Трассировка процессов

- Трассировка — способ полного контроля над выполнением процесса со стороны другого процесса
- Выполняется с помощью системного вызова ptrace
- Форматы данных и команд для ptrace зависят от архитектуры и ОС, ptrace не стандартизован POSIX
- ptrace используется утилитой strace для печати системных вызовов, выполняемых процессом
- Используется отладчиками (например, gdb)

Выполнение трассируемого процесса

- Трассируемый процесс приостанавливается (как при получении SIGSTOP) при следующих событиях:
  - Поступление сигнала, даже игнорируемого (кроме SIGKILL)
  - Вход в или выход из системного вызова
  - После выполнения одной инструкции процессора
  - При выполнении специальной инструкции трассировочного прерывания (например, на i386 — int $3 (код операции: 0xcc))

Системный вызов ptrace

```
long ptrace(int request, pid_t pid, void *addr,
            void *data);
```

- request — команда трассировки
- Некоторые команды возвращают произвольное значение, в том числе -1: необходима проверка значения переменной errno
- Все команды, кроме PTRACE_TRACEME, PTRACE_ATTACH, PTRACE_KILL, должны выполняться при остановленном трассируемом процессе

Включение трассировки

```
ptrace(PTRACE_TRACEME, 0, 0, 0);
```

- Сыновний процесс может разрешить трассировать себя родителю

```
ptrace(PTRACE_ATTACH, pid, 0, 0);
```

- Подключение и трассировка процесса с указанным pid
- Трассирующий процесс отслеживает остановку трассируемого процесса с помощью wait или waitpid, затем использует ptrace для модификации поведения сына

Команды трассировки

- PTRACE_PEEKTEXT, PTRACE_POKETEXT — доступ к адресному пространству
- PTRACE_PEEKUSER, PTRACE_POKEUSER — доступ к информации о процессе (содержимое регистров)
- PTRACE_CONT — продолжить выполнение
- PTRACE_SYSCALL — до следующего системного вызова
- PTRACE_SINGLESTEP — одна инструкция
- PTRACE_KILL — уничтожить процесс
- PTRACE_DETACH — отключиться от процесса

Средства System V IPC

- Включают в себя массивы семафоров, разделяемую память и очереди сообщений
- «Именем» объекта SysV IPC является значение типа key_t (реально — целое число)
- Каждый тип объектов находится в своем пространстве имен (массив семафоров, разделяемая память и очередь сообщений могут иметь одинаковые «имена»

Массивы семафоров

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

- Создание массива семафоров
- key - «имя» массива, nsems — число семафоров
- semflg — права доступа и флаги: IPC_CREAT, IPC_EXCL
- Начальное значение семафоров неопределено, но на Linux и многих других системах — 0
- Возвращается идентификатор созданного массива семафоров semid

Массивы семафоров: удаление

```
int semctl(int semid, int semnum, int cmd, ...);
```

- Управление массивом семафоров

```
semctl(semid, 0, IPC_RMID);
```

- Уничтожение массива семафоров: другие процессы, работающие с данным массивом семафоров, получат ошибку EIDRM

Массив семафоров: операции

```
int semop(int semid, struct sembuf *sops,
          unsigned nsops);
```

- Выполнение операций, заданных в массиве sops.
- Все операции выполняются как одно целое атомарно
- Поддерживаемые операции: увеличение значение, уменьшение значения, проверка на 0

struct sembuf:

```
  unsigned short sem_num;  /* номер семафора */
  short          sem_op;   /* операция*/
  short          sem_flg;  /* флаги */
```

Массив семафоров: операции

- sem_op > 0 — увеличение значение семафора на указанное число (не превышая максимально возможное значение)
- sem_op < 0 — блокирование процесса до тех пор, пока значение семафора не станет не меньше |sem_op|, затем уменьшение значения семафора на |sem_op|
- sem_op == 0 — ожидание значения 0 семафора

Разделяемая память

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

- Создание области разделяемой памяти
- key - «имя» разделяемой памяти, size — размер области
- shmflg — права доступа и флаги: IPC_CREAT, IPC_EXCL
- Возвращается идентификатор созданной области разделяемой памяти

