Лекция 32
Сигналы, часть 2

Системно-зависимые особенности signal(2)

- На время обработки (выполнения обработчика сигнала) повторный вызов текущего обработчика может блокироваться или не блокироваться
- Обработчик может сбрасываться после запуска или не сбрасываться
- Некоторые системные вызовы (read, write, accept...) могут перезапускаться или завершаться с errno == EINTR
- Вызов sigaction(2) позволяет управлять этим

Обработчики сигналов

- В обработчиках сигналов можно использовать только асинхронно-безопасные (async signal safe) стандартные функции
- Большинство системных вызовов (включая fork и exec) — асинхронно-безопасные
- Функции работы с динамической памятью (new, delete, malloc, free), функции работы с потоками (fopen, fprintf, <<) - **не асинхронно-безопасные**

Безопасная обработка сигналов

- Безопаснее всего в обработчике сигнала устанавливать глобальный флаг поступления сигнала, который обрабатывать в основной программе
- Для этого требуются доп. средства управления сигналами

```
volatile sig_atomic_t sigint_flag;
void hnd(int s)
{
  sigint_flag = 1;
}
```

Volatile, sig_atomic_t

- Ключевое слово volatile обозначает, что значение переменной может измениться «неожиданно» для компилятора программы
- Компилятор не должен пытаться оптимизировать обращения к переменной (например, загружая ее на регистр)
- Тип sig_atomic_t — это целый тип (обычно int), для которого гарантируется атомарная запись и чтение

Множества сигналов

// очистка множества

```
void sigemptyset(sigset_t *pset);
```

// заполнение множества

```
void sigfillset(sigset_t *pset);
```

// добавление сигнала в множества

```
void sigaddset(sigset_t *pset, int signo);
```

// удаление сигнала из множества

```
void sigdelset(sigset_t *pset, int signo);
```

Блокирование сигналов

- Если сигнал заблокирован, его доставка процессу откладывается до момента разблокирования

```
int sigprocmask(int how, const sigset_t *set,
                sigset_t *oldset);
```

- SIG_BLOCK — добавить сигналы к множеству блокируемых
- SIG_UNBLOCK — убрать сигналы из множества блокируемых
- SIG_SETMASK — установить множество

Ожидание поступления сигнала

```
int sigsuspend(const sigset_t *mask);
```

- На время ожидания сигнала выставляется множество блокируемых сигналов mask
- После доставки сигнала восстанавливается текущее множество блокируемых сигналов

Отображение сигналов на файловые дескрипторы

- Системный вызов signalfd(2) позволяет создать файловый дескриптор, работая с которым можно получать уведомления о поступлении сигналов
  - signalfd — создает файловый дескриптор
  - select/poll/epoll — ожидание события (прихода сигнала)
  - read — ожидание прихода сигнала и получения информации о нем
  - close — закрытие

Стратегия корректной работы с сигналами

- Функции-обработчики сигналов устанавливают флаг поступления сигнала
- Программа выполняется с заблокированными сигналами
- Сигналы разблокируются только на время ожидания прихода сигнала (с помощью sigsuspend или pselect) или используется signalfd а сигналы не нужно разблокировать

Сокеты

Сети TCP/IP

- Хост — компьютер, подключенный в сеть
- Интерфейс — физическое или логическое устройство, позволяющее принимать и отправлять данные в сеть (loopback, ethernet, …)
- Loopback-интерфейс есть на каждом хосте, IP-адрес 127.0.0.1
- Хост может иметь несколько физических интерфейсов

IP-адрес

- У протокола IP v4 размер адреса — 32 бита, адрес делится на 4 байта, каждый байт записывается в десятичном виде, байты разделяются точкой: 212.192.248.182
- IP v6 — 128 бит, адрес делится на группы по 16 бит, каждая записывается в шест. виде, разделяются двоеточием: fe80::21d:7dff:fe00:e4d6
- Символическая нотация — DNS имена (www.cs.hse.ru)

Порт

- Порт — число от 1 до 65535 (16 бит)
- Порты с номерами до 1024 зарезервированы для использования процессами с правами администратора
- Некоторые номера портов приписаны (то есть обычно используются) стандартными сервисами
- Каждому исходящему запросу или датаграмме присваивается номер порта

