Сокеты

Цели и задачи

Обеспечить унифицированное взаимодействие между процессами как выполняющимися на одном компьютере, так и на разных хостах в сети.

Понятие «Сокет»

Коммуникационный узел, обеспечивающий прием и передачу данных для процесса. Сокет имеет исключительно программный интерфейс. Сокет существует, пока счетчик ссылок на него не равен нулю.

Коммуникационный домен

Сокет создается в рамках коммуникационного домена, который определяет набор коммуникационных характеристик в некоторой унифицированной форме.

Коммуникационные характеристики (1)

- Упорядоченная доставка данных
- Отсутствие дублирования данных
- Надежная доставка данных

Коммуникационные характеристики (2)

- Сохранение границ сообщений
- Поддержка передачи экстренных сообщений
- Предварительное установление соединений

BSD-сокеты

- Дейтаграммный сокет (datagram socket)
- Потоковый сокет (stream socket)
- Пакетный сокет (packet socket)
- Сокет низкого уровня (raw socket)

Дейтаграммный сокет

SOCK_DGRAM

Теоретически ненадежная, несвязная передача пакетов (дейтаграмм)

Потоковый сокет

SOCK_STREAM

Надежная передача потока байтов без сохранения границ сообщений. Возможна передача экстренных данных.

Пакетный сокет

SOCK_SEQPACKET

Надежная последовательная передача данных без дублирования с предварительным установлением связи и сохранением границ сообщений.

Сокет низкого уровня

SOCK_RAW
Обеспечивает непосредственный доступ к коммуникационному протоколу.

Коммуникационные домены

AF_UNIX (PF_UNIX)
AF_INET (PF_INET)
AF_INET6 (PF_INET6)
AF_NS (PF_NS)

AF_UNIX

Локальное межпроцессное взаимодействие в пределах единой ОС. Используются внутренние протоколы.

AF_INET

Взаимодействие процессов удаленных систем.
Используется стек протоколов TCP/IP.

Двунаправленный канал

int socketpair(int domain, int type, int protocol, int sv[2]);

Создается двунаправленный канал, напоминающий два неименованных канала, направленных в противоположные стороны.

Создание сокета

int socket(int domain, int type, int protocol);

Создает сокет заданного типа в заданном домене. Протокол чаще всего выбирается по умолчанию.

Поддержка различных типов сокетов в доменах

Домен:	AF UNIX	**AF_INET**
Тип сокета		
SOCK STREAM	Да	Да
SOCK DGRAM	Да	Да
SOCK SEQPACKET	Нет	Нет
SOCK RAW	Нет	Да

Поддержка различных протоколов

Сокет	Протокол
SOCK_STREAM	IPPROTO_TCP (TCP)
SOCK_DGRAM	IPPROTO_UDP (UDP)
SOCK_RAW	IPPROTO_ICMP (ICMP)
SOCK_RAW	IPPROTO_RAW (IP)

Привязка сокета

int bind(int socket,

struct sockaddr * addr, int len);

Осуществляется привязка сокета к локальному адресу (получение сокетом имени). В некоторых случаях привязка происходит неявно.

Структура sockaddr

```
short sa_family;

char sa_data[14];
```

Структура sockaddr_un

```
short sun_family;

char sun_path[108];
```

Структура sockaddr_in

```
short sin_family;
u_short sin_port;
struct in_addr sin_addr;
char sin_zero[8];
```

Адреса сокетов в различных доменах

UNIX домен

sockaddr un

AF_UNIX	2 ба
Имя файла	до 10

Internet домен

sockaddr in

AF_INET	2 байта
port	2 байта
IP-адрес	4 байта
Не используется	8 байт

Преобразование различных форматов IP-адресов

```
int inet_aton(const char *cp,
        struct in_addr *inp);
in_addr_t inet_addr(const char *cp);
in_addr_t inet_network(const char *cp);
char *inet_ntoa(struct in_addr in);
struct in_addr inet_makeaddr(int net,
        int host);
in_addr_t inet_lnaof(struct in_addr in);
in_addr_t inet_netof(struct in_addr in);
```

Создание очереди входящих запросов

int listen(int socket, int backlog);

Параметр backlog определяет максимальное число запросов, которые могут ожидать соединения. (Уже установленные соединения не учитываются)

Прием входящего соединения

```
int accept(int socket,
   struct sockaddr * clntaddr,
   int * addrlen);
```

Блокирующий системный вызов.

