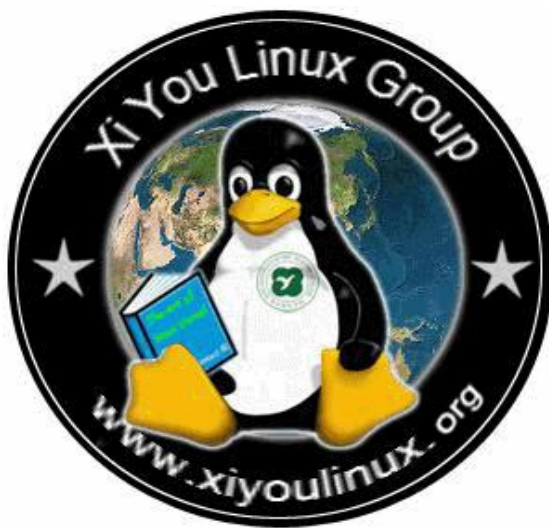

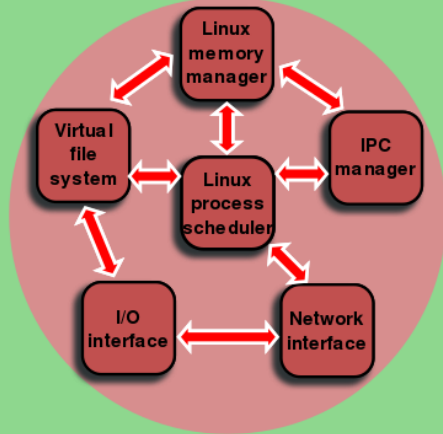
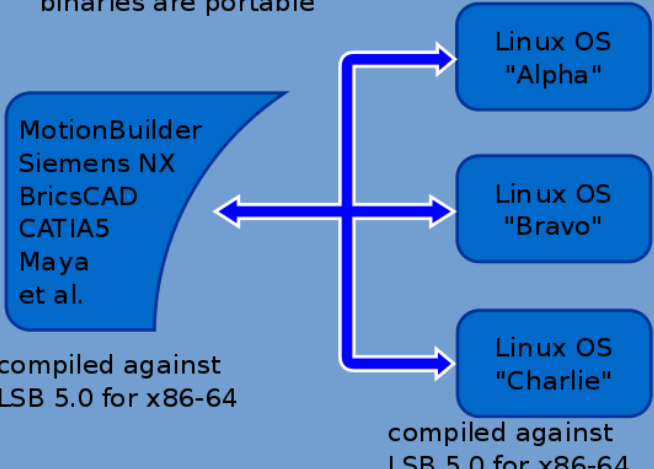
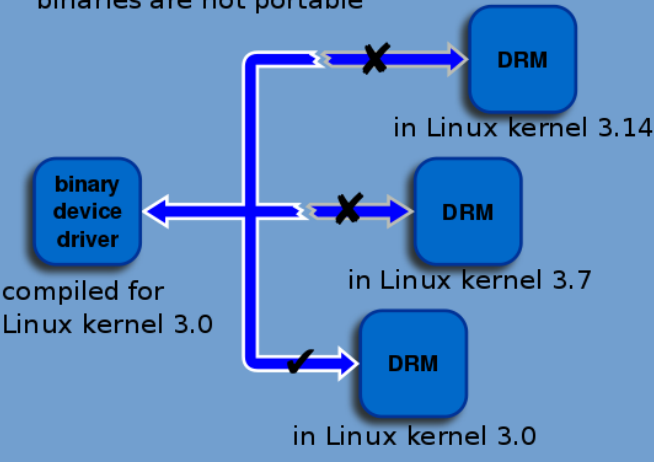


6.1 Linux中各种接口



西安邮电大学

Linux 中各种接口

	Linux kernel-to-userspace	Linux kernel-internal
API	<p>☑ API stability is guaranteed, source code is portable!</p> 	<p>☒ API stability is not guaranteed, source code portability is not given</p> 
ABI	<p>☑ compatible ABI can be guaranteed, binaries are portable</p> 	<p>☒ no stable ABI over Linux kernel releases, binaries are not portable</p> 

