Системное программирование

Лекция 1

Введение. Стандарты UNIX и POSIX.
Программы в POSIX.
Основы работы в терминале.

Операционная система

Операционная система (ОС) – программный комплекс, который управляет аппаратным обеспечением компьютера и предоставляет унифицированный доступ к аппаратному обеспечению другим программам.

Задачи ОС:

- управление аппаратным обеспечением: распознавание и настройка устройств, выполнение операций ввода-вывода и т.д.
- предоставление доступа к аппаратному обеспечению другим программам: предоставление унифицированного API, независимого от конкретного аппаратного обеспечения; контроль и разграничение доступа к аппаратному обеспечению.

Ядро ОС – часть ОС, непосредственно отвечающая за управление аппаратными ресурсами компьютера и обслуживание запросов на доступ к нему со стороны прикладных программ.

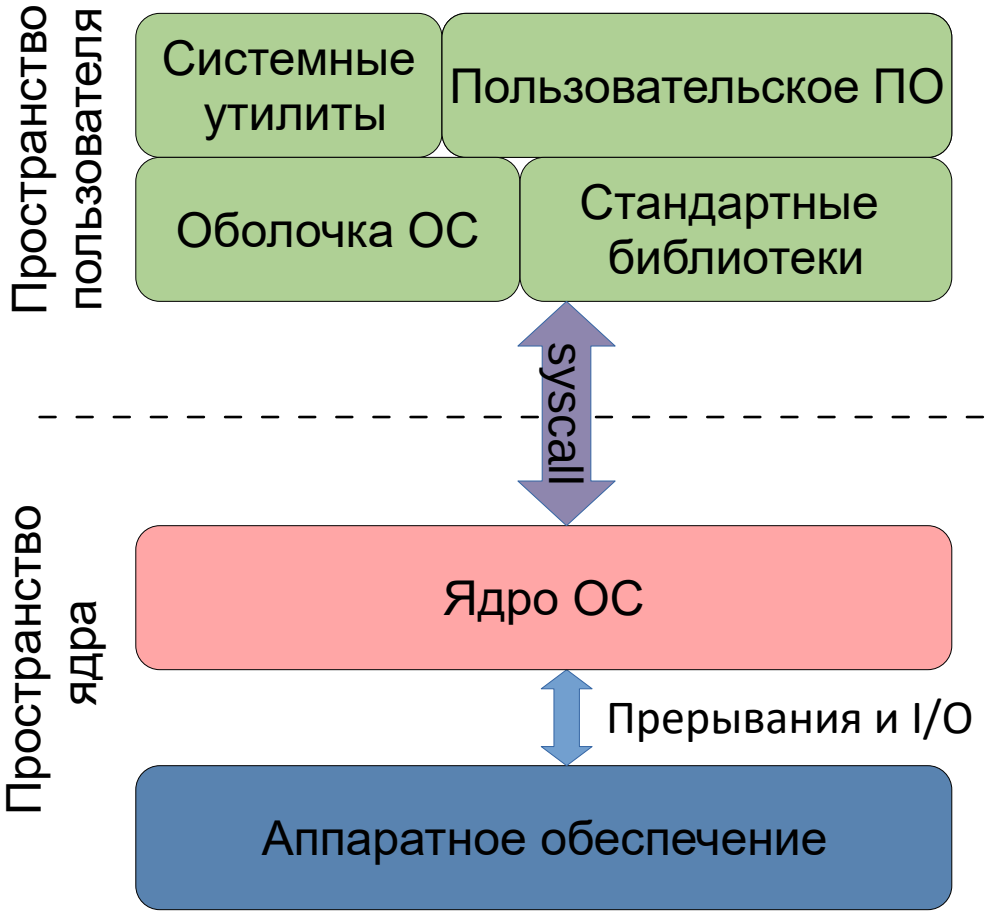
Пространства ядра и пользователя

Код ядра ОС и необходимые данные находятся в защищенной части памяти, которая называется **пространством ядра**.

Остальные части ОС а также пользовательские программы находятся в общем **пространстве пользователя**.

Переход между пространствами ядра и пользователя осуществляется посредством **системных вызовов**.

Набор системных вызовов определяет **API** (application programming interface) операционной системы.



