

Среда исполнения

Системные вызовы и библиотеки Unix SVR4

Иртегов Д.В.

ФФ/ФИТ НГУ

Электронный лекционный курс подготовлен в рамках реализации

Программы развития НИУ-НГУ на 2009-2018 г.г.

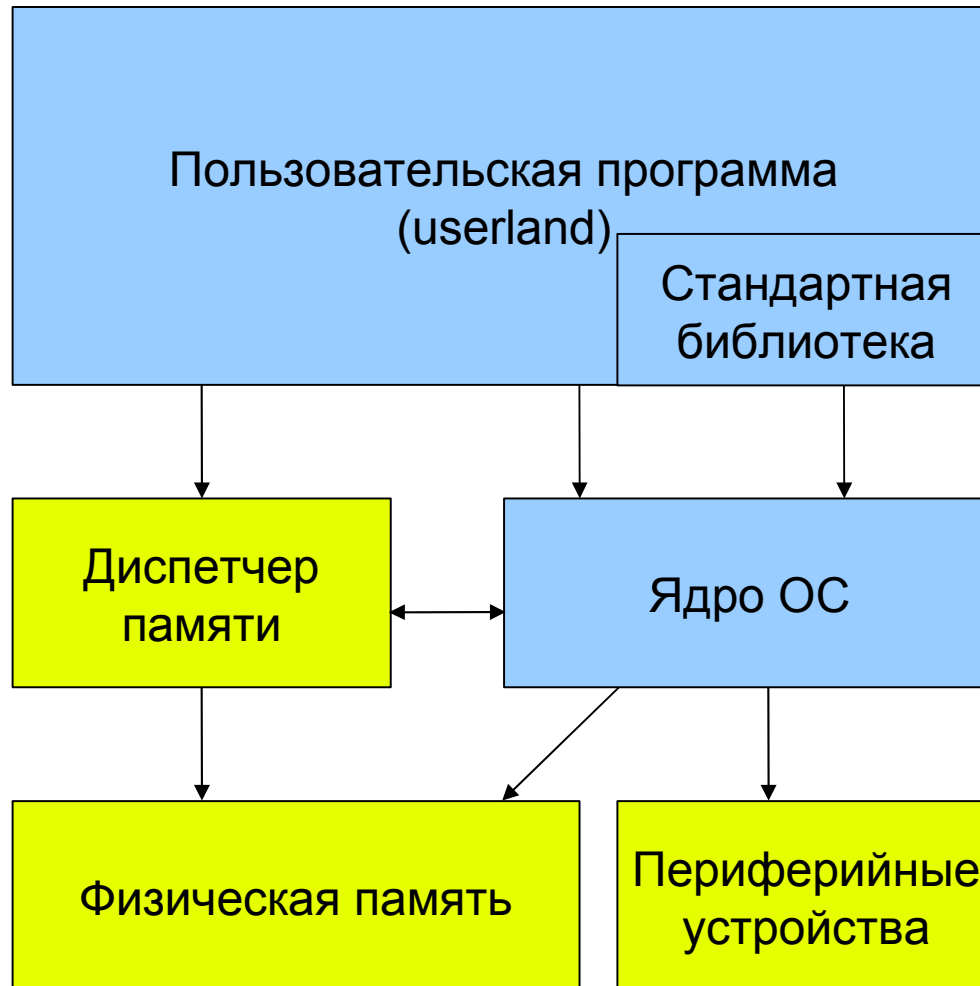
ЦЕЛИ РАЗДЕЛА

- Определить термин процесс
- Описать среду исполнения процесса
- Получать и изменять информацию о среде исполнения процесса с помощью системных вызовов и стандартных библиотечных функций.

