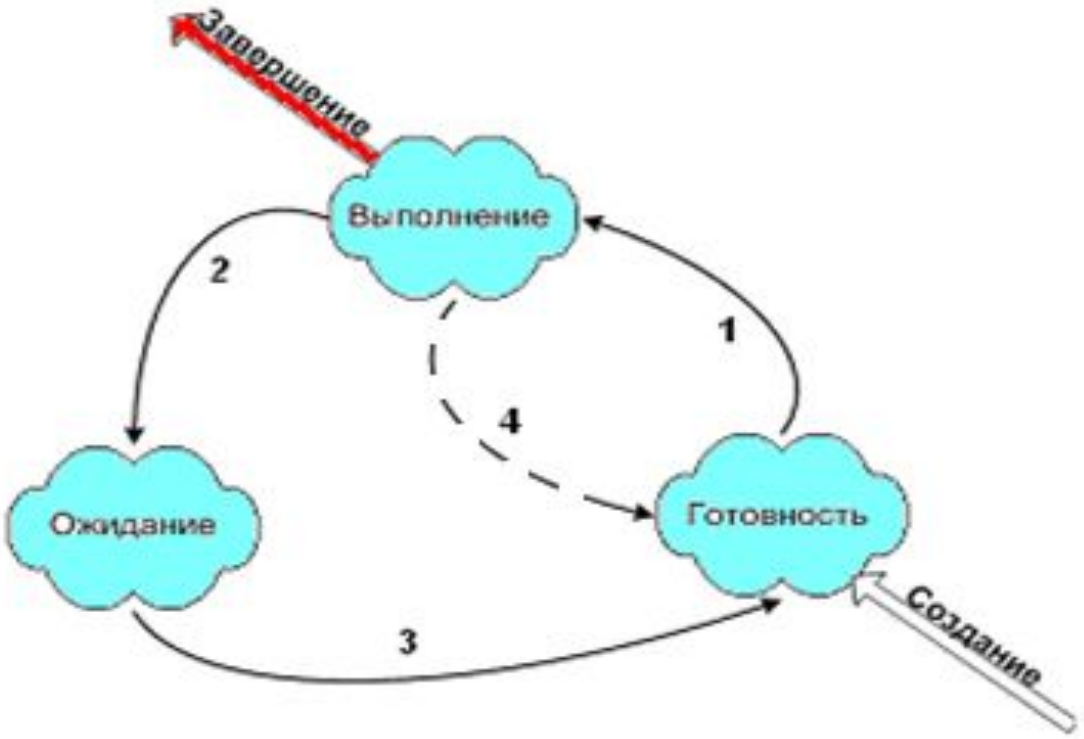
Название	Описание	Примечания
Задание	Коллекция процессов, у которых общие квоты и лимиты	Используется редко
Процесс	Контейнер для ресурсов	
Поток	Единица планирования для ядра	
Волокно	«Легкий» поток, управляемый полностью в пространстве пользователя	Используется редко

Состояния потоков

выполнение	активное состояние, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором
готовность	пассивное состояние, поток заблокирован в связи с внешними по отношению к нему обстоятельствами
ожидание	пассивное состояние, находясь в котором поток заблокирован по своим внутренним причинам

Граф состояний потока

1. Поток выбран на выполнение
2. Поток ожидает завершения ввода/вывода
3. Ввод/вывод завершен (событие произошло)
4. Поток вытеснен планировщиком



Создание процессов

События, приводящие к созданию процессов:		
загрузка системы	работающий процесс подает системный вызов на создание процесса	запрос пользователя на создание процесса

Создать процесс означает:

Идентификаторы, дескрипторы и контекст

- Дескриптор процесса содержит такую информацию о процессе, которая необходима ядру в течение всего жизненного цикла процесса независимо от того, находится он в активном или пассивном состоянии. В дескрипторе прямо или косвенно содержится информация о состоянии процесса, о расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, глобальном приоритете, об идентификаторе пользователя, создавшего процесс, о родственных процессах и некоторая др. информация.
- Контекст процесса содержит менее оперативную, но более объемную часть информации о процессе, необходимую для возобновления выполнения процесса с прерванного места: содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, таблица открытых файлов, информация о незавершенных операциях ввода/вывода и др.

Структура сегмента TSS

Битовая карта ввода - вывода (БКВВ)			8 Кбайт
Дополнительная информация ОС			
Относительный адрес БККВ	0 . . . 0	T	64
0 . . . 0	Селектор LDT		60
0 . . . 0	GS		5C
0 . . . 0	FS		58
0 . . . 0	DS		54
0 . . . 0	SS		50
0 . . . 0	CS		4C
0 . . . 0	ES		48
EDI			44
ESI			40
EBP			3C
ESP			38
EBX			34
EDX			30
ECD			2C
EAX			28
EFLAGS			24
EIP			20
CR3			1C
0 . . . 0	SS уровня 2		18
ESP2			14
0 . . . 0	SS уровня 1		10
ESP1			C
0 . . . 0	SS уровня 0		8
ESP0			4
0 . . . 0	Селектор TSS возврата		0

Уничтожение процессов

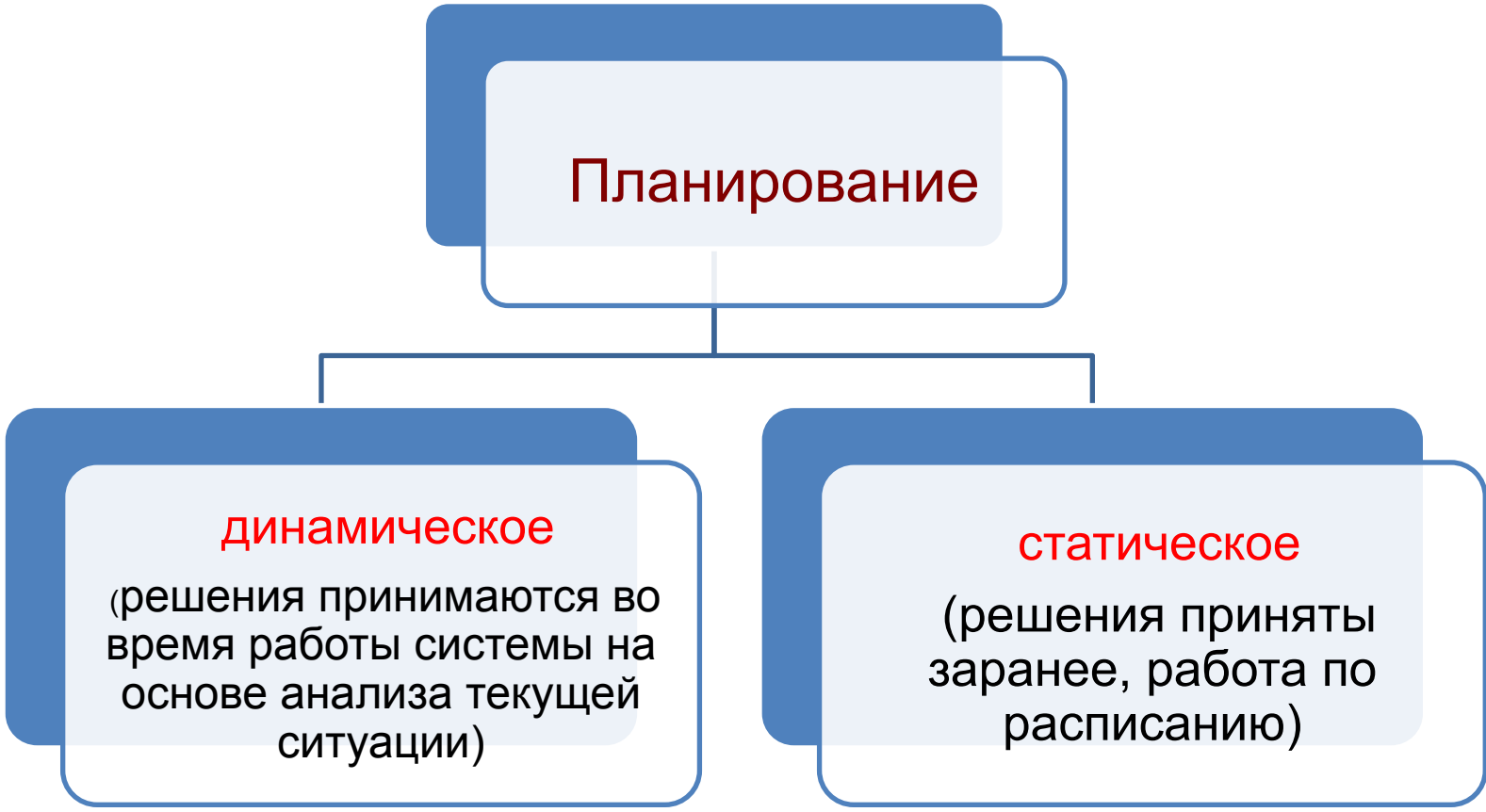
Планирование:

- определение момента времени для смены текущего активного потока
- выбор потока для выполнения из очереди готовых

Планирование и диспетчеризация потоков

Планирование:

- определение момента времени для смены текущего активного потока
- выбор потока для выполнения из очереди готовых


Планирование
динамическое
(решения принимаются во время работы системы на основе анализа текущей ситуации)
статическое
(решения приняты заранее, работа по расписанию)

Диспетчеризация

это реализация найденного в результате планирования решения, т.е.:

1. сохранение контекста текущего потока
2. загрузка контекста потока, выбранного в результате планирования
3. запуск нового потока на выполнение

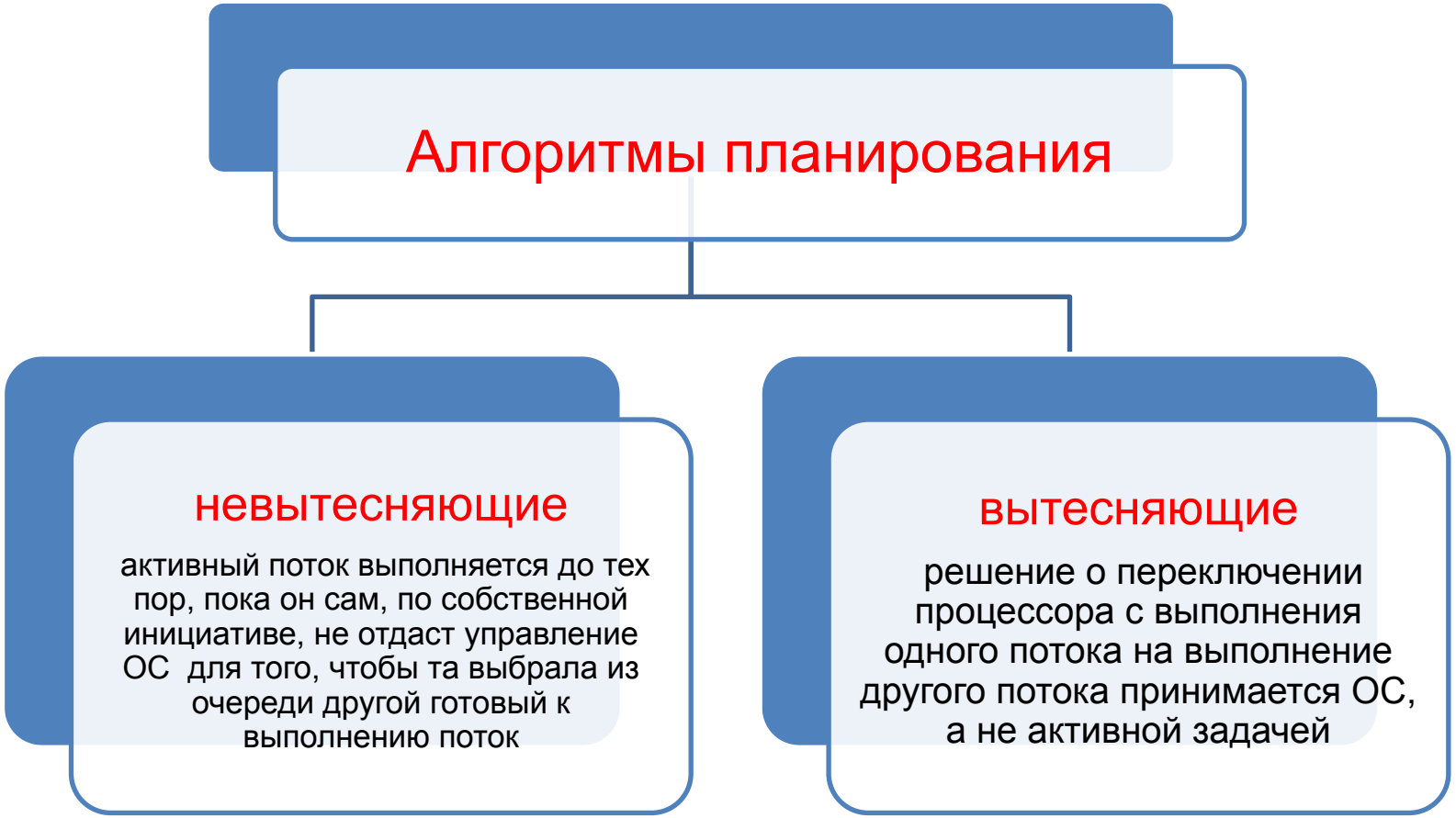
Моменты перепланировки

- Время, отведенное активной задаче на выполнение, закончилось. Планировщик переводит задачу в состояние готовности и выполняет перепланирование.
- Активная задача выполнила системный вызов, связанный с запросом на ввод/вывод или на доступ к ресурсу, который в настоящий момент занят. Планировщик переводит задачу в состояние ожидания и выполняет перепланирование.
- Активная задача выполнила системный вызов, связанный с освобождением ресурса. Если есть задача, ожидающая это событие, то она переводится из состояния ожидания в состояние готовность.
- Завершение периферийным устройством операции ввода/вывода переводит соответствующую задачу в очередь готовых и выполняется планирование.
- Внутреннее прерывание сигнализирует об ошибке, которая произошла в результате выполнения активной задачи. Планировщик снимает задачу и выполняет перепланирование.