Linux中的各种接口

如果将内核比作一座工厂，那么Linux中众多的接口就是通往这个巨大工厂的高速公路。这条路要足够坚固，禁得起各种破坏(Robust)。要能跑得了运货的卡车，还要能升降飞机。(Compatible)。当然了这条路要越宽越好(Performant)。如图所示，Linux中有四种类型的接口，应用编程接口(API)和应用二进制接口(ABI)。内核内部的API和ABI。下面我们逐一的来看看这些接口。

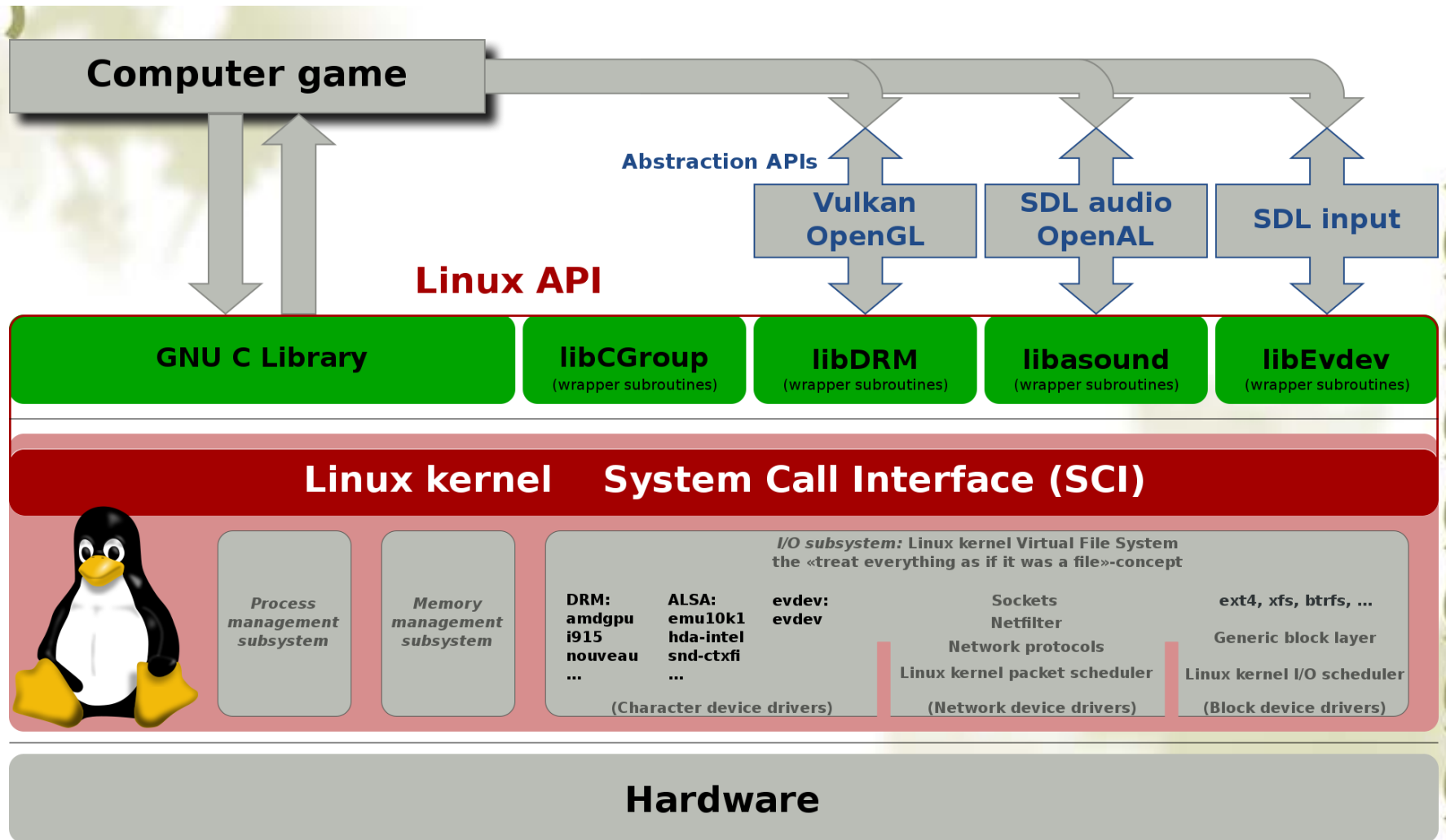
LSB (Linux Standards Base)



