Лекция 1.1 Обзор архитектуры Linux

Разработал: Максимов А.Н.

Версия 1.16. 02.2025

RISoft

Содержание

- Компоненты Linux
- Архитектура ядра и типы драйверов

История linux

Линуз Торвальдс (Linus Torvalds) начал работать на первой версией юниксоподобной операционной системы в апреле 1991.
Версия Linux 0.01 была выпущена в сентябре 1991 года и содержала 10,239 строк кода.
1992 март , Linux 0.95 первая версия Linux на которой выполнялась среда X-windows

1994 март Linux 1.0.0 176,250 строк кода
1995 март Linux 1.2.0 310,950 строк кода
1996 9 мая, пингвин выбран симовлом Linux.
1999 январь Linux 2.2.0 1,800,847 строк кода
2001 январь Linux 2.4.0 3,377,902 строк кода
2003 декабрь Linux 2.6.0 5,929,913 строк кода

2009 март Linux 2.6.29 11,010,647 строк кода
2011 февраль Linux 3.0
2012 январь Linux 3.3 > 15 млн. строк кода
2014декабрь Linux 3.17.4
2017 все суперкомпьютеры из Top500 работают на Linux.
2019: Linux 5.0
2022: Linux 6.0
2025 февраль Linux 6.13.4 текущая версия
В разработке релиза ядра участвует более 1000 разработчиков как независимых, так и представляющих компании

Свойства Linux

Перечень возможностей, реализуемые Linux:

- поддержка вытесняющей многозадачности в режиме пользователя и в режиме ядра;
- поддержка виртуальной памяти;
- поддержка разделяемых библиотек;
- поддержка загрузки модулей по запросу;
- гибкое управление памятью;
- поддержка множества коммуникационных протоколов;
- возможность использовать потоки (threads);
- поддержка многопроцессорности;
- и т.д.

Какой Linux взять

- Бинарный дистрибутив (AstraDebian, Ubuntu, fedora и др.)

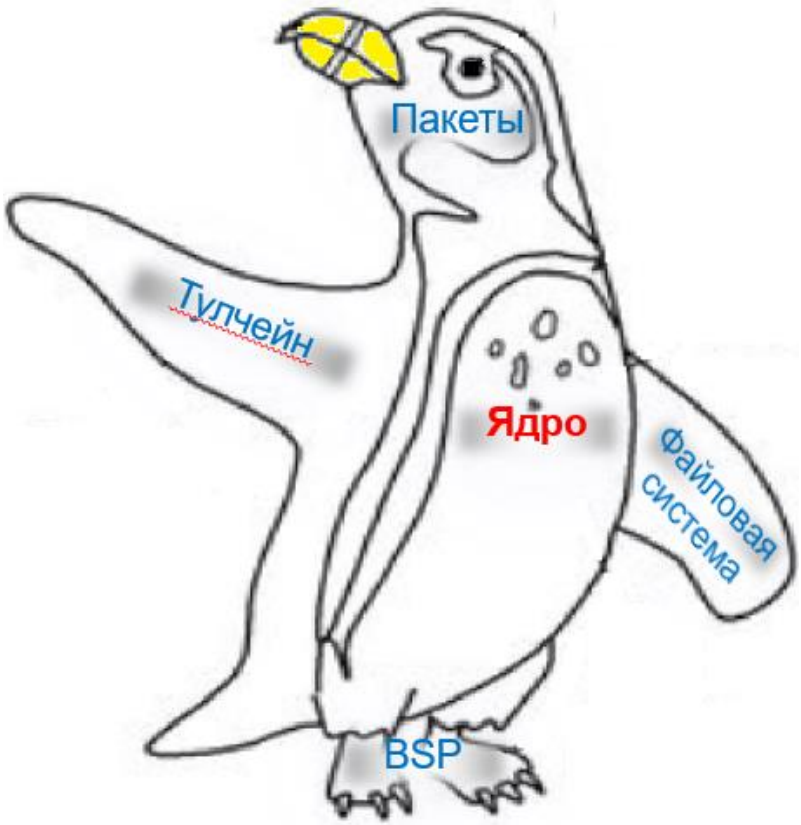
- Сборочную систему (yocto, buildroot, ptxdist и др.)

- Собрать все самому (см. http://www.linuxfromscratch.org/)

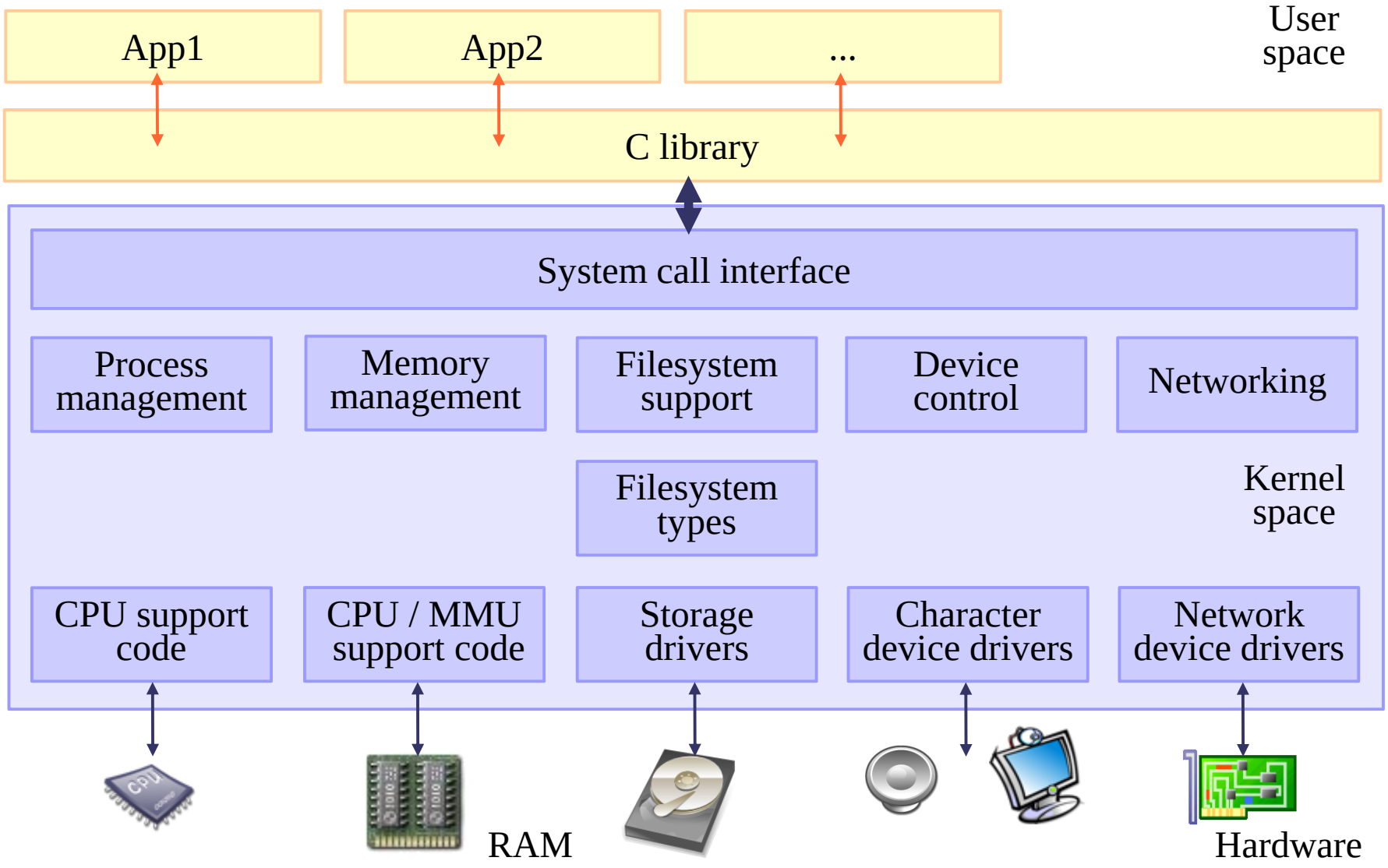
RISoft

Что включает Linux

- Тулчейн ☺ (binutils, gcc и т.д.)
- Ядро linux
- Файловая система
- Набора пакетов



Часть 1. Архитектура ядра



Архитектура ОС. Особенности linux

Linux — операционная система с **монолитным ядром**. Драйверы и модули ядра исполняются в едином адресном пространстве в 0 кольце защиты с полным доступом к аппаратному обеспечению. Пользовательские задачи работают в кольце 3.

Недостатки монолитного ядра обходятся в linux с использванием механизма загружаемых драйверов

Пример других операционных системы с монолитным ядром (LynxOS, OS-9)

Многие современные операционные системы используют концепцию **микроядра** (KasperskyOS, Minix, VxWorks, GNU Hurd)

Подсистемы ядра linux

В ядре linux может быть выделено несколько подсистем:

- Подсистема управления процессами

Планировщик

Механизмы межзадачного обмена

- Подсистема управления файлами

Виртуальная файловая система (VFS)

Модули файловых систем (ext3, vfat и т.д)

- Подсистема управления памятью.
- Сетевая подсистема.
- Подсистема ввода-вывода
- Другие подсистемы (video4linux, аудио подсистема, ….)