Установление соединения со стороны клиента

```
int connect(int socket,
   struct sockaddr * srvaddr, int len);
```

Блокирующий системный вызов.

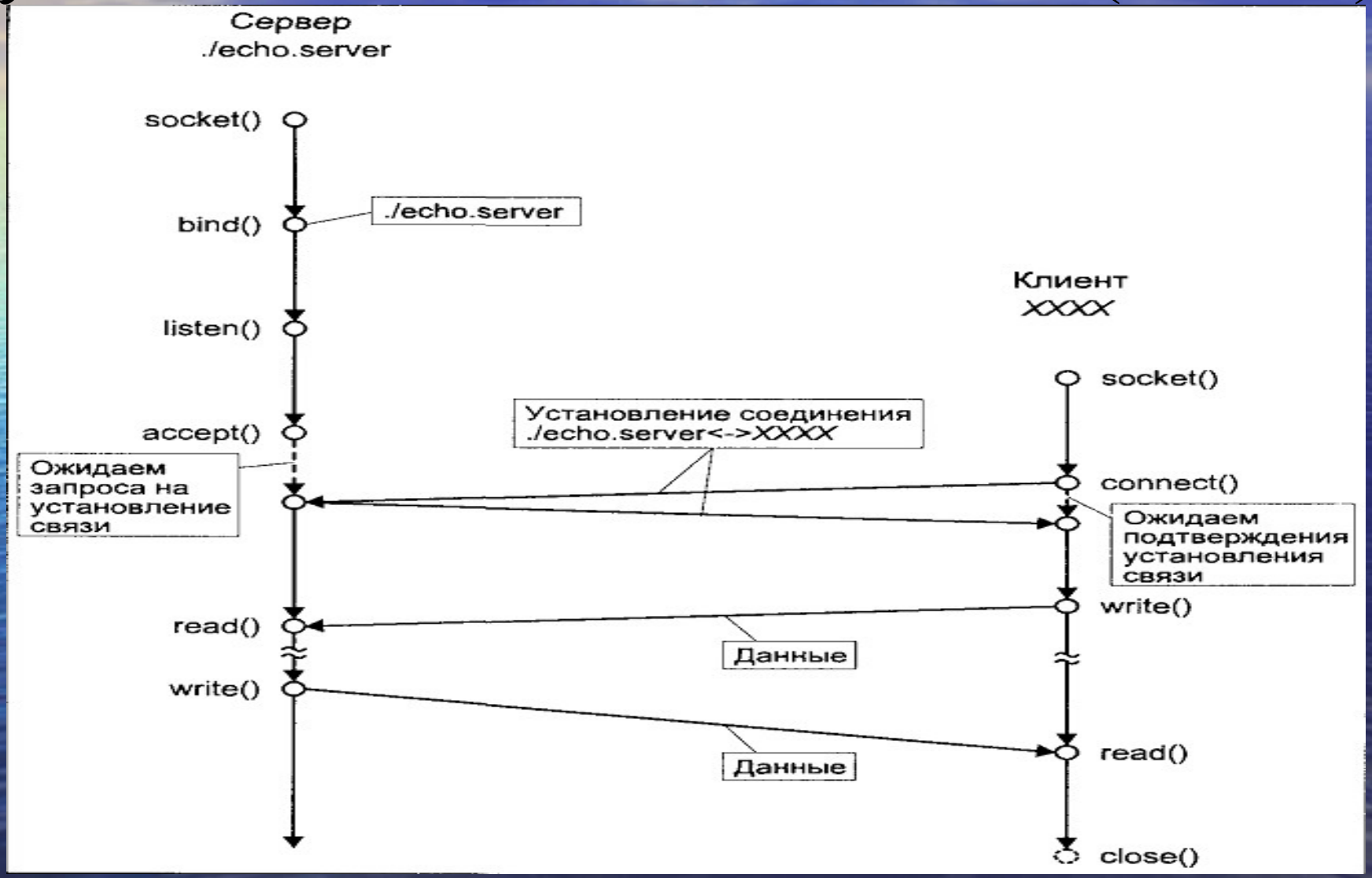
Получение данных

```
int recv(int s, char * msg, int len,
   int flags);
int recvfrom(int s, char * msg,
   int len, int flags,
   struct sockaddr * fromaddr,
   int * fromlen);
```