Лекция № 5

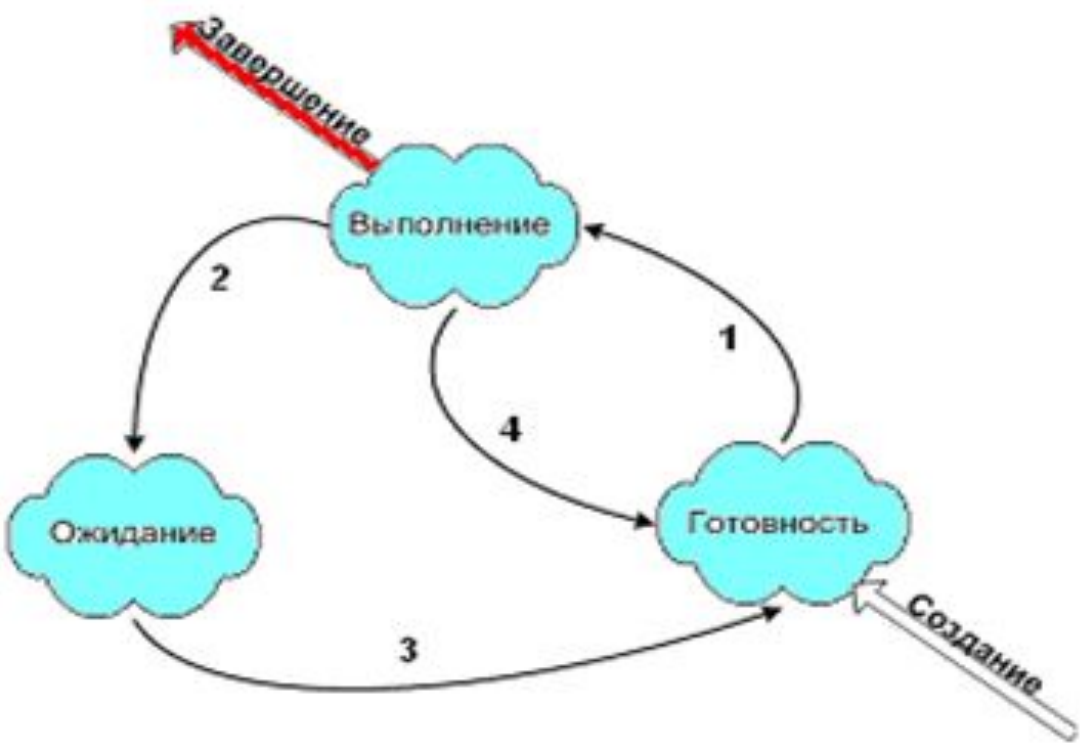
Планирование процессов

Планирование процессов



Алгоритмы планирования, основанные на квантовании

Квант – ограниченный непрерывный интервал процессорного времени, который поочередно предоставляется всем существующим в системе потокам.

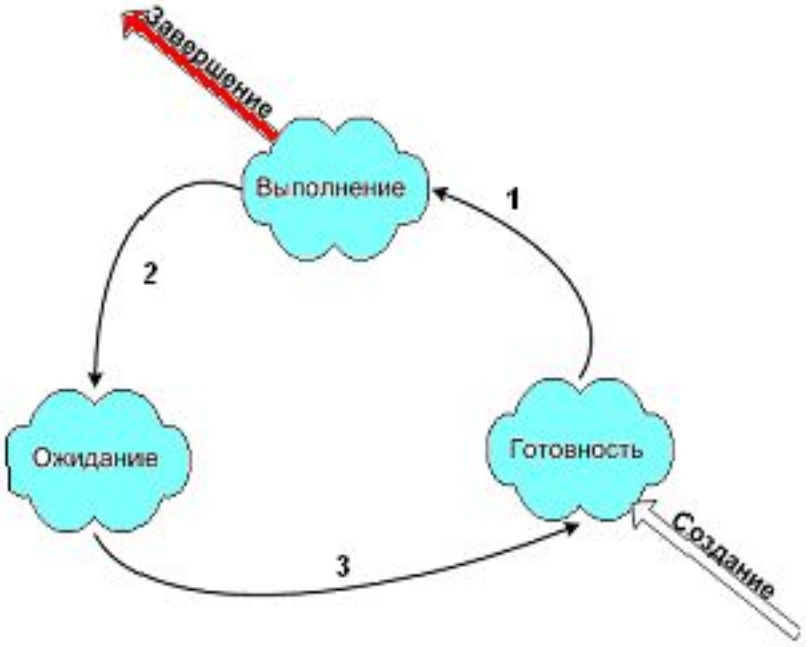


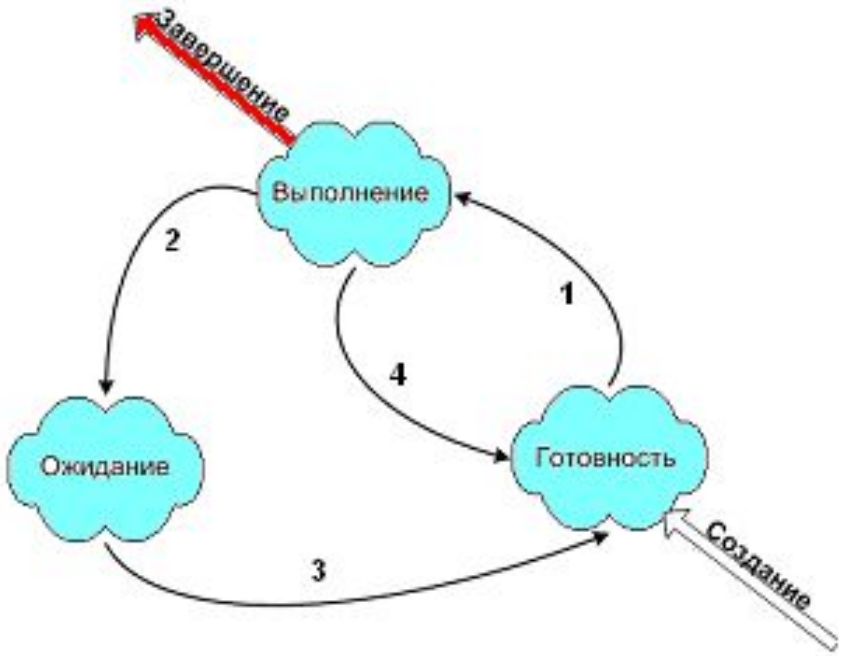
Алгоритмы планирования, основанные на приоритетах

Приоритет – это число, характеризующее степень привилегированности потока при использовании ресурсов ОС.

- Приоритеты
  - фиксированные
    - приоритет потоку назначается ОС при его создании и не изменяется за время существования потока
  - динамические
    - приоритет может быть изменен либо по инициативе самого потока, либо по инициативе пользователя, либо ОС изменяет приоритеты потоков

- Приоритетное планирование
  - с относительными приоритетами – приоритет учитывается только при выборе потока на выполнение из очереди готовых потоков
  - с абсолютными приоритетами выполнение активного потока прерывается, когда в очереди готовых к выполнению потоков появляется поток, приоритет которого выше, чем приоритет активного потока.