Подсистема управления процессами. Планировщик.

Планировщик – управляет доступом процессов к CPU. В составе планировщика можно выделить следующие компоненты:

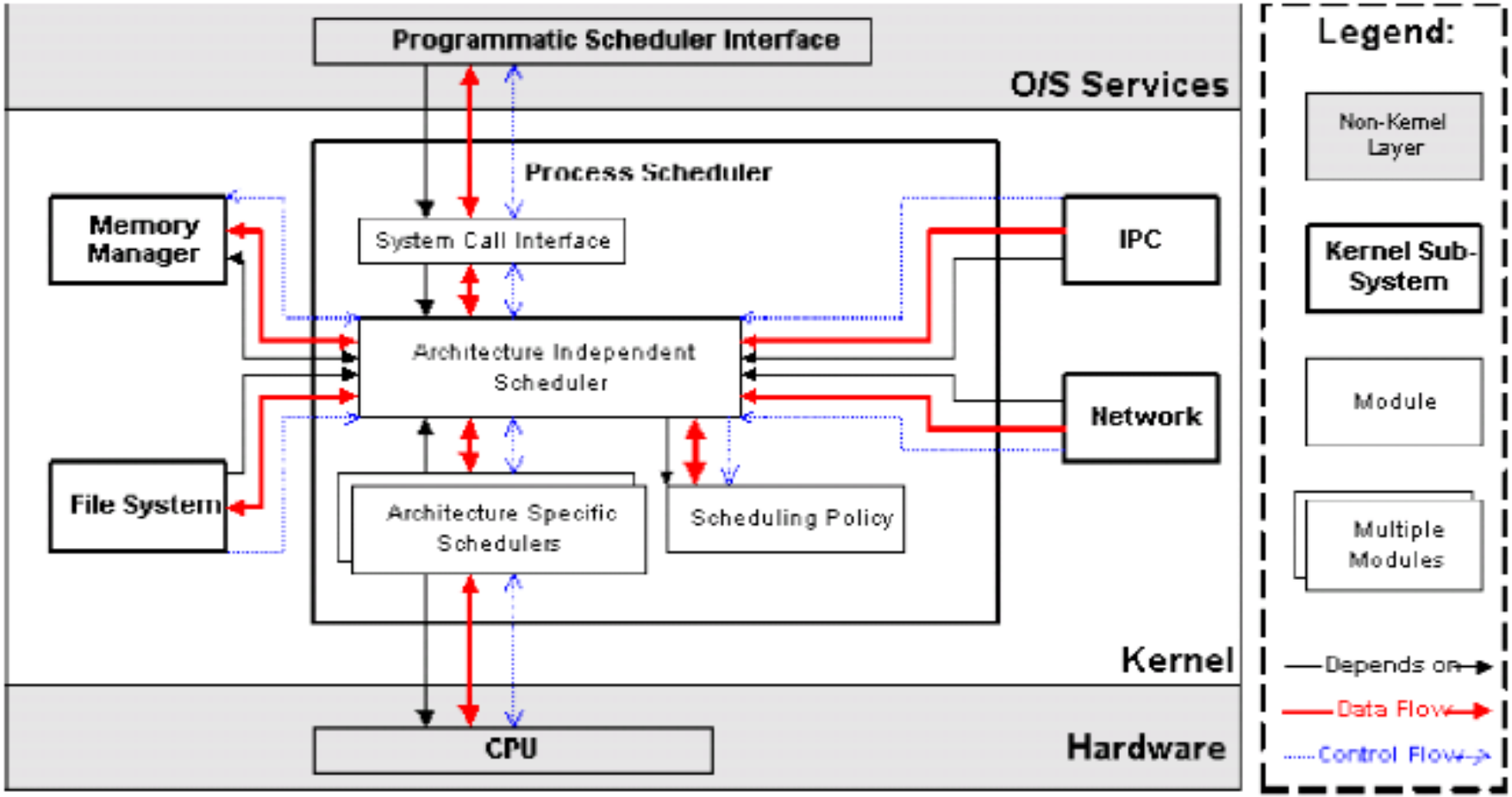
Политика диспетчеризации – модуль, определяющий какой процесс получит доступ к CPU

Аппаратно зависимый компонент диспетчера – инкапсулирует детали работы с конкретной платформой, взаимодействует с CPU для запуска и останова процессов

Аппаратно независимый компонент диспетчера – получает от модуля политики идентификатор следующего процесса для запуска, обращается к аппаратно зависимому модулю для запуска процесса. Дополнительно взаимодействует с подсистемой управления памятью.

Интерфейс системных вызовов. Предоставляет доступ к сервисам управления процессами пользовательским задачам.

Подсистема управления процессами. Планировщик.



Luca Pizzamiglio. The linux kernel: architecture and programming.

Процесс в Linux

Процессом называется программа, находящаяся в процессе выполнения совместно со своими данными и структурами ОС, необходимыми для управления для ей.

Представление процесса - tusk_struct

Вся необходимая информация о процессе содрежится в структуре task_struct

```
struct task_struct {
    volatile long state;
    void *stack;
    unsigned int flags;
    int prio, static_prio, normal_prio;
    unsigned int rt_priority;
    const struct sched_class *sched_class;
    struct list_head tasks;
    struct mm_struct *mm, *active_mm;
    pid_t pid;
    pid_t tgid;
    struct task_struct *real_parent;
    struct list_head children;                  /* список потомков процесса */
    struct list_head sibling;                   /* ссылка на одранговые процессов в списке предке */
    struct task_struct *group_leader;          /* лидер группы потоков */
    char comm[TASK_COMM_LEN];          /* executable name excluding path  - access with [gs]et_task_comm
       (which lock  it with task_lock()) */
    struct thread_struct thread;
    struct files_struct *files;
   ...
};
```