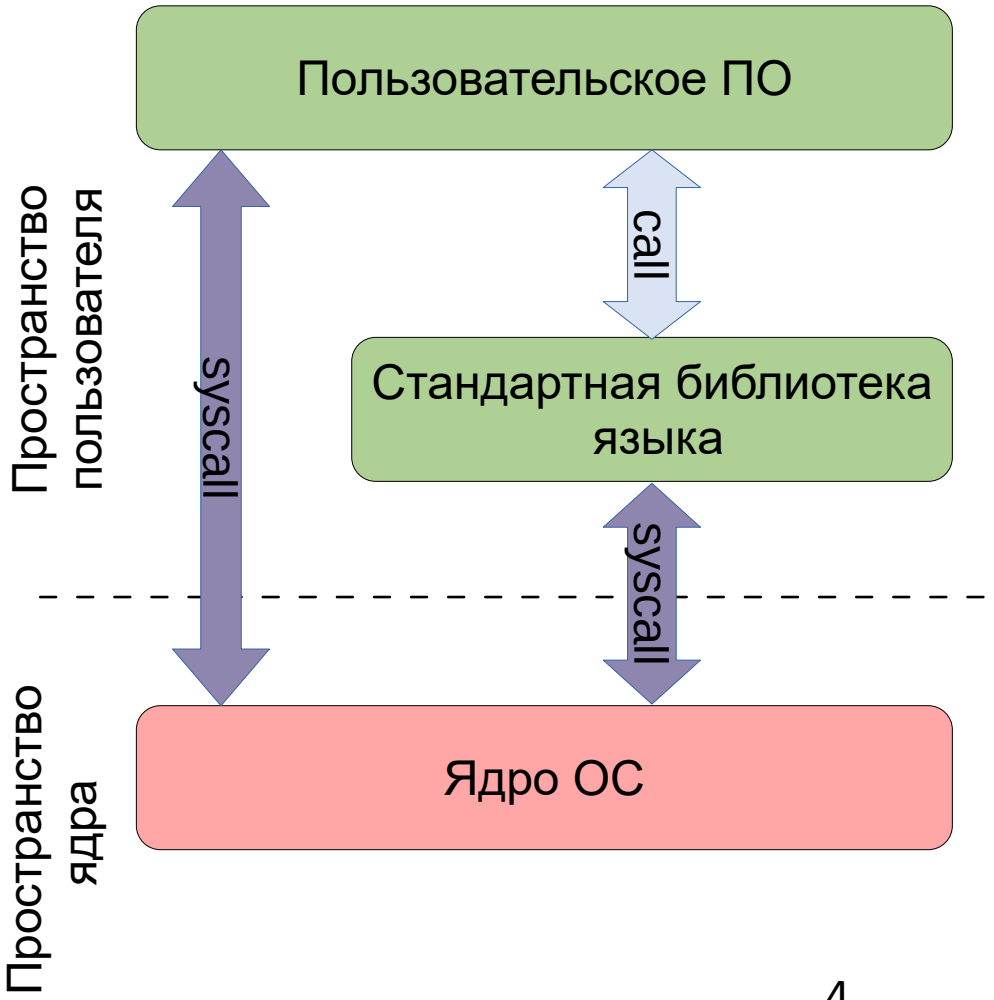
Системные вызовы

Пользовательские программы могут осуществлять системные вызовы напрямую (через инструкцию `syscall/int`).

Стандартные библиотеки некоторых языков предоставляют обертки системных вызовов в виде функций, которые вызываются обычным образом.

Кроме того, стандартные библиотеки языков обычно предоставляют собственные высокоуровневые программные интерфейсы (особенно в случае ООП-языков). Однако внутри, в конечном итоге, используются системные вызовы.

В дальнейшем, термином «вызов» будет обозначаться системный вызов или его обертка, термином «функция» - функция, не являющаяся системным вызовом.



ОС Unix. Стандарт POSIX

В 1970 году вышел первый выпуск операционной системы Unix. Данная система завоевала большую популярность и породила ряд ответвлений (BSD, Xenix и т.д.).

Системы, построенные на принципах, заложенных в Unix, называют **Unix-подобными**. Большинство Unix-подобных ОС используют ядро Linux.