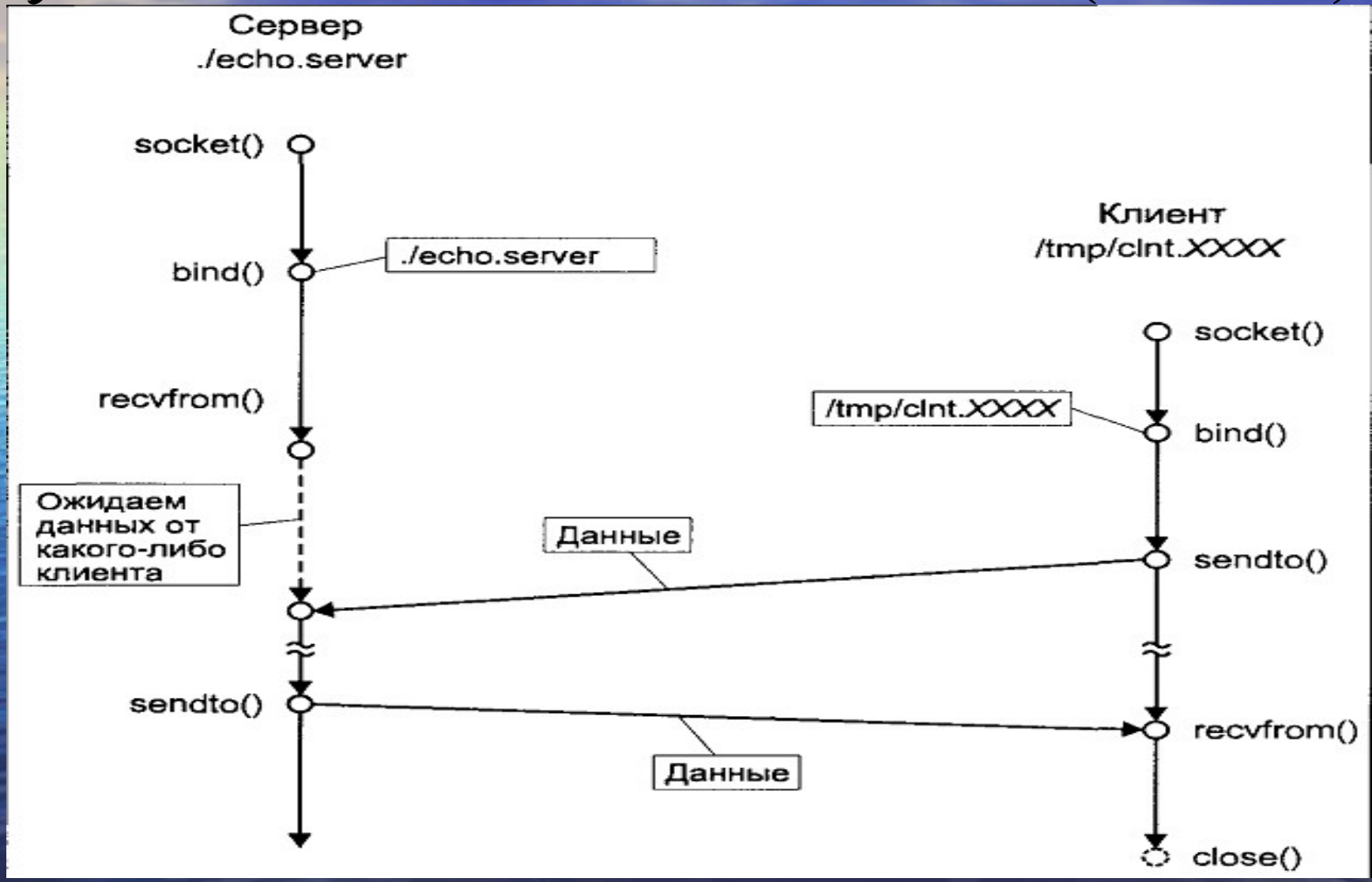
Отправка данных

```
int send(int s, char * msg, int len,
   int flags);
int sendto(int s, char * msg,
   int len, int flags,
   struct sockaddr * toaddr,
   int tolen);
```