Структура task_struct объявлена в /linux/include/linux/sched.h

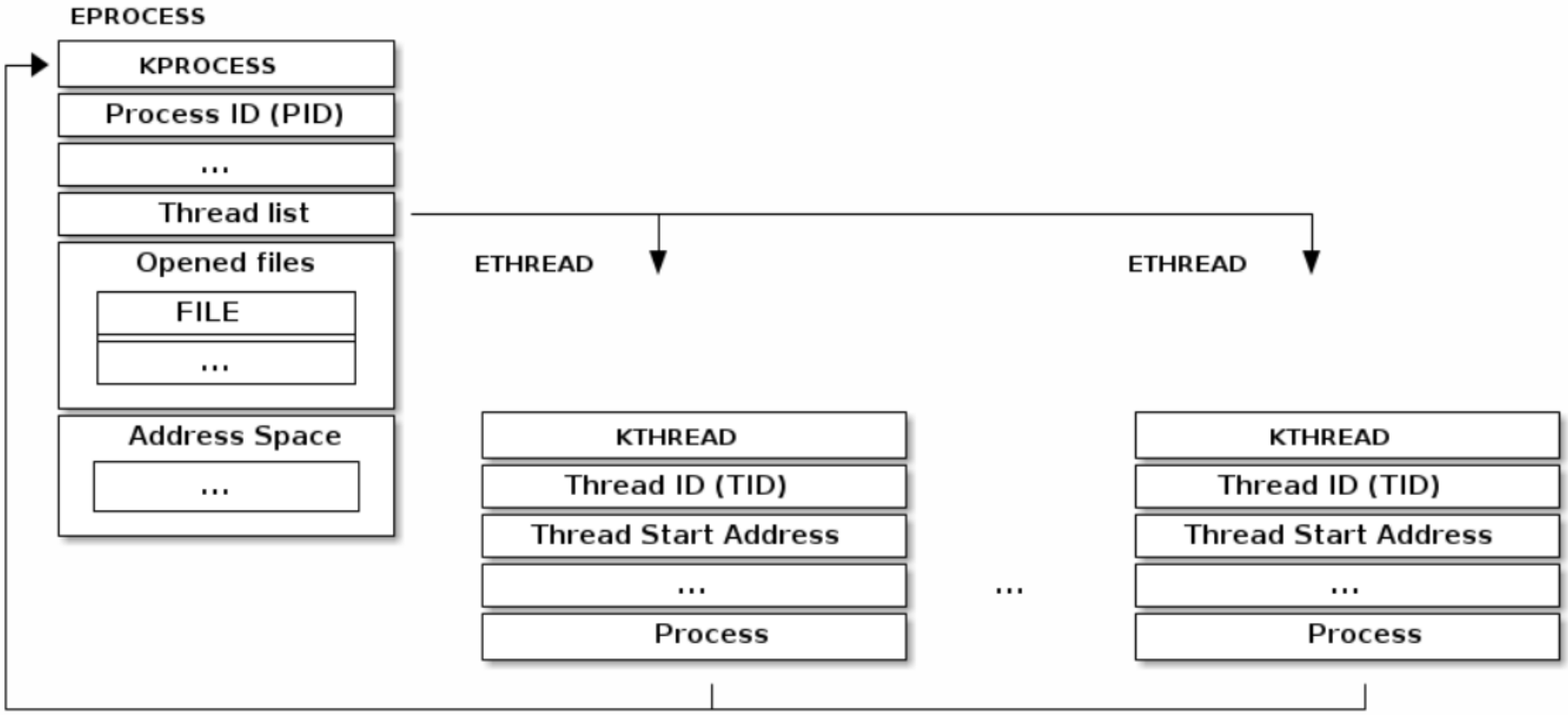
Состояния процесса

Активен (TASK_RUNNING)	Запрашивает процессорное время. Выполняется или ждет в очереди на выполнение.
Ждет (TASK_INTERRUPTIBLE)	Ожидает определенный интервал времени или внешнего события или сигнала.
Спит(TASK_UNINTERRUPTIBLE)	Ожидает определенный интервал времени или внешнего события или сигнал. Приход сигнала не пробуждает процесс .
Остановлен (TASK_STOPPED)	Выполнение процесса остановлено. Процесс поподает в это состояние, когда получает сигнал SIGSTOP, SIGTSTP, SIGTTIN или SIGTTOU .
Зомби(TASK_ZOMBIE)	Процесс завершился, но родительский процесс не еще не вызвал wait

Дополнительные состояния начиная

Авинеш Кумар. TASK_KILLABLE: Новое состояние процесса в Linux. http://www.ibm.com/developerworks/linux/library/l-task-killable/index.html?S_TACT=105AGX99&S_CMP=CP

Потоки и процессы



https://linux-kernel-labs.github.io/refs/pull/273/merge/lectures/processes.html

Изменение состояния процесса

Для изменения состояния процесса могут быть использованы следующие макросы:

set_task_state(task, state) - для указанной задачи

set_current_state(state) - для текущей задачи (часто используется)

Для однопроцессорных систем они эквивалентны:

task->state = state;

Для многопроцессорных систем макрос включет *memory barrier* для обеспечения корректного порядка операций

Планировщик. Критерии качества планировщика

Для систем общего назначения критериями качества планировщика могут быть:

- справедливый доступ задач к процессора (распределение процессорного времени);
- оптимальная пропускная способность (количество выполненных задач за единицу времени);
- оптимальное время отклика;
- оптимальная производительность;
- максимальную загрузку процессоров

Подсистема управления памятью.

Подсистема управления памятью позволяет задачам получать доступ к памяти. Реализует механизм виртуальной памяти.

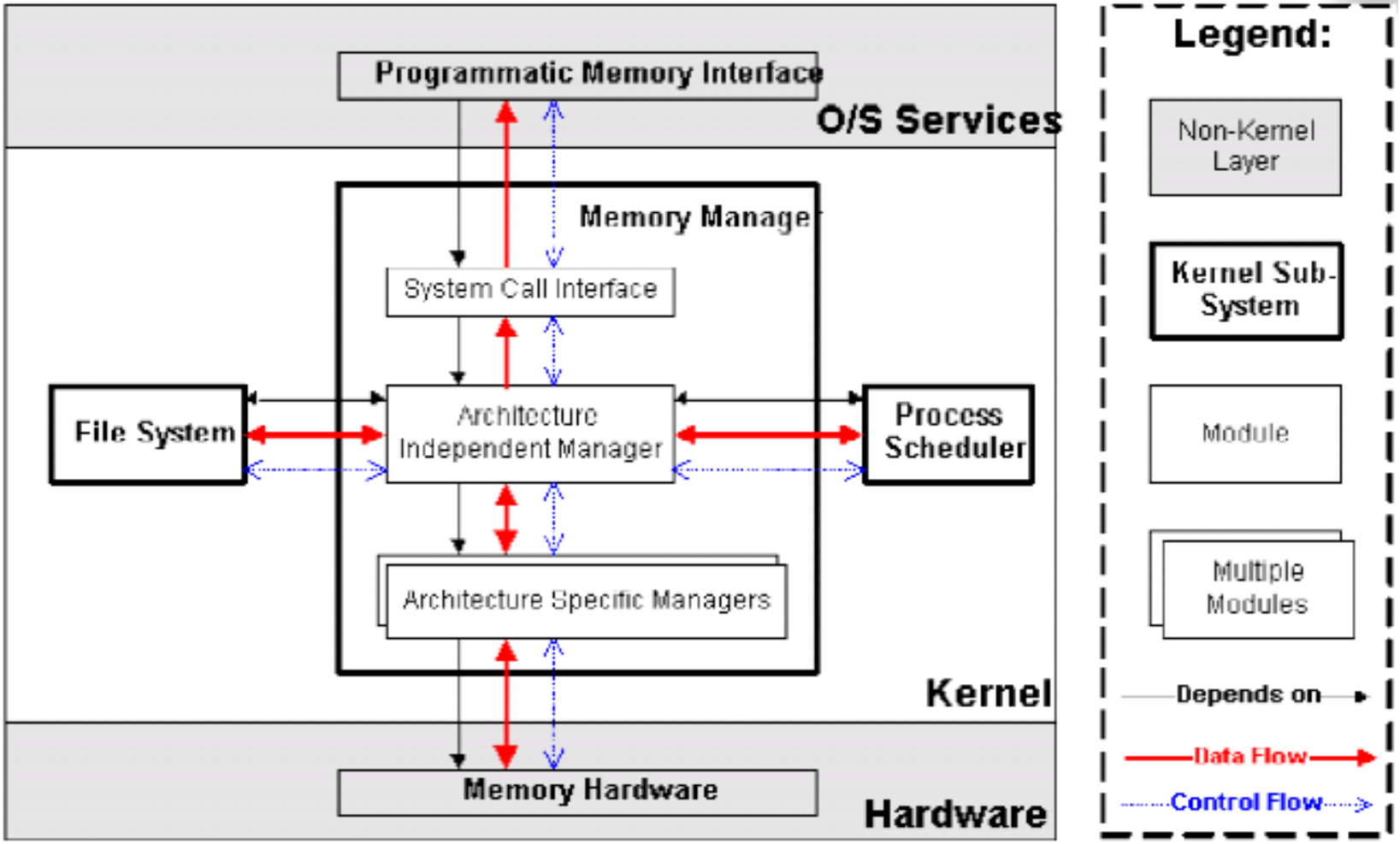
Подсистема управления памятью состоит из следующих модулей:

Архитектурно зависимый модуль – инкапсулирует работу с аппаратным обеспечением MMU

Архитектурно независимый менеджер памяти – осуществляет выделение памяти для процессов, реализует механизм свопирования виртуальной памяти. При возникновении ошибки доступа к странице определяет какую страницу извлекать.

Интерфейс системных вызовов – обеспечивает контролируемый доступ пользовательских процессов к сервисам подсистемы управления памятью. Позволяет выделять и освобождать память и выполнять отображение файлов в память.

Подсистема управления памятью. Структурная схема



Luca Pizzamiglio. The linux kernel: architecture and programming.

RISoft

Виртуальная файловая система

Осуществляет унифицированное представление данных хранящихся на устройствах хранения данных.

Позволяет монтировать любую файловую систему на любом устройстве.

Абстрагирует логические файловые системы от физических устройств и физические устройства от логических файловых систем.

Отвечает за загрузку выполняемых программ. Позволяет linux поддерживать различные форматы исполняемых файлов.

Виртуальная файловая система. Ключевые компоненты.

Драйверы устройств. Для поддерживаемых аппаратных контроллеров.