Процесс

- Процесс (задача) — это «песочница» для исполнения программ с ограниченными привилегиями
- Unix-системы используют защиту памяти или, что почти то же самое, виртуальную память
- Каждый процесс имеет свое виртуальное адресное пространство

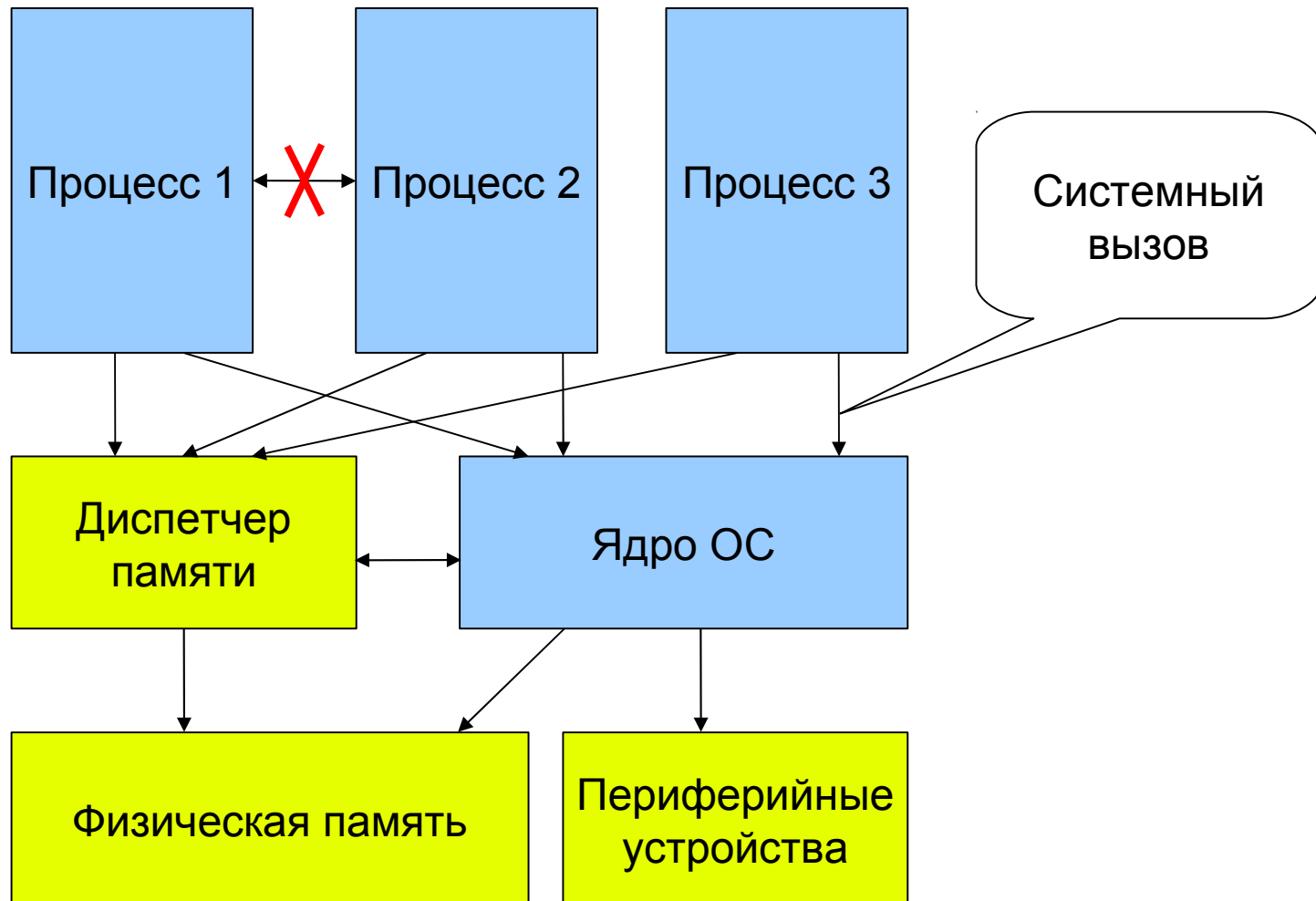
Защита памяти



Что такое ядро ОС

- Библиотека:
 - Код, находящийся в адресном пространстве процесса,
 - Исполняется с пользовательскими привилегиями
 - Вызывается обычной командой call
- Ядро:
 - Код, расположенный в привилегированной области памяти
 - Исполняется с повышенными привилегиями
 - Вызывается специальной командой, например `syscall/sysenter` (это и есть системный вызов)
 - В Unix, все системные вызовы имеют «обертки», которые выглядят как обычные вызовы функций.

