Виды ПО

ПО

Прикладное

Системное

Утилиты

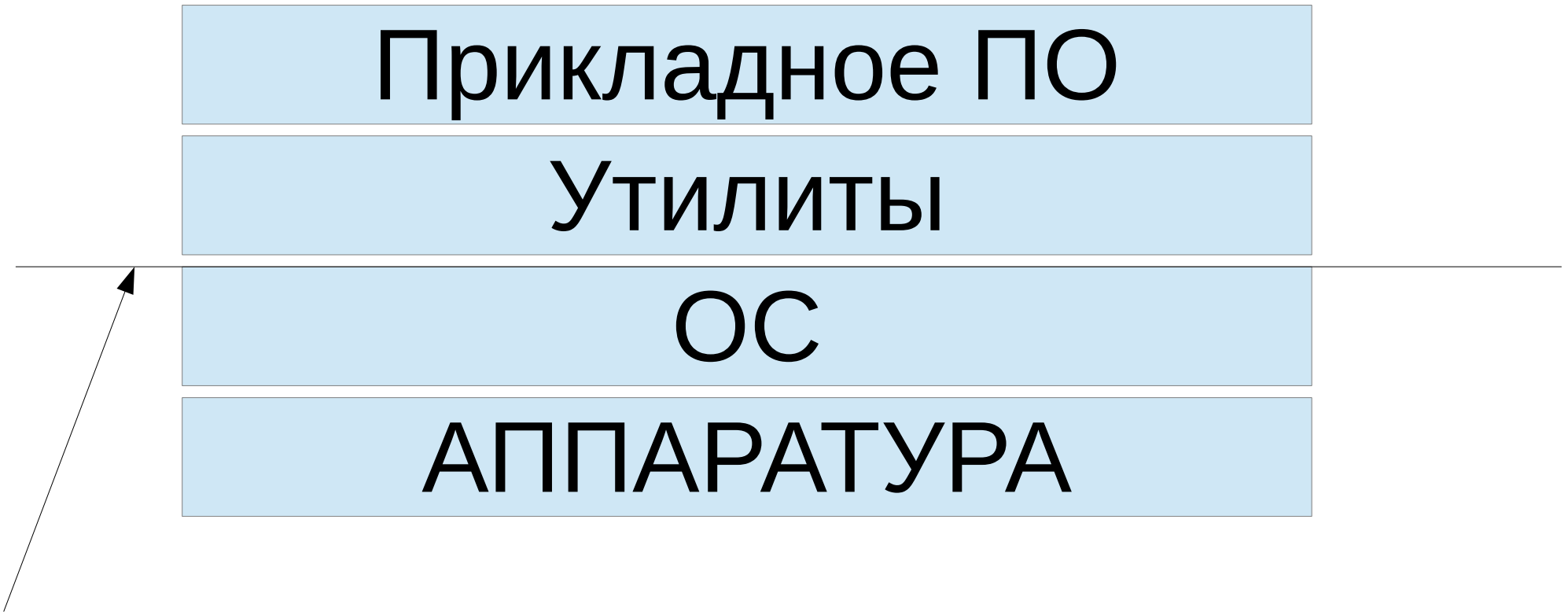
Библиотеки

Трансляторы

Демоны

ОС

Структура ПО



Стандартизация

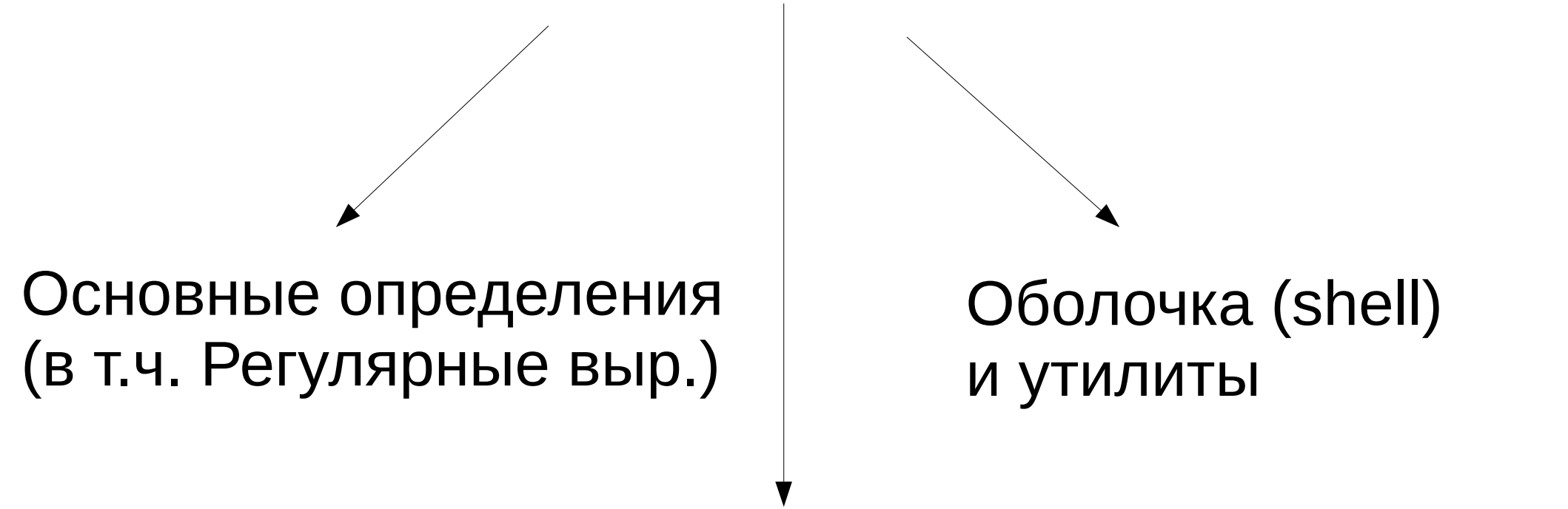
POSIX (Cygwin...), WinAPI

1. Перенос приложений на широкий диапазон систем
2. Совместная работа приложений в локальной или глобальной сети
3. Взаимодействие с пользователем в стиле, облегчающим ему переход от одной системы к другой

Открытое ПО ≠ Свободное ПО

POSIX

IEEE Std 1003.1, 2013 Edition



WinAPI

А давайте зафигачим еще одну функцию, т.к. предыдущая оказалась …., и чтобы в нее обязательно можно было передать не менее 15-ти обязательных параметров, по ссылкам, некоторые из них она поменяет. и половина параметров может быть 0, вторая Null. И еще чтоб ее два раза вызывать — первый чтобы определить необходимые размеры буферов. И чтобы она делала все-все и kitchen sink в придачу! А, и еще обязательно чтобы этой функции было три-четыре версии, разных поколений, и чтобы w64 обязательно что-то делала не так как w32 и если эта функция вернёт FALSE (не путать с false), то обязательно нужно будет сразу же вызвать GetLastError(), чтобы узнать - это по приколу так, или действительно ошибка.