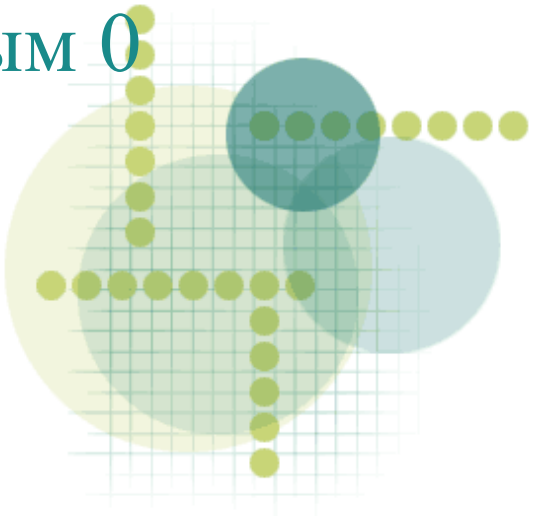
Разделяемая память: уничтожение

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- Управление разделяемой памятью

```
shmctl(shmid, IPC_RMID, 0);
```

- Уничтожение разделяемой памяти
- Разделяемая память будет уничтожена, когда счетчик использований станет равным 0



Разделяемая память: использование

```
void *shmat(int shmid, const void *shmaddr, int flg);
```

- Подключение разделяемой памяти к адресному пространству процесса
- Допустимые флаги: SHM_RDONLY, SHM_REMAP, SHM_RND

```
int shmdt(const void *shmaddr);
```

- Отключение разделяемой памяти от адресного пространства
- Разделяемая память отключается автоматически при завершении процесса

Очередь сообщений: создание

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

- Создание очереди сообщений
- key - «имя» очереди сообщений
- msgflg — права доступа и флаги: IPC_CREAT, IPC_EXCL
- Возвращается идентификатор созданной очереди сообщений

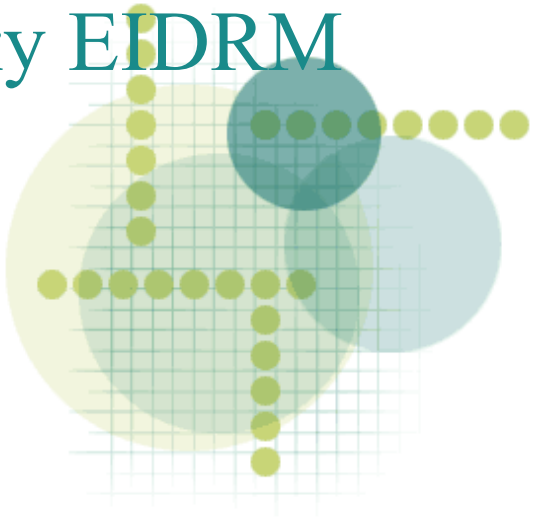
Очередь сообщений: уничтожение

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- Управление очередью сообщений

```
msgctl(msqid, IPC_RMID, 0);
```

- Удаление очереди сообщений
- Другие процессы, работающие с данной очередью сообщений, получат ошибку EIDRM



Отправка сообщений

```
int msgsnd(int msqid, const void *msgp,
           size_t msgsz, int msgflg);
```

- Передаваемое сообщение должно начинаться с поля (его размер не учитывается в msgsz): long mtype;
- Тип сообщения — положительное целое число
- Очередь сообщений имеет ограниченный размер
- IPC_NOWAIT в msgflg — неблокирующий режим