Защита памяти



Процесс

- Экземпляр исполнения программы в операционной системе с атрибутами, содержащимися в структурах данных операционной системы
- Новый процесс для каждого запуска программы
- Каждый процесс имеет свой уникальный идентификатор процесса pid (Process ID)

Основные структуры процесса

- TEXT (код программы)
- DATA (инициализированные статические данные)
- BSS (неинициализированные статические данные)
- STACK
- Куча
- Динамические сегменты
- User Area (дескриптор процесса в ядре)
 - Стек процесса в ядре
 - Дескрипторы открытых файлов
 - Атрибуты процесса

Адресное пространство процесса x86

<http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/i86pc/os/startup.c>

```
* 32-bit Kernel's Virtual memory layout.
*
*      +-----+
*      |      Kernel Context      |
* 0xC3002000 -|-----|- segmap_start (floating)
*      |      Red Zone      |
* 0xC3000000 -|-----|- kernelbase / userlimit (floating)
*      |      Shared objects  | \ /
*      |      |      |      |
*      |      :      :      |
*      |      :      :      |
*      |      user heap      | (grows upwards)
*      |      user data      |
*      |-----|
*      |      user text      |
* 0x08048000 -|-----|-
*      |      user stack      |
*      |      :      :      |
*      |      invalid        |
* 0x00000000 +-----+
```

Адресное пространство процесса x64

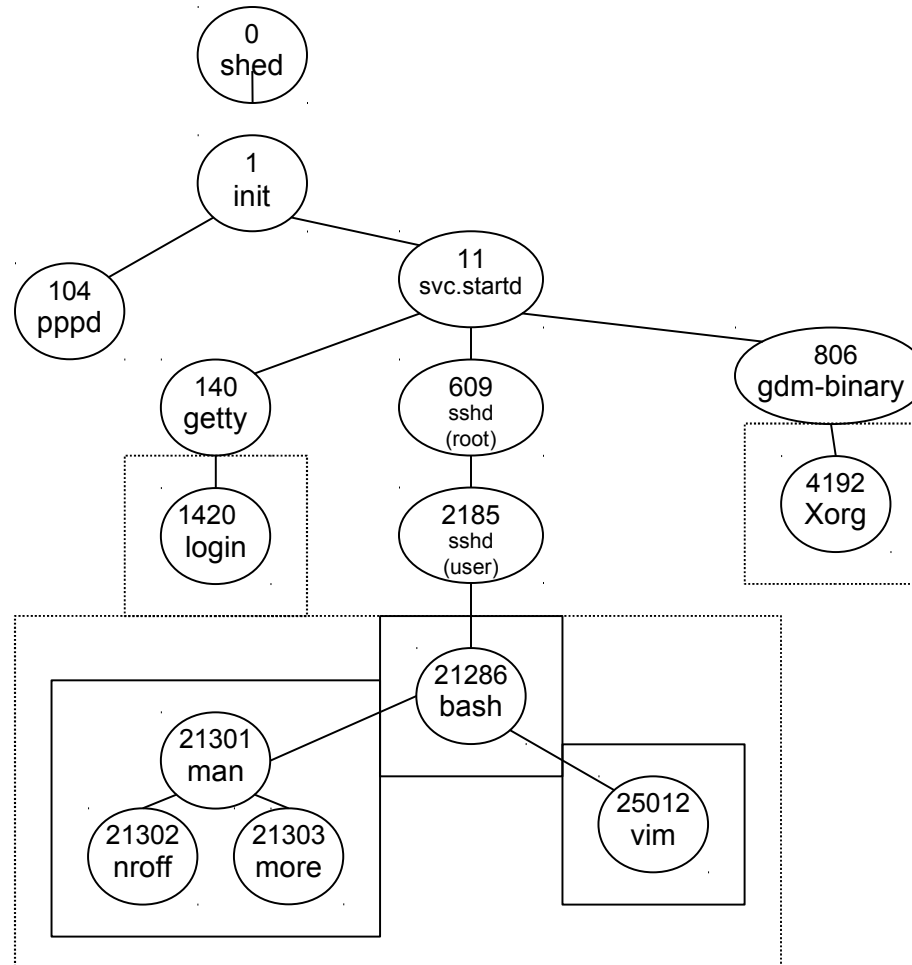
<http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/i86pc/os/startup.c>

```
* 64-bit Kernel's Virtual memory layout. (assuming 64 bit app)
*
*          +-----+
*          | Kernel Context |
* 0xFFFFFE00.00000000 |-----|
*          | Red Zone |
* 0xFFFFFD80.00000000 |-----| - KERNELBASE (lower if > 1TB)
*          | User stack | - User space memory
*          | shared objects, etc | (grows downwards)
*          | : |
*          | : |
* 0xFFFF8000.00000000 |-----|
*          | VA Hole / unused |
*          | : |
* 0x00008000.00000000 |-----|
*          | : |
*          | user heap | (grows upwards)
*          | user data |
*          |-----|
*          | user text |
* 0x00000000.04000000 |-----|
*          | invalid |
* 0x00000000.00000000 +-----+
```