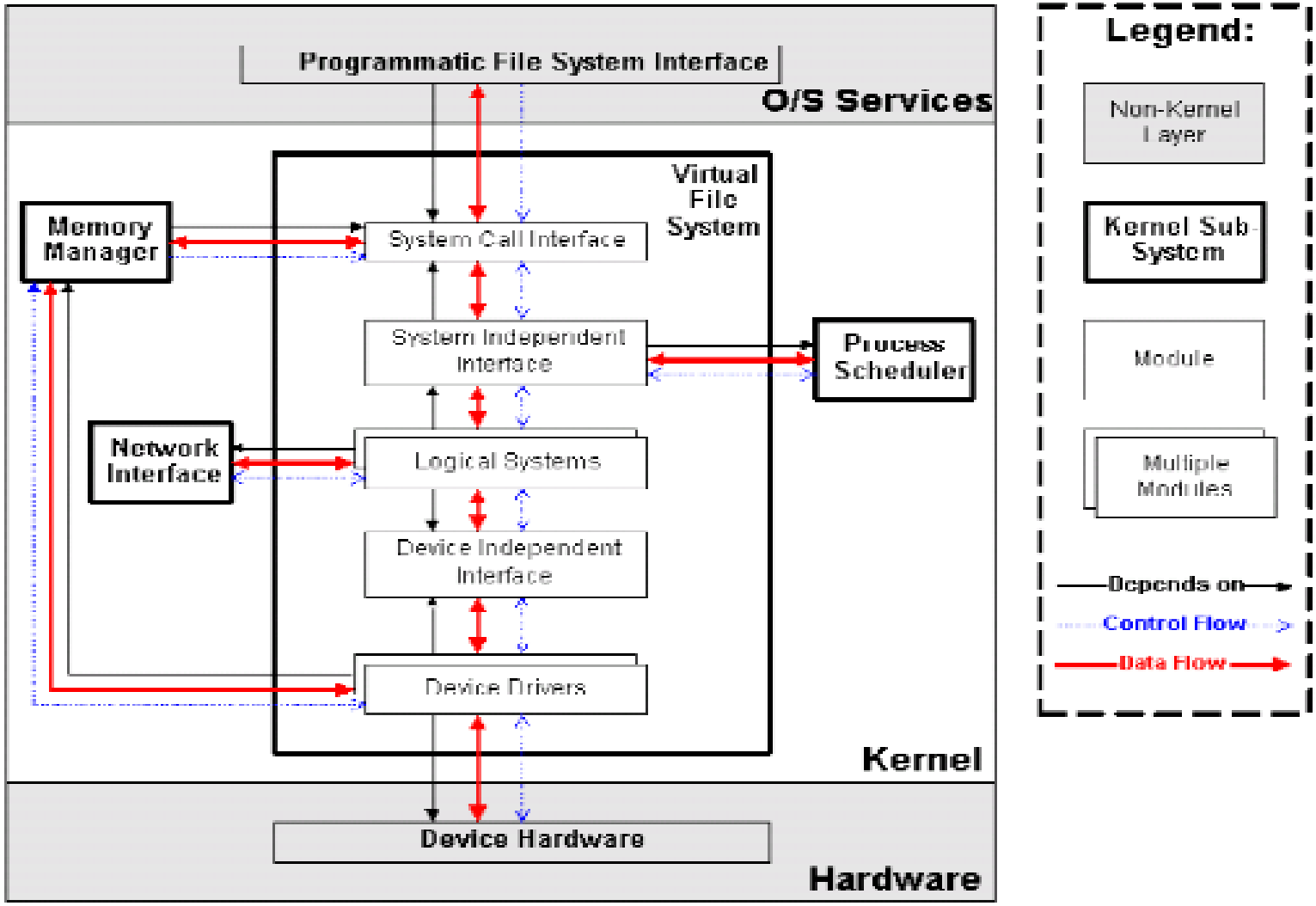
Независимый от устройства интерфейс. Обеспечивает инкапсуляцию работы с аппаратным обеспечением.

Модуль логической файловой системы. Отдельный модуль для каждого типа файловой системы.

Унифицированный интерфейс к файловым системам. Обеспечивает доступ через интерфейс блочных устройств или через интерфейс символьных устройств.

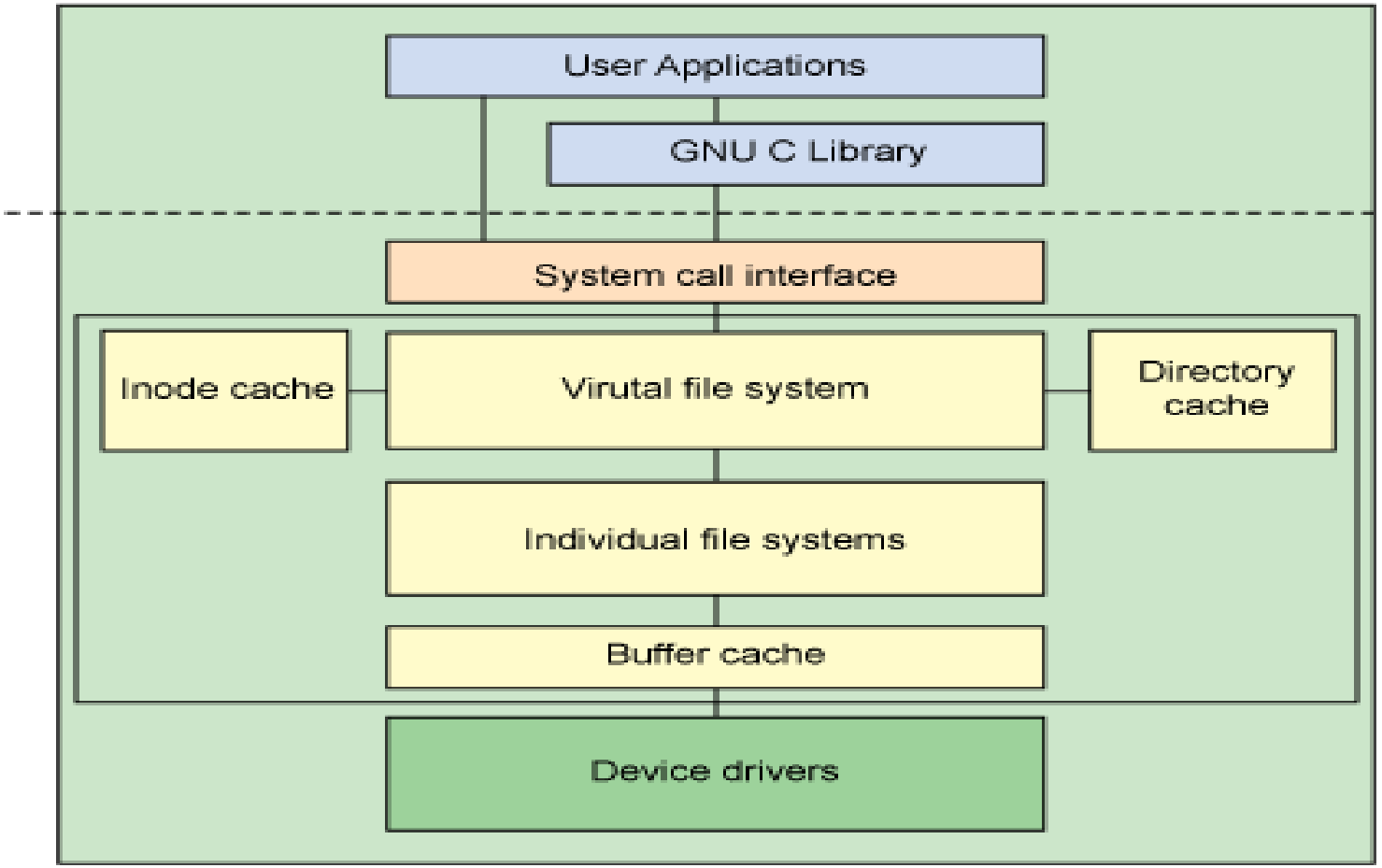
Интерфейс системных вызовов.

Виртуальная файловая система. Структурная схема



Luca Pizzamiglio. The linux kernel: architecture and programming.

Виртуальная файловая система. Структурная схема



M. Tim Jones. M. Tim Jones Anatomy of the Linux file system. http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/

VFS. Обобщенная модель файла.