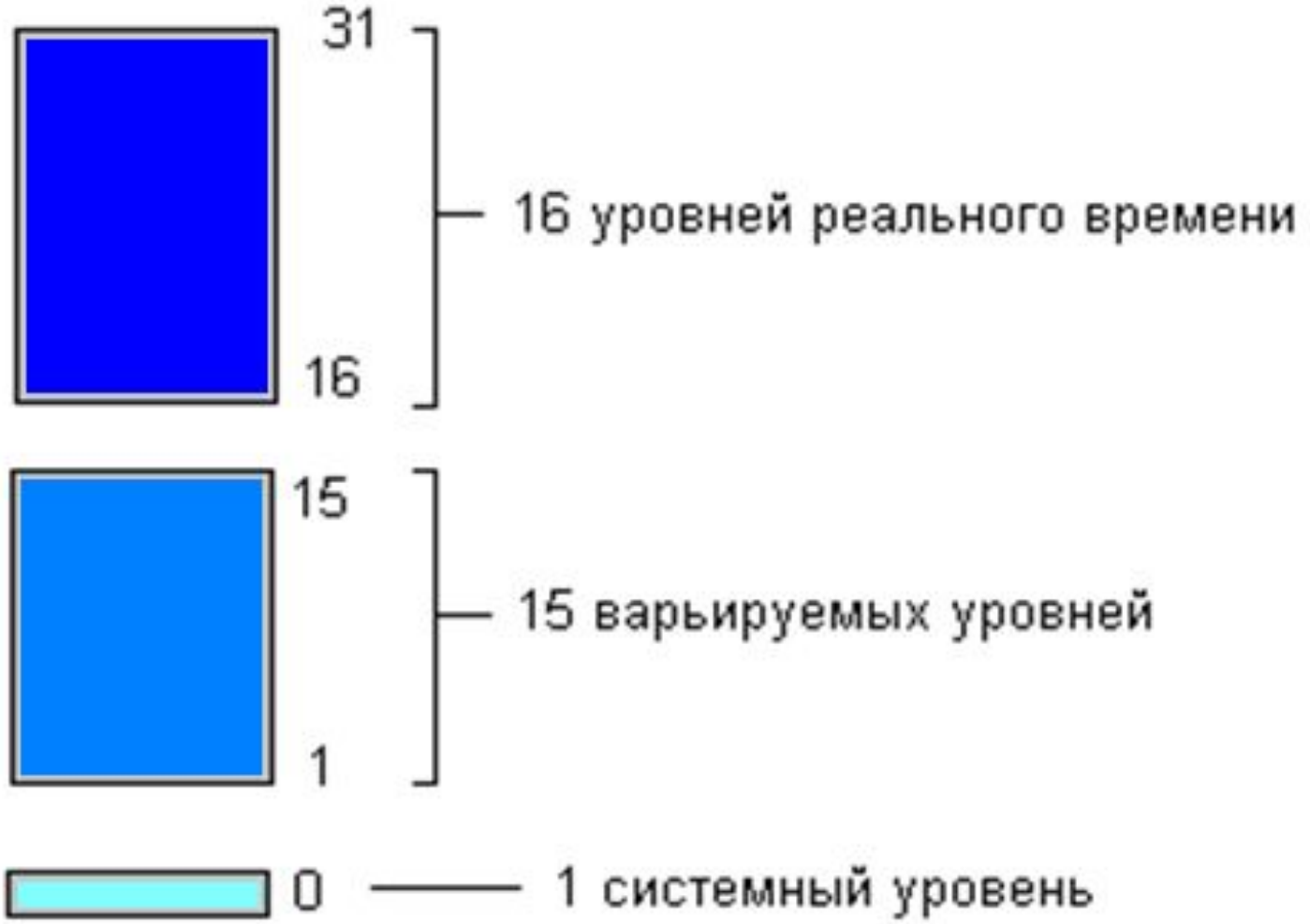




Смешанный алгоритм планирования

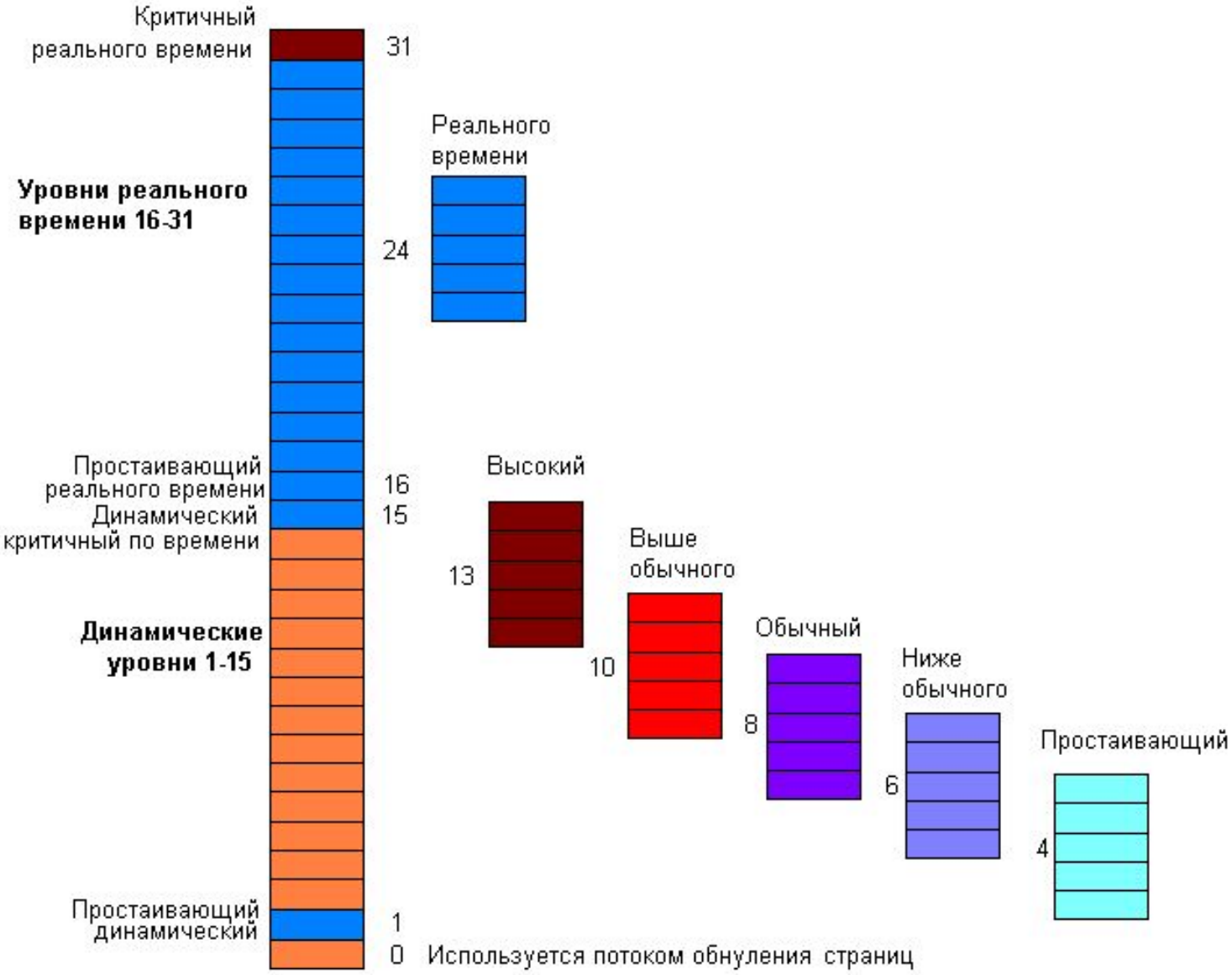
Квантование +приоритеты

Схема назначения приоритета потокам в Windows NT



Классы приоритетов процессов и приоритеты потоков Win32

Приоритеты потоков Win32	Классы приоритетов процессов Win32					
	Real-time	High	Above Normal	Normal	Below Normal	Idle
Time critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below normal	23	12	9	7	5	3
Lowest	22	11	8	6	4	2
Idle	16	1	1	1	1	1


Критичный реального времени
31
Реального времени
Уровни реального времени 16-31
24
Простаивающий реального времени
16
Динамический критичный по времени
15
Высокий
13
Выше обычного
Динамические уровни 1-15
10
Обычный
8
Ниже обычного
6
Простаивающий
4
Простаивающий динамический
1
0
Используется потоком обнуления страниц

Алгоритм планирования Linux

- В операционной системе Linux поддерживаются три класса потоков:
- 1. потоки реального времени, обслуживаемые по алгоритму FIFO;
- 2. потоки реального времени, обслуживаемые в порядке циклической очереди;
- 3. потоки разделения времени.

Linux различает 140 уровней приоритета.

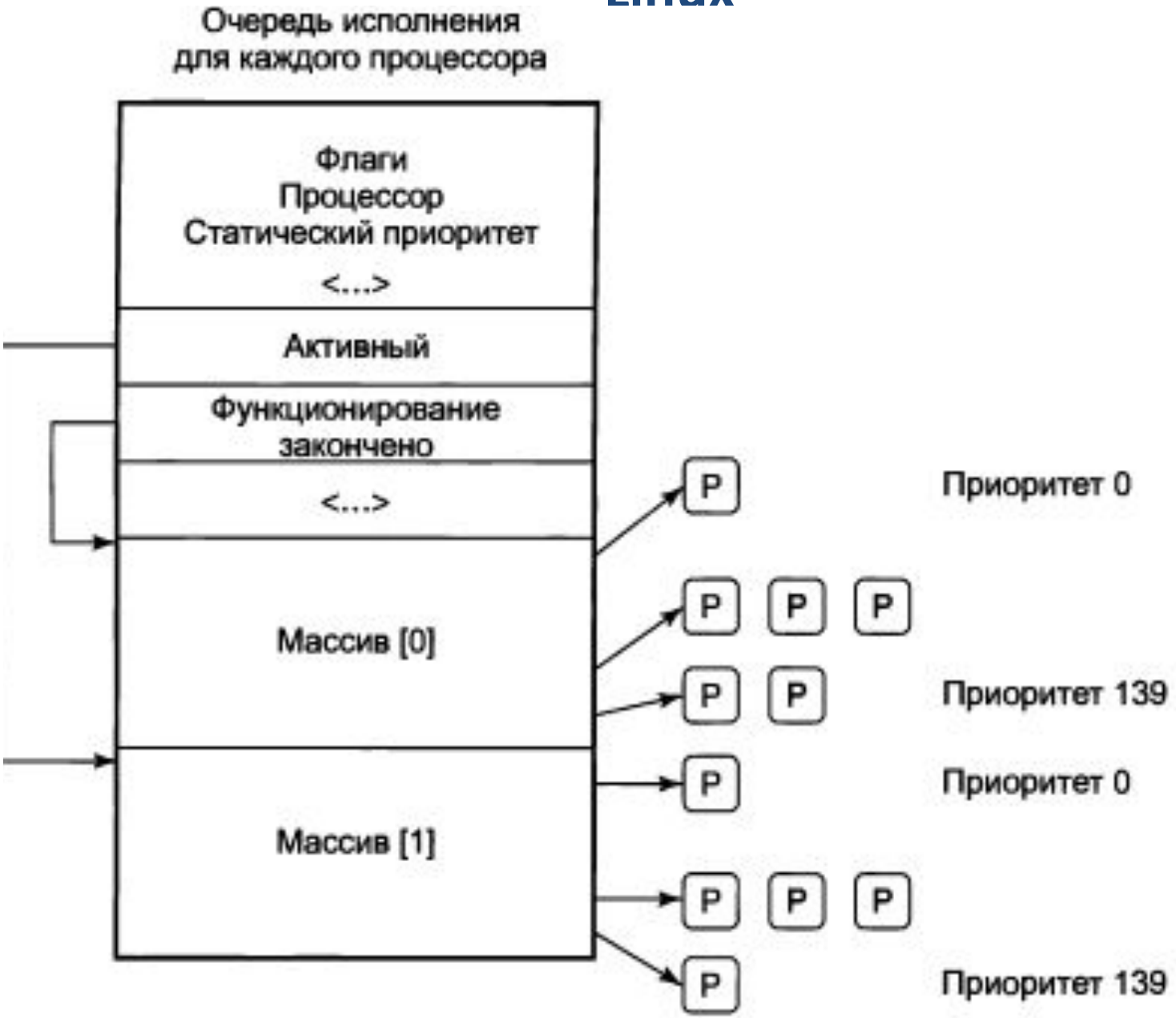
Потоки реального времени имеют приоритеты от 0 до 99, причем 0 – самый высокий приоритет.

Обычному потоку ставится в соответствие уровень приоритета от 100 до 139.

Каждому уровню приоритета обычных потоков соответствует свое значение длительности кванта времени.

Linux связывает с каждым потоком значение nice, которое определяет статический приоритет каждого потока. По умолчанию он равен 0, но его можно изменить при помощи системного вызова nice(value), где value меняется от -20 до +19.

Очередь исполнения и массивы приоритетов в Linux



Алгоритмы планирования в ОС пакетной обработки информации

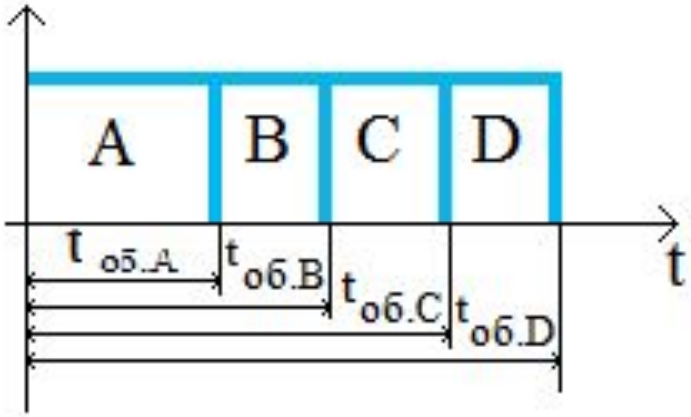
1. "Первый пришел - первым обслужен" (FIFO)

+ Достоинства:
- простота;
- справедливость.

2. "Кратчайшая задача – первая»

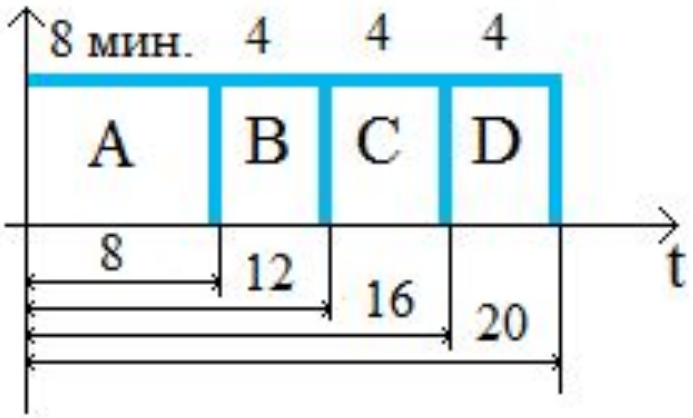
Минимизирует среднее оборотное время выполнения задачи.

Оборотное время – время, прошедшее от момента запуска всего пакета на выполнение до получения результата задачи.

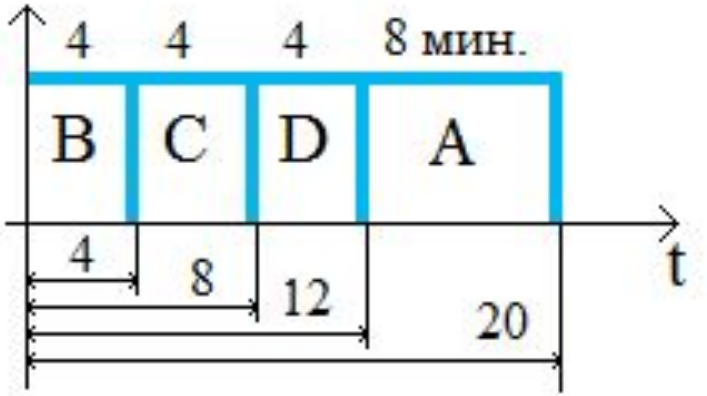


алгоритма: первой на выполнение запускается самая короткая задача из пакета

Задачи:	A	B	C	D
Время выполнения:	8 мин.	4 мин.	4 мин.	4 мин.



$$t_{об.\,ср.} = \frac{t_{об.A} + t_{об.B} + t_{об.C} + t_{об.D}}{4} = \frac{8 + 12 + 16 + 20}{4} = 14 \text{ мин.}$$



$$t_{об.\,ср.} = \frac{t_{об.A} + t_{об.B} + t_{об.C} + t_{об.D}}{4} = \frac{4 + 8 + 12 + 20}{4} = 11 \text{ мин.}$$