Идентификация соединений и запросов

- Каждое соединение или датаграмма идентифицируется 4 числами: IP отправителя, порт отправителя, IP получателя, порт получателя

TCP (Transmission Control Protocol)

- TCP — протокол транспортного уровня
- Протокол с установлением виртуального соединения
- Протокол гарантирует доставку данных (при необходимости повторяя пересылку пакета) и гарантирует корректность данных
- После установления соединения TCP предоставляет двунаправленный канал (то есть границы блоков данных, пересылаемых по TCP, не сохраняются)

Протоколы прикладного уровня

- HTTP (hypertext transfer protocol) — TCP
- FTP (file transfer protocol) — TCP
- SSH (secure shell) — TCP
- SMTP (simple mail transfer protocol) — TCP
- DNS — UDP
- NFS (network file system) — UDP или TCP
- NTP (network time protocol) — UDP
- И так далее...

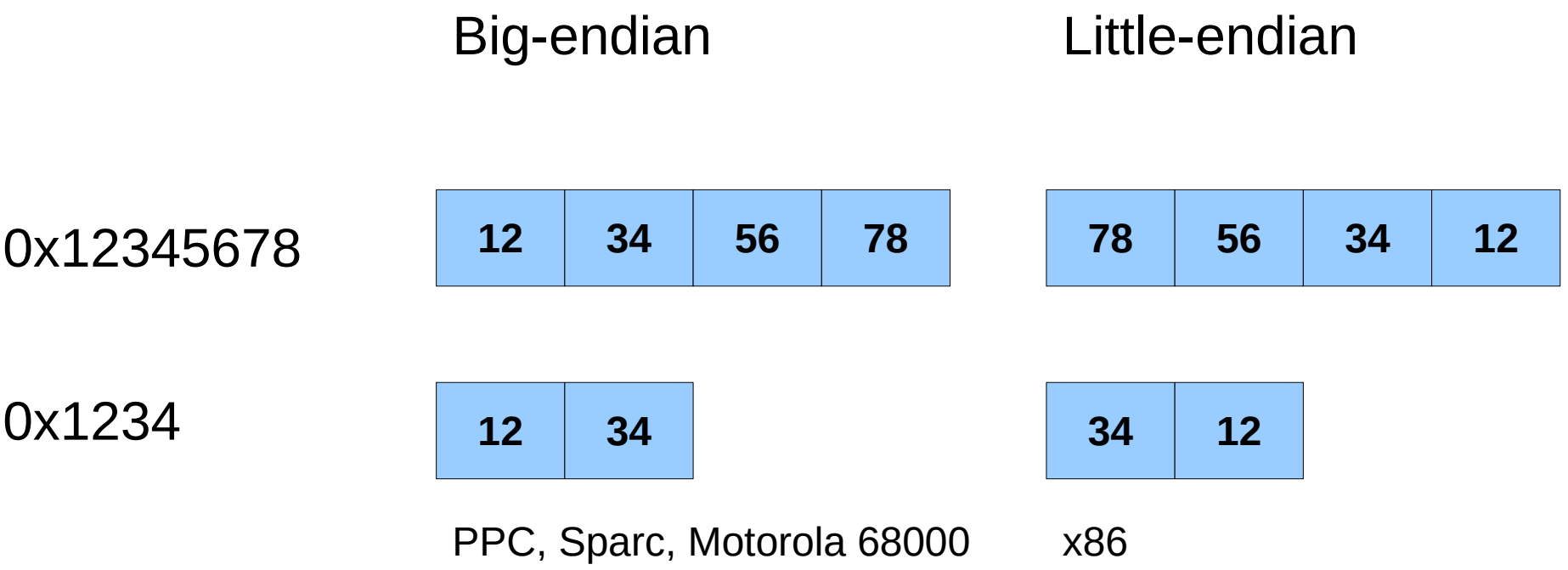
Стандартные номера портов

- TCP/80, TCP/8080 — httpd — веб-сервер
- TCP/22 — sshd — сервер удаленного доступа
- TCP/21, TCP/20 — ftpd — ftp-сервер
- TCP/25 — sendmail — почтовый сервер
- UDP/53 — named — DNS-сервер
- UDP/123 — ntpd — сервер времени
- И так далее

Кодирование данных

- На разных хостах сети могут использоваться разные способы хранения целых и вещественных чисел, разные требования к выравниванию полей структур, разные кодировки символов...
- Преобразование данных в формат передачи данных по сети — сериализация (serialization) или маршаллинг (marshalling)
- Обратное преобразование — десериализация или демаршаллинг

Представление целых чисел



В качестве сетевого формата представления целых чисел принят Big-endian формат.

