Обзор структуры операционной системы UNIX

Михайленко К. И.

Уфимский государственный авиационный технический университет
Институт механики Уфимского научного центра РАН

8 сентября 2005 г.

Положительные стороны UNIX

- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Положительные стороны UNIX

- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Положительные стороны UNIX

- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Положительные стороны UNIX

- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Положительные стороны UNIX

- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Положительные стороны UNIX

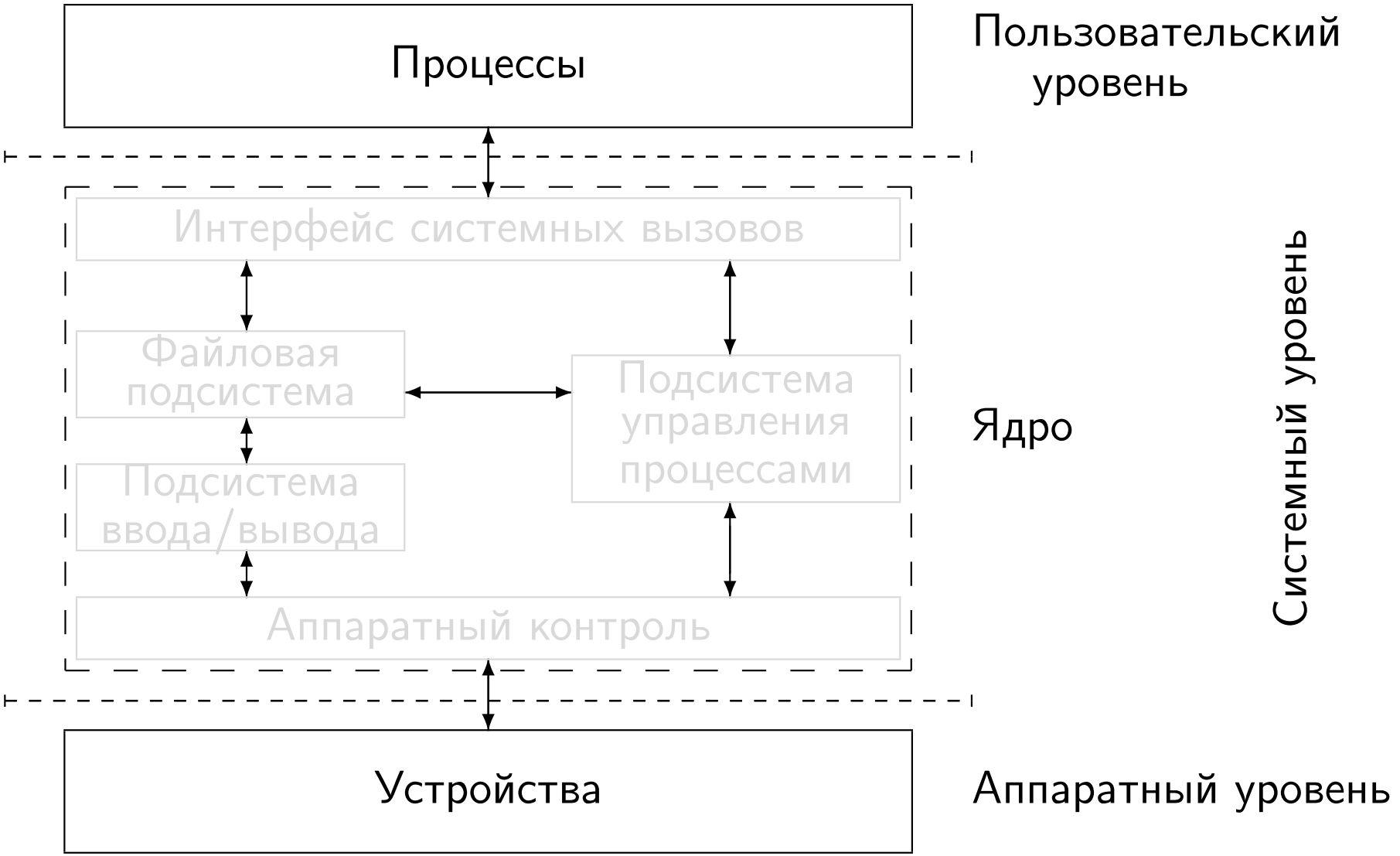
- Переносимость (код системы написан на языке высокого уровня C).
- Полные многозадачность и многопользовательность.
- Стандартизация.
- Простой модульный интерфейс пользователя.
- Единая файловая система.
- Большое количество приложений.