•Достоинства:
- уменьшение оборотного времени

•справедливость

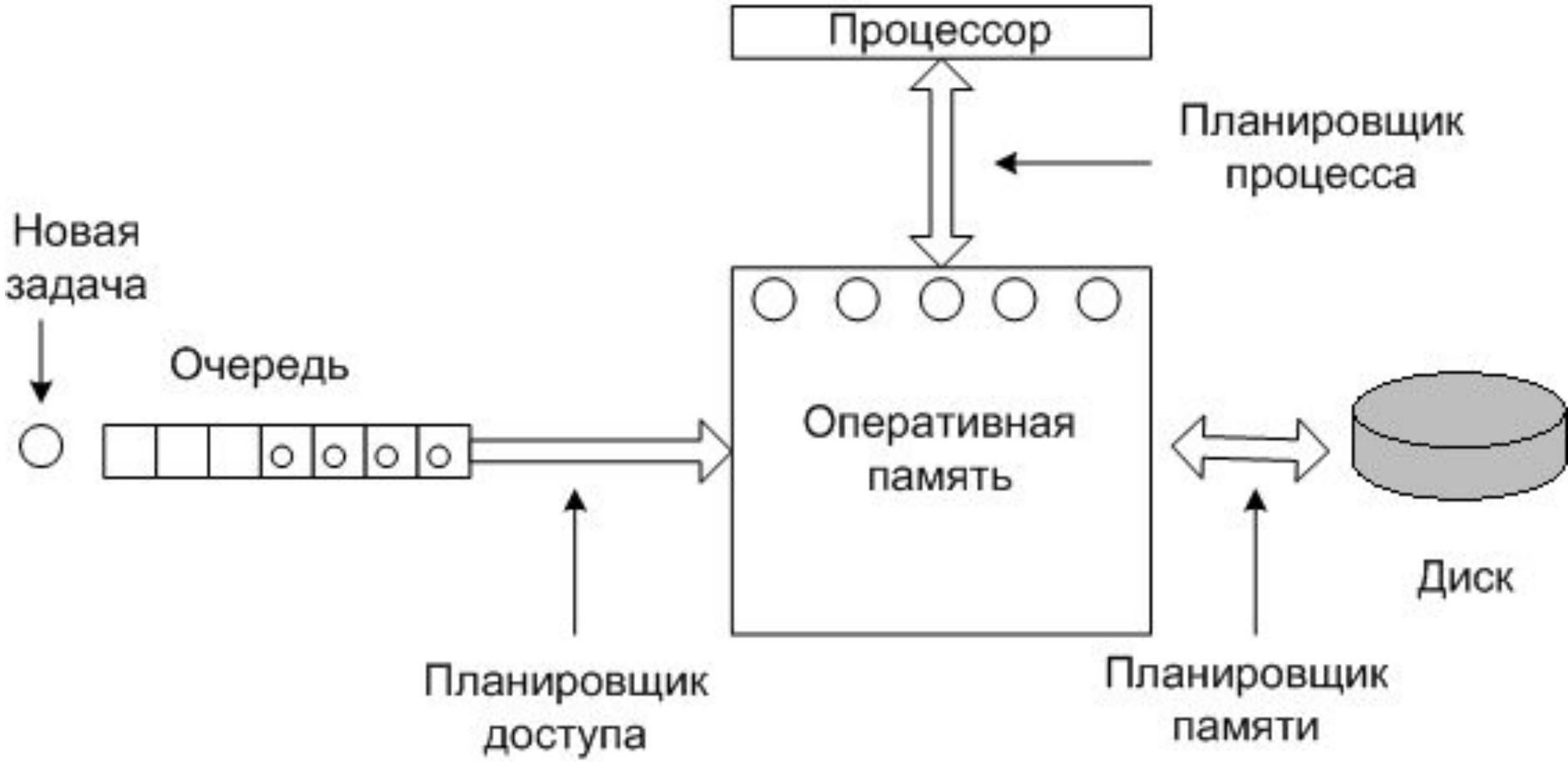
•Недостатки:

•требуется превентивная информация о времени выполнения задач

•длинный процесс, занявший процессор, не пустит более новые краткие процессы, которые пришли позже.

3. Наименьшее оставшееся время выполнения

4. Трехуровневое планирование



Планирование в интерактивных системах

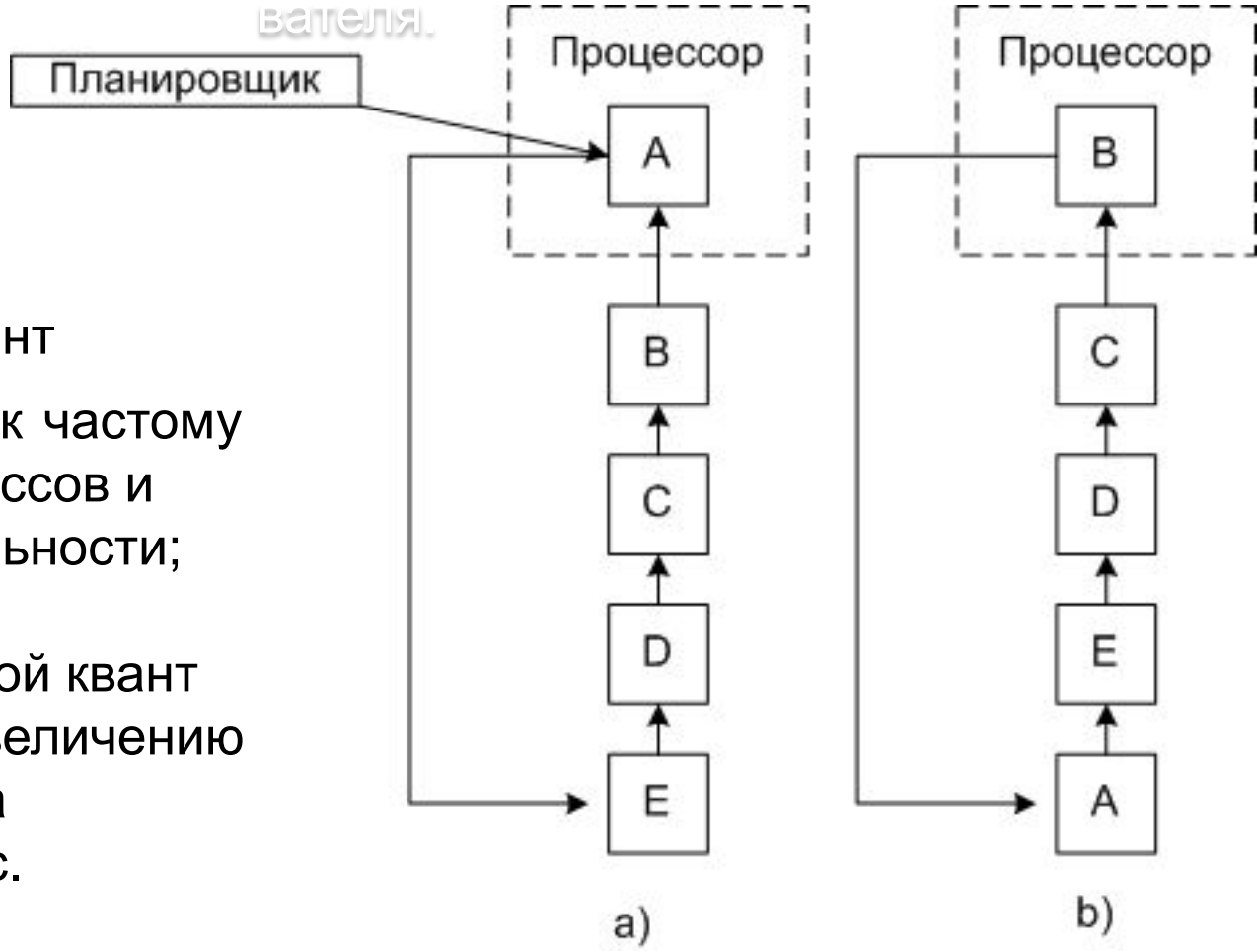
1. Циклическое планирование (квантование)

+ простота;

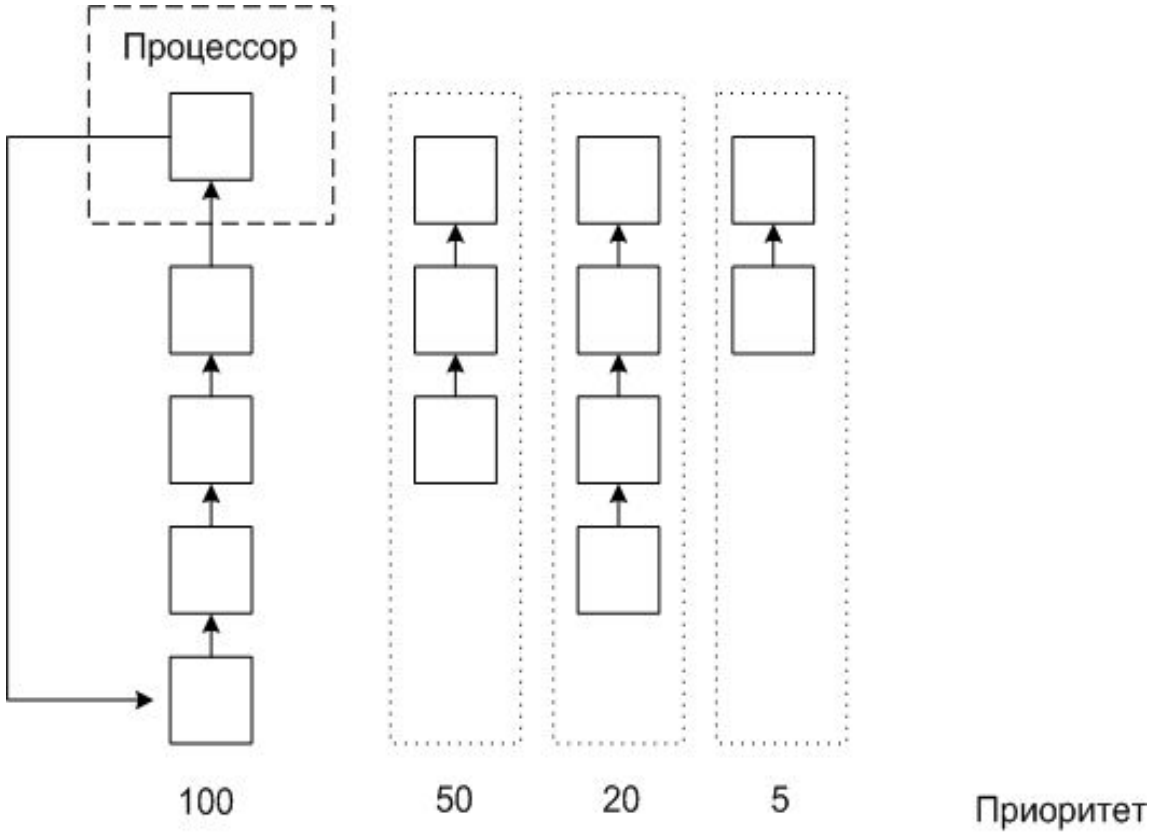
справедливость.

- - слишком малый квант времени приводит к частому переключению процессов и снижению производительности;

- слишком большой квант может привести к увеличению времени ответа на интерактивный запрос.



2.Приоритетное планирование



3. Самый короткий процесс - следующий

4.Гарантированное планирование

Если в процессе работы в системе зарегистрировано n пользователей, то вы получите 1/n от мощности центрального процессора. Аналогично этому, в однопользовательской системе, имеющей n работающих процессов, при прочих равных условиях каждый из них получит 1/n от общего числа процессорных циклов.

Суть алгоритма

Необходимо отслеживать, сколько процессорного времени затрачено на каждый процесс с момента его создания. Затем вычисляют отношение времени, фактически полученного процессом к количеству времени, на которое он имел право.

На выполнение выбирается процесс с наименьшим отношением, который будет работать до тех пор, пока его соотношение не превысит соотношение его ближайшего конкурента.

5. Лотерейное планирование

Основная идея состоит в раздаче процессам лотерейных билетов на доступ к различным системным ресурсам, в том числе и к процессорному времени. Когда планировщику нужно принимать решение, в случайном порядке выбирается лотерейный билет, и ресурс отдается процессу, обладающему этим билетом. Применительно к планированию процессорного времени система может проводить лотерейный розыгрыш 50 раз в секунду, и каждый победитель будет получать в качестве приза 20 мс процессорного времени.

6. Справедливое планирование

Некоторые системы перед планированием работы процесса берут в расчет, кто является его владельцем. В этой модели каждому пользователю распределяется некоторая доля процессорного времени и планировщик выбирает процессы, соблюдая это распределение. Таким образом, если каждому из двух пользователей было обещано по 50% процессорного времени, то они его получат, независимо от количества имеющихся у них процессов.

Планирование в системах реального времени

- *Критерий эффективности – способность системы выдерживать заранее заданные интервалы времени между запуском программы и получением результата (реактивность системы).*

•Системы реального времени

•жесткие

• несоблюдение временных ограничений приводит к катастрофическим последствиям

•гибкие

• нарушения временного графика нежелательны, но допустимы

- Внешние события
  - периодические – начиная с момента первоначального запроса все будущие моменты возникновения задачи можно определить заранее
  - спорадические - моменты возникновения запросов заранее неизвестны

- Задачи
  - независимые
  - взаимосвязанные

$\{T_i\}$ - *периодический набор задач*

p_i - *периоды*

d_i - *предельные сроки*

c_i - *требования к времени выполнения*

μ *– коэффициент использования процессора*

$$\mu_i = c_i / p_i$$

Необходимое условие существования расписания:

$$\mu = \sum c_i / p_i \leq k,$$

где k - количество доступных процессоров.