Пример: запуск программы

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx

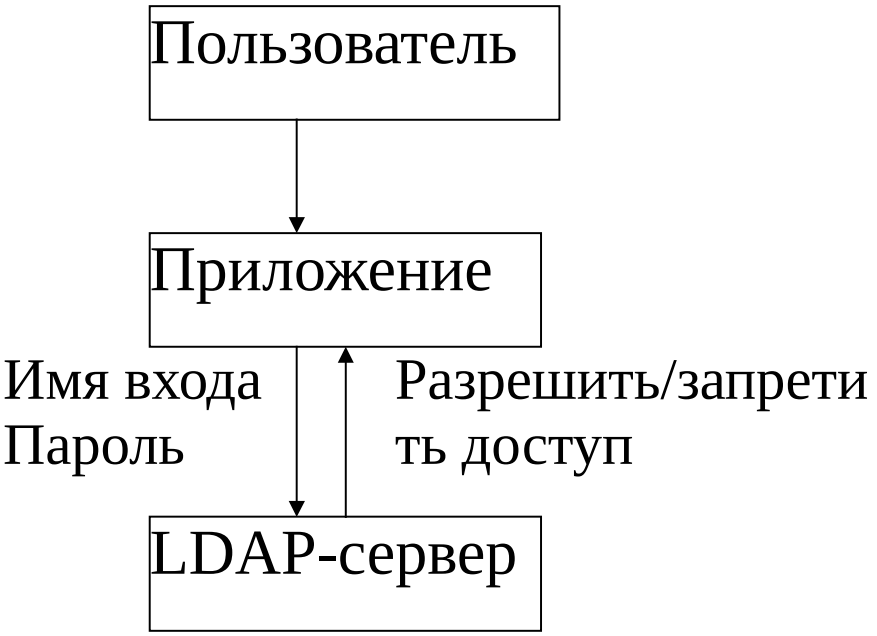
```
BOOL WINAPI CreateProcess(
  _In_opt_      LPCTSTR lpApplicationName,
  _Inout_opt_   LPTSTR lpCommandLine,
  _In_opt_      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_          BOOL bInheritHandles,
  _In_          DWORD dwCreationFlags,
  _In_opt_      LPVOID lpEnvironment,
  _In_opt_      LPCTSTR lpCurrentDirectory,
  _In_          LPSTARTUPINFO lpStartupInfo,
  _Out_         LPPROCESS_INFORMATION lpProcessInformation
);
```