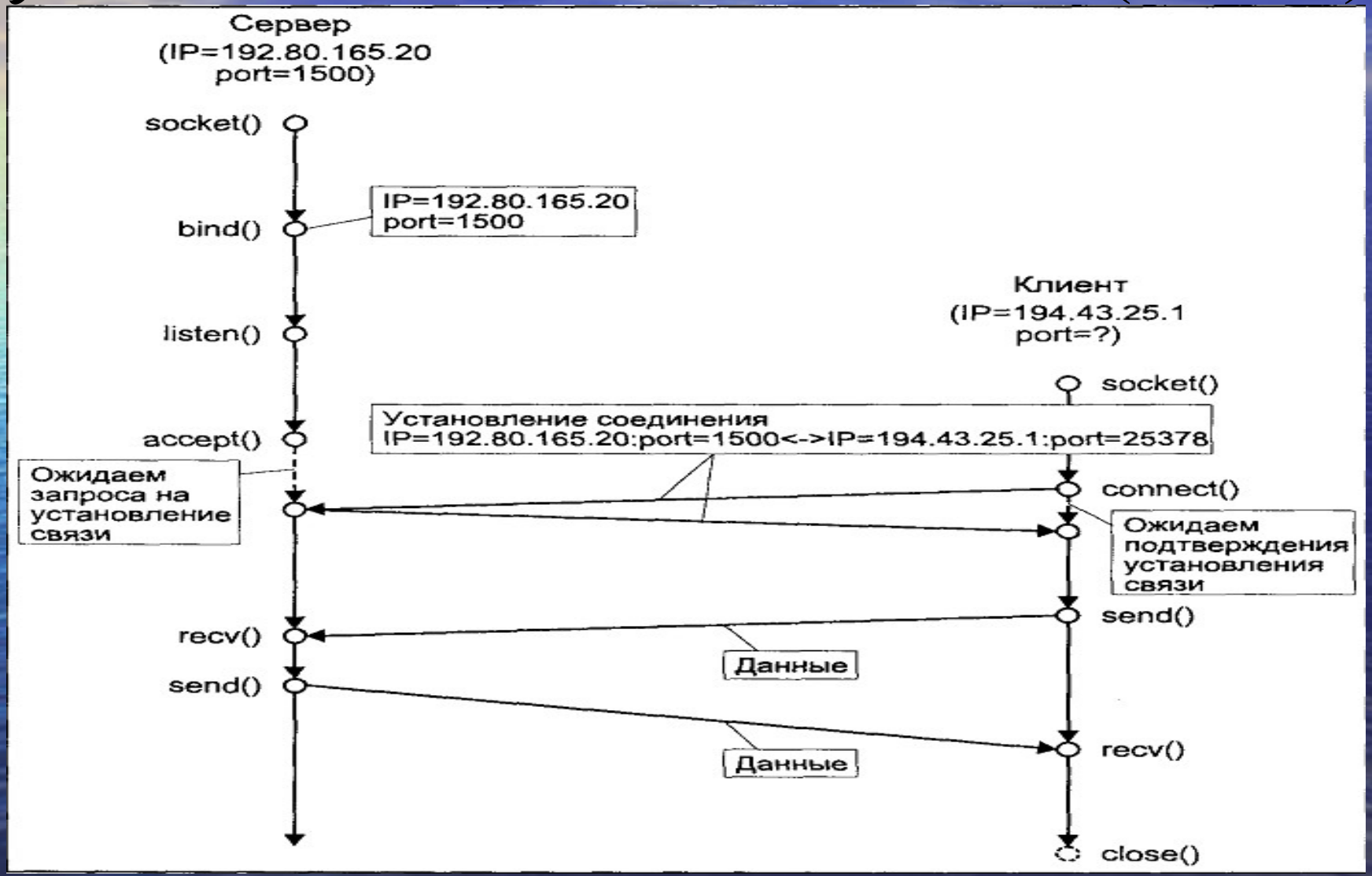
Взаимодействие процессов с установлением соединения (UNIX)