Для того, чтобы минимизировать отличия между Unix-подобными ОС, был создан стандарт POSIX.

POSIX (Portable Operating System Interface) – стандарт IEEE, описывающий системный API (набор системных вызовов), набор команд оболочки ОС, набор системных утилит и системных библиотек (в т.ч. библиотеку языка С), которые функционируют поверх системного API. POSIX делится на обязательную часть и набор необязательных расширений.

Более общий стандарт **Single UNIX Specification** (SUS) включает в себя POSIX.

Философия UNIX

Пишите программы, которые делают что-то одно и делают это хорошо.

Среди стандартных утилит UNIX нет программ, которые бы делали несколько вещей сразу, каждая программа сфокусирована на своей задаче.

Пишите программы, которые бы работали вместе.

Следует из первого пункта — для того, чтобы делать сложные вещи, можно использовать несколько простых программ.

Пишите программы, которые бы поддерживали текстовые потоки, поскольку это универсальный интерфейс.

Использование единого формата представления данных позволяет облегчить взаимодействие программ с пользователем и взаимодействие программ между собой.

Файловая система UNIX

- Каталоги образуют дерево с единственным корнем.
- **Корневой каталог** всегда имеет имя “/”.
- В любом каталоге есть два подкаталога:
  “.” - **ссылка на самого себя**
  “..” - **ссылка на родительский каталог**.
- Путь, который начинается с “/”, называется **абсолютным**. Абсолютный путь однозначно идентифицирует файл.
- Путь, который не начинается с “/”, называется **относительным**.
- Относительные пути отсчитываются от **рабочего каталога**.

```
/home/ivanov/./ = /home/ivanov/
/home/ivanov/../ = /home
```

Оболочка ОС и терминал

Оболочка ОС (shell) – специальное ПО, предназначенное для организации непосредственного взаимодействия пользователя и ОС посредством специального набора команд.

Стандарт POSIX строго определяют минимальный набор команд оболочки и их поведение, но язык написания скриптов у разных оболочек может отличаться.

Изначально оболочка была единственным способом взаимодействия с ОС. Подключение к компьютеру осуществлялось через выделенное устройство, называемое **терминалом**. В настоящее время, для того, чтобы не менять внутреннюю структуру ОС, вместо терминалов используются программные *псевдотерминалы*.

Отличие между оболочкой и (псевдо)терминалом заключается в назначении: терминал предназначен для приема пользовательского ввода и вывода результата, а оболочка – для интерпретации ввода и выполнения требуемых команд.

Запуск программ в оболочке

Запуск программы в оболочке осуществляется путем указания абсолютного либо относительного имени исполняемого файла.

Если указано просто имя программы, без символа “/”, то программа ищется в каталогах, отмеченных в переменной окружения **PATH** и только в них, текущая директория по умолчанию не просматривается.

Из этого следует, что чтобы запустить файл foo из текущего рабочего каталога, его нужно указывать как “*./foo*”.

Функция main. Аргументы программы(пример 1)

Согласно стандарту языка C, точкой входа в программу является функция `main()`:

```
int main(int argc, char** argv)
```

Аргумент **`argc`** равен числу аргументов программы. Если `argc` > 0, то первым аргументом будет исходная команда запуска программы.

Массив строк **`argv`** содержит аргументы команды. Размер массива равен `argc`+1, последний элемент массива равен NULL.

Возвращаемое значение `main()` является **кодом завершения** программы.

По общему соглашению, код завершения должен быть равен 0 при успешном завершении программы, иначе — не 0. При ошибке конкретное значение кода возврата определяет программист.

Примечание: в других языках точка входа может определяться иначе.

Переменные окружения (пример 1)

Переменные окружения/переменные среды (environment variables) - параметры окружения в котором запускается программа.

Переменные среды хранятся в виде строк вида *”<имя переменной>=<значение>”*. Переменные среды могут изменяться оболочкой ОС. Некоторые переменные влияют на поведение самой оболочки или системных утилит (в т.ч. динамическогоко компоновщика).

Список переменных окружения можно вывести в оболочке командой *env*.