Перекодирование целых

- Network byte order — порядок байт в сети
- Host byte order — локальный порядок байт

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Сокеты (гнезда)

- Универсальный механизм межпроцессного взаимодействия
- Используется для локального взаимодействия (аналогично именованным каналам)
- Используется для сетевого взаимодействия

Создание сокета

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- Возвращается файловый дескриптор сокета
- В зависимости от типа создаваемого сокета может потребоваться дополнительная настройка параметров
- Файловый дескриптор может копироваться с помощью dup, наследоваться через fork/exec
- В конце использования файловый дескриптор должен быть закрыт с помощью close

Параметры создания сокета

- Параметр domain — домен сокета
  - PF_UNIX, PF_LOCAL — локальный сокет
  - PF_INET — IPv4
  - PF_INET6 — IPv6
- Параметр type — тип соединения
  - SOCK_STREAM — потоковый сокет
  - SOCK_DGRAM — датаграммный сокет
- Параметр protocol уточняет используемый протокол для пары (domain,type). 0 — выбрать протокол по умолчанию

Примеры создания сокета

```
fd = socket(PF_LOCAL, SOCK_STREAM, 0);
```

- Локальный потоковый сокет (аналог именованных каналов)

```
fd = socket(PF_INET, SOCK_STREAM, 0);
```

- Сокет для работы по протоколу TCP

```
fd = socket(PF_INET, SOCK_DGRAM, 0);
```

- Сокет для работы по протоколу UDP

Адрес соединения

- Для указания адреса как отправителя, так и получателя используется struct sockaddr
- При работе с каждым конкретным доменом сокета должна использоваться структура адреса, специфичная для этого домена

```
struct sockaddr_in {
  sa_family_t    sin_family; /* AF_INET */
  uint16_t       sin_port;   /* port in NBO*/
  struct in_addr sin_addr;   /* internet address */
};
struct in_addr {
  uint32_t       s_addr;     /* address in NBO*/
};
```

Получение информации об адресе

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

- node — строка имени или IP-адреса хоста
- service — строка номера порта или имени сервиса
- hints — флаги для управления трансляцией
- res — адрес указателя, в который помещается указатель на голову списка результатов трансляции

Исходящий порт

- Для полной идентификации датаграммы или соединения необходим номер исходящего порта
- Если номер исходящего порта не задан, он назначается автоматически
- Для взаимодействия по протоколу TCP исходящий порт, как правило, не требуется
- Для взаимодействия по протоколу UDP может потребоваться назначить исходящий порт
- Привязку необходимо выполнять, если планируется принимать сообщения