Взаимодействие процессов без установления соединения (UNIX)



Взаимодействие процессов с установлением соединения (INET)



Сравнение различных систем межпроцессного взаимодействия

	Каналы	FIFO	Сообще-ния	Разделя-емая па-мять	**Сокеты** (домен UNIX)
Простран-ство имен	—	Имя файла	Ключ	Ключ	Имя файла
Объект	Системный канал	Именован-ный канал	Очередь сообщений	Разделяемая область па-мяти	Коммуника-ционный узел
Создание объекта	`pipe()`	`mknod()`	`msgget()`	`shmget()`	`socket()`
Связывание	`pipe()`	`open()`	`msgget()`	`shmat()`	`bind()` `connect()`
Передача данных	`read()` `write()`	`read()` `write()`	`msgrcv()` `msgsnd()`	Непосред-ственный до-ступ `memcpy()`	`read()` `write()` `recv()` `send()` `recvfrom()` `sendto()`
Уничтоже-ние	`close()`	`close()` `unlink()`	`msgctl()`	`shmdt()`	`close()` `unlink()`

Планирование процессов

Типы мультипрограммирования

- Системы пакетной обработки
- Системы разделения времени
- Системы реального времени

Разделение времени

Каждому процессу выделяется некоторый квант времени, так чтобы у каждого пользователя создавалась иллюзия, что он один работает в системе.

Когда выделять квант?

Какой выделять квант?

Планирование по наивысшему приоритету (1)

HPF (Highest priority first):

- С вытеснением процессов
- Без вытеснения процессов

Проблема организации очереди.

Планирование по наивысшему приоритету (2)

- Стратегия SJF (Shorted job first)
- Динамический приоритет:
  - ✓ коэффициент a – ожидание
  - ✓ коэффициент b – выполнение

Планирование по наивысшему приоритету (3)

$0 < a < b$ – FIFO

$0 > b > a$ – LIFO

Круговорот (1)

RR (Round robin)

K – текущая длина очереди.

T – время цикла планирования (эпохи).

t = T / K – величина временного кванта.

Круговорот (2)

Если T – бесконечно, то метод вырождается в FIFO с приоритетами, соответствующими порядку поступления.

Круговорот (3)

Время T должно быть не очень большое, чтобы не страдали интерактивные пользователи, но и не очень маленькое, чтобы не возрастали накладные расходы на переключение контекстов.

Модификации круговорота

- Круговорот со смещением – квант времени каждого процесса пропорционален его статическому (задаваемому пользователем) приоритету.
- Использование для вычисления приоритета формулы исходя из стратегии SJF

Модификации круговорота

- Использование двух очередей: ведущей и фоновой.
- Использование нескольких очередей. (Многоуровневый метод очередей с обратной связью)

Очереди с обратной связью (1)

Новый процесс попадает в первую очередь. Отработав квант времени, перемещается во вторую и т.д. Таким образом, новый процесс работает до прихода следующего процесса, но не более, чем успел проработать предыдущий процесс.

Очереди с обратной связью (2)

Обобщение метода ведущей и фоновой очереди. Каждый процесс несколько раз проходит одну очередь, прежде чем перемещается в другую очередь. Получаем несколько очередей с принципом планирования RR.

Очереди с обратной связью (3)

Использование предыдущего метода побуждает пользователей разбивать сложные задания на несколько коротких простых.

Сравнение подходов

Очереди с обратной связью и круговорот хорошо обслуживают короткие задания. Планирование по наивысшему приоритету позволяет повысить загруженность устройств.

Можно использовать смешанные подходы.

Многоуровневое планирование (1)

- Диспетчер (передача управления первому процессу в очереди) очень короткий.
- Краткосрочный планировщик (Поместить готовый процесс в очередь) выполняется часто.