图中的在ABI提到LSB，那么是什么LSB？目前Linux 的发行版非常繁多，为了促进Linux 不同发行版间的兼容性，LSB开发了一系列标准，使各种软件可以很好地在兼容LSB 标准的系统上运行，从而可以帮助软件供应商更好地在Linux 系统上开发产品，或将已有的产品移植到Linux系统上。

LSB 是Linux 标准化领域中事实上的标准，它的图标（请参看图）非常形象地阐述了自己的使命：对代表自由的企鹅（Linux）制定标准。给定企鹅的体形和三维标准之后，软件开发者就可以设计并裁减出各色花样的衣服（应用程序），这样不管穿在哪只企鹅身上，都会非常合身。

Linux API



Linux API

1. Linux API

Linux API是Linux内核与用户空间的API，也就是让用户空间的程序能够通过这个接口访问系统资源和内核提供的服务。Linux API由两部分组成：Linux内核的系统调用接口和GNU C库（glibc）中的例程。

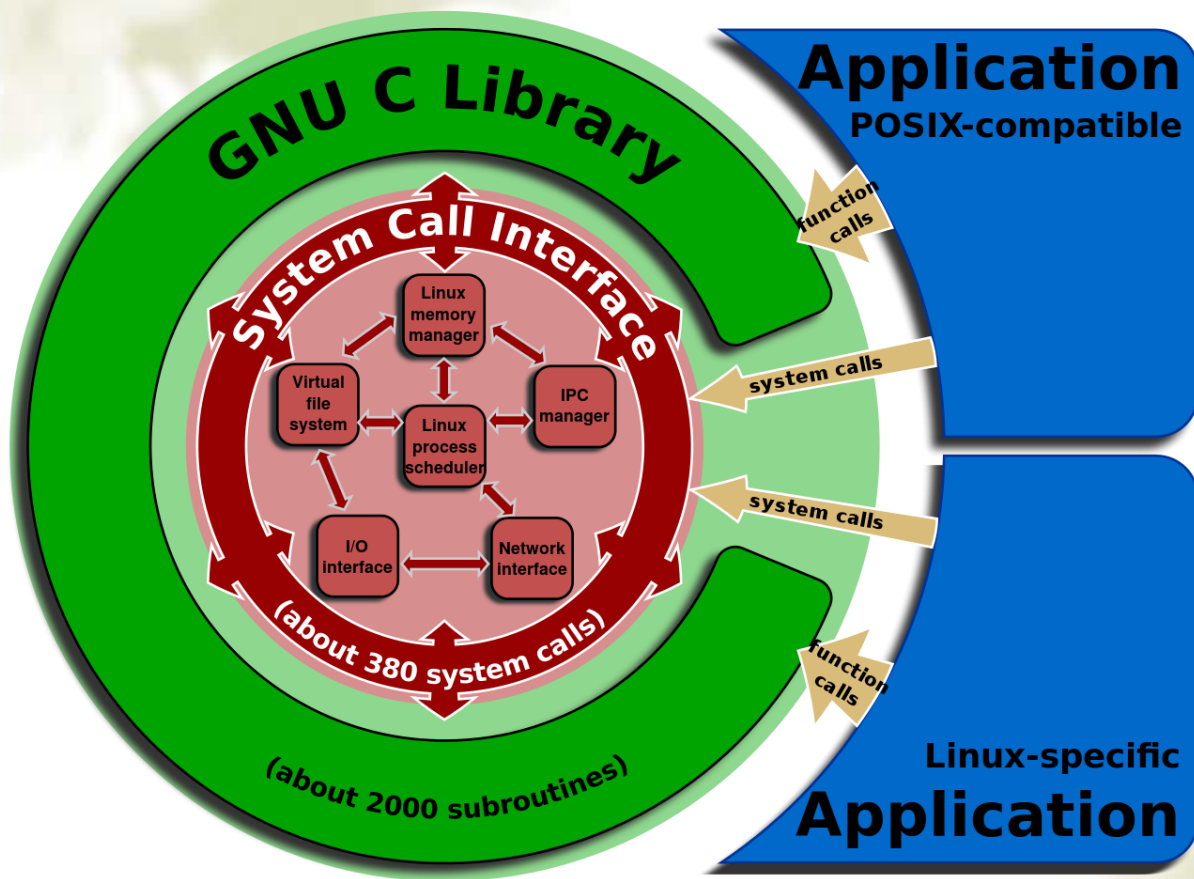
2. Linux内核系统调用接口

系统调用接口是内核中所有已实现和可用系统调用的集合，这是本章重点要介绍的，

3. C标准库

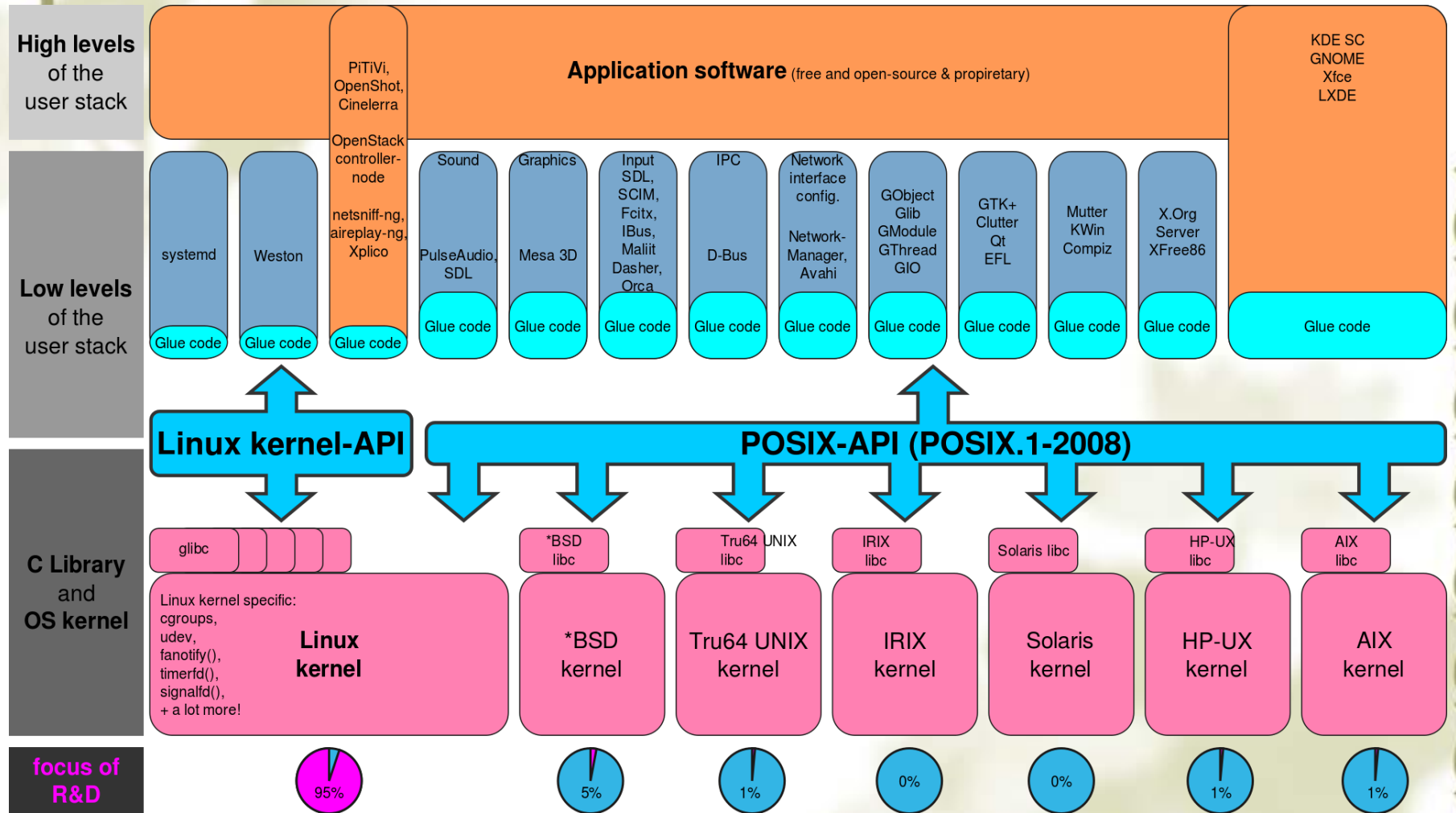
GNU C 库是Linux内核系统调用接口的封装，下面进一步介绍。Linux内核系统调用接口和glibc库合在一起就构成了Linux API

GNU C库



GNU C 库是Linux内核系统调用接口的封装。其中包括POSIX兼容应用函数调用和Linux 专用应用的函数调用，目前最新Linux内核5.0系统调用有大约380个左右，GNU C库大约有2000个左右的函数