Привязка порта

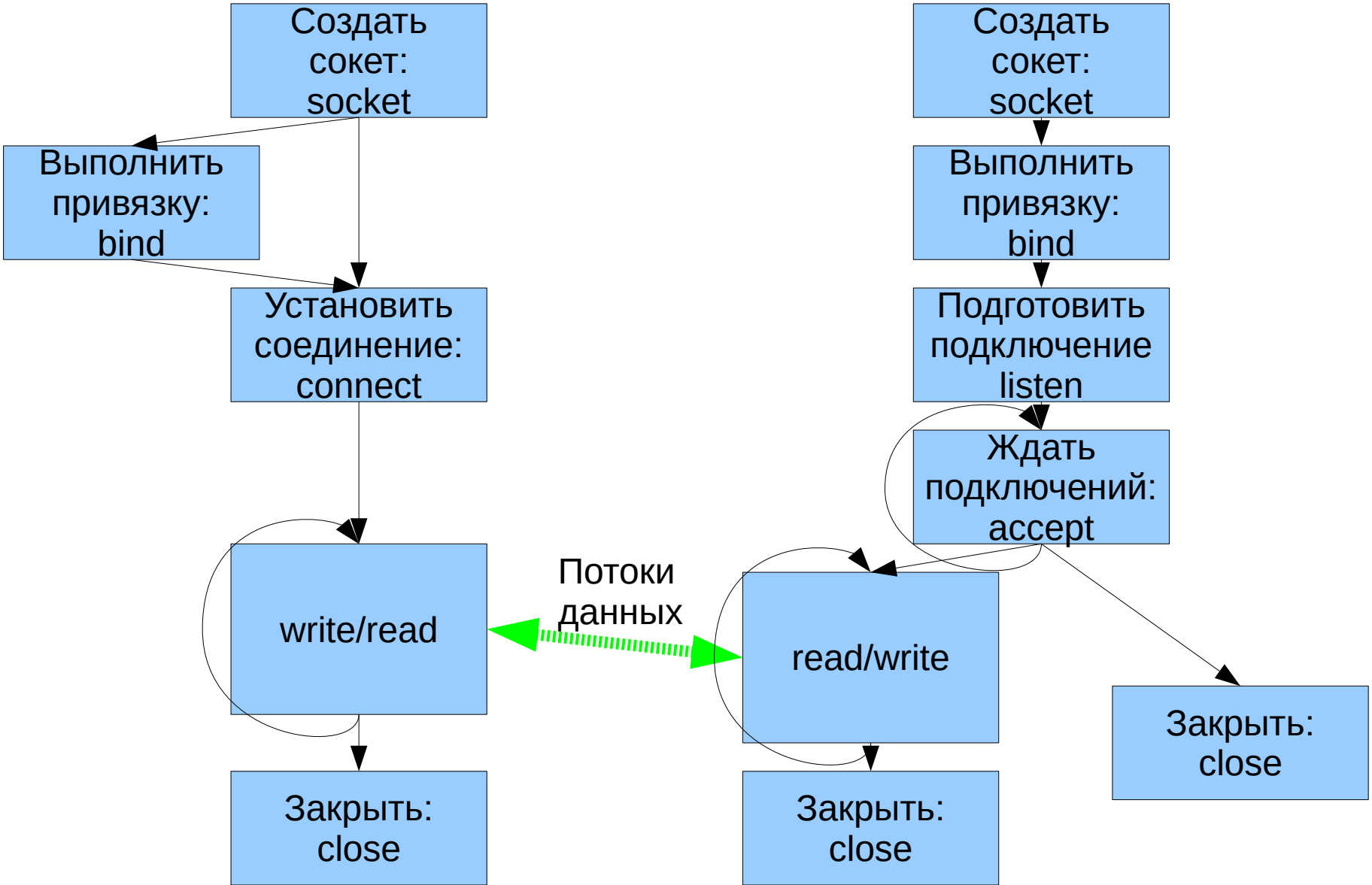
```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

- addr — структура, задающая параметры привязки
- addrlen — размер структуры адреса

Архитектура клиент-сервер

- Техническое понимание:
  - Клиент — сторона, которая инициирует соединение
  - Сервер — сторона, которая ожидает подключения
- Логическое понимание
  - Клиент — сторона, запрашивающая выполнение некоторого сервиса
  - Сервер — сторона, предоставляющая сервис
- Как правило, техническое = логическое

Взаимодействие с установлением соединения



Подключение к серверу

```
int connect(int sockfd, const struct sockaddr *sa,
            socklen_t addrlen);
```

- Пример: подключение к хосту/порту, указанным в командной строке

```
int res = connect(sockfd, reslist->ai_addr,
                  reslist->ai_addrlen);
if (res < 0) {
  // обработать ошибочную ситуацию
}
```

Переключение сокета в режим прослушивания

```
int listen(int sockfd, int backlog);
```

- Параметр backlog — длина очереди запросов, ожидающих обработки, в ядре
- Если программа не успевает обрабатывать входящие запросы на подключение, ядро будет отвергать запросы, которые бы приводили к превышению длины очереди значения backlog
- Обычное «магическое» значение — 5

Ожидание подключения