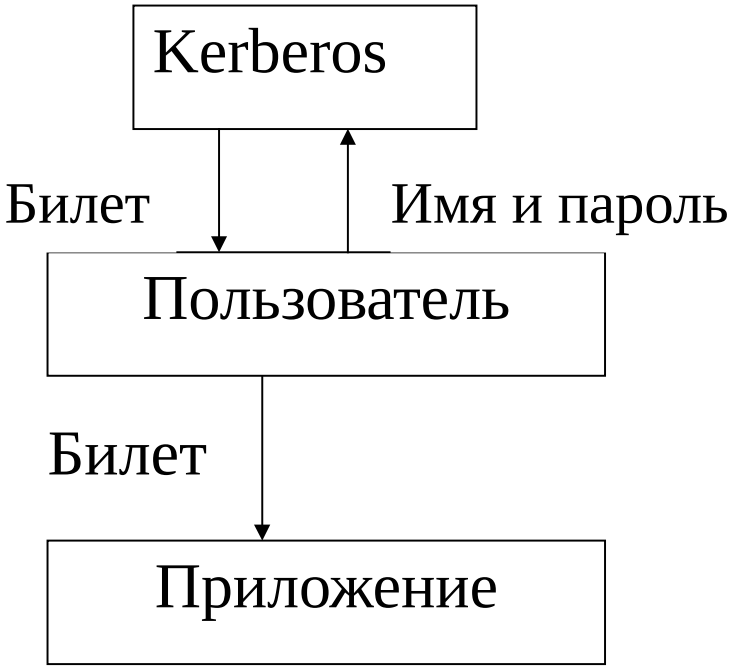
```
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

Аутентификация

/etc/passwd:

имя пользователя, домашний каталог, оболочка,UID и GID





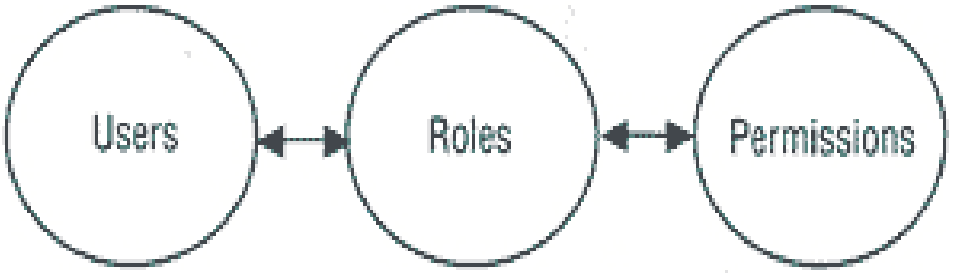
Модели прав доступа

Мандатная
MAC
(Mandatory access control)

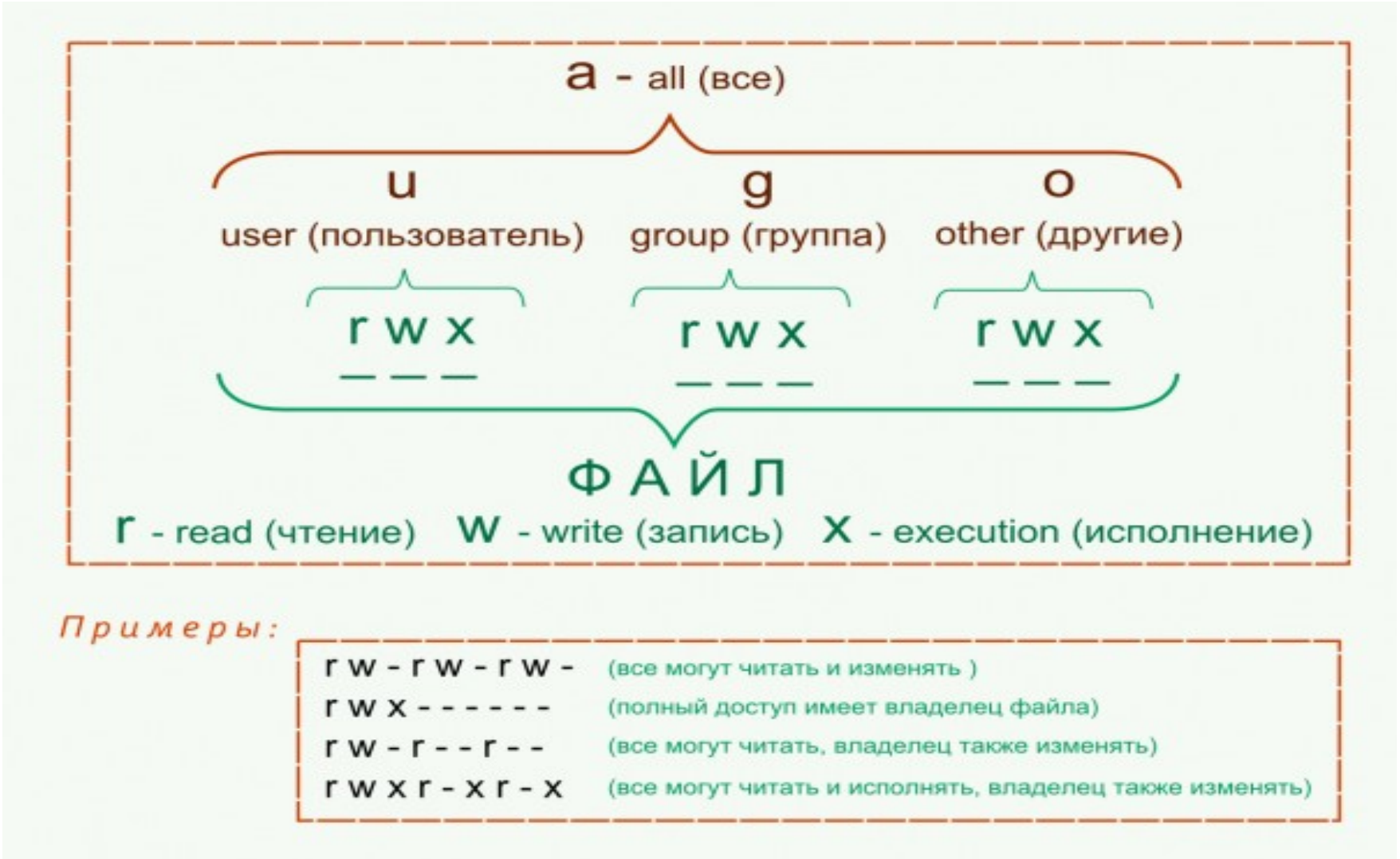
SELinux AstraLinux

Ролевая
RBAC
(Role Based Access Control)

Избирательная
(Дискреционная, DAC,