Модель системы

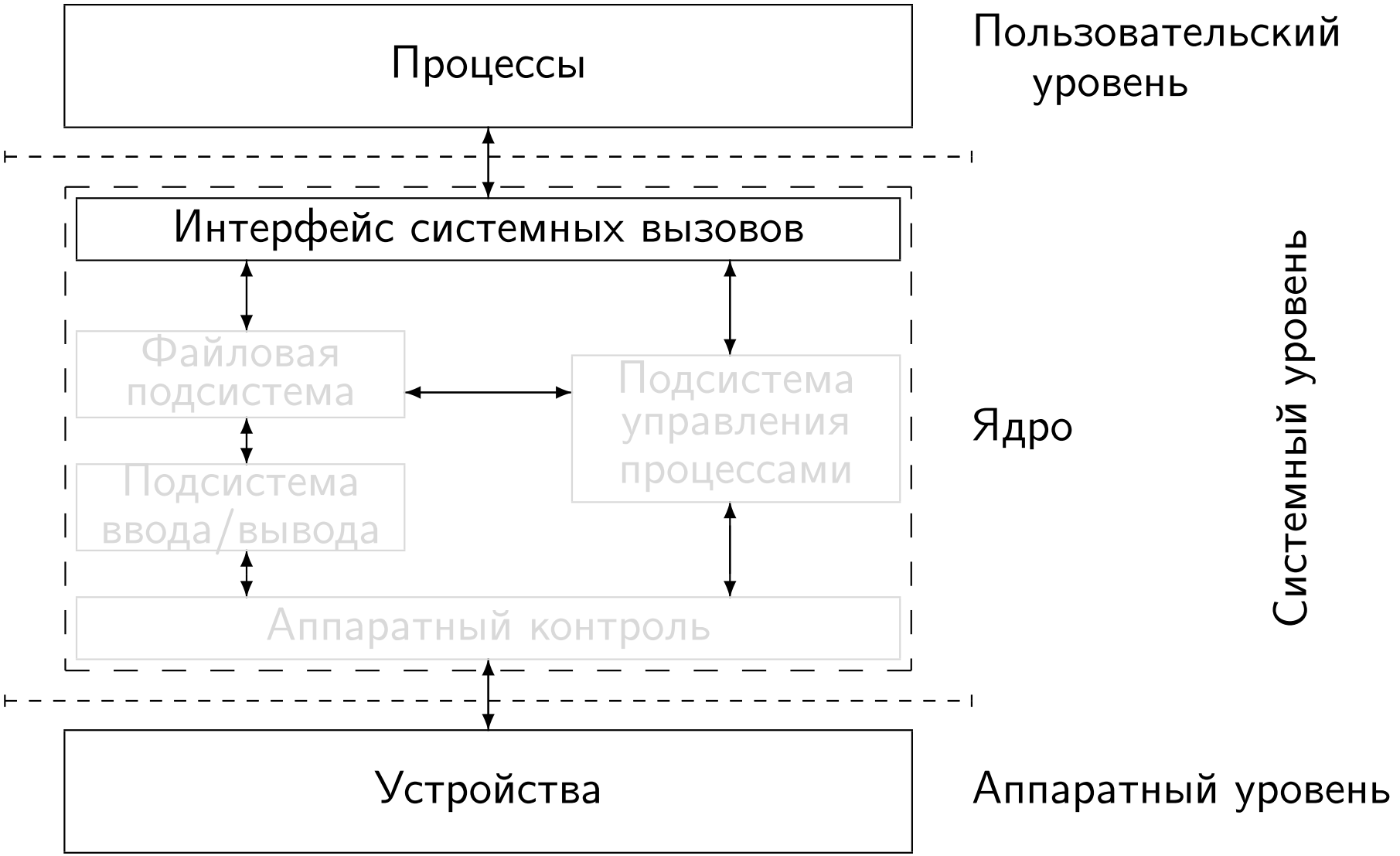
Модель системы

Модель системы

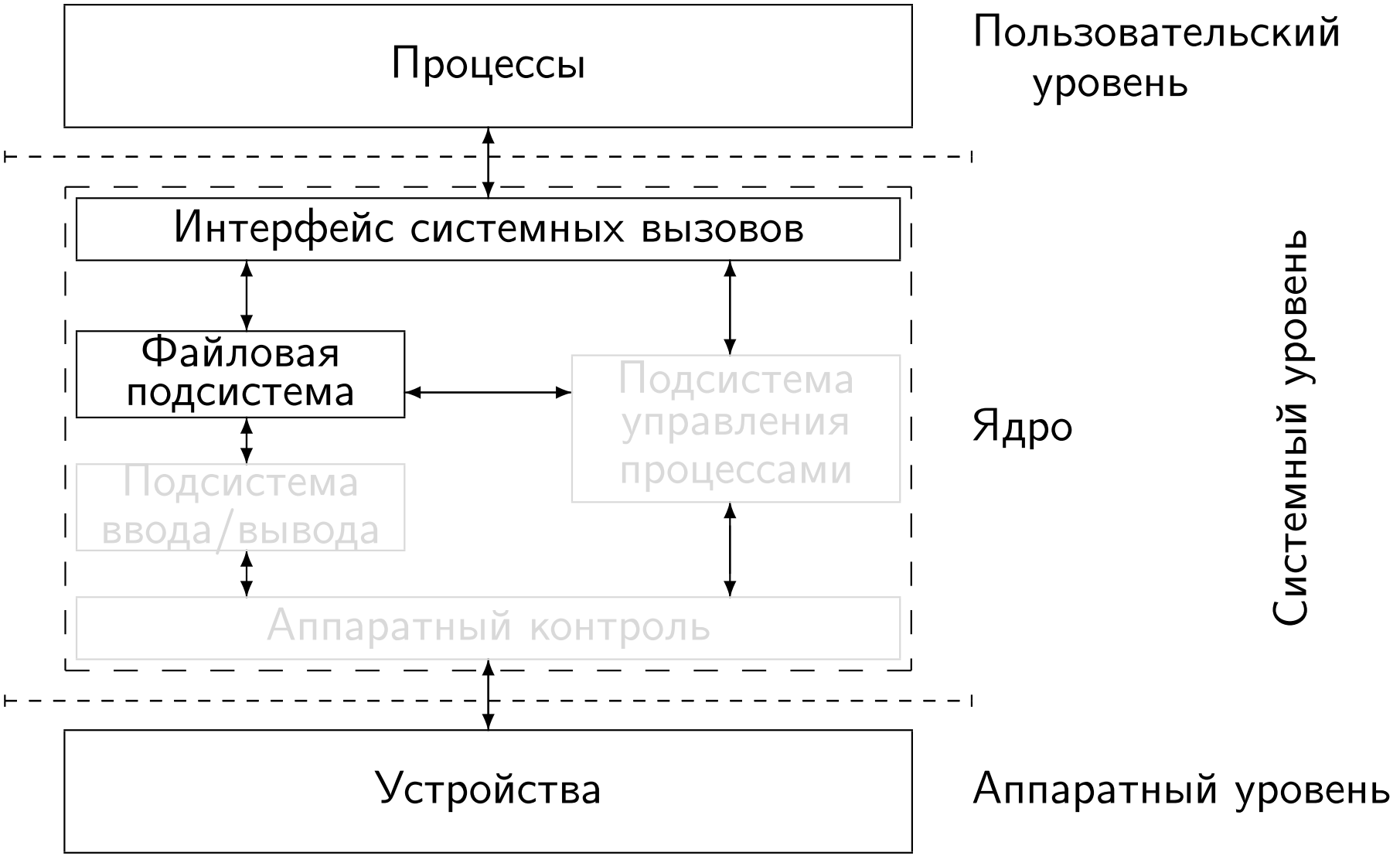
Структура ядра UNIX



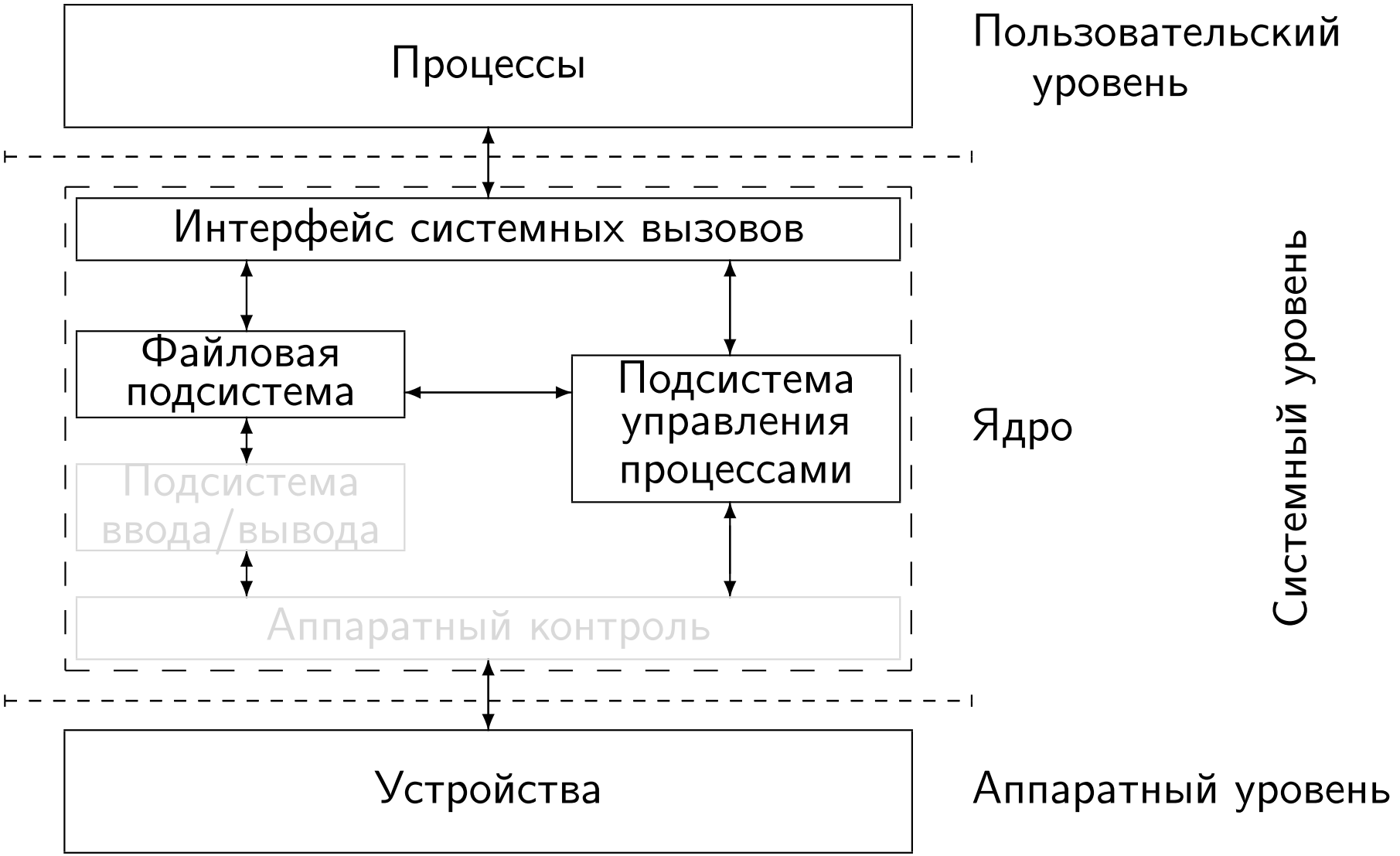
Структура ядра UNIX



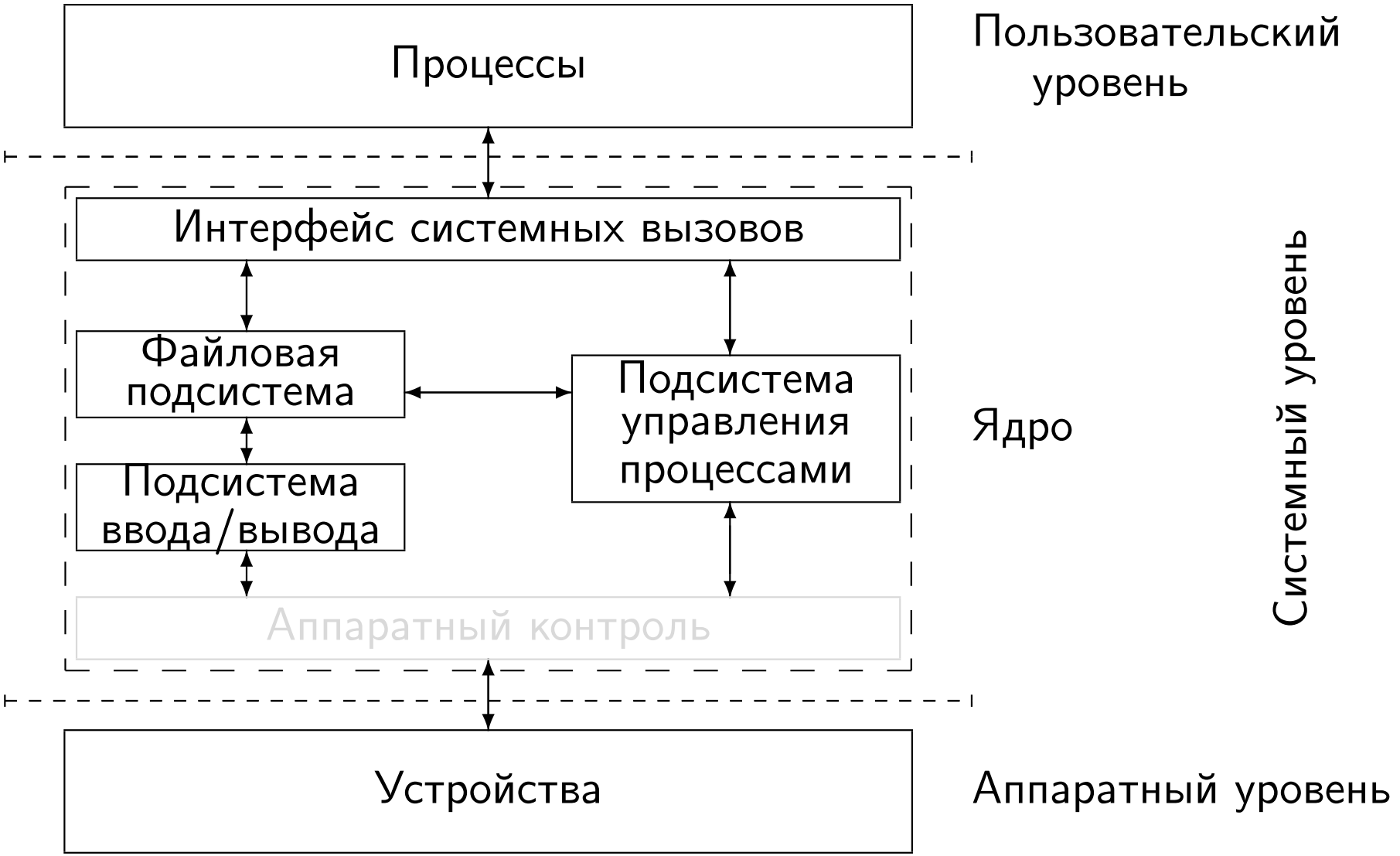
Структура ядра UNIX



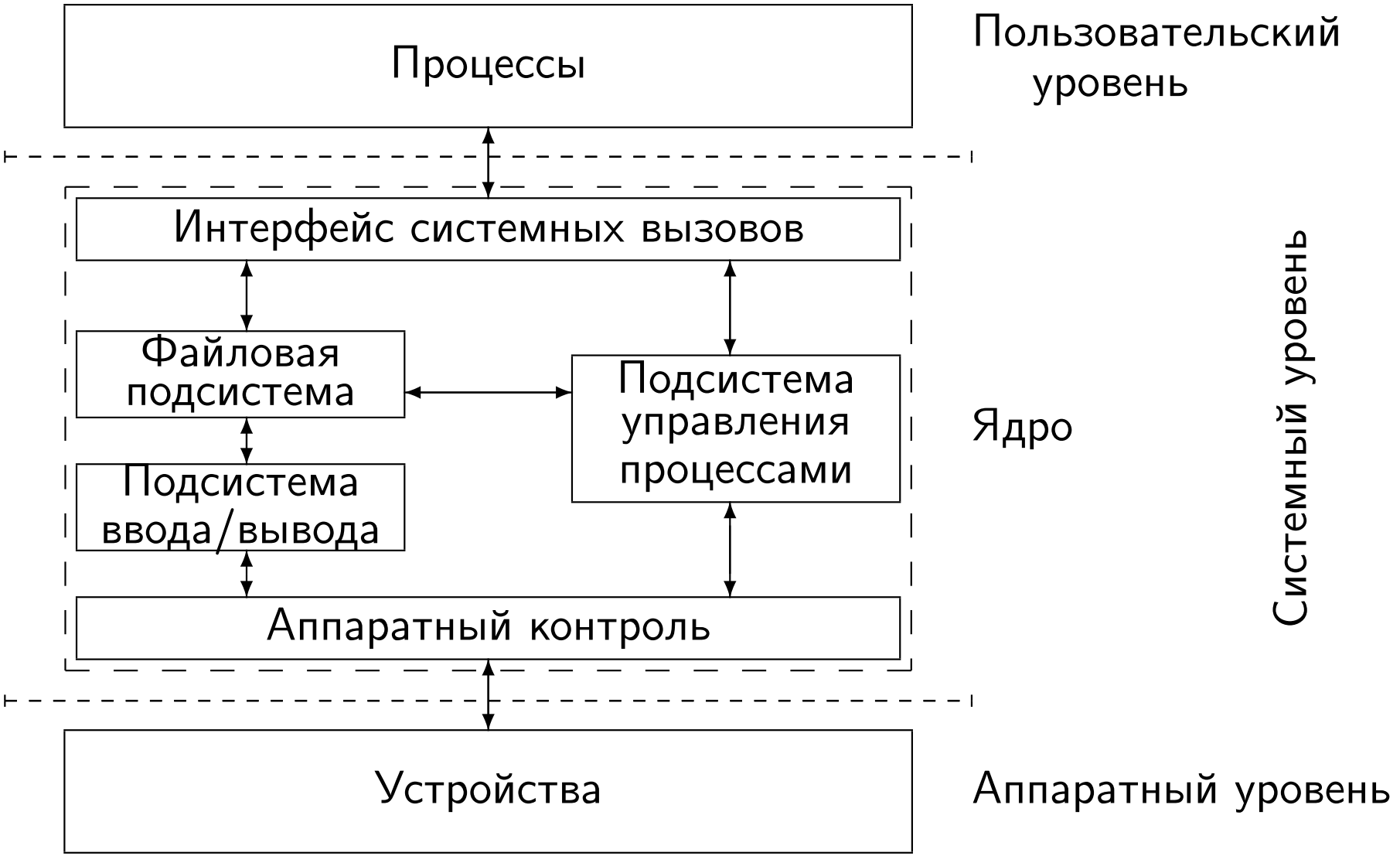
Структура ядра UNIX