Еще информация о процессах

- Каждый процесс, кроме `init`, имеет родителя
- `init` – процесс с `pid==1`, обычно `/sbin/init`
- `init` запускается при старте системы ядром и запускает все остальные процессы
- В Unix SVR4 `init` определяет, что запускать, по файлу `/etc/inittab`

Взаимосвязь между процессами



Терминальные сессии

- Физические терминалы обслуживаются демоном `ttymon` (в старых юниксах – `getty`).
- Виртуальные терминалы (сессии `xterm`, `ssh`) создаются динамически соответствующими сервисами, например для сессий `ssh` – демоном `sshd`
- При входе пользователя в систему создается сессия (`id` сессии равен `id` процесса, который создал эту сессию), принимается авторизация, устанавливается идентификатор пользователя и запускается шелл.
- Шелл берется из учетной записи пользователя, например из `/etc/passwd`, NIS или LDAP
- У большинства из вас шелл - `/bin/bash`
- Шеллы с управлением заданиями (`csh`, `zsh`, `ksh`, `bash`) создают на каждую команду так называемую группу процессов (будет разбираться в теме «управление терминальными устройствами»).

Атрибуты процесса

В пользовательской области (в ядре)

- процесс
 - номер
 - родитель
 - группа
 - терминал
 - сессия
 - ограничения (например, время ЦП, открытые файлы) .
- пользователь
 - идентификатор (реальный/эффективный)
 - группа (реальный/эффективный) .
- файловая система
 - открытые файлы
 - текущая директория
 - ограничения (umask, ulimit)
 - корневая директория (setroot)
- обработка сигналов

В пользовательском стеке

- параметры командной строки
- экспортированные shell-переменные (переменные среды)

Доступ к атрибутам процесса

Информация	Метод
Идентификатор процесса	getpid(2) getppid(2) getpgid(2) setpgid(2) getsid(2) setsid(2)
Пользователь/группа	getuid(2) geteuid(2) getgid(2) getegid(2) setuid(2) setgroups(2)+ getgroups(2) initgroups(3C)+
Ресурсы процесса	getrlimit(2) setrlimit(2)
Терминал процесса	ttyname(3C)

+ доступно только суперпользователю (uid==0)

Доступ к системным переменным

ИМЯ

`sysconf` — получить значение системной переменной

`pathconf` — получить настраиваемые параметры файла или файловой системы

СИНТАКСИС

```
#include <unistd.h>
```

```
long sysconf(int name);
```

```
long fpathconf(int fildes, int name);
```

```
long pathconf(const char *path, int name);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

значение переменной или параметра

При ошибке, -1 и `errno` установлена

Пользовательский стек

- Переменные среды (environ)
- Позиционные аргументы (argv)
- Запись активации main
- Записи активации функций
- Ограничен защитной областью

Доступ к переменным среды

- третий аргумент `main()`:

```
main(int argc, char **argv, char *envp[])  
    /* or char **envp */
```

- внешняя переменная `environ`