POSIX ACL)



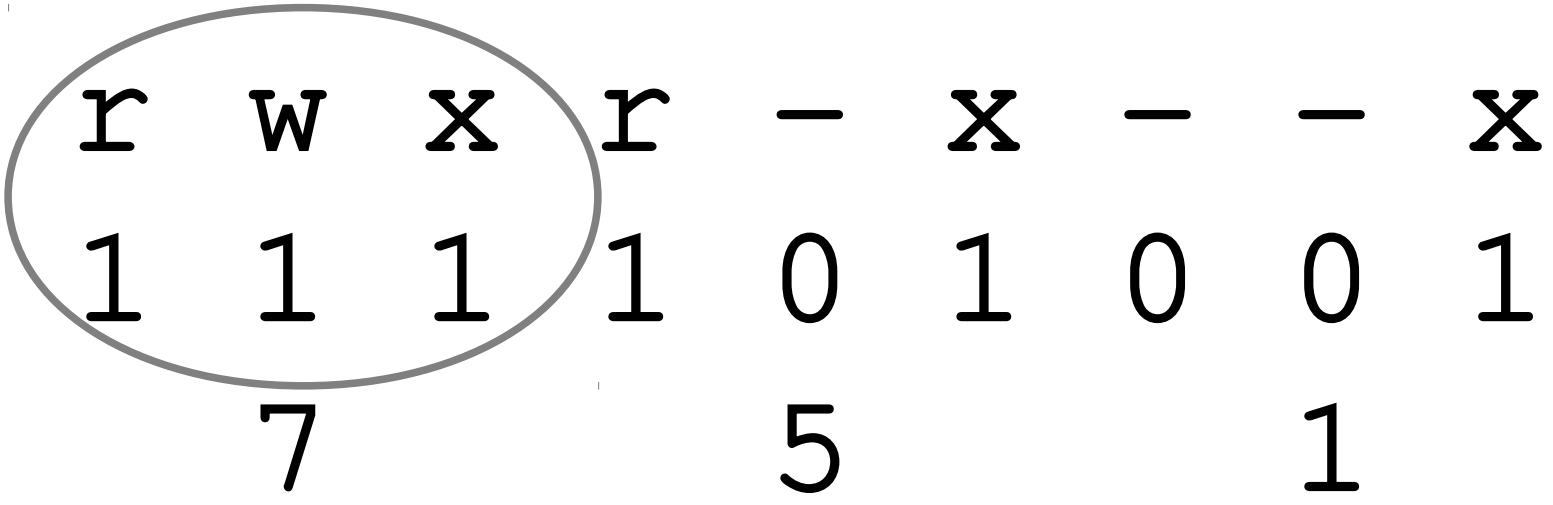
Права доступа файлов



Примеры:

r w - r w - r w -	(все могут читать и изменять)
r w x - - - - - -	(полный доступ имеет владелец файла)
r w - r - - r - -	(все могут читать, владелец также изменять)
r w x r - x r - x	(все могут читать и исполнять, владелец также изменять)

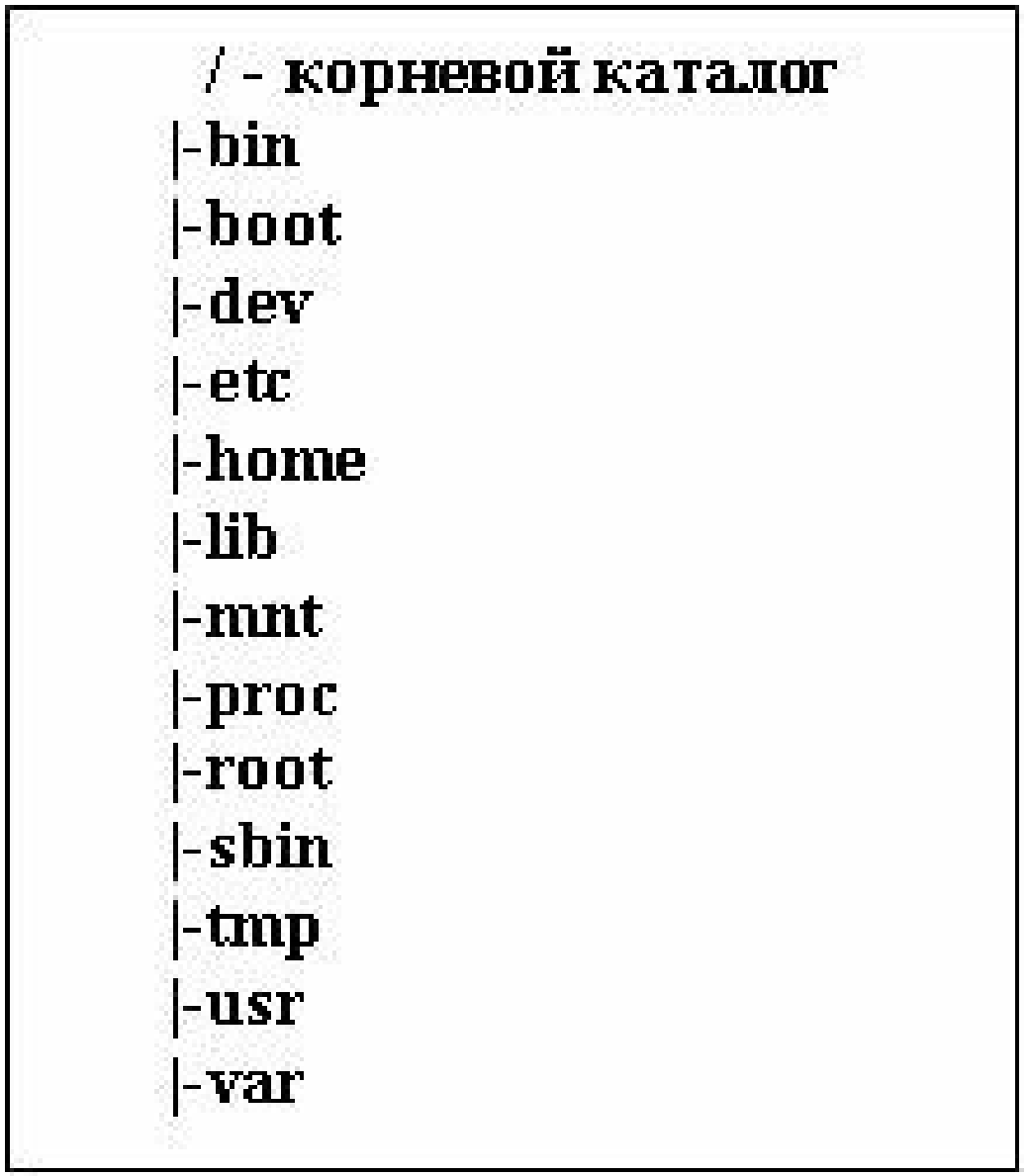
Кодирование прав доступа



Примеры установки прав доступа