Структура ядра UNIX



Структура ядра UNIX



- Файл
- Процесс

Типы файлов

- Regular file (обычный файл)
- Directory (каталог)
- Special device file (специальный файл устройства)
- FIFO (named pipe — именованный канал)
- Link (связь)
- Socket (сокет)

Обычный файл

Каталог

```
$ ls -i1a /home/const/Tests
  1011878   .
    32641   ..
  1011879   first
  1011880   second
    32732   decode.sh
   962981   11-19-RVB.jpg
```

inode — Имя файла

Специальный файл устройства

```
$ ls -l /home/const/Tests
-rw-r--r-- 1 const const 124 Июл 12 23:46 decode.sh
-rw-r--r-- 1 const const   8 Сен  2 16:17 first
-rw-r--r-- 1 const const   8 Сен  2 16:17 second
```

```
ls -l /dev
brw-rw---- 1 root  disk  3,   0 Июл 26 2004 /dev/hda
brw-rw---- 1 root  disk  3,   1 Июл 26 2004 /dev/hda1
brw-rw---- 1 root  disk  3,  10 Июл 26 2004 /dev/hda10
...
crw------- 1 const root 11,   0 Июл 26 2004 /dev/kbd
crw-r----- 1 root  kmem  1,   2 Июл 26 2004 /dev/kmem
crw------- 1 root  root 10, 152 Июл 26 2004 /dev/kpoll
```

Именованный канал (FIFO)