Алгоритм Лью - Лейланда

Классический алгоритм для жестких систем реального времени с одним процессором.

Алгоритм основан на следующих предположениях:

- Запросы на выполнение всех задач набора, имеющих жесткие ограничения на время реакции, являются периодическими.
- Все задачи независимы.
- Срок выполнения задачи равен ее периоду.
- Максимальное время выполнения каждой задачи c_i известно и постоянно.
- Время переключения контекста можно игнорировать.
- Максимальный суммарный коэффициент загрузки процессора $\sum c_i / p_i \leq n(2^{1/n} - 1)$, где n – число задач.

Суть алгоритма: задача с самым коротким периодом получает наивысший приоритет, задача с наибольшим периодом получает наименьший приоритет.

Межпроцессное взаимодействие

- согласование действий процессов;
- передача информации от одного процесса другому;
- контроль над деятельностью процессов.

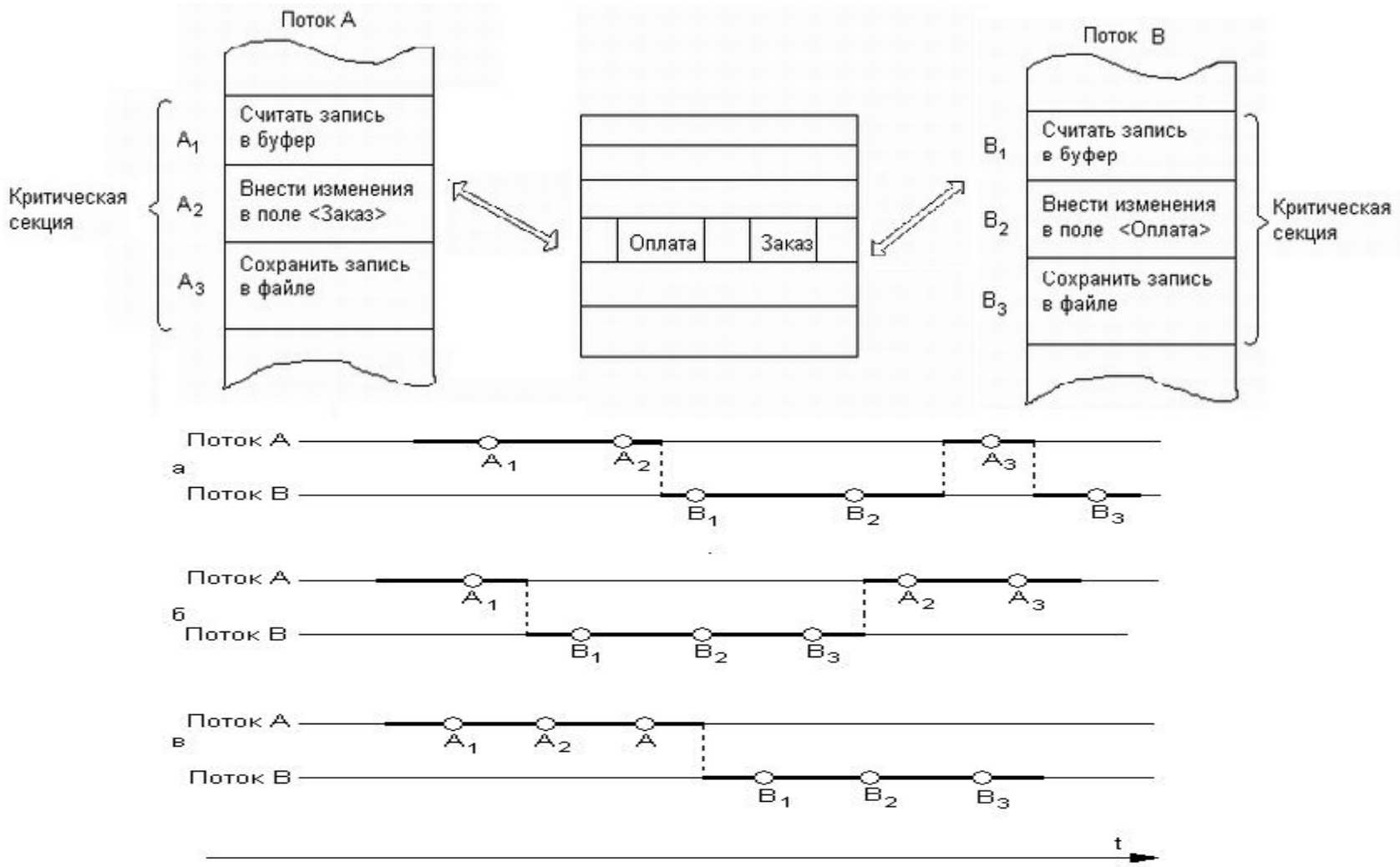
Лекция № 6

Синхронизация процессов и потоков

- Потребность в синхронизации потоков возникает только в мультипрограммной операционной системе и связана с совместным использованием аппаратных и информационных ресурсов вычислительной системы.

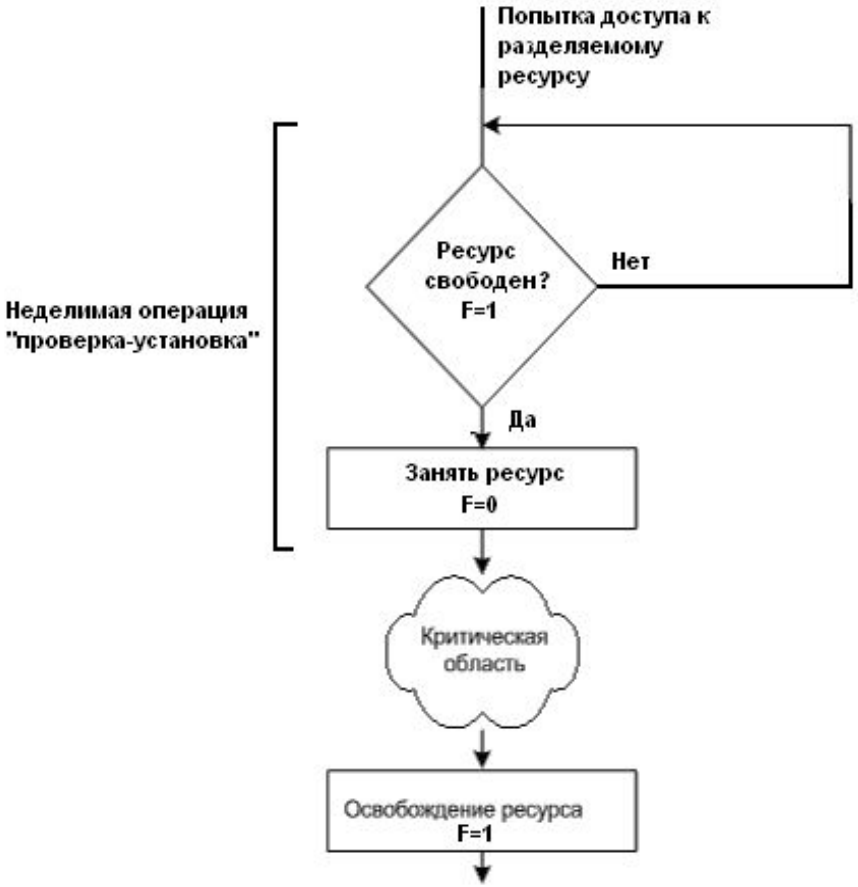
Гонки (взаимные состязания)

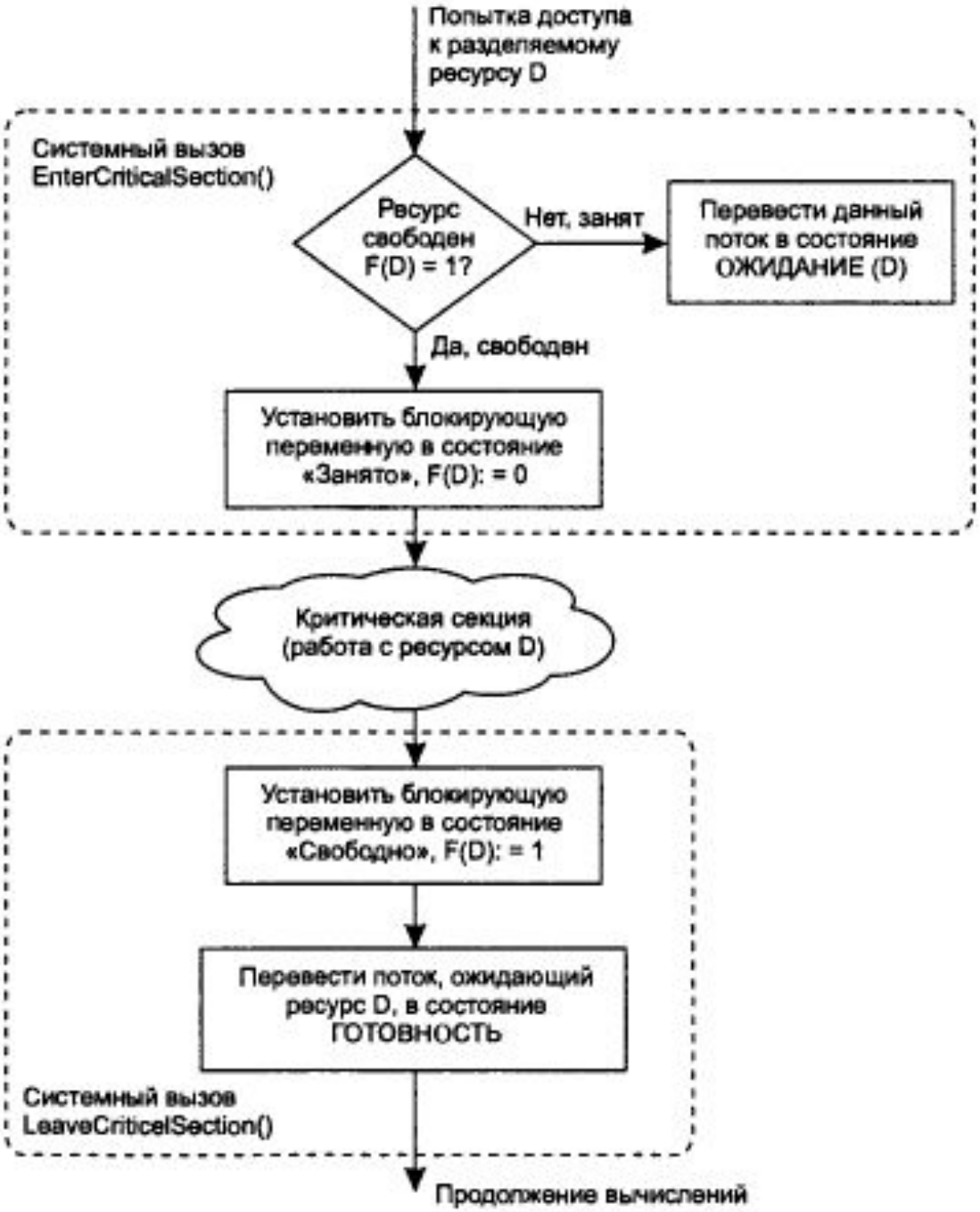
Состязания (гонки)– ситуация, когда два или более потока обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей потоков.



Способы реализации взаимного исключения

1. Запрет прерываний
2. Блокирующие переменные




Попытка доступа к разделяемому ресурсу D
Системный вызов EnterCriticalSection()
Ресурс свободен F(D) = 1?
Нет, занят
Перевести данный поток в состояние ОЖИДАНИЕ (D)
Да, свободен
Установить блокирующую переменную в состояние «Занято», F(D): = 0
Критическая секция (работа с ресурсом D)
Установить блокирующую переменную в состояние «Свободно», F(D): = 1
Перевести поток, ожидающий ресурс D, в состояние ГОТОВНОСТЬ
Системный вызов LeaveCriticelSection()
Продолжение вычислений

Семафоры Дейкстры

Семафоры – переменные, которые могут принимать целые неотрицательные значения и используются для синхронизации вычислительных процессов.