```
$ ls -l f1.txt
-rw-r--r-- 1 kdb kdb 10 Янв 29 22:12 f1.txt
$ chmod 755 f1.txt
$ ls -l f1.txt
-rwxr-xr-x 1 kdb kdb 10 Янв 29 22:12 f1.txt
$ chmod g+w f1.txt
$ ls -l f1.txt
-rwxrwxr-x 1 kdb kdb 10 Янв 29 22:12 f1.txt
```

Структура каталогов Unix



Имена каталогов Unix

.. — вышестоящий каталог
. — текущий каталог
/ — корневой каталог

```
$ pwd
/tmp/someplace/t1
$ cd ..
$ pwd
/tmp/someplace
$ cd .
$ pwd
/tmp/someplace
```

Команды shell

Любой исполняемый файл!!!!

Типовой вызов команды

имяКоманды -опции --опции -- аргументы

```
ls -ltr --color /tmp
```

Связывание команд

; последовательное выполнение команд

$ k1; k2

& асинхронное (фоновое) выполнение предшествующей команды

$ k1 & k2

$ gcc pr.c & man gcc

&& выполнение команды при условии нормального завершения предыдущей

$ k1 && k2

$ mkdir doki && cd doki

|| выполнение команды при условии ненормального завершения предыдущей

$ cp a b || echo “Error”

{} для группировки команд в блок

$ k1 && {k2; k3}

() группировка + новый экземпляр интерпретатора shell

- для выполнения программы из файла в текущем Shell–е

$. k1.sh

Перенаправление ввода/вывода

`>` перенаправление вывода

$ k1 > f1

```
$ ls > f1.test
```

`>>` перенаправление вывода с дописыванием в файл

$ k1 > f1

```
$ pwd >>f1
```

`<` и `<<` перенаправление ввода

$ k1 < f1

```
$ wc -l < f1
```

$ k1 << спецСтрока

произвольный
многострочный текст
спецСтрока

```
$ cat « cat » cat
```

Конвейер

Конвейер объединяет стандартный выход k1 со стандартным входом k2

$ k1 | k2

$ k1 | k2 | … | kn

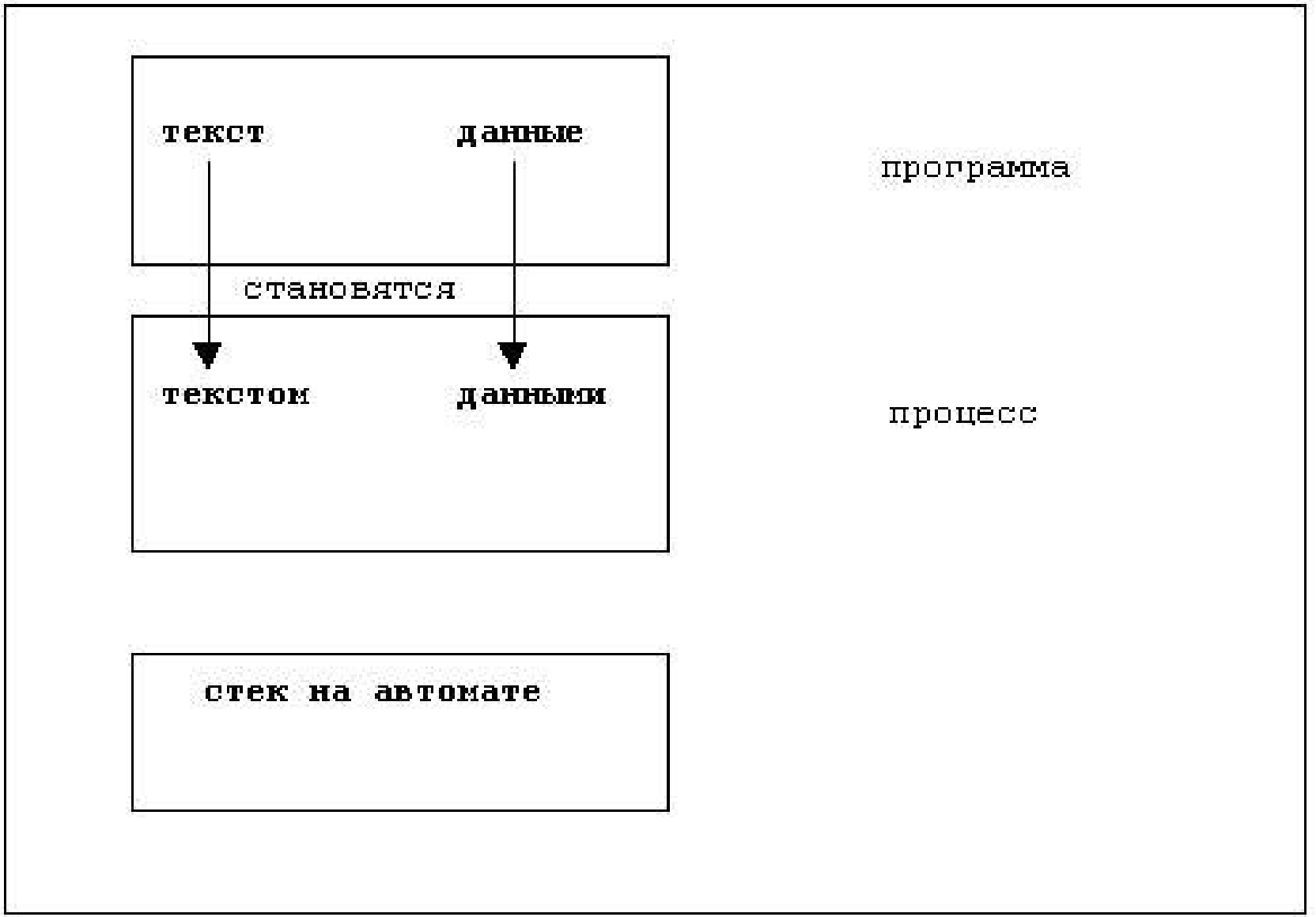
$ ls -l | wc -l