FIFO: First In First Out

```
$ mknod FIFO.file p
  или
$ mkfifo FIFO.file
```

```
int mknod(const char *pathname,
          mode_t mode,
          dev_t dev);
$ man 2 mknod
```

```
$ ls -l ~/const/Tests
prw-r--r--   1 const const     0 Сен  2 18:43 FIFO.file
```

Именованный канал (FIFO)

1. Если процесс считывает число байтов меньшее, чем имеется в канале, — остаток сохраняется для последующих чтений.
2. Если процесс пытается прочитать количество байтов большее, чем имеется в канале, — он получит имеющееся количество байтов. Процесс должен уметь обработать такую ситуацию.
3. Если канал пуст и не был открыт для записи, при чтении возвращается 0 байтов.

Именованный канал (FIFO)

1. Если процесс считывает число байтов меньшее, чем имеется в канале, — остаток сохраняется для последующих чтений.
2. Если процесс пытается прочитать количество байтов большее, чем имеется в канале, — он получит имеющееся количество байтов. Процесс должен уметь обработать такую ситуацию.
3. Если канал пуст и не был открыт для записи, при чтении возвращается 0 байтов.

Именованный канал (FIFO)

1. Если процесс считывает число байтов меньшее, чем имеется в канале, — остаток сохраняется для последующих чтений.
2. Если процесс пытается прочитать количество байтов большее, чем имеется в канале, — он получит имеющееся количество байтов. Процесс должен уметь обработать такую ситуацию.
3. Если канал пуст и не был открыт для записи, при чтении возвращается 0 байтов.

Именованный канал (FIFO)

4. При открытом на запись канале чтение блокируется до получения данных. (Блокировка может быть отменена флагом O_NDELAY).
5. Запись в канал, не превыщающая его ёмкость, атомарна. (Записи нескольких процесов в один канал не перемешиваются).
6. Попытка записи в канал большего числа байтов, чем позволяет канал, вызов блокируется до высвобождения требуемого места. (Атомарность операци при этом не гарантруется).
7. Попытка записи в канал, не открытый на чтение, генерирует сигнал SIGPIPE. Процесс должен уметь его обработать.

Именованный канал (FIFO)

4. При открытом на запись канале чтение блокируется до получения данных. (Блокировка может быть отменена флагом O_NDELAY).
5. Запись в канал, не превыщающая его ёмкость, атомарна. (Записи нескольких процесов в один канал не перемешиваются).
6. Попытка записи в канал большего числа байтов, чем позволяет канал, вызов блокируется до высвобождения требуемого места. (Атомарность операци при этом не гарантруется).
7. Попытка записи в канал, не открытый на чтение, генерирует сигнал SIGPIPE. Процесс должен уметь его обработать.

Именованный канал (FIFO)

4. При открытом на запись канале чтение блокируется до получения данных. (Блокировка может быть отменена флагом O_NDELAY).
5. Запись в канал, не превыщающая его ёмкость, атомарна. (Записи нескольких процесов в один канал не перемешиваются).
6. Попытка записи в канал большего числа байтов, чем позволяет канал, вызов блокируется до высвобождения требуемого места. (Атомарность операци при этом не гарантруется).
7. Попытка записи в канал, не открытый на чтение, генерирует сигнал SIGPIPE. Процесс должен уметь его обработать.

Именованный канал (FIFO)

4. При открытом на запись канале чтение блокируется до получения данных. (Блокировка может быть отменена флагом O_NDELAY).
5. Запись в канал, не превыщающая его ёмкость, атомарна. (Записи нескольких процесов в один канал не перемешиваются).
6. Попытка записи в канал большего числа байтов, чем позволяет канал, вызов блокируется до высвобождения требуемого места. (Атомарность операци при этом не гарантруется).
7. Попытка записи в канал, не открытый на чтение, генерирует сигнал SIGPIPE. Процесс должен уметь его обработать.

Именованный канал (FIFO)

server.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO "FIFO.1"
#define MAXBUFF 80
main() {
    int readfd, n;
    char buff[MAXBUFF];
    if (mknod(FIFO, S_IFIFO | 0666, 0) < 0) {
        printf("Невозможно создать FIFO\n");
        exit(1);
    }
    if ((readfd = open(FIFO, O_RDONLY)) < 0) {
        printf("Невозможно открыть FIFO\n");
        exit(1);
    }
    while ((n = read(readfd, buff, MAXBUFF)) > 0)
        if (write(1, buff, n) != n) {
            printf("Ошибка вывода\n");
            exit(1);
        }
    close(readfd);
    exit(0);
}
```

client.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO "FIFO.1"
main() {
    int writefd, n;
    if ((writefd = open(FIFO, O_WRONLY)) < 0) {
        printf("Невозможно открыть FIFO\n");
        exit(1);
    }
    if (write(writefd, "Hello, World!\n", 15)
            != 15) {
        printf("Ошибка вывода\n");
        exit(1);
    }
    close(writefd);
    if (unlink(FIFO) < 0) {
        printf("Невозможно удалить FIFO\n");
        exit(1);
    }
    exit(0);
}
```