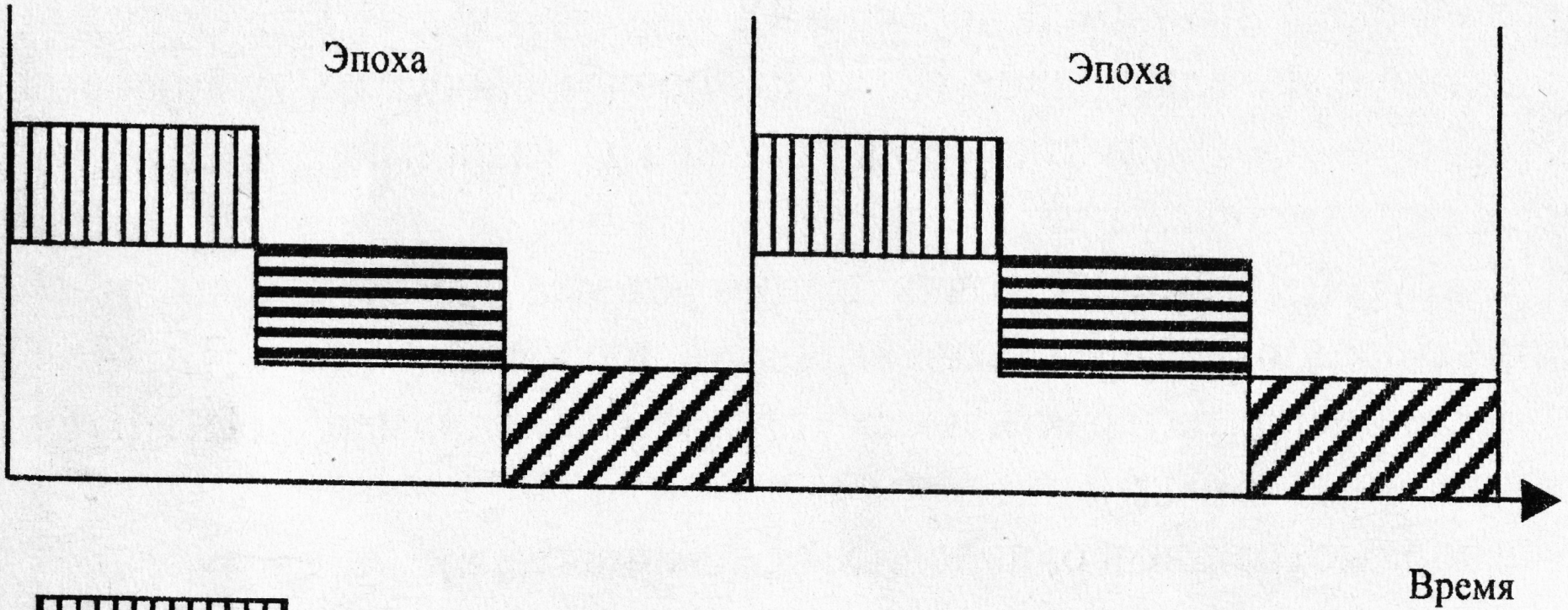
Многоуровневое планирование (2)

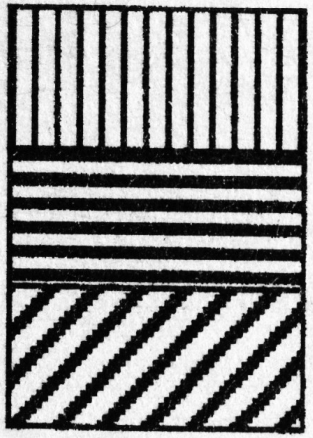
- Долгосрочный планировщик (Пересчет приоритетов) выполняется долго, вызывается редко.

Планирование в Linux

Время процессора делится на **эпохи**. В пределах каждой эпохи каждому процессу предоставляется квант времени. Квант времени напрямую зависит от приоритета. Когда все процессы израсходуют свой квант времени, эпоха заканчивается.