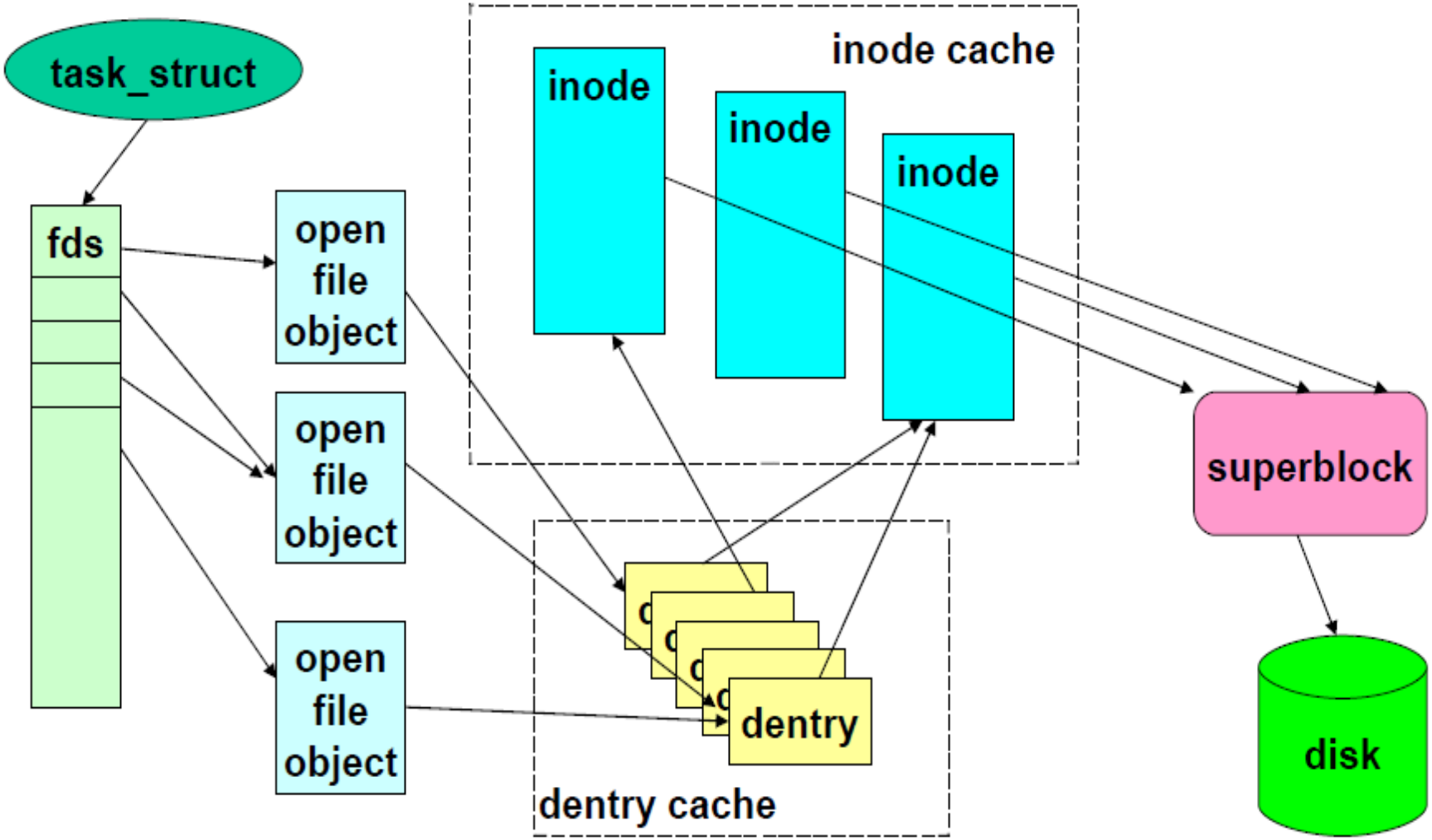
VFS предоставляет обобщенную модель файловой системы для работы со всеми поддерживаемыми файловыми системами

В основе лежит файловая система Unix остальные файловые системы приводятся к данной модели.

Основные компоненты обобщенной файловой системы:

- superblock (информация о смонтированной файловой системе)
- inode (информация о файле)
- file (информация об открытом файле)
- dentry (информация о директории)

Взаимосвязь структур данных VFS.



Сетевая подсистема

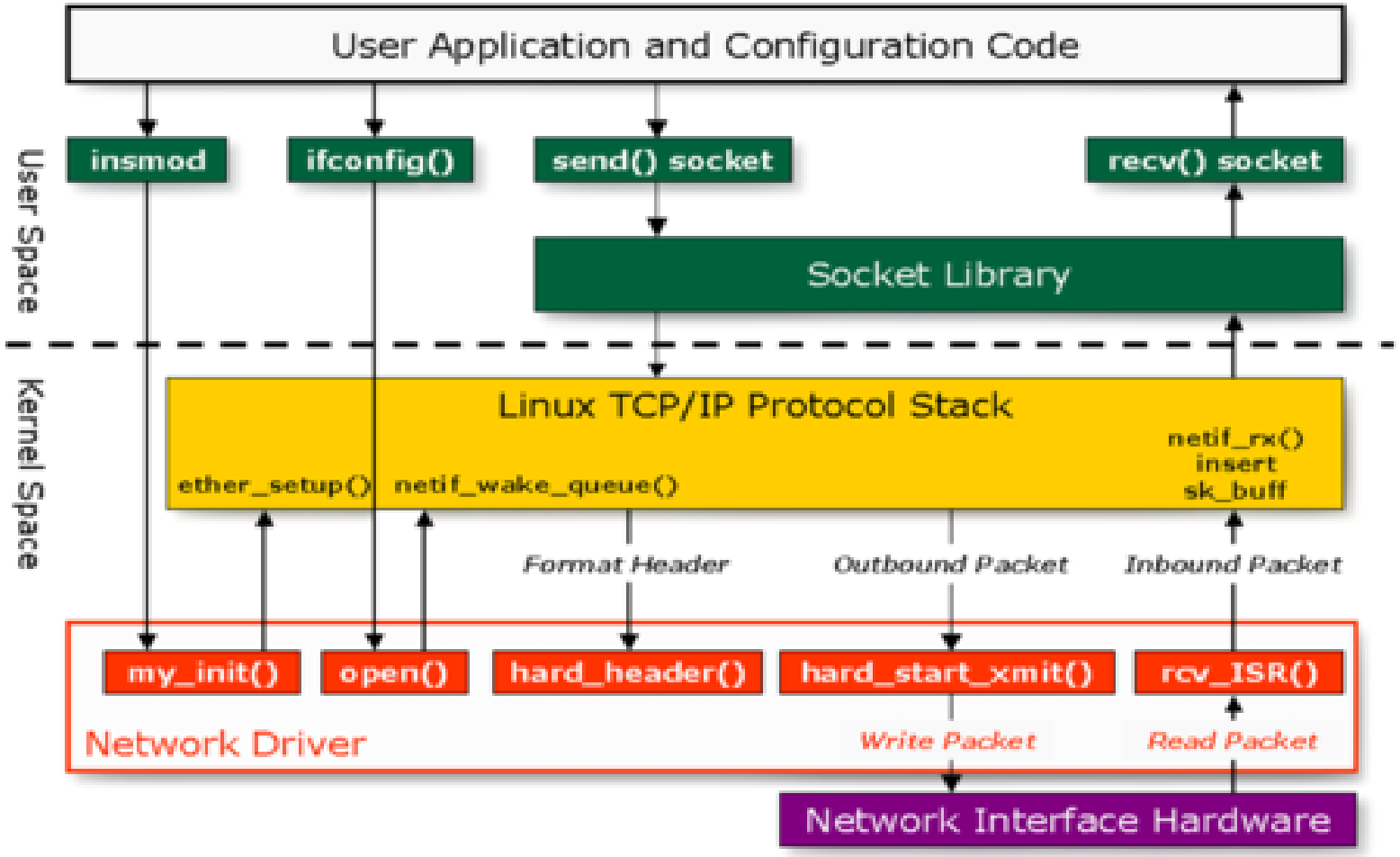
Представляет унифицированный доступ к сервисам передачи данных по сети.

Отделяет стек протоколов от особенностей аппаратной реализации сетевого оборудования.

Поддерживает множество сетевых протоколов.

Поддерживает множество сетевых устройств.

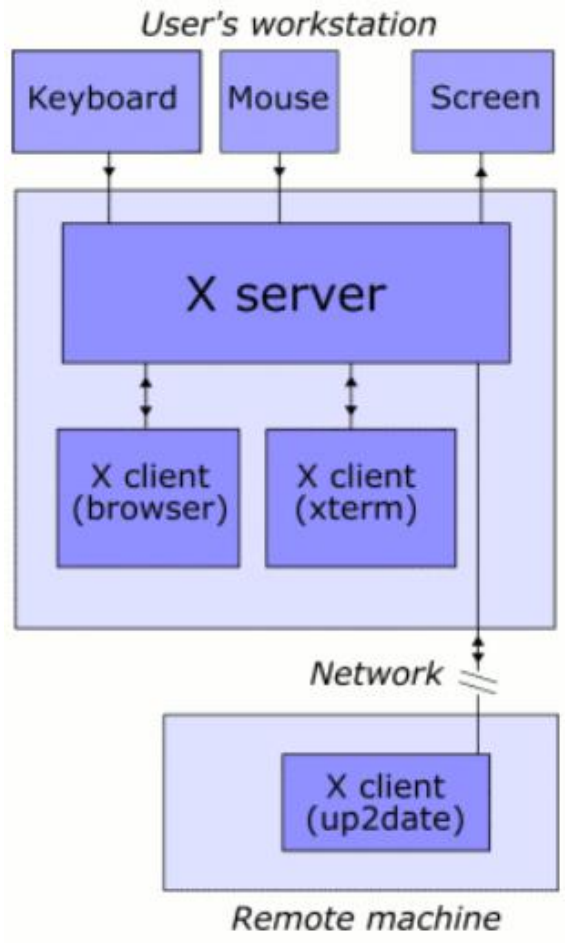
Сетевая подсистема. Структурная схема



На основе статьи Bill Weinberg Porting RTOS Device Drivers to Embedded Linu. http://www.linuxjournal.com/article/7355