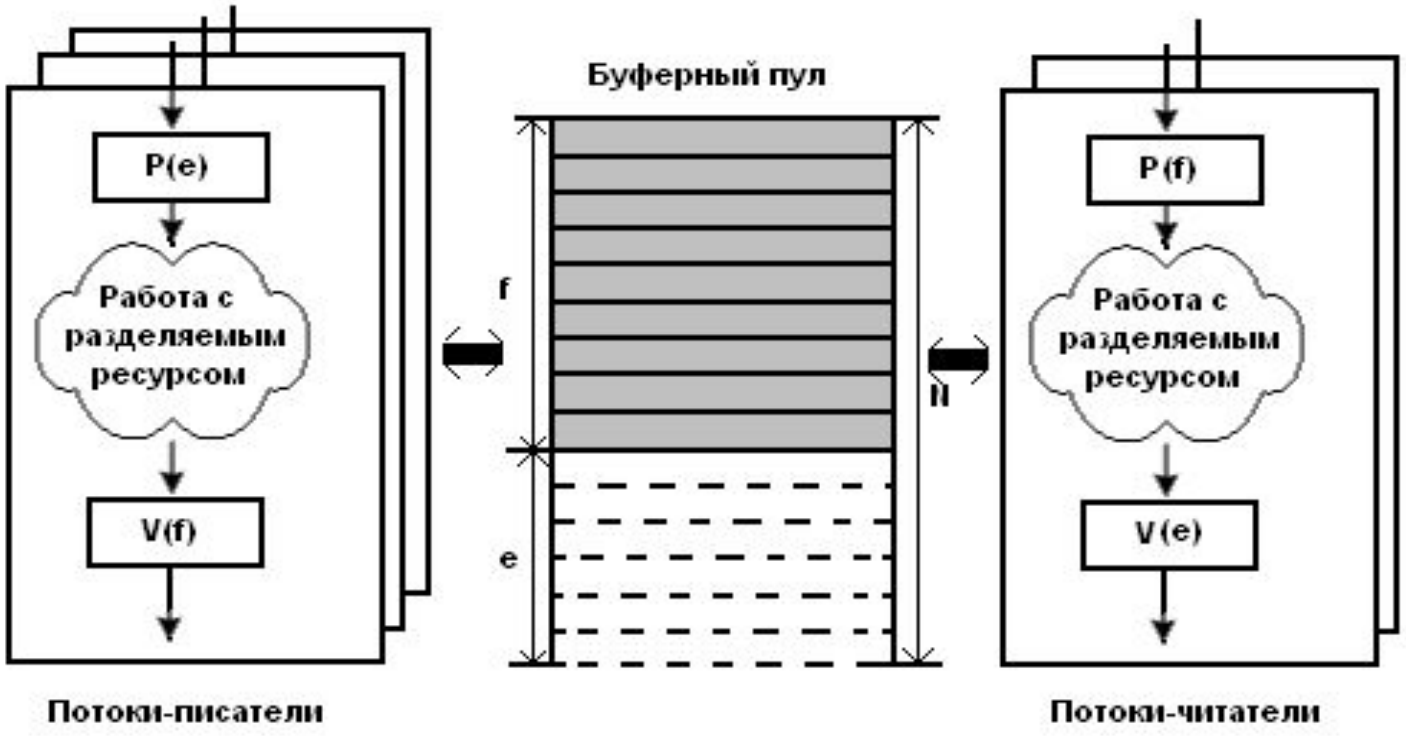
Для работы с семафорами определены два примитива:

- V(S): S=S+1 единым действием;
- P(S): S=S-1 , если возможно; если это невозможно, то поток, вызвавший P(S) переводится в состояние ожидания.

Решение классической задачи синхронизации «читатели – писатели» с помощью семафоров

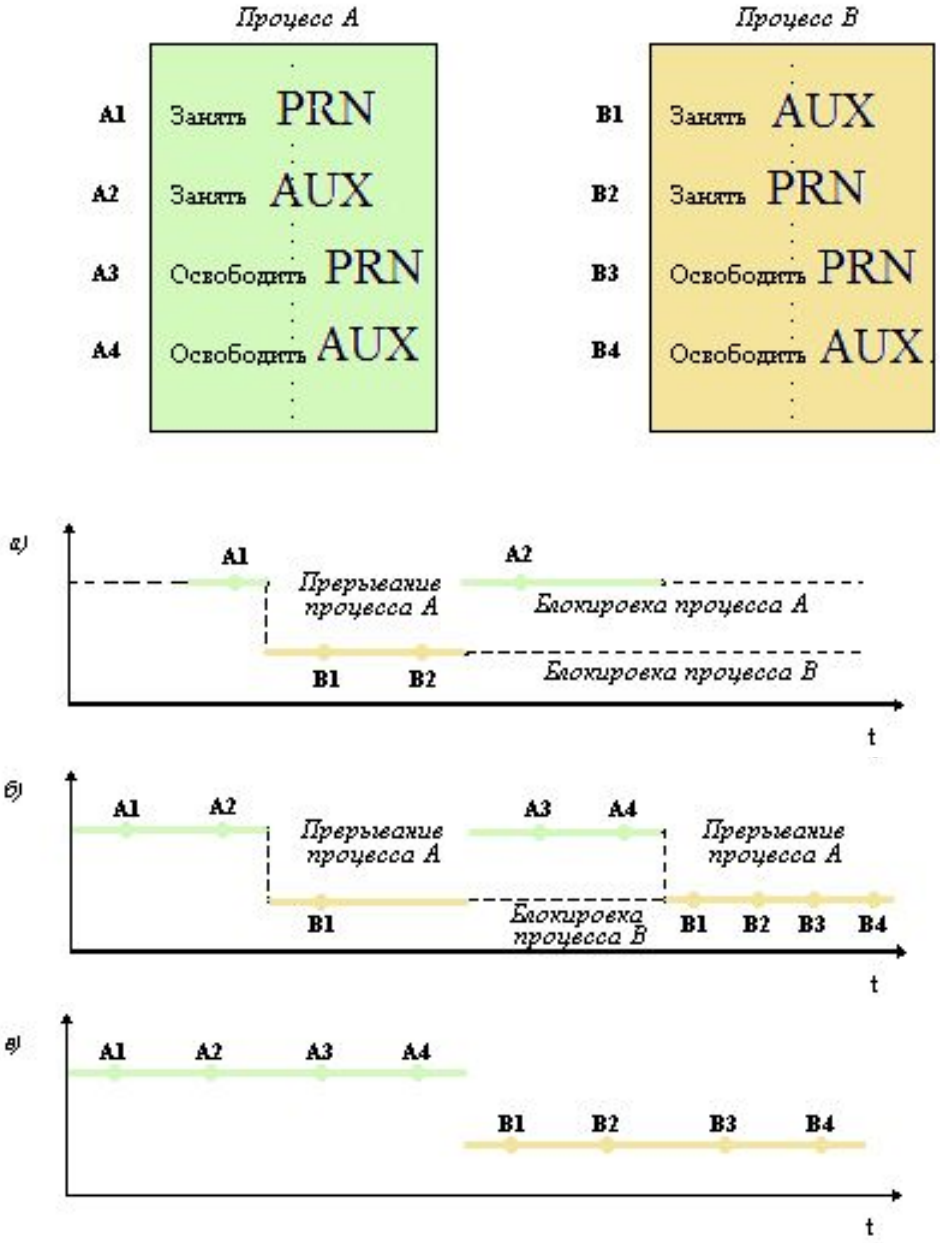
буферный пул состоит из N буферов.

e - число пустых буферов и *f* - число заполненных буферов



Взаимные блокировки (тупики, клинчи, дедлоки)

Взаимная блокировка – ситуация, когда несколько процессов борются за ресурсы, и ни один из них не может завершить начатую работу.


Процесс A
A1 Занять PRN
A2 Занять AUX
A3 Освободить PRN
A4 Освободить AUX
Процесс B
B1 Занять AUX
B2 Занять PRN
B3 Освободить PRN
B4 Освободить AUX
а)
A1
A2
Прерывание процесса A
Блокировка процесса A
B1
B2
Блокировка процесса B
t
б)
A1
A2
Прерывание процесса A
A3
A4
Прерывание процесса A
B1
Блокировка процесса B
B1
B2
B3
B4
t
в)
A1
A2
A3
A4
B1
B2
B3
B4
t

Условия взаимоблокировки:

- **Условие взаимного исключения.**

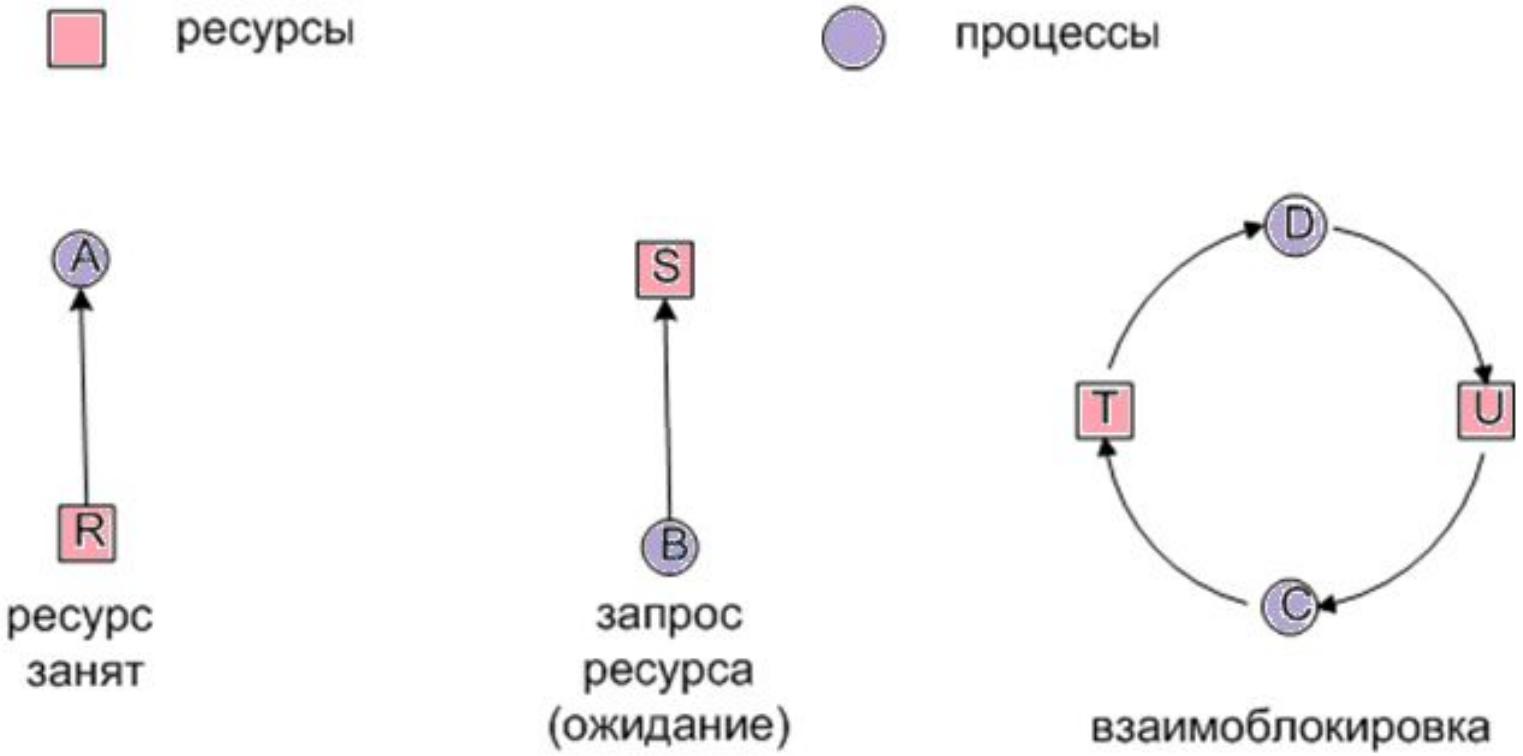
  Каждый ресурс в данный момент или отдан одному процессу или свободен.
- **Условие удержания и ожидания.**

  Процесс, удерживающий в данный момент ресурс, может запрашивать новые ресурсы.
- **Условие отсутствия принудительной выгрузки ресурса.**

  У процесса нельзя забрать ранее полученные ресурсы.
- **Условие циклического ожидания .**

  Должна существовать круговая последовательность из процессов, каждый, из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

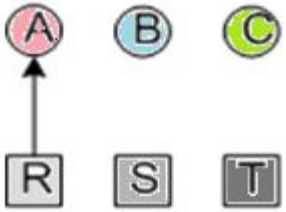
Моделирование взаимоблокировок

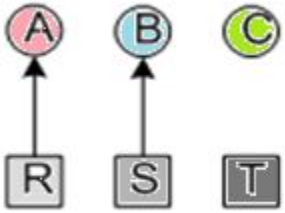


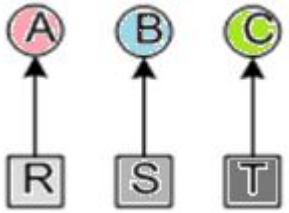
процессы A, B, C

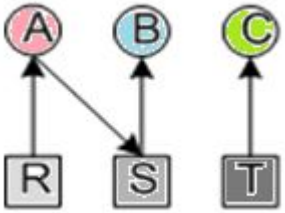
ресурсы R, S, T

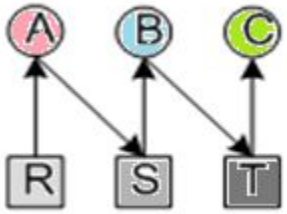
A	B	C
Запросить R	Запросить S	Запросить T
Запросить S	Запросить T	Запросить R
Освободить R	Освободить S	Освободить T
Освободить S	Освободить T	Освободить R