Linux API VS POSIX API



Linux API VS POSIX API

1. 从下往上看，最下面 (focus of R &D) 是研发人员的比重，Linux内核占比最高，95%，其他占比就很小了；
2. 第二层是C库和OS内核 (C Library and OS kernel)，不同的内核都遵循POSIX的接口，但是就像Linux 内核中cgroups，udev这些就不属于POSIX的功能，我们还看到，POSIX-API和Linux kernel-API连接着Linux Kernel并向上提供服务；并且就glibc而言，glibc把Linux系统调用封装进相应API中提供给上层；
3. 第三层是用户栈底层 (Low levels of the user stack)，像systemd，IPC, QT都不是直接的应用程序，它们依靠胶水函数 (Glue code) 把上一层提供的API整合成平台工具供用户使用；
4. 最上层是用户栈高层 (High levels of the user stack) 就到应用软件层了，这一层可以直接调用Linux 内核的 API 和POSIX-API完成，也可以依托用户栈底层的这些平台工具来构建。

POSIX标准

POSIX

..... stands for

**Portable Operating System
Interface based on UNIX**



Abbreviations.com

这里对POSIX做一个简介。POSIX表示可移植操作系统接口（Portable Operating System Interface of UNIX，缩写为 POSIX），POSIX标准定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称，其正式称呼为IEEE 1003。

Linux ABI

```
static int collect(coret char *root) {
    enum {
        FD_FANOTIFY, /* Get the actual fs events */
        FD_SIGNAL,
        FD_NOTIFY, /* We get notifications to quit early via this fd */
        _FD_MAX
    };
    struct pollfd pollfd[_FD_MAX] = {0};
    int fanotify_fd = -1, signal_fd = -1, inotify_fd = -1, r = 0;
    pid_t my_pid;
    HANDLE_FILES = NULL;
    Iterator i;
    char *p, *q;
    sigset_t mask;
    FILE *fpack = NULL;
    char *pack_fn_new = NULL, *pack_fn = NULL;
    bool on_aid, on_btrfs;
    struct statfs sfs;
    usec_t not_after;
    uint64_t previous_block_readhead;
    bool previous_block_readhead_set = false;

    assert(root);

    if (asprintf(&pack_fn, "%s/.readhead", root) < 0) {
        r = log_oom();
        goto finish;
    }
}
```

systemd
(source code)

```
static int collect(coret char *root) {
    enum {
        FD_FANOTIFY, /* Get the actual fs events */
        FD_SIGNAL,
        FD_NOTIFY, /* We get notifications to quit early via this fd */
        _FD_MAX
    };
    struct pollfd pollfd[_FD_MAX] = {0};
    int fanotify_fd = -1, signal_fd = -1, inotify_fd = -1, r = 0;
    pid_t my_pid;
    HANDLE_FILES = NULL;
    Iterator i;
    char *p, *q;
    sigset_t mask;
    FILE *fpack = NULL;
    char *pack_fn_new = NULL, *pack_fn = NULL;
    bool on_aid, on_btrfs;
    struct statfs sfs;
    usec_t not_after;
    uint64_t previous_block_readhead;
    bool previous_block_readhead_set = false;

    assert(root);

    if (asprintf(&pack_fn, "%s/.readhead", root) < 0) {
        r = log_oom();
        goto finish;
    }
}
```

Linux kernel & GNU C Library
(source code)

Available documentation, e.g.:
Linux manual pages: system calls
The GNU C Library Reference Manual
»The Linux Programming Interface, Michael Kerrisk (2010, No Starch Press)
etc.

```
static int collect(coret char *root) {
    enum {
        FD_FANOTIFY, /* Get the actual fs events */
        FD_SIGNAL,
        FD_NOTIFY, /* We get notifications to quit early via this fd */
        _FD_MAX
    };
    struct pollfd pollfd[_FD_MAX] = {0};
    int fanotify_fd = -1, signal_fd = -1, inotify_fd = -1, r = 0;
    pid_t my_pid;
    HANDLE_FILES = NULL;
    Iterator i;
    char *p, *q;
    sigset_t mask;
    FILE *fpack = NULL;
    char *pack_fn_new = NULL, *pack_fn = NULL;
    bool on_aid, on_btrfs;
    struct statfs sfs;
    usec_t not_after;
    uint64_t previous_block_readhead;
    bool previous_block_readhead_set = false;

    assert(root);

    if (asprintf(&pack_fn, "%s/.readhead", root) < 0) {
        r = log_oom();
        goto finish;
    }
}
```