Прием сообщений

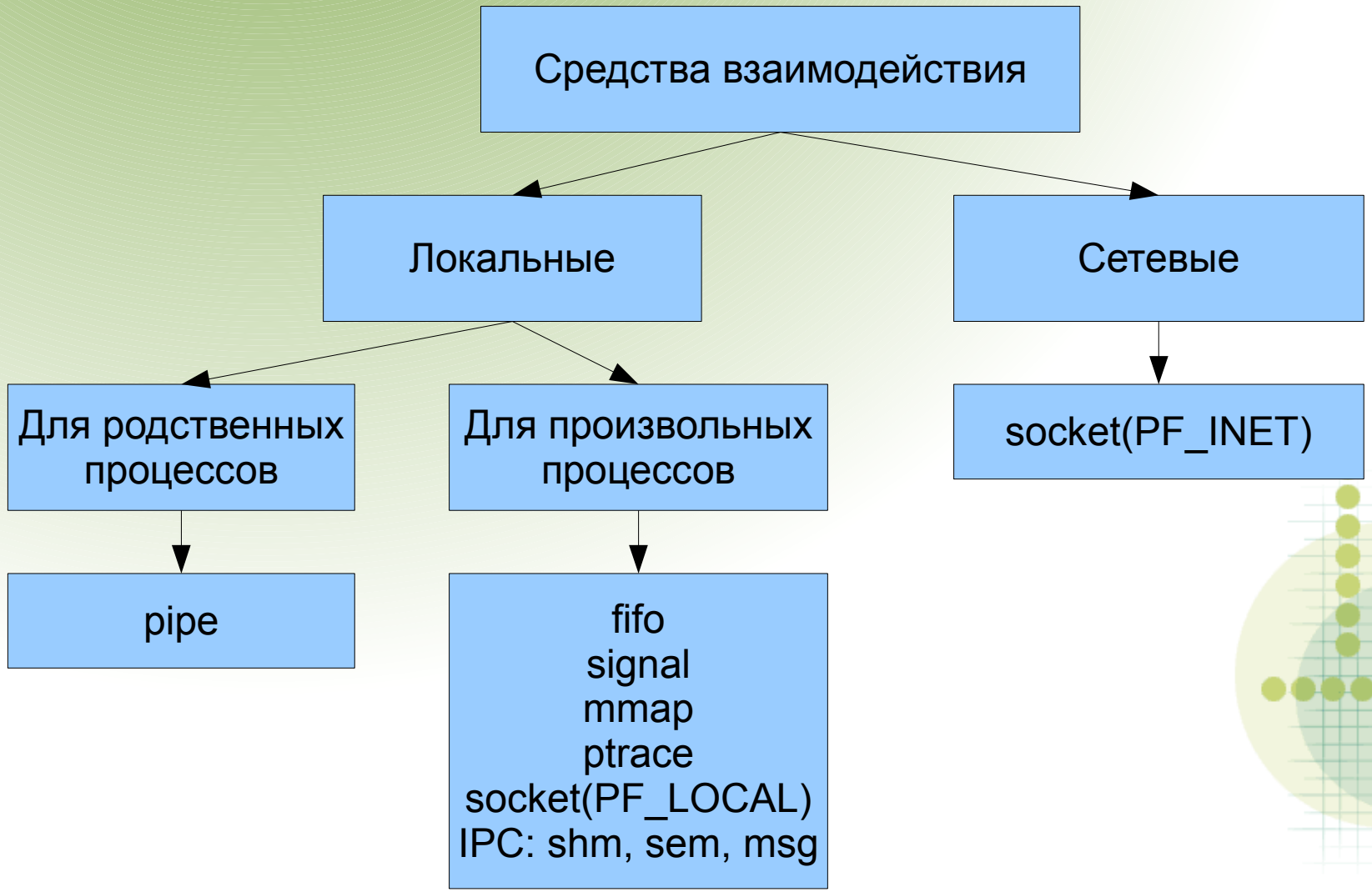
```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz,
               long msgtyp, int msgflg);
```

- msgp — буфер для приема сообщений, msgsz — размер буфера
- msgrcv возвращает размер сообщения (без учета поля type)
- msgflg: IPC_NOWAIT, MSG_NOERROR (обрезать сообщение, если не помещается в буфер), MSG_EXCEPT — модифицирует фильтр сообщений