Внутри программы переменные среды доступны через переменную `environ` и с помощью спец. функций:

```
#include <stdlib.h>

extern char** environ;          //указатель на переменные среды

char* getenv(const char* name);
int setenv(const char* name, const char* value, int overwrite);
int unsetenv(const char* name);
```

Стоит посмотреть: команда export, переменные PATH, LD_LIBRARY_PATH и LD_PRELOAD

Стандартные потоки

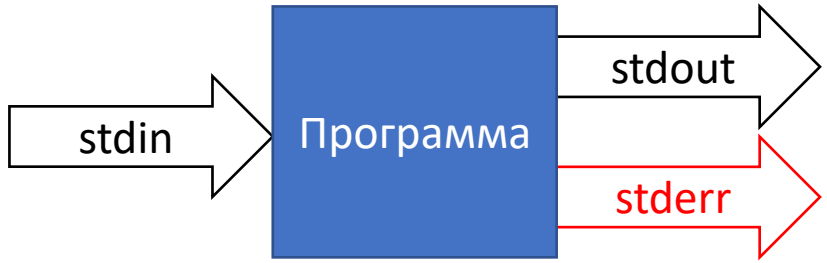
Считается, что программа имеет 3 стандартных потока:

Стандартный поток ввода (stdin) служит для передачи данных программе. Функции `scanf`, `gets`, `readln`, `Console.ReadLine` из стандартных библиотек разных языков программирования берут данные именно из потока ввода.

Стандартный поток вывода (stdout) служит для вывода данных программой. С ним работают функции `printf`, `puts`, `writeln`, `Console.WriteLine`, и т. п.

Стандартный поток вывода ошибок (stderr) служит для вывода сообщений об ошибках. По умолчанию поток ошибок направлен туда же, куда и stdout.

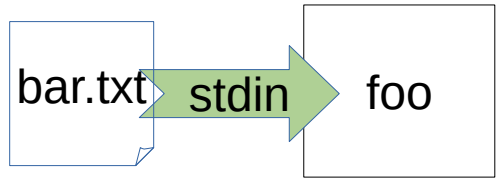
По умолчанию, все 3 потока направлены в терминал, из которого программа была запущена.



Перенаправление потоков

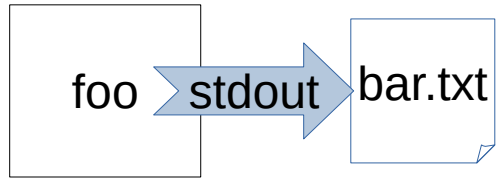
Для перенаправления стандартного ввода служит символ **<** .

```
$ ./foo <bar.txt      # содержание bar.txt отправлено в stdin.
```



Для перенаправления стандартного вывода служит символ **>**.
Если приемником является файл, он будет перезаписан. Если файла нет, он будет создан.
Если нужно добавить данные в конец файла без перезаписи, используется комбинация **>>**.

```
$ ./foo >bar.txt      # bar.txt создан/перезаписан
$ ./foo >>bar.txt     # bar.txt создан/дополнен
```



Для перенаправления стандартного потока ошибок используется комбинация **2>**.

```
$ ./foo >result.txt 2>errors.txt      # вывод программы запишется в result.txt,
                                      # сообщения об ошибках  - в errors.txt
```

Конвейеры

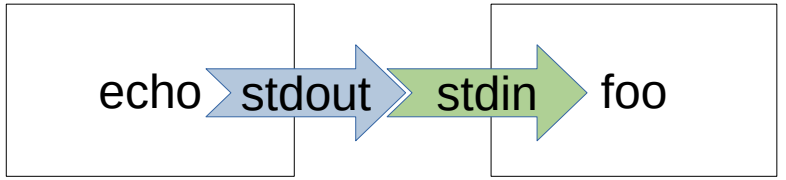
Для объединения выхода одной программы со входом другой программы используется конвейер (символ |)

При объединении программ в конвейер программы запускаются одновременно и обрабатывают данные по мере получения.

Пример:

```
$ echo “Hello world” | ./foo
```

“Hello world”станет входными данными foo



Обработка ошибок. Переменная errno (пример 2)

В случае ошибки функции и системные вызовы возвращают некоторое *определённое* значение (обычно -1 или NULL), сигнализирующее об ошибке.