Именованный канал (FIFO)

plot.f

```
      program plot

      integer nframe
      real x

      open(1, file = 'FIFOfile', status = 'unknown')
        write(1,*) 'set xrange [-5:5]'
        write(1,*) 'set yrange [-1:1.2]'
        write(1,*)'set title \'Demonstrating animation Fortran-gnuplot'
     1  ,' via named pipe\''
      close(1)

      nframe=0
 1    call second(x)
      if(15*x .gt. nframe) then
        nframe = nframe + 1
        open(1, file = 'FIFOfile', status = 'unknown')
        write(1,*) 'plot sin(x-',x,')'
        close(1)
      endif
      goto 1

      end
```

terminal

```
$ gnuplot < FIFOfile
```

terminal

```
$ ./plot > FIFOfile
```

Связь — жёсткая связь

```
$ ln second fird
$ ls -il
```

```
$ ls -il
 962981 -rw-r--r--  1 const const 195020 Мар  3  2004 11-19-RVB.jpg
  32732 -rw-r--r--  1 const const    124 Июл 12 23:46 decode.sh
1011881 prw-r--r--  1 const const      0 Сен  2 18:43 FIFO.file
1011880 -rw-r--r--  2 const const      8 Сен  2 16:17 fird
1011879 -rw-r--r--  1 const const      8 Сен  2 16:17 first
1011880 -rw-r--r--  2 const const      8 Сен  2 16:17 second
```

```
$ rm -f fird
$ ls -il
```

```
$ ls -il
 962981 -rw-r--r--  1 const const 195020 Мар  3  2004 11-19-RVB.jpg
  32732 -rw-r--r--  1 const const    124 Июл 12 23:46 decode.sh
1011881 prw-r--r--  1 const const      0 Сен  2 18:43 FIFO.file
1011879 -rw-r--r--  1 const const      8 Сен  2 16:17 first
1011880 -rw-r--r--  1 const const      8 Сен  2 16:17 second
```

Связь — символическая связь

```
$ ln -s second fird
$ ls -il
```

```
$ ls -il
 962981 -rw-r--r--  1 const const 195020 Мар  3  2004 11-19-RVB.jpg
  32732 -rw-r--r--  1 const const    124 Июл 12 23:46 decode.sh
1011881 prw-r--r--  1 const const      0 Сен  2 18:43 FIFO.file
1011894 lrwxrwxrwx  1 const const      6 Сен  5 17:44 fird -> second
1011879 -rw-r--r--  1 const const      8 Сен  2 16:17 first
1011880 -rw-r--r--  2 const const      8 Сен  2 16:17 second
```

Сокет

Используется для межпроцессных связей подобно именованным каналам. (В BSD-like системах межпроцессное взамодействие строится полностью на сокетах).

Отличия от FIFO:

- Предварительное установление соединения.
- Сохранение границ сообщений.
- Поддержка передачи экстренных сообщений.

Основные типы сокетов:

- Datagram socket (сокет датаграм)
- Stream socket (сокет потока)
- Packet socket (сокет пакетов)
- Raw socket (сокет низкого уровня)

Атрибуты файла

Дополнительные атрибуты обычного файла

t	Sticky bit	Сохранение образа выполняемого файла в памяти после завершения.
s	Set UID (SUID)	Установить UID процесса при выполнении.
s	Set GID (SGID)	Установить GID процесса при выполнении.
1	Блокирование	Установить обязательное блокирование файла.

Атрибуты файла

Дополнительные атрибуты каталога

t	Sticky bit	Пользователь может удалять только собственные файлы и файлы для которых он имеет право на запись.
s	Set GID (SGID)	Наследование владельца-группы для создаваемых вложенных каталогов (BSD-like).

Типы процессов

- Системные процессы.
- Демоны (службы).
- Прикладные программы.