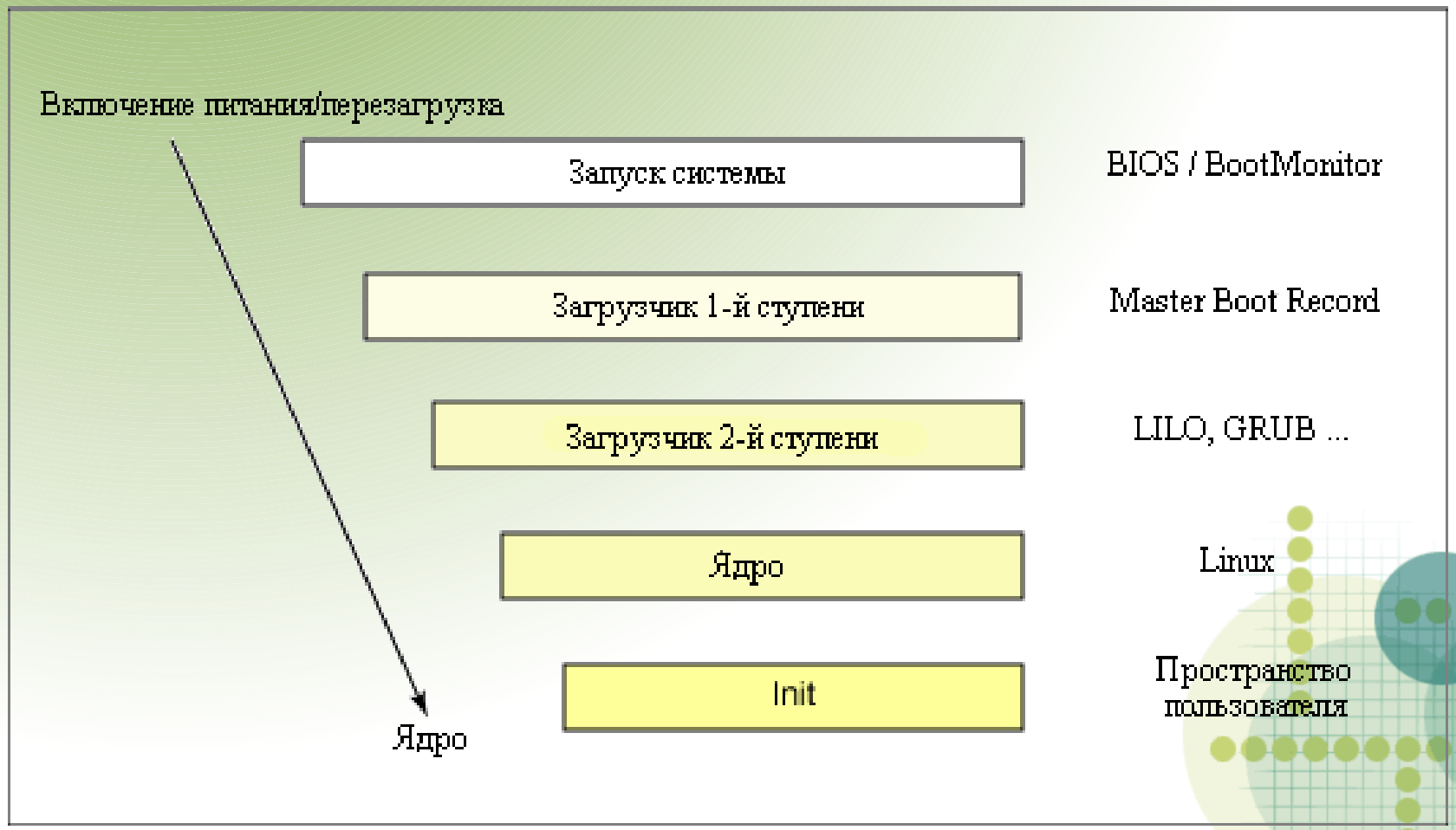
Фильтрация типа сообщений:

- msgtyp == 0 — выбрать первое сообщение
- msgtyp > 0 — выбрать первое сообщение указанного типа
- msgtyp > 0, MSG_EXCEPT — выбрать первое сообщение типа, отличающегося от указанного
- msgtyp < 0 — выбрать первое сообщение с типом <= |msgtyp|

Классификация средств взаимодействия



Загрузка Linux



Инициализационные скрипты

- /sbin/init запускает скрипты для инициализации сервисов системы
- Инициализация зависит от ОС:
  - BSD — один большой скрипт
  - Linux, Solaris, System V — отдельные скрипты для каждого сервиса

Уровни работы Linux

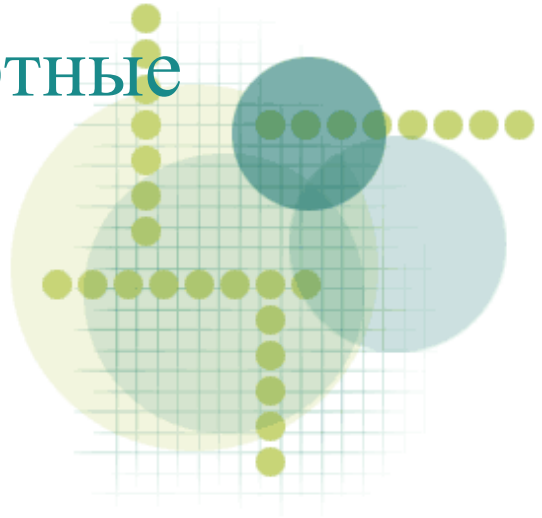
- Система может работать на нескольких «уровнях» (runlevel), набор запущенных программ может различаться
- Уровни выполнения:
  - 0 — завершение работы системы
  - 6 — перезагрузка системы
  - 1, S — однопользовательский режим
  - 3 — многопольовательский текстовый режим
  - 5 — многопользовательский графический режим