Причина ошибки записывается в глобальную переменную **errno**. Значение в errno совпадает с одной из констант, определенных в **<errno.h>** (например, EACCES – доступ запрещен). *При успешном завершении функции значение errno не меняется.*

Для вывода текстового имени ошибки с дополнительным сообщением используется функция perror().

```
int fd = open("file", O_RDWR);
if (fd == -1) {
    if(errno == ENOENT)
        fputs("File does not exist", stderr);
    else{
        perror("Failed to open file"); exit(1);
    }
}
```

Параметры и ограничения ОС (пример 3)

Для определения особенностей конкретной ОС в POSIX используются функции из заголовочного файла <unistd.h>.

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

В качестве параметра `name` выступает именованная целочисленная константа, определенная в виде макроса.

Имена констант начинаются с `_SC` (например, `_SC_OPEN_MAX`) для `sysconf` и с `_PC` (`_PC_PATH_MAX`) – для `pathconf`.

Парной командой терминала является *`getconf`*.

Команда man

Документация по системе и программам доступна через команду *man* (**man**ual).

man [опции] [раздел] <имя>

Полезные разделы man:

1 – общие команды;
2 – системные вызовы;
3 – библиотека языка C.

```
$ man mkdir      # документация по команде mkdir
$ man 2 mkdir    # документация по системному вызову mkdir
```

Часто при установке программы из репозитория вместе с ней автоматически устанавливается документация, доступная через *man*.

Опция --help

Все системные программы и команды оболочки поддерживают опцию *--help*.

Данная опция заставляет команду распечатать информацию по использованию команды (назначение команды, опции, обязательные и необязательные аргументы).

```
$ grep --help
```

```
$ cat --help
```

```
$ ls --help
```

Трассировка программ

Трассировка программы – отслеживание процесса выполнения программы.

Утилита *strace* позволяет отслеживать системные вызовы, выполняемые трассируемой программой.

Общий вид команды: *strace [аргументы strace] <программа> [аргументы программы]*

$ strace echo “Hello world”

Утилита *ltrace* позволяет отслеживать вызовы функций из системных библиотек. Общий вид команды аналогичен strace.

$ ltrace echo “Hello world”

Полезные команды

pwd – печать рабочего каталога

cd – перемещение между каталогами

ls – просмотр содержимого каталога

touch – создание файла

cat – вывод содержимого файла

less – просмотр файла с прокруткой

head – вывод начала файла

tail – вывод конца файла

find – поиск файлов

grep – поиск в тексте

env – вывод переменных окружения

mkdir – создание каталога

rm – удаление файлов и каталогов

cp – копирование файлов и каталогов

mv – переименование и перемещение файлов и каталогов

sort – сортировка строк

ln – создание ссылок

echo – повтор ввода

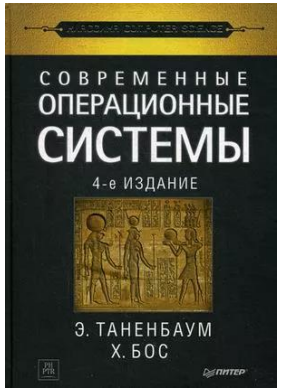
chmod – изменение прав доступа к файлу

stat – просмотр информации о файле

sed – модификация текста по шаблону

tee – дублирование входных данных в файл

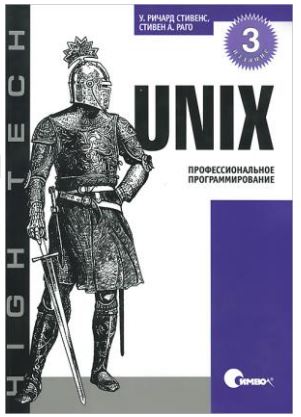
Полезные ресурсы и литература



Э. Таненбаум, Х. Бос.Современные операционные системы. 4-е изд.

Книга по внутреннему устройству и общим концепциям ОС (в т.ч. Windows и Android).

У. Стивенс, С. Раго. UNIX. Профессиональное программирование. 3-е изд.



Книга-справочник, ориентированная на практическое применение. Рассматривается POSIX-интерфейс современных ОС и его использование.

man ☺ online-версия