Идеальная система



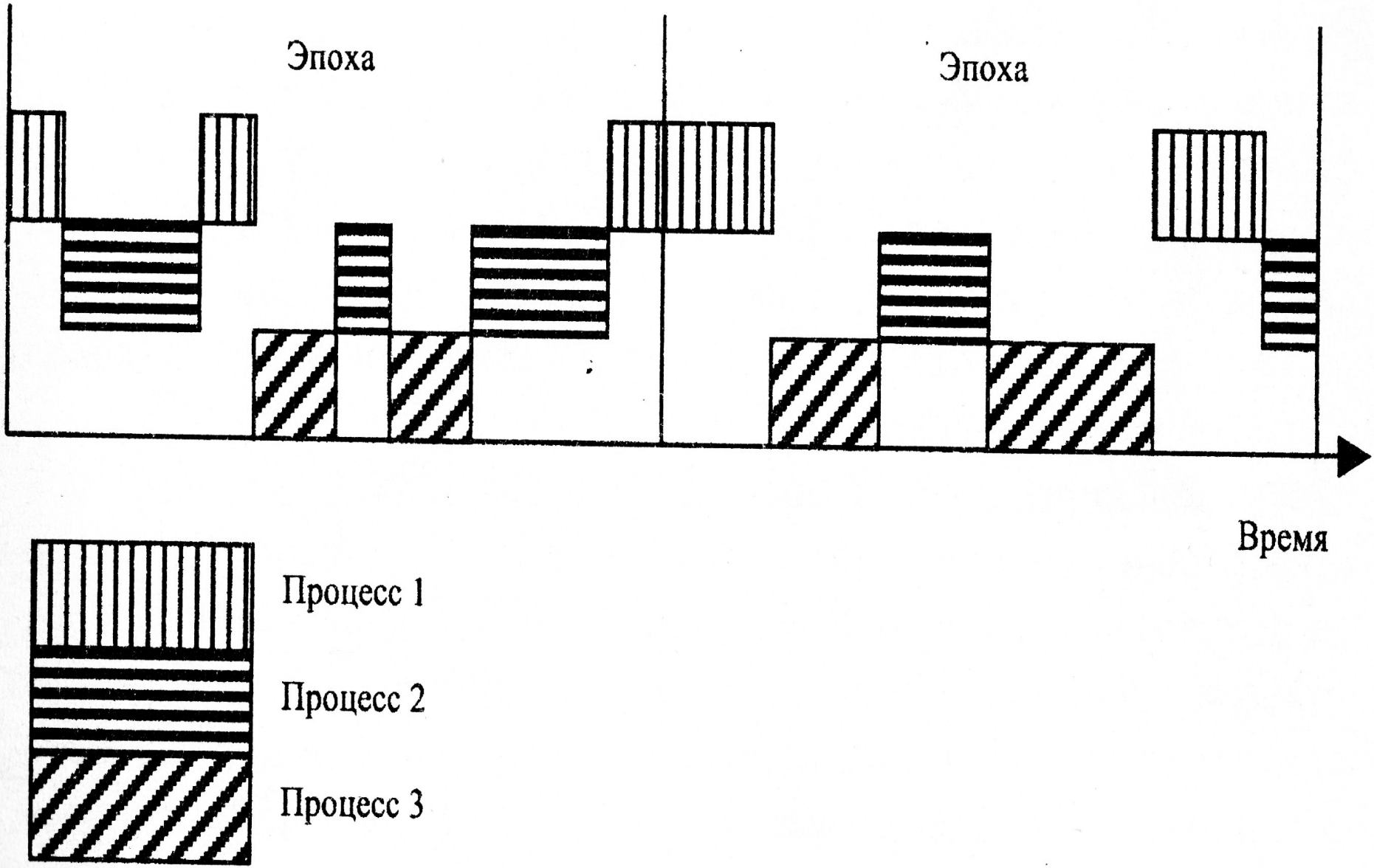


Процесс 1

Процесс 2

Процесс 3

Реальная система



Две задачи планировщика

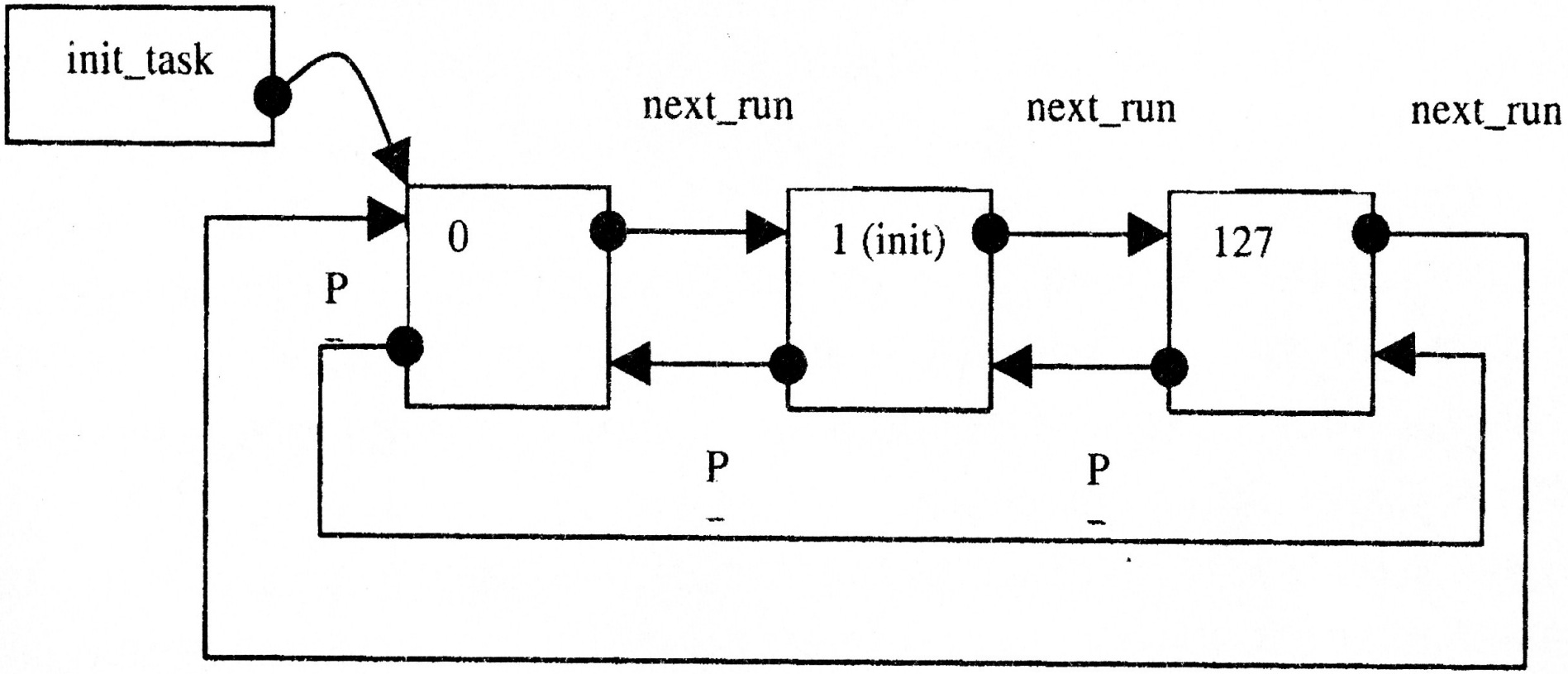
- Реализация режима разделения времени (Квантование времени процессора)
- Ранжирование процессов (Вычисление и назначение приоритетов)

Случаи освобождения процессора

Происходит вызов планировщика

- Переход в состояние ожидания
- По истечении кванта времени
- При появлении более приоритетного процесса

Очередь готовности



Политики планирования

- Традиционное планирование UNIX (приоритет динамический)
- Планирование по дисциплине FIFO (приоритет статический)
- Планирование по дисциплине RR (приоритет статический)

Дескриптор процесса (некоторые поля)

- Флаг перепланирования
- Политика планирования
- Базовый приоритет
- Счетчик тиков
- Приоритет реального времени

Работа планировщика

- Подготовительный этап (обработка текущего процесса и определение состава очереди готовности)
- Основной этап (выбор из очереди готовности процесса для передачи ему процессора)

Значения приоритетов

- 0 – 40 – обычные процессы
- 1001 – 1099 – процессы реального времени

Некоторые особенности

- После fork() оставшийся родительскому процессу квант времени делится между родительским и дочерним процессом
- «Подыгрывание» приостановленным процессам:

счетчик=(счетчик>>1)+приоритет