Сервисные скрипты

- Для каждого установленного сервиса, который может запускаться при загрузке, в каталоге /etc/init.d находится вспомогательный скрипт

```
/etc/init.d/httpd stop # остановить веб-сервер
/etc/init.d/httpd start # запустить веб-сервер
```

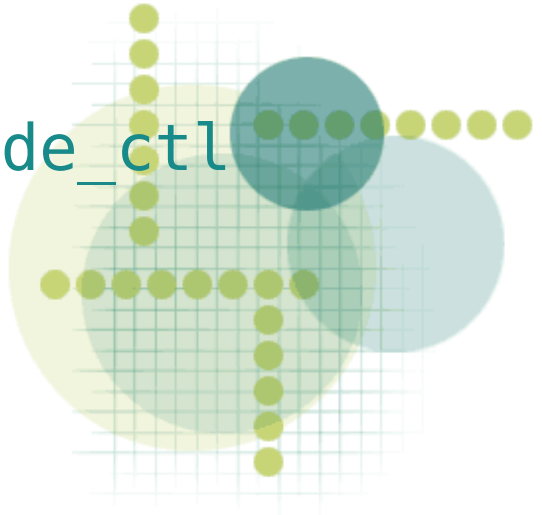
- Все скрипты поддерживают стандартные опции: start, stop, restart, status



Инициализация уровней работы

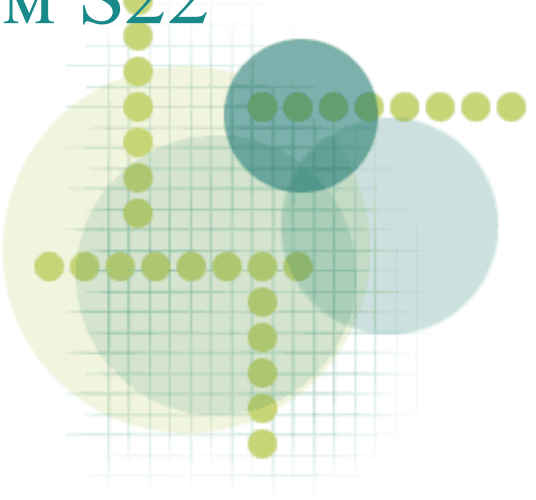
- Каждому уровню работы соответствует каталог /etc/rcN.d, который содержит символические ссылки на сервисные скрипты

```
# ls -l /etc/rc5.d
K89netplugd -> ../init.d/netplugd
K89rdisc -> ../init.d/rdisc
K90network -> ../init.d/network
K95firstboot -> ../init.d/firstboot
S00microcode_ctl -> ../init.d/microcode_ctl
S06cpuspeed -> ../init.d/cpuspeed
S07iscsid -> ../init.d/iscsid
```



Инициализация уровня работы

- SNN — скрипт должен быть выполнен при входе системы в этот уровень работы
- KNN — скрипт должен быть выполнен при выходе системы из этого уровня работы
- Число задает порядок выполнения, например, S12 должен выполниться раньше, чем S22



Управление консолями

- /sbin/init управляет консолями, подключенными к компьютеру
  - Виртуальные консоли (Ctrl-Alt-1 … Ctrl-Alt-9)
  - Консоли на последовательных портах
- Для каждой зарегистрированной в системе консоли запускается программа getty
  - Для виртуальных консолей — mingetty
  - Для остальных - agetty

Работа getty

- getty настраивает устройство и ожидает ввода логина пользователя, после получения логина пользователя запускается /bin/login
- /bin/login выводит приглашение ко вводу пароля, проверяет пароль, проверяет остальные условия допустимости входа пользователя в систему
- При успешной аутентификации для пользователя запускается его командный процессор (обычно /bin/bash)
- После завершения командного процессора /sbin/init снова создает getty для устройства