RTSoft

Графический интерфейс в Linux – X Window



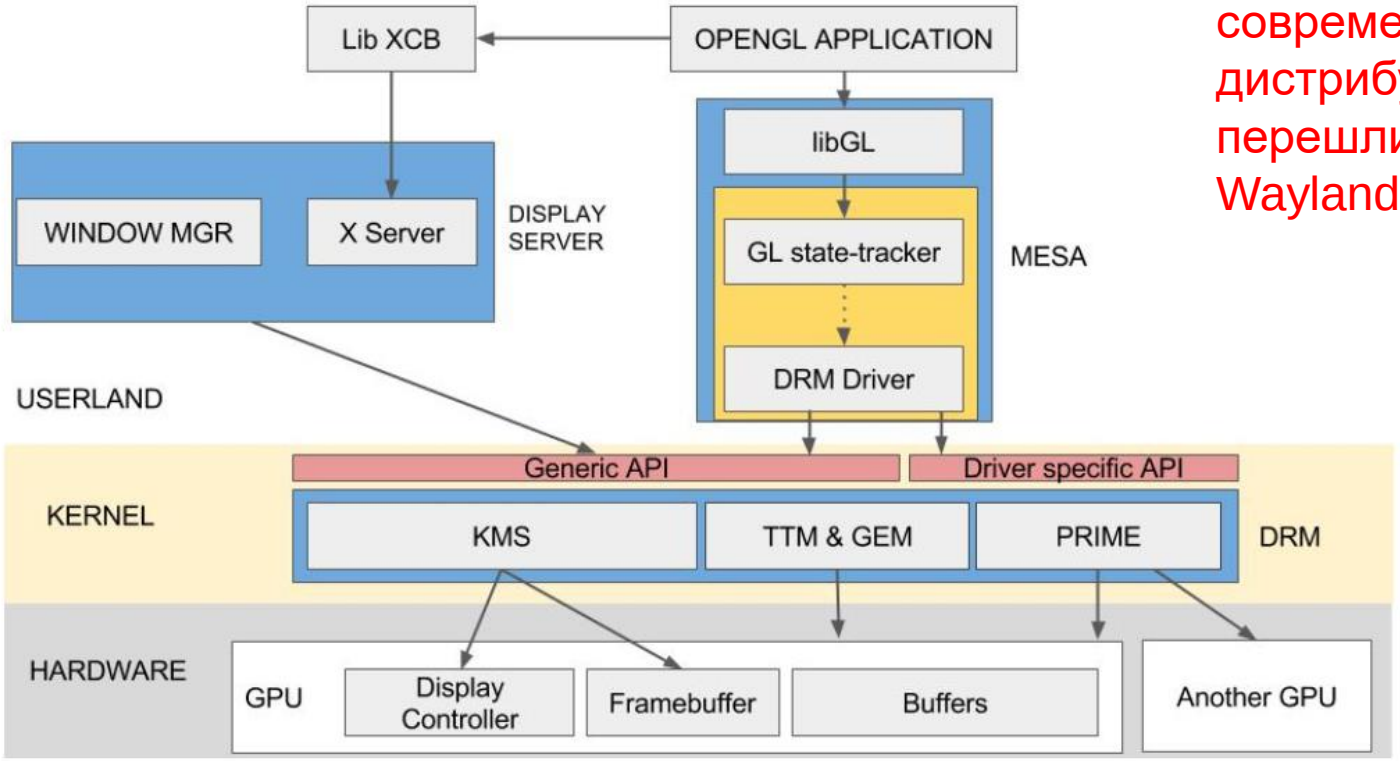
X Window System — Система оконного интерфейса X, создана в недрах MIT летом 1984 г. В иксах определен протокол X.

X11 — Одиннадцатая версия X, которая была выпущена в 1987 г. и в различных ипостасях благополучно дожила до наших дней. В частности, от нее отпочковался XFree86, из которого затем появился по настоящему свободный X.Org Server, с лицензией GPL, как положено.

https://www.cs.mcgill.ca/~rwest/wikispeedia/wpcd/wp/x/X_Window_System_protocols_and_architecture.htm

Графический стек Linux

Многие современные дистрибутивы перешли на Wayland



GLSL example

```
uniform vec4 args1, args2;

void main()
{
        gl_FragColor = log2(args1) + args2;
}
```

GLSL IR

```
GLSL IR for native fragment shader 3:
(
(declare (location=2 shader_out ) vec4 gl_FragColor)
(declare (location=0 uniform ) vec4 args1)
(declare (location=1 uniform ) vec4 args2)
( function main
  (signature void
    (parameters)
    (
      (assign  (xyzw)
               (var_ref gl_FragColor)
               (expression vec4 + (expression vec4 log2 (var_ref args1) )
                                  (var_ref args2) ) )
    ))
)
)
```

Intel i965 instruction set

```
   START B0 (54 cycles)
math log(16)     g3<1>F          g2<0,1,0>F      null<8,8,1>F
math log(16)     g5<1>F          g2.1<0,1,0>F    null<8,8,1>F
math log(16)     g7<1>F          g2.2<0,1,0>F    null<8,8,1>F
math log(16)     g9<1>F          g2.3<0,1,0>F    null<8,8,1>F
add(16)          g120<1>F        g3<8,8,1>F      g2.4<0,1,0>F
add(16)          g122<1>F        g5<8,8,1>F      g2.5<0,1,0>F
add(16)          g124<1>F        g7<8,8,1>F      g2.6<0,1,0>F
add(16)          g126<1>F        g9<8,8,1>F      g2.7<0,1,0>F
sendc(16)        null<1>UW       g120<8,8,1>UD   0x90031000
                 render MsgDesc: RT write SIMD16 LastRT mlen 8 rlen 0
   END B0
```

RISoft

Типы драйверов

- Символьные драйверы
- Блочные драйверы
- Сетевые драйверы

Символьный драйвер

Символьный драйвер обрабатывает поток байт. Обычно драйвер отвечает за реализацию следующих системных вызовов open, close, read, and write .

Примеры:

- console /dev/console
- serial ports /dev/ttyS0, /dev/ttyUSB0
- IrDA
- Bluetooth
- web камера /dev/video0
- GPIO /dev/gpiochip0
- ...

Доступ к символьным устройствам осуществляется через файловую систему. Они представляются в виде файла.

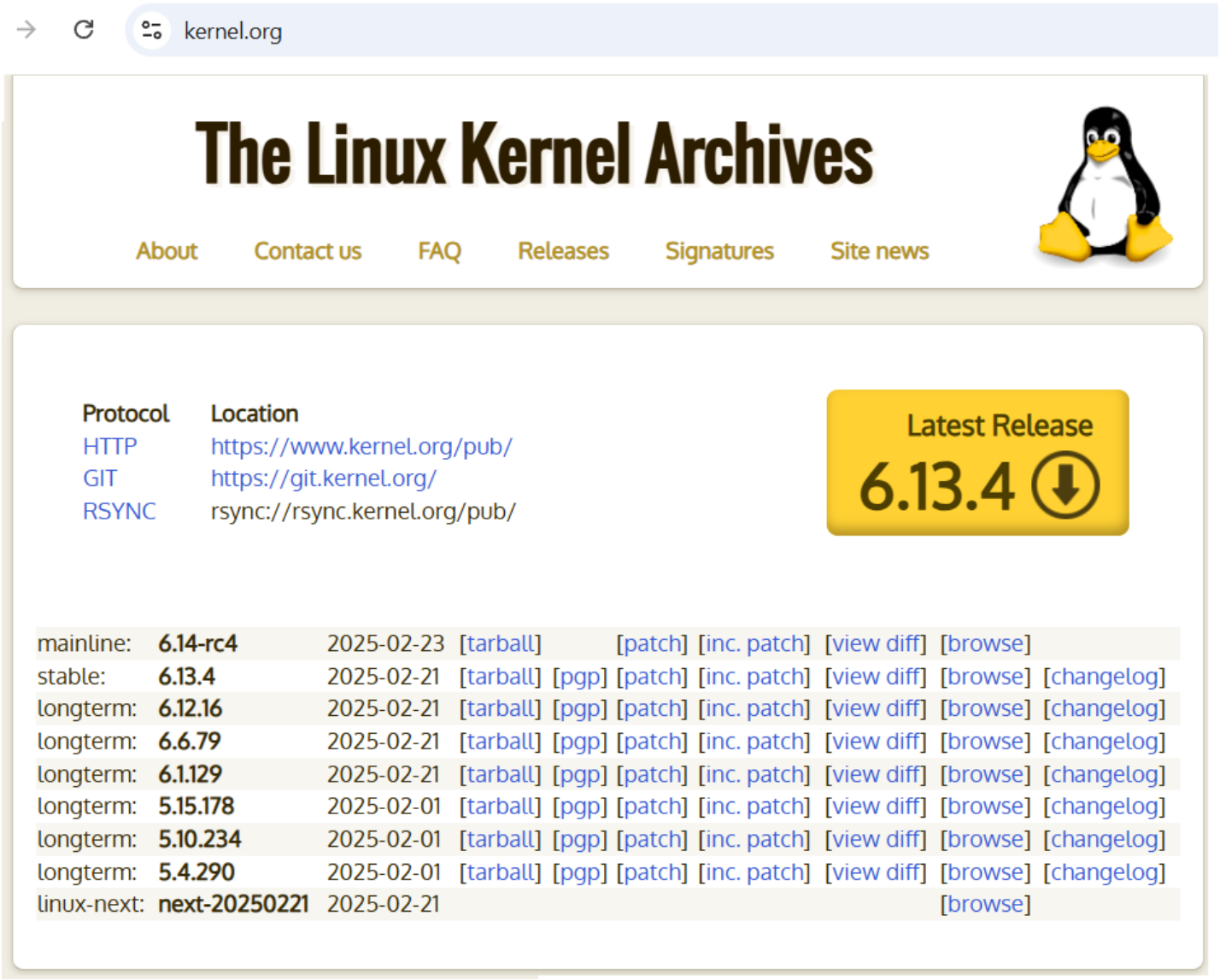
Блочное устройство

- Блочное устройство может иметь файловую систему
- Как и символьное устройство блочное представляется файлом в каталоге /dev
- Блочное устройство оперирует блоками информации (обычно по 512 байт)
- В linux оно может работать с блоками любого размера