```
pipe(pd);
 if( fork() ) {
  close(1);
  dup(pd[1]);
  execl("/bin/ls", "LS", "-1", 0);
 }
 else {
  close(0);
  dup(pd[0]);
  execl("/bin/grep", "GGG", ".c", 0);
 }
```

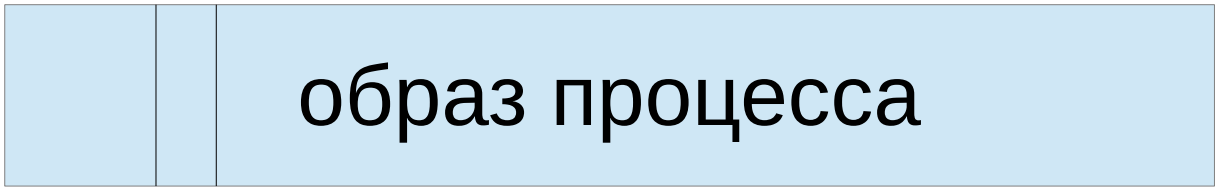
Понятие процесса

- Процесс
- Задача
- Легкий процесс (lightweight)
- Нить (thread)
- Тяжелый процесс (havy)
- Образ процесса
- Зомби
- Демоны

Образ процесса

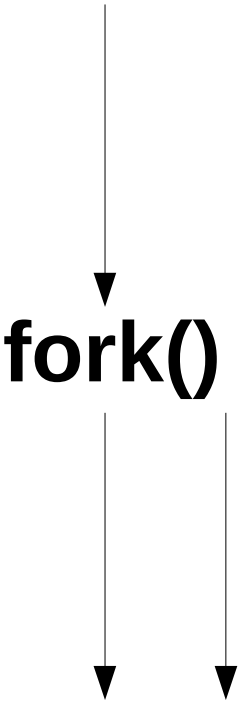


Нити

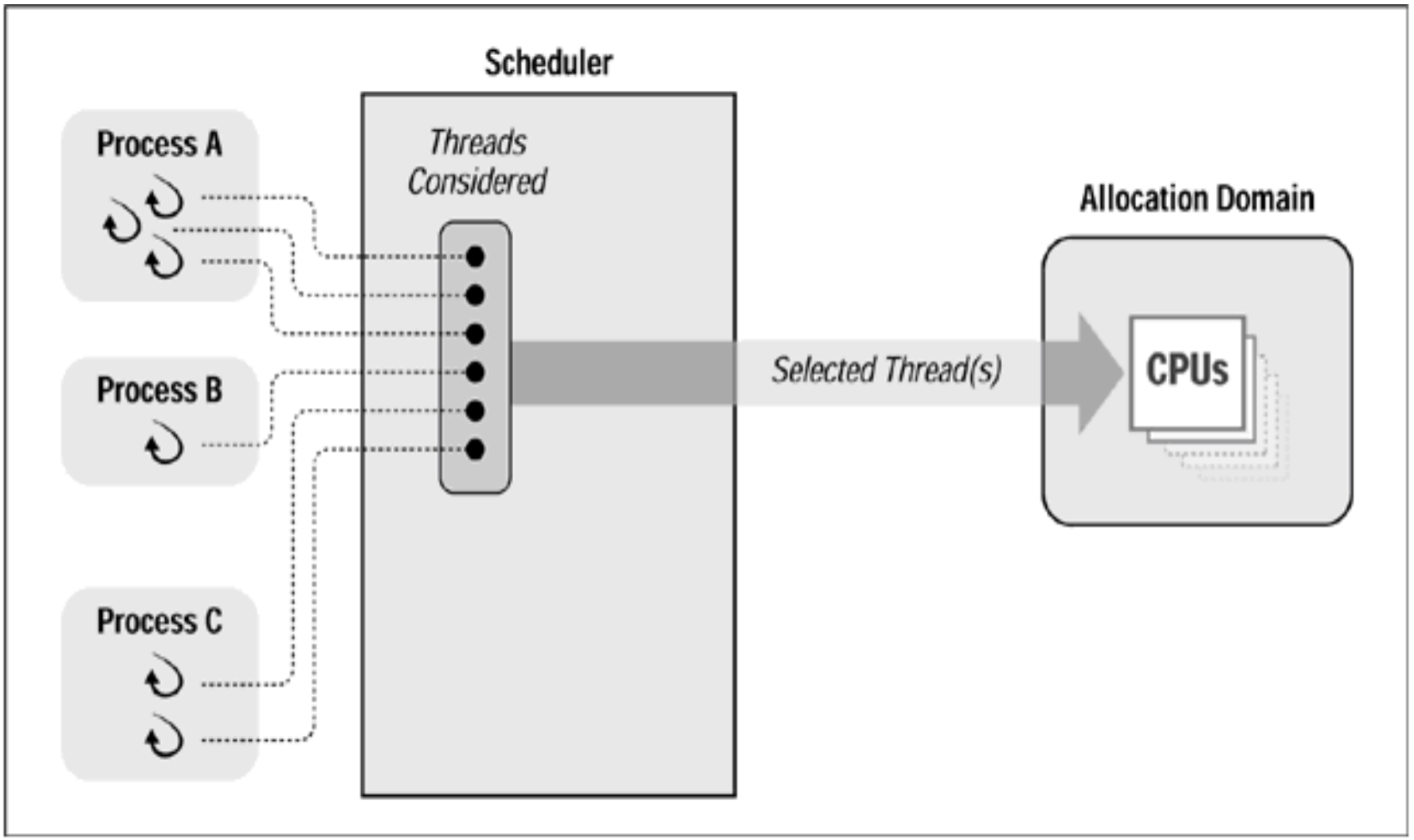


Тяжелый процесс

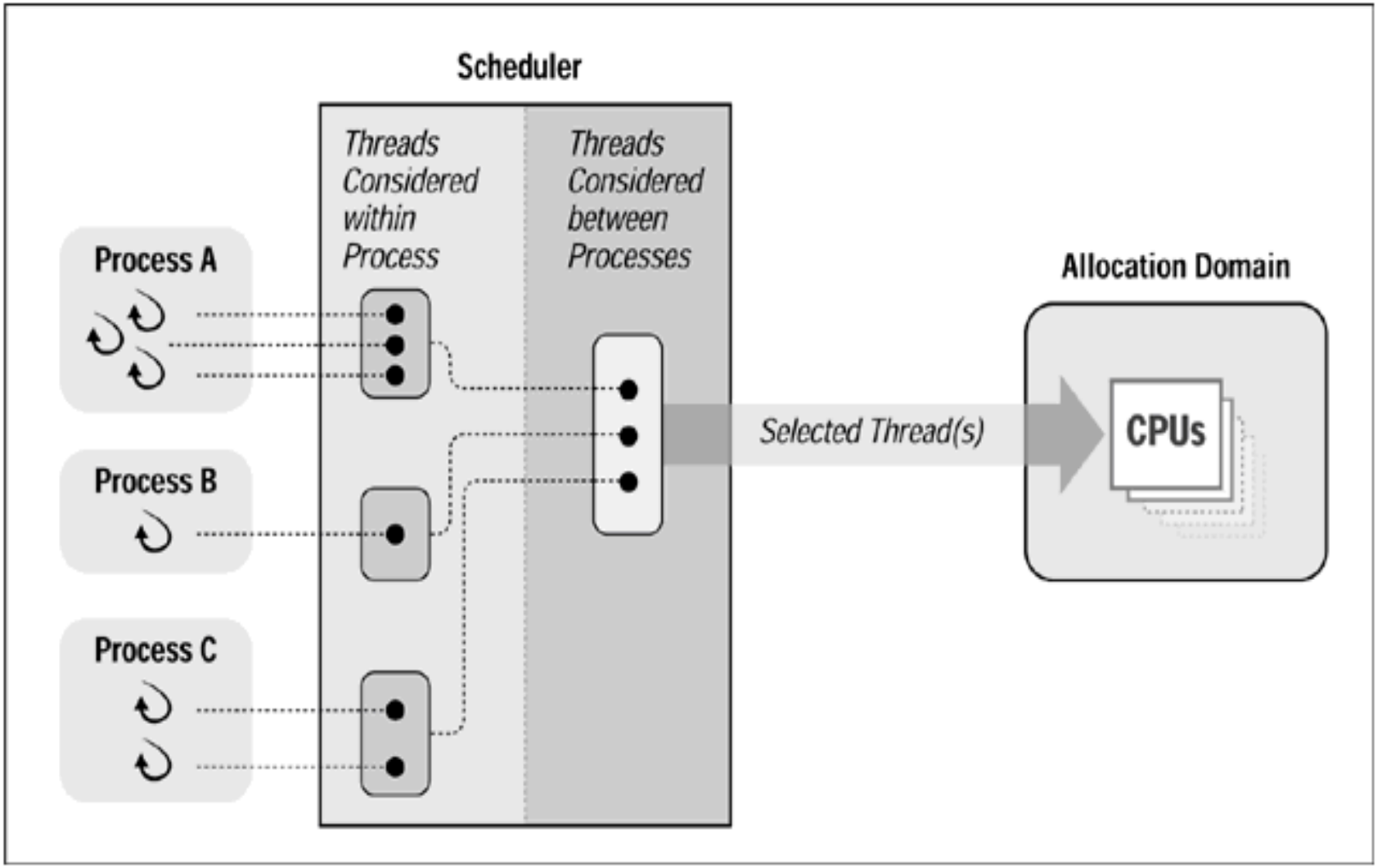
```
....
pid=fork();
if( pid == 0 )
{
 Родительский процесс
 wait(0);
}
{
Порожденный процесс
}
```



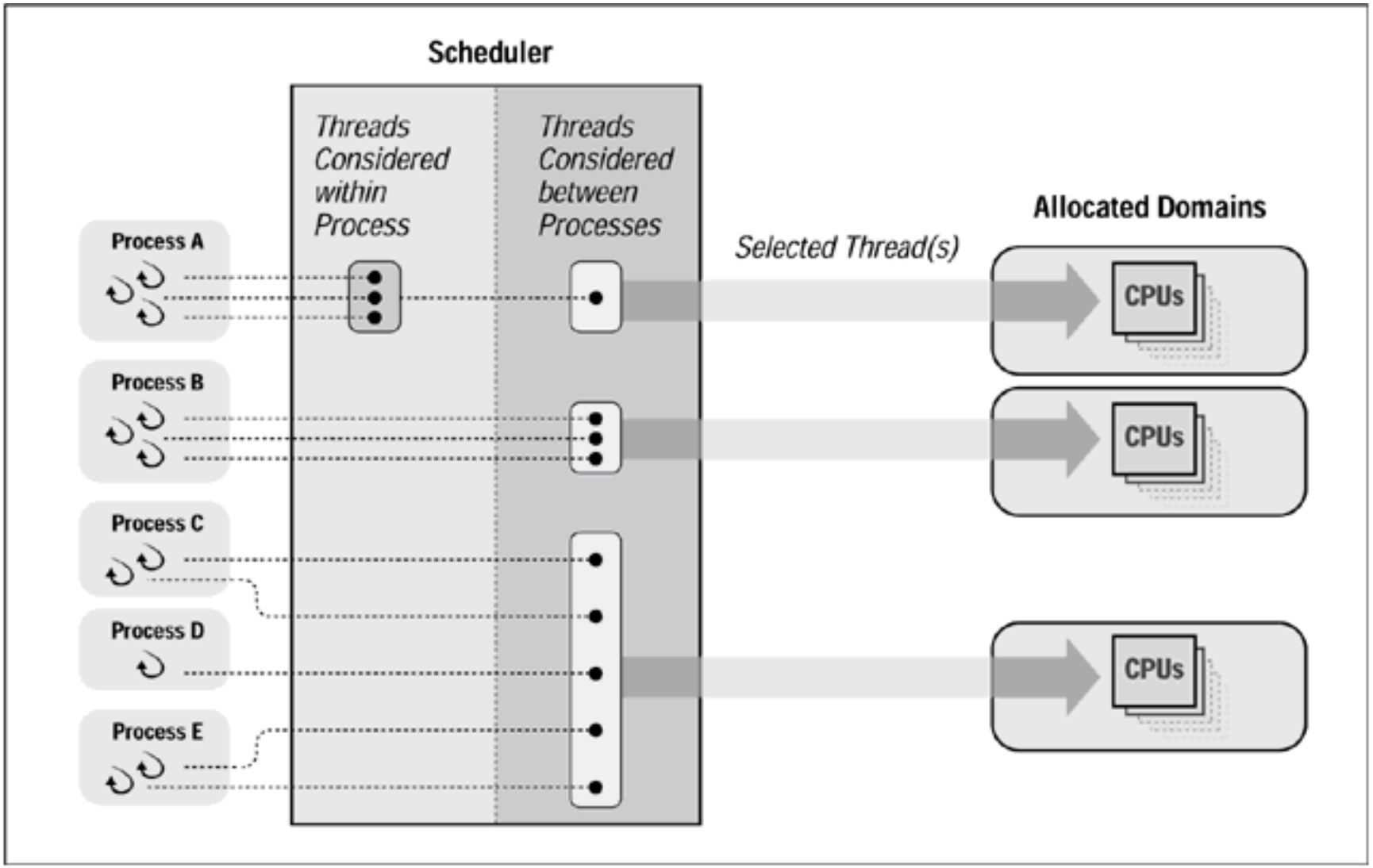
Управление нитями: system scope



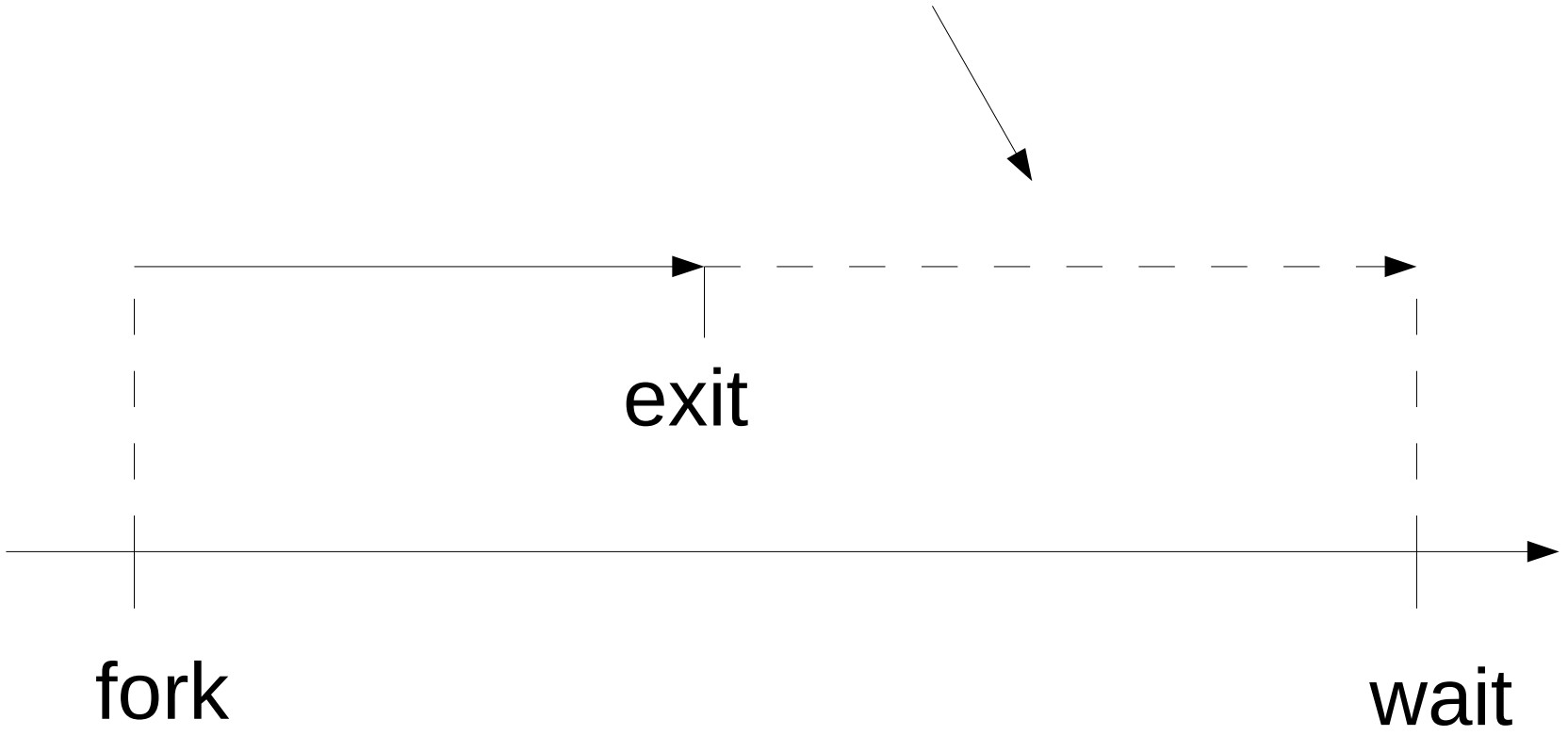
Управление нитями: process scope



Управление нитями: «гибрид»



Зомби



Способы взаимодействия процессов

- Сигналы
- Неименованные каналы
- Именованные каналы
- Пакет IPC
- Сокеты
- Отладка

Передача сигнала

kill (<номер процесса>, <номер сигнала>)

SIGHUP (Hang Up) опускание трубки телефона , заверш. управл. процесс

SIGINT (Interact) Ctrl+C прерывание с клавиатуры

QUIT- выход

ILL – неверная инструкция

FPE – деление на 0

KILL – нельзя обработать процессом

SEGV – нарушение сегментации

PIPE – возникновение проблем в конвейере

Обработка сигнала

Сигналы обрабатываются асинхронно. Обработчик сигнала устанавливается с помощью вызова:

signal (<номер сигнала>,<обработчик>)

SIG_INT SIG_DFL указатель на функцию

(игнорирование) (по умолчанию)

Функция – обработчик:

void <имя> (int <номер сигнала>)

Пример обработки сигнала