```
extern char **environ;
```

- библиотечные функции `getenv(3C)` и `putenv(3C)`

Переменные среды

- PATH – список каталогов, где ищутся исполняемые файлы
- TERM – тип терминала (используется экранными редакторами и рядом других программ)
- TZ – временная зона
- HOME – ваш домашний каталог
- USER – имя пользователя

PATH

```
-bash-3.00$ echo $PATH
```

```
/opt/SUNWspro/bin:/usr/sfw/bin:/opt/csw/bin:\n  /usr/bin:/opt/SUNWspro/bin:/opt/sfw/bin:\n  /home/teachers/fat/bin
```

- Текущий каталог в пути нужно добавлять явно:

```
export PATH=$PATH:.
```

- Лучше не добавлять, особенно в начало

TZ

- Asia/Novosibirsk
- PST8PDT (Pacific Standart Time, 8 часов на запад от Гринвича, с переводом летнего/зимнего времени)
- QQQ-6 (три любые буквы, 7 часов на восток от Гринвича, перевода летнего/зимнего времени нет)
- Пустая строка – брать из /etc/TIMEZONE

TZ (продолжение)

- Каждый процесс живет в своей временной зоне, определяемой его значением TZ
- Ядро живет по Гринвичу (GMT, UTC).
- `time(2)` возвращает время в секундах по Гринвичу
- Временные штампы (например, даты файлов) тоже отсчитываются по Гринвичу
- Очень удобно при удаленном входе в систему.

Выделение опций командной строки getopt(3C)

ИМЯ

getopt - выделить опцию из командной строки

СИНТАКСИС

```
#include <stdlib.h>
```

```
int getopt (int argc, char *const *argv,  
            const char *optstring);
```

```
extern char *optarg;
```

```
extern int optind, opterr, optopt;
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

буква верной опции

'?' если обнаружена недопустимая опция

-1 если все опции были обработаны.

Использование getopt(3C)

- Вызывается в цикле, пока не обработает все опции.
- Опция – аргумент, начинающийся с символа ‘-’
- Getopt(3C) разбирает однобуквенные опции, например, `ls -l`, `ls -lh`
- Опция может иметь аргумент, например `cc -o a.out`, `cc -oa.out`
- В GNU libc есть `getopt_long`, который понимает опции вида `--all`
- стандартный `getopt` такой формы не имеет

Использование getopt(3C)

- `int argc`. Обычно первый аргумент `main()`.
- `char **argv`. Обычно второй аргумент `main()`.
- `char *optstring`. Это строка допустимых опций. Если у опции есть аргументы, то за этой опцией будет стоять двоеточие.

Использование getopt(3C)

- `char *optarg`. Когда `getopt(3C)` обрабатывает опцию, у которой есть аргументы, `optarg` содержит адрес этого аргумента.
- `int optind`. Когда `getopt(3C)` возвращает `-1`, `argv[optind]` указывает на первый аргумент не-опцию.
- `int opterr`. Когда `getopt(3C)` обрабатывает недопустимые опции, сообщение об ошибке выводится на стандартный вывод диагностики. Печать может быть подавлена установкой `opterr` в ноль.
- `int optopt`. Когда `getopt(3C)` возвращает '?', `optopt` содержит значение недопустимой опции.

Бит setuid

- доступ к ресурсам основывается на эффективном идентификаторе пользователя и группы
- с помощью механизма установки идентификатора пользователь может быть чьим-то "представителем"
- При запуске программы с битом setuid, эффективный идентификатор приравнивается uid хозяина файла программы
- Действует от начала исполнения процесса до конца или до использования setuid(2).

Бит `setuid` (продолжение)

- `setuid`-бит для `a.out` устанавливается с помощью `chmod(1)`, `chmod(2)` и `open(2)`
- `setuid`

04711 восьмеричное число

`u+s` символьная запись для `chmod(1)`

`rws--x--x` выводит `ls -l`

- `setgid`

02711 восьмеричное число

`g+s` символьная запись для `chmod(1)`

`rws--s--x` выводит `ls -l` .