Сетевой драйвер

- Служит для обмена информацией с сетевым устройством.
- Обрабатывает пакеты
- Интерфейс сетевого устройства отличен от символьного и блочного

Исходные код ядра на www.kernel.org



Исходные код ядра на www.kernel.org

Longterm release kernels

Version	Maintainer	Released	Projected EOL
6.12	Greg Kroah-Hartman & Sasha Levin	2024-11-17	Dec, 2026
6.6	Greg Kroah-Hartman & Sasha Levin	2023-10-29	Dec, 2026
6.1	Greg Kroah-Hartman & Sasha Levin	2022-12-11	Dec, 2027
5.15	Greg Kroah-Hartman & Sasha Levin	2021-10-31	Dec, 2026
5.10	Greg Kroah-Hartman & Sasha Levin	2020-12-13	Dec, 2026
5.4	Greg Kroah-Hartman & Sasha Levin	2019-11-24	Dec, 2025

https://www.kernel.org/category/releases.html

Часть 2. Определение модуля

Модуль — фрагмент кода, который может быть загружен или выгружен из ядра по запросу. Модули расширяют функции системы без необходимости перезагрузки ядра.

Просмотреть список модулей:

```
lsmod
```

Установить модуль

```
insmod hello.ko  (/sbin/insmod hello.ko)
```

```
modprobe hello
```

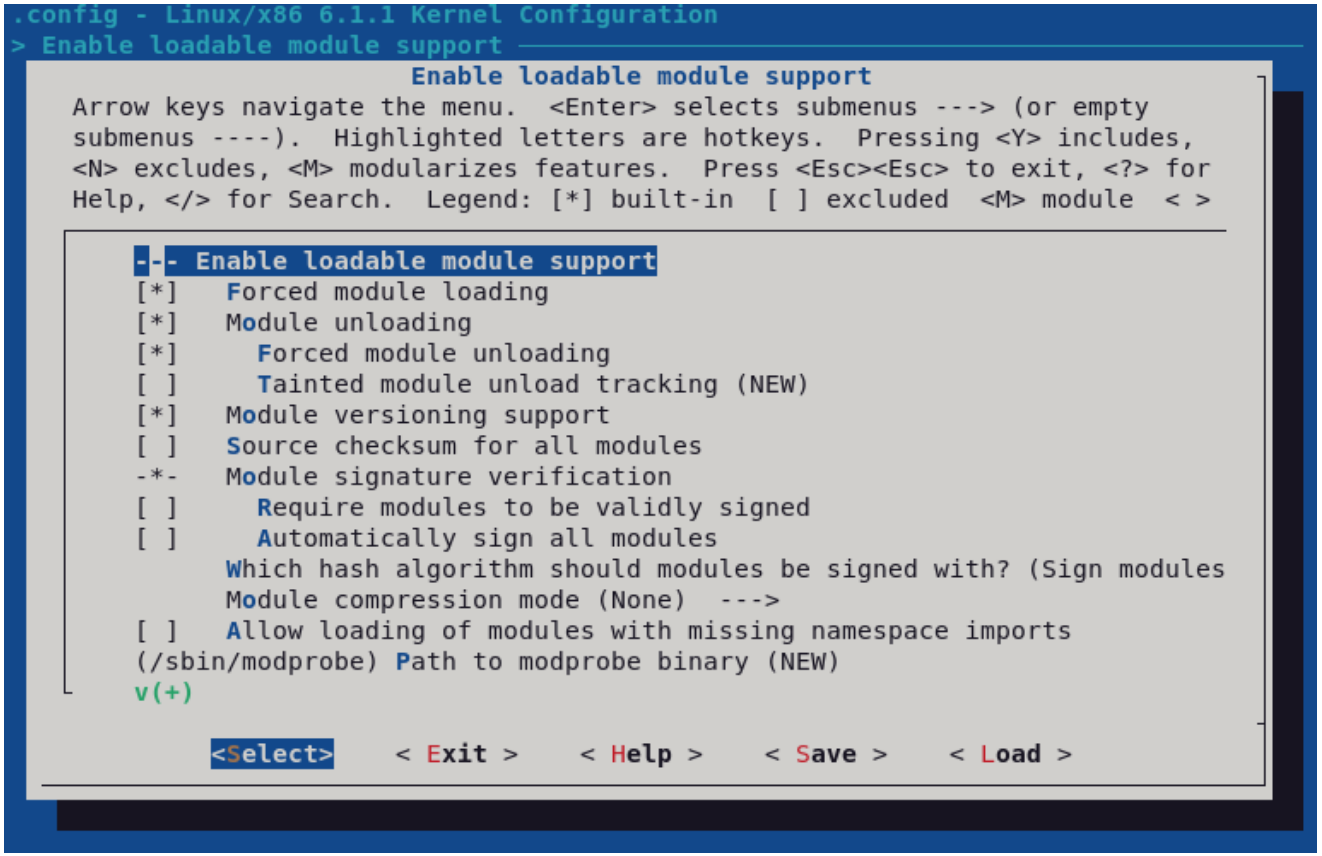
Удалить модуль

```
rmmod hello.ko
```

Module dependency

```
depmod hello.ko
```

Конфигурирование поддержки модулей



- Поддержка модулей может быть выключения для повышения безопасности
- Использование модулей теоретически может приводить к фрагментации памяти

Пример модуля

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
MODULE_LICENSE("GPL");
int init_module(void) {
    printk(KERN_INFO "Hello world 1.\n");
/*
*  Модуль должен возвращать  0 в случае успешной загрузки.
*/
   return 0;
}
void cleanup_module(void) {
   printk(KERN_INFO "Goodbye world 1.\n");
}
```

Компиляция модуля. Makefile

Для компиляции компонентов ядро Makefile должен отличаться от обычного.
Пример:

```
obj-m += hello.o
all:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Что нужно для сборки на Devnian/Ubuntu

Для компиляции компонентов ядра нужны заголовочные файлы ядра.

```
sudo apt update

sudo apt install linux-headers-$(uname -r)
```

Для нашего debian:

apt-get install linux-headers-5.10.0-22-amd64

Метаданные и макросы

Добавление информации о модуле

MODULE_AUTHOR(author);

MODULE_DESCRIPTION(description);

MODULE_VERSION(version_string);

MODULE_LICENSE(lizense);

Посмотреть параметры модуля можно так:

```
modinfo helloworld.ko
```

и др.

Определение воходной и выходной точек модуля:

module_init(init_function);

module_exit(exit_function);

Утилита modinfo

Программа для просмотра информации о модуле Linux

```
[root@lab]# modinfo module5.ko
filename: module5.ko
description: This module is a homework
license: GPL
author: Ivanov Ivan
srcversion: 7C328D0F123A8B302DDCD43
depends:
vermagic: 2.6.37.emb SMP mod_unload 686
```

Возможные типы лицензий

include/linux/module.h

- GPL
  GNU Public License v2 or later
- GPL v2
  GNU Public License v2
- GPL and additional rights
- Dual MIT/GPL
  GNU Public License v2 or MIT
- Dual BSD/GPL
  GNU Public License v2 or BSD
- Dual MPL/GPL
  GNU Public License v2 or Mozilla
- Proprietary
  Non free products

printk — печать сообщений

Прототип функции (include\linux\printk.h):

int printk(const char * fmt, ...)

Поддерживаемые форматы можно посмотреть в Documentation/printk-formats.txt

Пример:

printk(**KERN_INFO** "fb%d: %s frame buffer device\n", info->node,info->fix.id);

Сообщения printk имеют приоритет:

Name	String	Alias function
KERN_EMERG	“0”	`pr_emerg()`
KERN_ALERT	“1”	`pr_alert()`
KERN_CRIT	“2”	`pr_crit()`
KERN_ERR	“3”	`pr_err()`
KERN_WARNING	“4”	`pr_warn()`
KERN_NOTICE	“5”	`pr_notice()`
KERN_INFO	“6”	`pr_info()`
KERN_DEBUG	“7”	`pr_debug()` and `pr_devel()` if DEBUG is defined
KERN_DEFAULT	“”	
KERN_CONT	“c”	`pr_cont()`

Просмотр сообщений от printk

- Сообщение помещаются в циркулярный буфер
- Посмотреть можно разными способами:

1) Посмотреть файл

/var/log/messages

Файл формируется демоном

syslogd / klogd

Посмотреть можно так:

tail -f /var/log/messages

Дла контроля роста файла можно использовать logrotate

- cat /proc/kmsg
- **dmesg** (“**d**iagnostic **mes**sa**g**e”) Отображает содержимое kernel log buffer

Просмотр на экране

Сообщение помещаются в циркулярный буфер.