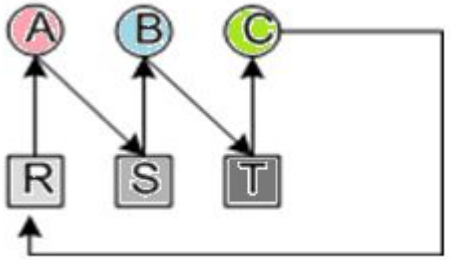


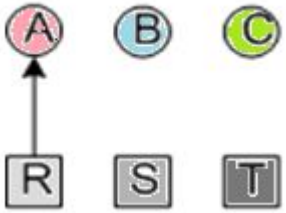


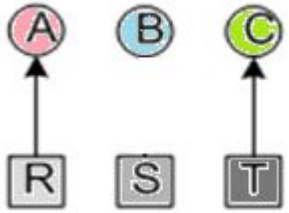


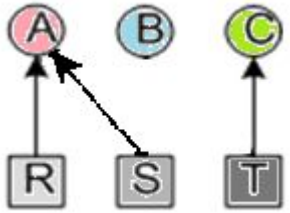


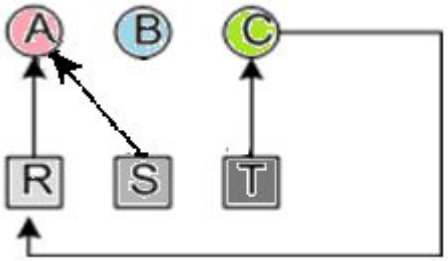


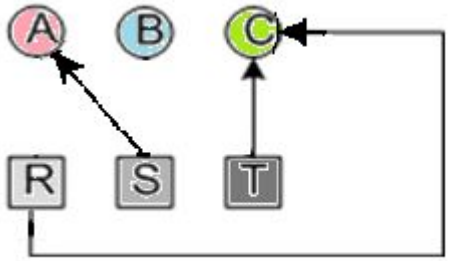


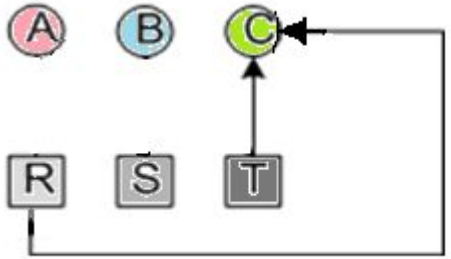

A
B
C
R
S
T


A
B
C
R
S
T


A
B
C
R
S
T


A
B
C
R
S
T


A
B
C
R
S
T


A
B
C
R
S
T

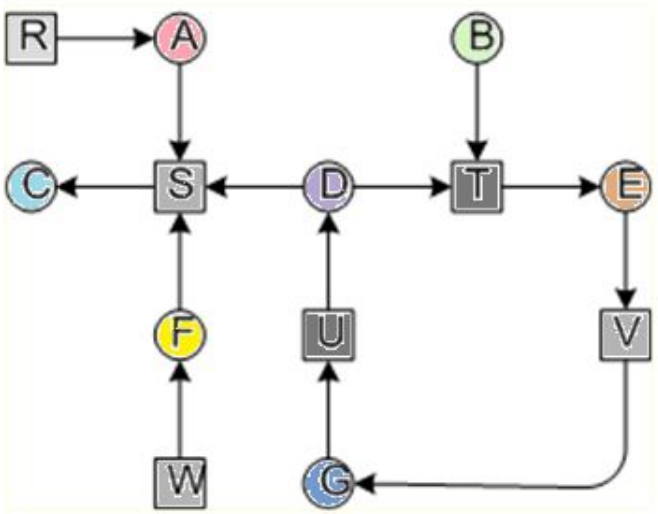
Стратегии при столкновении с взаимными блокировками

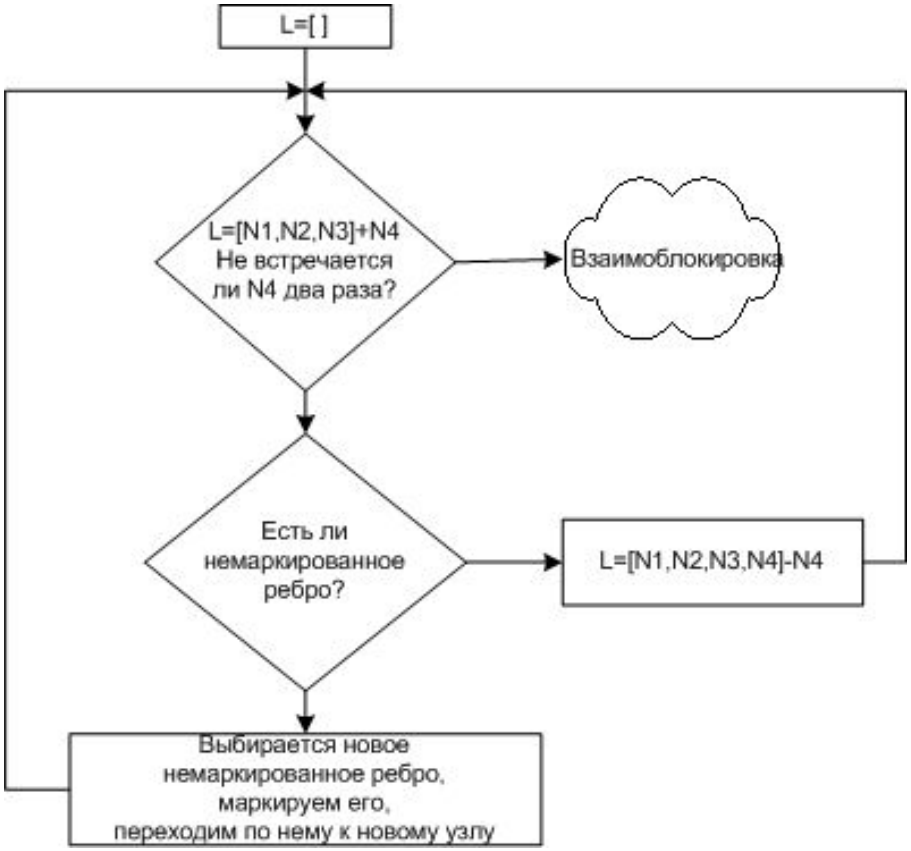
- 1
  - Пренебрежением проблемой в целом (страусовый алгоритм)
- 2
  - Обнаружение и восстановление
- 3
  - Динамическое избежание тупиков
- 4
  - Предотвращение с помощью опровержения хотя бы одного условия, необходимого для возникновения тупика

Обнаружение и устранение взаимоблокировок

1. Обнаружение взаимоблокировки при наличии одного ресурса каждого типа

Для каждого узла N в графе выполняются следующие 5 шагов





2. Обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа

- *m* - число классов ресурсов
- *n* - количество процессов, $P_1 \ldots P_n$
- **E = (**$E_1, E_2, E_3, \ldots, E_m$) - вектор существующих ресурсов, где E_i - количество ресурсов класса *i,*
- **A = (**$A_1, A_2, A_3, \ldots, A_m$) - вектор доступных ресурсов, A_i - количество доступных ресурсов класса *i,*
- ***C*** - матрица текущего распределения ***R*** - матрица запросов

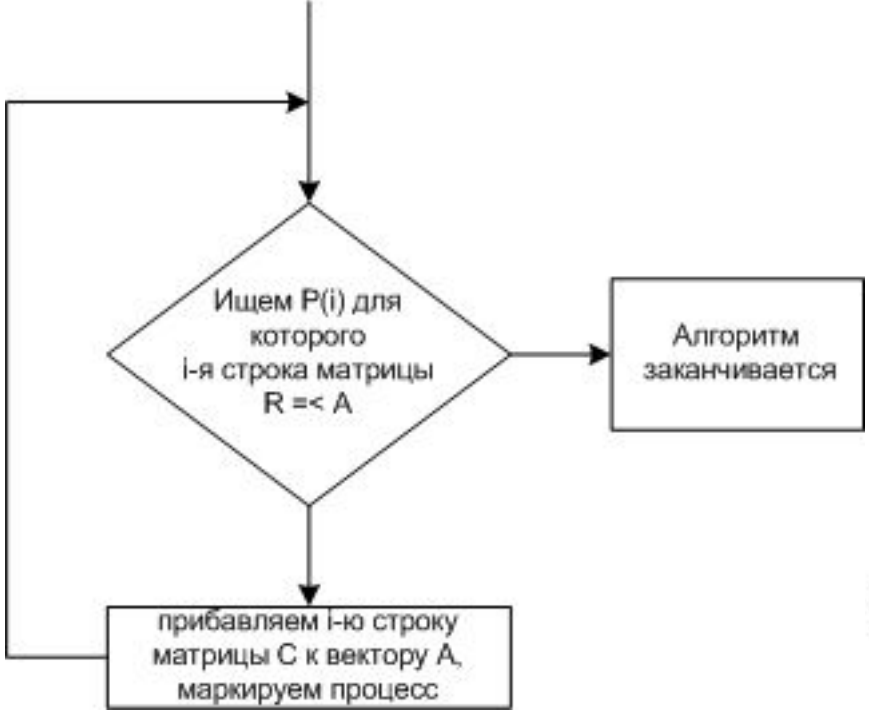
$$\begin{bmatrix} C_{11} & C_{12} & \ldots & C_{1m} \\ C_{21} & C_{22} & \ldots & C_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ C_{n1} & C_{n2} & \ldots & C_{nm} \end{bmatrix}$$

Строка **n** предоставлена процессу **n**

$$\begin{bmatrix} R_{11} & R_{12} & \ldots & R_{1m} \\ R_{21} & R_{22} & \ldots & R_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{n1} & R_{n2} & \ldots & R_{nm} \end{bmatrix}$$

Строка **n** нужна процессу **n**

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$



$$(\underset{\text{принтеры}}{4} \quad \underset{\text{плоттеры}}{2} \quad \underset{\text{сканеры}}{3} \quad \underset{\text{компакт диски}}{1})$$

$$A = (\underset{\text{принтеры}}{2} \quad \underset{\text{плоттеры}}{1} \quad \underset{\text{сканеры}}{0} \quad \underset{\text{компакт диски}}{0})$$

Матрица текущего распределения

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Матрица запросов

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

A= (2 2 2 0)

A= (4 2 2 1)

Когда следует искать тупики:

- Когда запрашивается очередной ресурс
- Периодически, через какой-то определенный промежуток времени
- Когда загрузка процессора слишком мала

Выход из взаимной блокировки

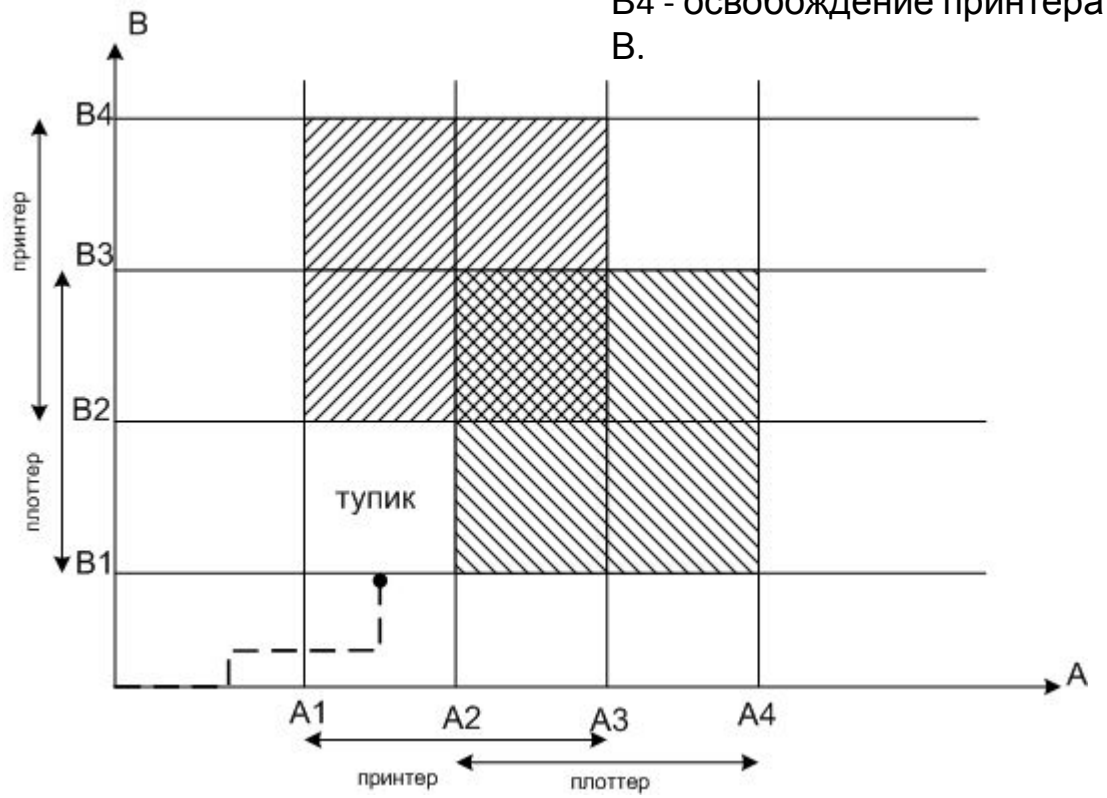
- 1
  - Восстановление при помощи принудительной выгрузки ресурса
- 2
  - Восстановление через откат
- 3
  - Восстановление путем уничтожения процесса

Динамическое избежание взаимоблокировок

Траектории ресурсов

A1 - запрос принтера процессом A,
A2 - запрос плоттера процессом A,
A3 - освобождение принтера процессом A,
A4 - освобождение плоттера процессом A

B1 - запрос плоттера процессом B,
B2 - запрос принтера процессом B,
B3 - освобождение плоттера процессом B,
B4 - освобождение принтера процессом B.



