Архитектура ОС UNIX

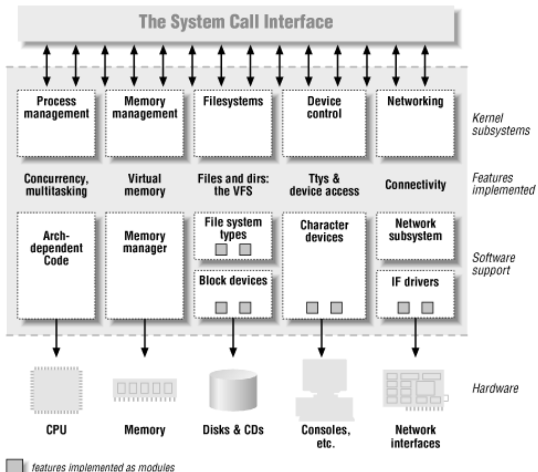
Санкт-Петербургский государственный политехнический университет

15 декабря 2011

Драйвера устройств

Драйвера устройств играют специальную роль в ядре Linux. Они представляют собой «черные ящики», которые обрабатывают определенную часть запросов к аппаратной части Linux ядра через хорошо организованный внутренний программный интерфейс.
Программный интерфейс реализован таким образом, что драйвера могут быть построены отдельно от ядра, и связаны с ядром в запущенное ядро по мере надобности. Это свойство драйверов в Linux называется модульностью и сильно упрощает написание и управление драйверами.

Структурная схема ядра



Классы устройств и модулей

В Unix, устройства подразделяются на три класса: символьные, блочные и сетевые устройства.

Символьные устройства

Символьным называется устройство, которое может быть представлено потоком байт.
Такие драйвера реализуют, системные вызовы open(), close(), read() и write().
Текстовая консоль (/dev/console) и последовательные порты (/dev/ttyS0) представляют собой примеры символьных устройств.

Блочные устройства

На блочных устройствах, как правило, размещаются файловые системы. В большинстве Unix систем, блочные устройства могут быть представлены только как множество блоков. Разница заключается в том, что при обращении к блочному устройству передается блок данных, а не один байт.

Сетевые интерфейсы

На сетевой интерфейс, который управляется сетевой подсистемой ядра, наложены функции приема и передачи пакетов данных.
Не будучи потокоориетированным, сетевой интерфейс не может быть легко представлен через файловый интерфейс устройства, наподобии /dev/tty1. В ОС Unix, доступ к сетевому интерфейсу осуществляется через уникальное имя, такое как eth0, которое не является элементом файловой системы.

Модули ядра

```
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>
MODULE_LICENSE("GPL");
static int __init my_init(void)
{
        printk("Hello␣world\n");

        return 0;
};
static void __exit my_cleanup(void)
{
        printk("Good␣bye\n");
};
module_init(my_init);
module_exit(my_cleanup);
```

Компиляция

Makefile:

```
obj-m := hello-2.6.o
```

Сборка модуля:
make -C /usr/src/linux SUBDIRS=‘pwd‘ modules

Связь модулей с ядром

Модули, при их загрузке, линкуются в ядро, поэтому все, внешние функции должны быть объявлены в заголовочных файлах ядра и присутствовать в ядре.
Исходники модулей никогда не должны включать обычные заголовочные файлы из библиотек пользовательского пространства.

Связь модулей с ядром

Весь интерфейс ядра, описан в заголовочных файлах, находящихся в каталогах include/linux и include/asm внутри исходников ядра, обычно находящихся в /usr/src/linux-x.y.z.

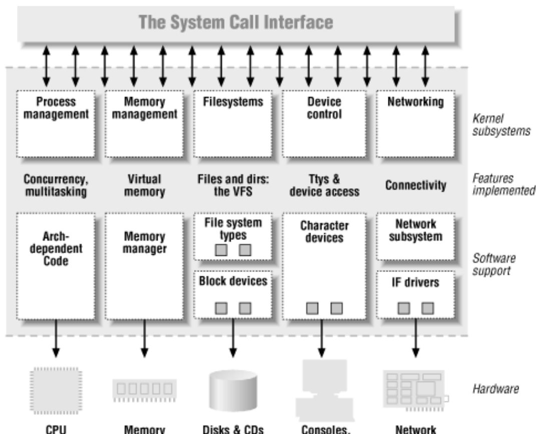
Информация о текущем процессе

Код ядра может опознать вызвавший его процесс обратившись к глобальному указателю который указывает на структуру struct task_struct, определенную.
Указатель current указывает на текущий исполняющийся пользовательский процесс. При исполнении таких системных вызовов как open() или close(), обязательно существует процесс вызвавший их.

```
printk("The process is \"%s\" (pid %i)\n", current
    ->comm, current->pid);
```

Таблица символов ядра

При загрузке модуля, символы экспортируемые модулем становятся частью таблицы символов ядра, и вы сможете просмотреть из в /proc/modules.
Новые модули могут использовать символы экспортируемые вашим модулем.



Инициализация и завершение модулей

Функция init_module() регистрирует функциональные компоненты модуля в ядре.

Если при инициализации модуля возникает какого-либо рода ошибка, то программист должен отменить уже совершенную инициализацию перед остановом загрузки модуля. Ошибка может возникнуть, например, из-за недостатка памяти.

Инициализация и завершение модулей

```
int init_module(void)
 {
 int err;
  /* registration takes a pointer and a name */
  err = register_this(ptr1, "skull");
  if (err) goto fail_this;
  err = register_that(ptr2, "skull");
  if (err) goto fail_that;
  err = register_those(ptr3, "skull");
  if (err) goto fail_those;
  return 0; /* success */
  fail_those: unregister_that(ptr2, "skull");
  fail_that: unregister_this(ptr1, "skull");
  fail_this: return err; /* propagate the error */
}
```

Счетчик использования модуля

Система содержит счетчик использования каждого модуля для того, чтобы определить возможность безопасной выгрузки модуля.
При выгрузке модуля выполняется системный вызов delete_module(), который либо выполняет вызов функции cleanup_module() выгружаемого модуля в случае, если его счетчик использования равен нулю, либо прекращает работу с ошибкой.