Изменить границу приоритета для отображаемых на экране сообщений можно в файле /etc/sysctl.conf

kernel.printk = 4 4 1 7

Расшифровка значений:

- console_loglevel: сообщения с приопритетом выще чем у указанного печатаются на консоле
- default_message_level: сообщения без приоритета будут иметь данный приоритет
- minimum_console_loglevel: минимальное(highest) значение в которое может быть установлено console_loglevel
- default_console_loglevel: значение по умолчанию для console_loglevel

(дополнительно можно посмотреть http://linux.die.net/man/2/syslog)

Для динамического изменения можно:

```
echo "7 1 1 7" > /proc/sys/kernel/printk
```

посмотреть текущее состояние можно так:

```
cat /proc/sys/kernel/printk
```

Часть 3. Передача параметров в модуль

```
#include <linux/moduleparam.h>
module_param(variable, type, perm);
```

variable — имя переменной
type - тип (можно посмотреть в linux/moduleparam.h)
perm — права доступа (можно посмотреть в linux/stat.h)

Параметр можно изменить из sysfs

Макрос для modinfo:

```
MODULE_PARM_DESC(debug_enable, "Enable deb mode");
```

Пример передачи параметров

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
MODULE_LICENSE("GPL");

static char *whom = "world";
module_param(whom, charp, 0);
static int howmany = 1;
module_param(howmany, int, 0);          // Передается пара параметров - сколько
раз и  кому передается привет
static int __init hello_init(void) {
    int i;
    for (i = 0; i < howmany; i++)
      printk(KERN_ALERT "(%d) Hello, %s\n", i, whom);
    return 0;
}
static void __exit hello_exit(void) {
    printk(KERN_ALERT "Goodbye, cruel %s\n", whom);
}
module_init(hello_init);
module_exit(hello_exit);
```

Запуск модуля с параметрами

- Вызов insmod:

```
sudo insmod ./hello_param.ko howmany=2 whom=universe
```

- Вызов modprobe:
  Set parameters in /etc/modprobe.conf or in any file in /etc/modprobe.d/:

```
options hello_param howmany=2 whom=universe
```

- Случай статической линковки модуля к ядру!:

```
options hello_param.howmany=2 hello_param.whom=universe
```

Стиль кодирования

/usr/src/linux/Documentation/CodingStyle

Ключевые особенности:

- Избегать трюков в выражениях
- Размер строки 80 символов
- Разбивать длинные выражения
- Открывающая скобка блока — последняя в строке
- В функциях открывающая скобка с начала строки
- В if скобки не нужны только если и в if и в else по одному выражению
- В указателях * ближе к переменной, чем к типу.
- Отступ в 1 символ вокруг бинарных операций
- Именование с использованием нескольких слов осуществляется с помощью _
- EXPORT_SYMBOL пишется сразу после функции, которую он экспортирует
- Goto используется для обработок ошибок
- Комментарии C99 не используются

Стиль кодирования

Пример:

```
/*
*     This function does ...
*/
int get_user_id(struct user *u)
{
if  (u->condition)  {
do_smth();
do_smth_again();
}
else  {
goto error;
                    }
/* … */
}
EXPORT_SYMBOL(get_user_id);
```

Динамическое выделение памяти

malloc - не существует.

kmalloc/kfree выделяет до 128k физической памяти в последовательном блоке

kcalloc(. . .) аналогично kmalloc, но память обнуляется

void *kmalloc (size t size, int flags);

flags определены в <linux/mm.h>

GFP_USER associated userspace process sleeps until free memory available

GFP_KERNEL associated kernel function sleeps until free memory available

GFP_ATOMIC doesn't sleep (used in ISRs)

Таблица символов модуля

Таблицы символом модуля содержатся в следующих сегментах:

__kstrtab – имена символов

__ksymtab – адреса символов, доступных всем типам модулей

__ksymtab_gpl - адреса символов, доступных модулям с лицензией GPL

Для экспорта символом из модуля можно использоваться макросы:

EXPORT_SYMBOL

EXPORT_SYMBOL_GPL

Пример (linux-xxx\drivers\net\mii.c):

```
int mii_link_ok (struct mii_if_info *mii) {….}
EXPORT_SYMBOL(mii_link_ok);
```

Что может показать objdump

objdump –section-headers module1.ko

```
module1.ko:     file format elf32-i386
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .note.gnu.build-id 00000024  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .text         00000018  00000000  00000000  00000058  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  2 .init.text    00000011  00000000  00000000  00000070  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  3 .rodata.str1.1 00000028  00000000  00000000  00000081  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 __mcount_loc  00000004  00000000  00000000  000000ac  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
  5 .modinfo      00000064  00000000  00000000  000000b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .data         00000000  00000000  00000000  00000114  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .gnu.linkonce.this_module 00000164  00000000  00000000  00000114  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, DATA, LINK_ONCE_DISCARD
  8 .bss          00000000  00000000  00000000  00000278  2**2
                  ALLOC
  9 .note.GNU-stack 00000000  00000000  00000000  00000278  2**0
                  CONTENTS, READONLY, CODE
 10 .comment      0000005a  00000000  00000000  00000278  2**0
                  CONTENTS, READONLY
```

Структура module

```
struct module {
        enum module_state state;
        struct list_head list;          /* Member of list of modules */
        char name[MODULE_NAME_LEN];     /* Unique handle for this module */
        struct module_kobject mkobj;            /* Sysfs stuff. */
        struct module_attribute *modinfo_attrs;
        const char *version;
        const char *srcversion;
        struct kobject *holders_dir;
        const struct kernel_symbol *syms; /* Exported symbols */
        const unsigned long *crcs;
        unsigned int num_syms;
        struct kernel_param *kp;            /* Kernel parameters. */
        unsigned int num_kp;
        unsigned int num_gpl_syms;      /* GPL-only exported symbols. */
        const struct kernel_symbol *gpl_syms;
        const unsigned long *gpl_crcs;
        unsigned int num_exentries;     /* Exception table */
        struct exception_table_entry *extable;
        int (*init)(void);                      /* Startup function. */
        void *module_init;                      /* If this is non-NULL, vfree after init() returns */
        void *module_core;                      /* Here is the actual code + data, vfree'd on unload. */
        unsigned int init_size, core_size;      /* Here are the sizes of the init and core sections */
        unsigned int init_text_size, core_text_size; /* The size of the executable code in each section.  */
        struct mod_arch_specific arch;          /* Arch-specific module values */
        unsigned int taints;                    /* same bits as kernel:tainted */
        void *percpu;                           /* Per-cpu data. */
        char *args; /* The command line arguments (may be mangled).  People like          keeping pointers to this stuff */
#ifdef CONFIG_MODULE_UNLOAD
        struct list_head modules_which_use_me; /* What modules depend on me? */
        struct task_struct *waiter; /* Who is waiting for us to be unloaded */
        void (*exit)(void); /* Destruction function. */
        local_t ref;
#endif
};
```

Литература

1. Пособие по программированию модулей ядра Linux. Ч.1 https://habr.com/ru/company/ruvds/blog/681880/
2. Пособие по программированию модулей ядра Linux. Ч.2 https://habr.com/ru/company/ruvds/blog/683106/
3. Writing a Linux Kernel Module — Part 2: A Character Device http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/
4. The linux-kernel mailing list FAQ http://www.tux.org/lkml.
5. Rob Day The Kernel Newbie Corner: What's in That Loadable Module, Anyway? http://www.linux.com/learn/linux-training/32867-the-kernel-newbie-corner-whats-in-that-loadable-module-anyway
6. Andrew Murray Init Call Mechanism in the Linux Kernel

http://linuxgazette.net/157/amurray.html

7. Kernel modules https://wiki.archlinux.org/index.php/Kernel_modules (полезные команды и конфигурирование)
8. https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html#overview
9. Automatic create /dev file https://gist.github.com/strezh/b01fcd50875c214e510a81c6aa6d2a2a
10. https://embetronicx.com/tutorials/linux/device-drivers/device-file-creation-for-character-drivers/#Automatically_Creating_Device_File
11. https://sysprog21.github.io/lkmpg/#avoiding-collisions-and-deadlocks
12. Understanding the Structure of a Linux Kernel Device Driver https://www.youtube.com/watch?v=XoYkHUnmpQo
13. Тернистый путь Hello World https://habr.com/ru/articles/339698/