Опасные и безопасные состояния

Состояние безопасно, если система не находится в тупике и существует некоторый порядок планирования, при котором каждый процесс может работать до завершения, даже если все процессы захотят получить свое максимальное количество ресурсов.

	имеет	max
A	3	9
B	2	4
C	2	7

Свободно:3

	имеет	max
A	3	9
B	4	4
C	2	7

Свободно:1

	имеет	max
A	3	9
B	0	-
C	2	7

Свободно:5

	имеет	max
A	3	9
B	0	-
C	7	7

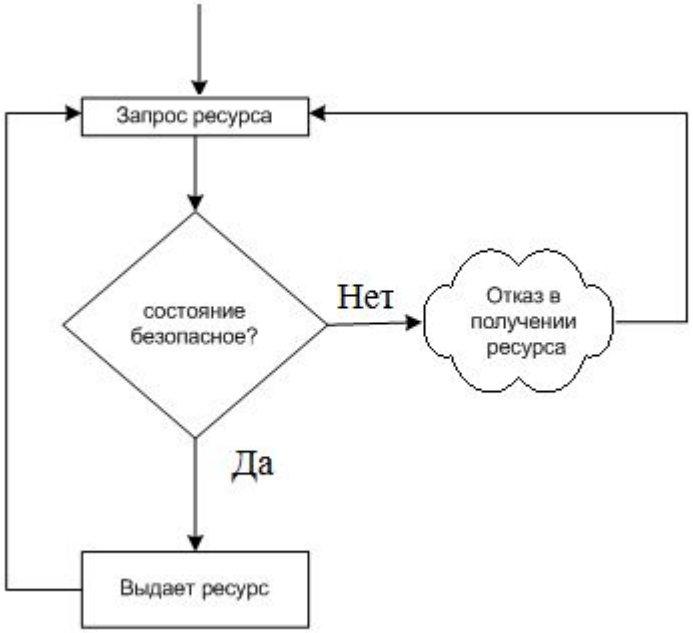
Свободно:0

	имеет	max
A	3	9
B	0	-
C	0	-

Свободно:7

В безопасном состоянии система может гарантировать, что все процессы закончат свою работу, в небезопасном состоянии такой гарантии дать нельзя.

Алгоритм банкира для одного вида



	имеет	max
A	0	6
B	0	5
C	0	4
D	0	7

Свободно: 10

	имеет	max
A	1	6
B	1	5
C	2	4
D	4	7

Свободно: 2

	имеет	max
A	1	6
B	2	5
C	2	4
D	4	7

Свободно: 1

Алгоритм банкира для несколько видов ресурсов

E=(6342) - существующие ресурсы,
P=(5322) - занятые ресурсы,
A=(1020) - доступные ресурсы

	принтеры	плоттеры	сканеры	устройства компакт-дисков
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Распределенные ресурсы

	принтеры	плоттеры	сканеры	устройства компакт-дисков
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Ресурсы, которые еще нужны

R

Предотвращение условий, необходимых для взаимоблокировок

Системные средства синхронизации

- Системные семафоры;
- мьютексы;
- события;
- таймеры;
- файлы, процессы, потоки…

объекты ядра

ОС	Wait ( )	Set ( )
Windows NT	WaitForSingleObjekt ( ) WaitForMultipleObjekt ( )	SetEvent ( )
UNIX	sleep ( )	wakeup ( )
OS/2	DosSemWait ( )	DosSemSet ( )

- **Мьютексы** (от MUTual Exclusion - взаимоисключения) – объекты ядра позволяют координировать взаимное исключение доступа к разделяемому ресурсу.
- **Системные семафоры** - принцип действия мьютексов, но в них заложена возможность подсчета ресурсов, что позволяет заранее определенному числу потоков одновременно войти в синхронизуемый участок кода.

События используются в качестве сигналов о завершении какой-либо операции.

- События
  - с ручным сбросом
  - (все потоки,ожидающие наступления события, переходят в состояние «готовность»
  - с автоматическим сбросом
  - (как и в случае мьютекса, в состояние «готовность» переводится только один поток

Сигнал

или виртуальное прерывание является сообщением, которое система посылает процессу или один процесс посылает другому.

Мониторы Хоара

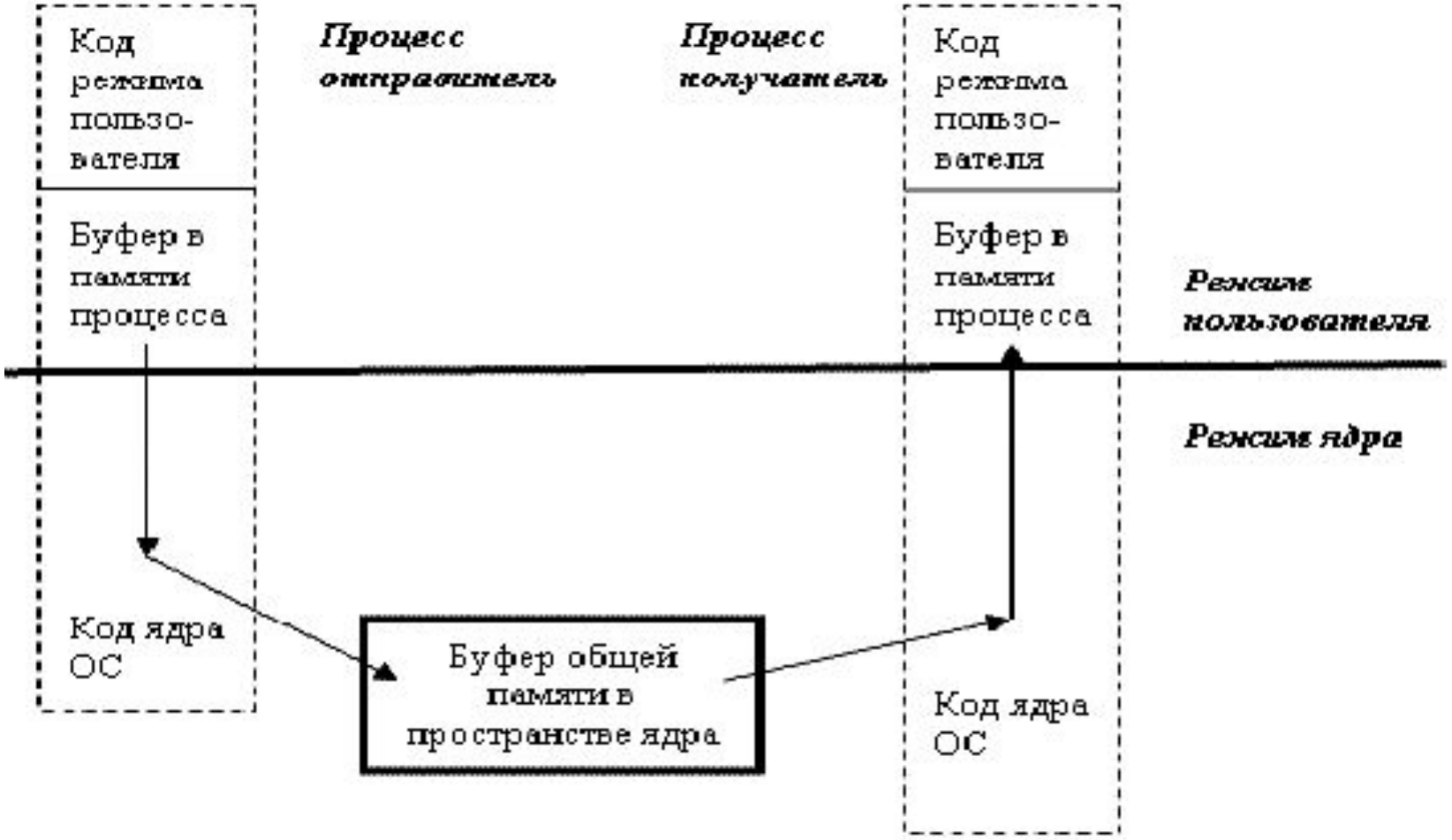
Монитор – это пассивный набор разделяемых переменных и повторно входимых процедур доступа к ним, которыми процессы пользуются в режиме разделения, причем в каждый момент времени им может пользоваться только один процесс.

Ждущие таймеры

- Ждущий таймер
  - Режим «ручного сброса»
  - таймер переходит в установленное состояние при истечении заданной задержки и остается установленным до тех пор, пока функция SetWaitableTimer не задаст новую задержку
  - Режим «автоматического сброса»
  - таймер переходит в установленное состояние при истечении заданной задержки и остается установленным до первого успешного вызова функции ожидания. Каждый раз при истечении времени задержки разрешается выполнение лишь одной нити
  - Режим интервального таймера, который перезапускается с заданной задержкой после каждого срабатывания объекта

Обмен данными между процессами и потоками

•Конвейер (канал, pipe) – представляет собой буфер в оперативной памяти, поддерживающий очередь байт по алгоритму FIFO.



Каналы

- **Каналы**
  - **Безымянные каналы**
  - позволяют обмениваться данными только родственным процессам
  - **Именованные каналы**
  - имеют имя, которое является записью в каталоге файловой системы ОС, поэтому пригодны для обмена данными между двумя произвольными процессами или потоками этих процессов.

Очереди сообщений

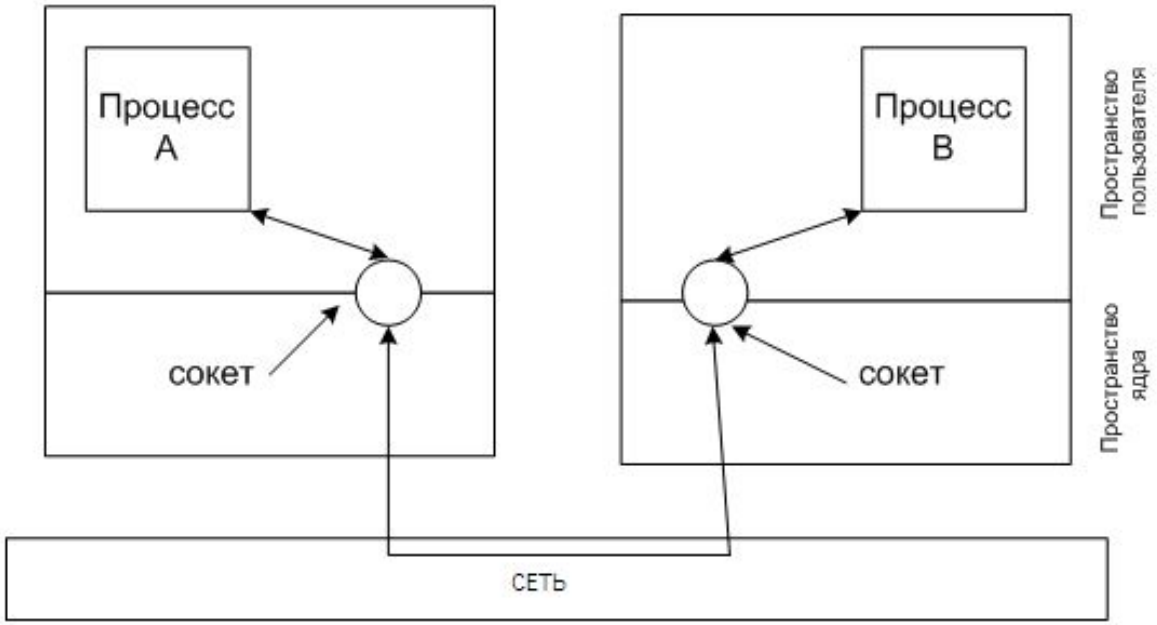
- позволяют процессам и потокам обмениваться структурированными сообщениями;

- являются глобальными средствами коммуникаций для процессов, так как каждая очередь в пределах ОС имеет уникальное имя.

Почтовые ящики

Почтовые ящики обеспечивают только однонаправленные соединения.

Сокеты


Процесс
A
Процесс
B
сокет
сокет
Пространство
пользователя
Пространство
ядра
СЕТЬ

Разделяемая память