```
int accept(int sockfd, struct sockaddr *addr,
           socklen_t *addrlen);
```

- В структуру addr возвращается адрес подключившегося клиента
- При успехе accept возвращает *новый* файловый дескриптор, который используется для обмена данными с клиентом
- Файловый дескриптор sockfd можно использовать для подключения других клиентов

Обработка входящих подключений

- При каждом успешном выполнении accept создается новый файловый дескриптор для обмена данными с клиентами
- Сервер должен выполнять операции ввода/вывода с несколькими файловыми дескрипторами и обрабатывать новые подключения
- Каждая операция может заблокировать процесс на неопределенное время

Именованные каналы

- Канал, доступ к которому выполняется через точку привязки файловой системы
- Ядро создает по одному объекту именованного канала для каждой записи в файловой системе

```
int mkfifo(const char *path, mode_t mode);
```

- Создание специального объекта в файловой системе
- Удаление — с помощью unlink

Типичные ошибки во взаимодействующих процессах

- Deadlock (тупик) — несколько процессов не может продолжить выполнение, так как процессы ждут друг друга
- Race Condition (гонки) — результат работы зависит от порядка переключения выполнения между параллельными процессами
- **Очень сложно обнаруживаемые ошибки**
- **Могут проявляться очень редко при редкой комбинации условий**

Открытие именованного канала

- Открывается с помощью системного вызова open:

```
fdr = open(path, O_RDONLY, 0); // на чтение
fdw = open(path, O_WRONLY, 0); // на запись
```

- Операции открытия блокируются до выполнения противоположной операции
  - Открытие на чтение заблокирует процесс, пока другой процесс не откроет на запись
  - Открытие на запись заблокирует процесс, пока другой процесс не откроет на чтение
- После открытия работа — как с обычным каналом

Открытие канала

- Допускается открытие именованного канала в режиме O_RDWR, тогда канал откроется независимо от наличия читателей.

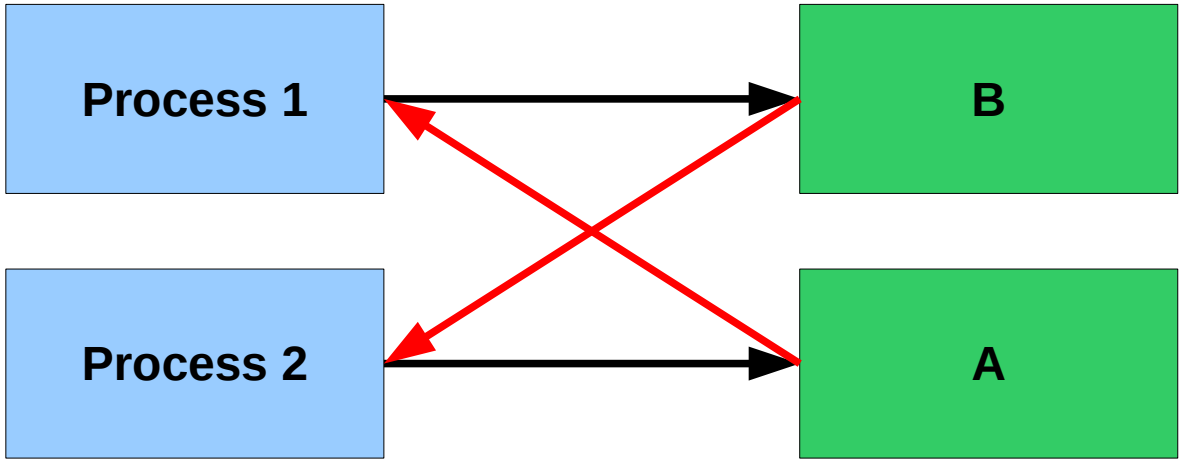
```
fdw = open(path, O_RDWR, 0);
fdr = open(path, O_RDONLY, 0);
```

- Таким образом можно полностью открыть канал в одном процессе

Обнаружение тупиков

Process 1:
wait(&A);
notify(&B);

Process 2:
wait(&B);
notify(&A);



- Захваченный ресурс — дуга от процесса к ресурсу
- Ожидаемый ресурс — дуга от ресурса к процессу
- Если в графе есть цикл, система попала в состояние тупика