```
#include <signal.h>
#include <stdio.h>
void obr (int n)
{ printf ("%d",n);
}
main ()
{ signal(SIGUSR2, obr);
while (1);}
```

Неименованный канал

Применяются только при взаимодействии между _родственными_ процессами.

Создается дескриптор, состоящий из двух элементов:

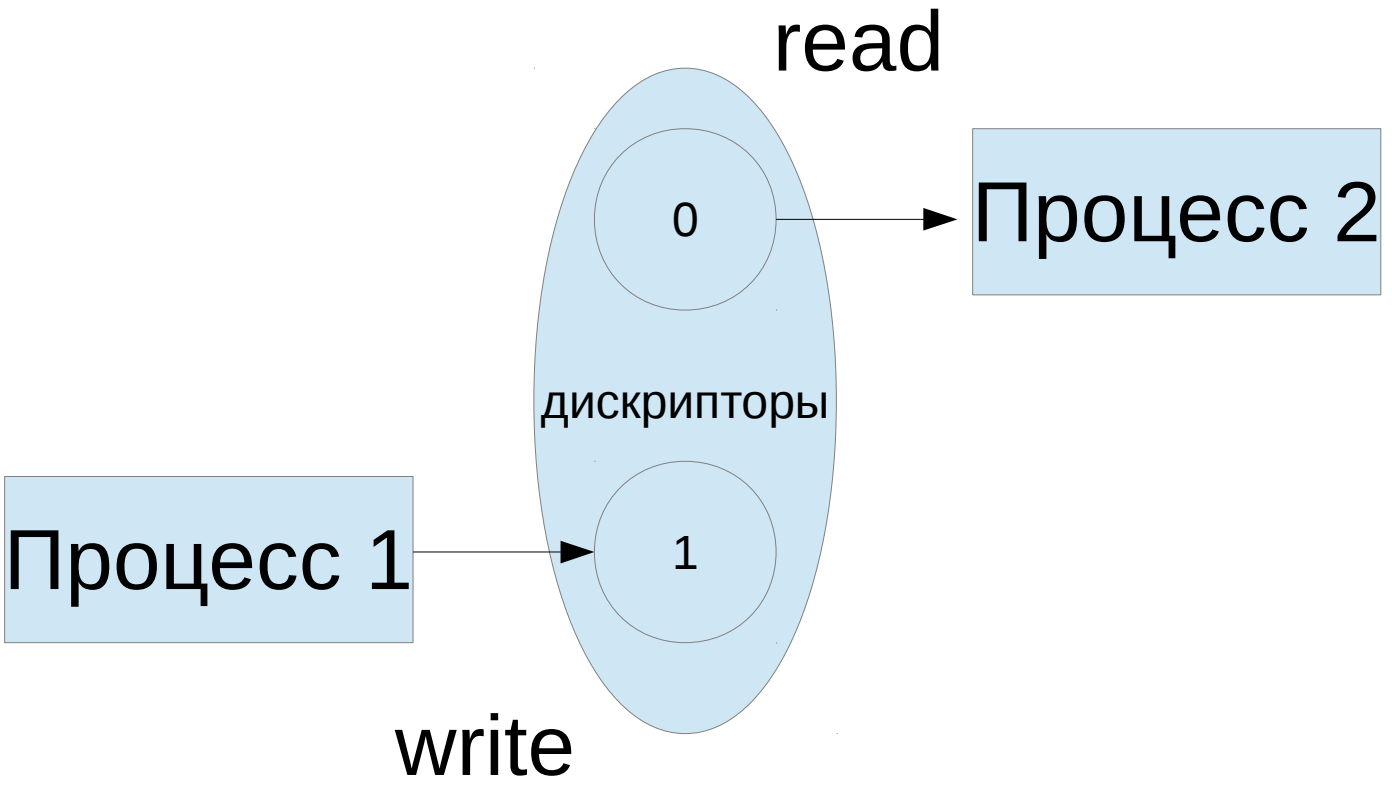
Командой pipe (fdp) он определяется.

pipe - создание неименованного канала

int pipe (fildes)

int fildes [2];

Дескрипторы канала



Именованный канал

При работе с именованным каналом создаем специальный файл:
mknod filename p
, где p - тип файла.

Сокеты

Сокеты - универсальные методы взаимодействия процессов на основе использования многоуровневых сетевых протоколов.

Сокеты предназначены для работы по сети.

Создание сокета

У сокета 3 атрибута:

1) домен

2) тип

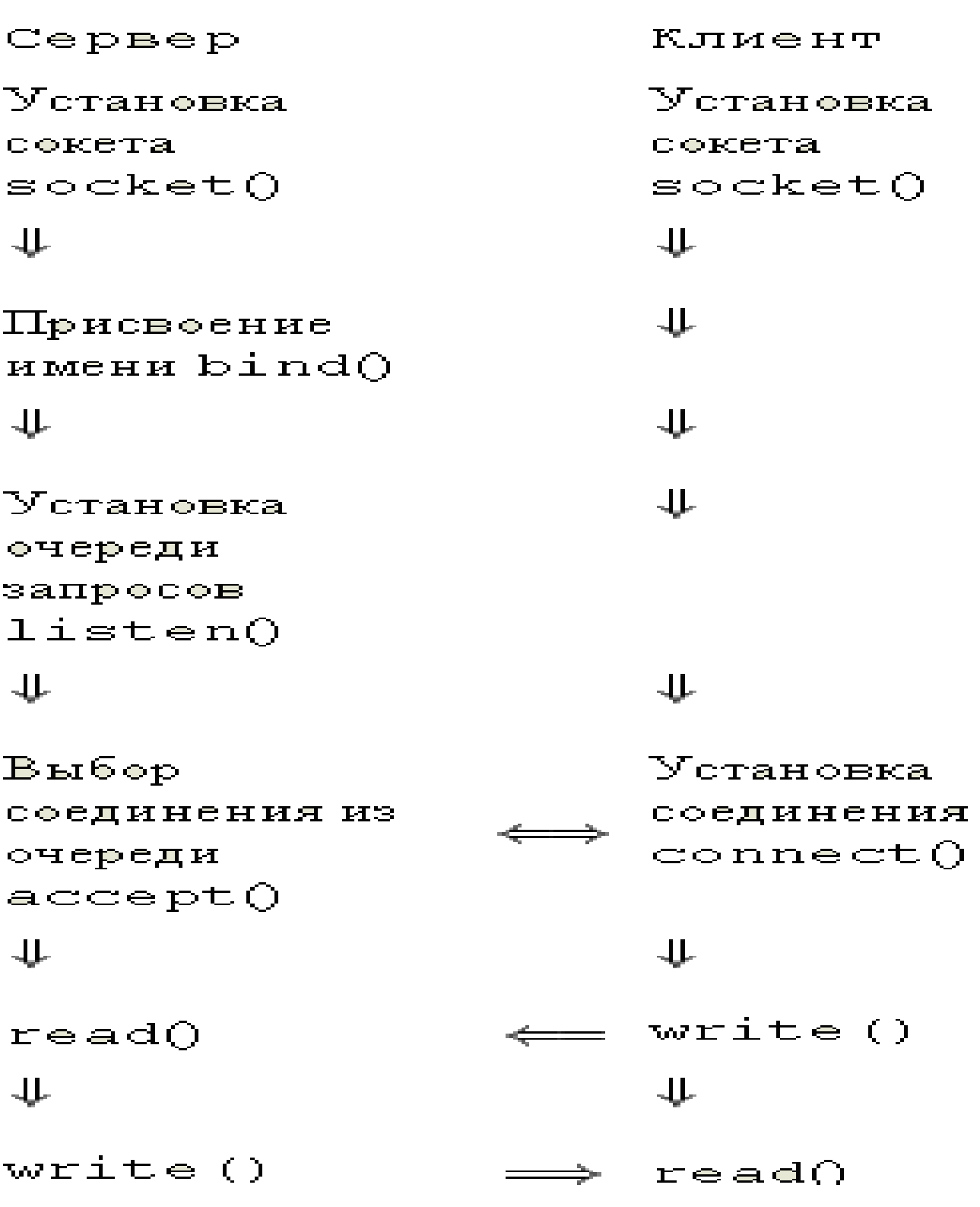
3) протокол

Для создания сокета используется системный вызов socket.

s = socket(domain, type, protocol);

Пример:

s = socket(AF_INET, SOCK_STREAM, 0);


Сервер
Клиент
Установка сокета socket()
Установка сокета socket()
Присвоение имени bind()
Установка очереди запросов listen()
Выбор соединения из очереди accept()
Установка соединения connect()
read()
write ()
write ()
read()