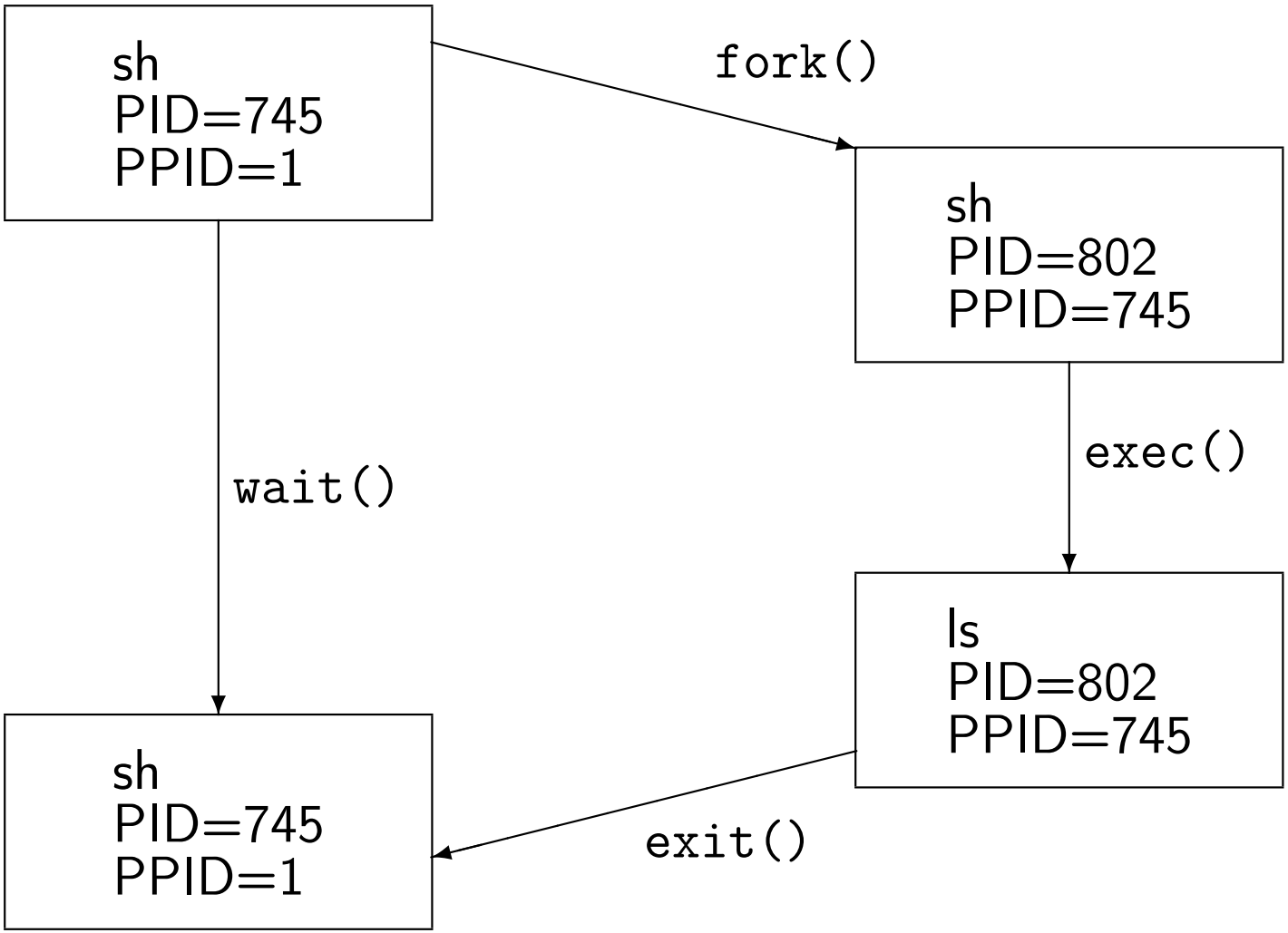
Атрибуты процессов

- Идентификатор процесса (PID — Process ID).
- Идентификатор родительского процесса (PPID — Parent Process ID).
- Приоритет процесса (Nice Number).
- Терминальная линия (TTY).
- Реальный (RID) и эффективный (EUID) идентификаторы пользователя.
- Реальный (RGID) и эффективный (EGID) идентификаторы группы.

Жизнь процесса

Жизнь процесса

```
$ ls
```



Жизнь процесса

Родитель всех процессов: init

fork

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);

$ man 2 fork
$ man 2 clone
$ man 2 vfork
```

Жизнь процесса

exec

```
#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[]);

$ man 3 exec
```

Сигналы

POSIX.1 сигналы

1	SIGHUP	Обнаружен обрыв связи с управляющим терминалом либо завершение управляющего процесса
2	SIGINT	Прерывание с клавиатуры
3	SIGQUIT	Выход с клавиатуры
4	SIGILL	Несуществующая инструкция
6	SIGABRT	Сигнал прерывания, посланный функцией abort
8	SIGFPE	Ошибка операций с плавающей запятой
9	SIGKILL	Сигнал безусловного завершения процесса
11	SIGSEGV	Обращение к запретной области памяти
13	SIGPIPE	Оборванный канал: запись в канал, из которого не читают
15	SIGTERM	Сигнал завершения
18	SIGCONT	Продолжить выполнение, если процесс был остановлен
19	SIGSTOP	Приостановить выполнение процесса

Сигналы

Реакция процесса на полученный сигнал

- Игнорировать сигнал
  (Сигналы SIGSTOP и SIGKILL не могут быть проигнорированы)
- Произвести действие по умолчанию:
  - прекратить выполнение процесса;
  - игнорировать сигнал;
  - прекратить выполнение процесса и записать дамп памяти;
  - приостановить выполнение процесса.
- Перехватить и самостоятельно обработать сигнал
  (Сигналы SIGSTOP и SIGKILL не могут быть перехвачены)

```
$ man 7 signal
```

Сигналы

Отправка сигнала из командной строки

```
$ kill -signum PID
$ killall -signum name
```

Отправка сигнала зерез вызов функции

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
int killpg(int pgrp, int sig);
```

Сигналы

После получения сигнала SIGKILL продолжают существовать:

- процессы-зомби;
- процессы, обращающиеся к недоступным ресурсам NFS (в этом случае могут помочь сигналы SIGINT или SIGQUIT);
- процессы,ожидающие завершения операции с устройством ввода-вывода.