Siemens NX
(source code)

stable **API** *is* guaranteed,
source code remains portable

Compilation

compatible **ABI** *can be* guaranteed,
machine code becomes portable

binary compatible
(same instruction set,
same compilation environment)

Linux kernel & GNU C Library
(machine code)

Available cross-distribution ABIs, e.g.:
LSB (Linux Standard Base)

binary compatible
(same instruction set,
same compilation environment)

systemd *not*
(machine code) binary compatible

```
45 78 3C 1E A6 4F 30 06 80 E9 D3 A6 B2 A7 A8 88 55 A8 D7 34
71 02 89 83 98 76 CC 14 12 12 42 97 30 9F 32 79 12 93 26 4C
C8 03 F7 83 65 EB 56 AE 5A 50 6F FF 88 FE 70 01 93 27 4E
F9 DE 82 6D 5B 8B 7D F2 A4 89 8C 19 30 8A 8C CC 4C 26 4F 9C
AA 2A 56 AC 5A C5 CE BC 3C BE 9B 39 83 F2 8A 0A 34 5D A3 6F
88 D1 E3 4C 4D 49 E1 C2 08 2E E0 ED F7 DE E7 91 C7 1F 27 39
5D 90 18 63 05 AA 05 AC 5D 87 8E BB 6E BF 8D 19 D3 A6 32 76
CC 68 1E 78 F8 21 34 5D E7 83 FF 7D C4 9A 85 EB D8 82 75 28
37 5D 77 1D 80 7A F6 24 3E 3E 8E EE 5D 88 86 F8 1F C9 FC A0
69 35 E4 71 8E 50 20 E8 09 18 07 85 ED 71 87 3E 44 99 55 05
35 5C 6E 02 1F AF 0F 05 EA 31 76 85 69 80 6D E1 F0 29 4A E3
FB 08 8D 71 78 31 5B 53 EF 37 67 DC 48 07 90 57 54 BA 89 EF
A9 BF 83 63 07 6E B2 32 02 C9 CA 48 27 2F BF 80 39 4F FE 90
F2 CA CA 16 08 4F 83 25 28 87 BB EC E1 D4 29 53 70 39 5D 3C
F8 B7 47 F9 F2 EB 6F 28 2F 2F E7 F6 98 6F C2 66 83 35 DE B9
46 7D 09 76 21 84 E8 48 0C C3 A0 7F BF 8E 9C 7F EE 39 BC F1
D6 08 DC F7 D0 C3 75 02 D9 6D D8 77 00 F0 C3 BC 1F F9 61 DE
8F 75 FA EE CE CF A7 58 07 AE E1 F7 C0 60 30 88 CF E7 08 87
9F 7E EA 29 6C F8 E5 11 E6 C0 9F CF A2 C5 48 C8 CA CC 64 F6
09 C7 F3 FC 8B 2F B1 76 FD 7A 96 FD FC 33 C3 87 00 25 28 B3
13 1F 7E FC 31 00 83 06 0C 08 8F D1 AF 5F 5F 00 76 EC CC A8
53 5E 80 7F FD 68 ED F6 D2 02 8D 6C D4 19 80 59 C7 1F 27
C1 F8 51 E0 A8 6F BE 05 E0 C3 73 CE AB 88 FD D8 6F 19 36 74
```

ABI

ABI

not binary compatible
Siemens NX
(machine code)

```
45 78 3C 1E A6 4F 30 06 80 E9 D3 A6 B2 A7 A8 88 55 A8 D7 34
71 02 89 83 98 76 CC 14 12 12 42 97 30 9F 32 79 12 93 26 4C
C8 03 F7 83 65 EB 56 AE 5A 50 6F FF 88 FE 70 01 93 27 4E
F9 DE 82 6D 5B 8B 7D F2 A4 89 8C 19 30 8A 8C CC 4C 26 4F 9C
AA 2A 56 AC 5A C5 CE BC 3C BE 9B 39 83 F2 8A 0A 34 5D A3 6F
88 D1 E3 4C 4D 49 E1 C2 08 2E E0 ED F7 DE E7 91 C7 1F 27 39
5D 90 18 63 05 AA 05 AC 5D 87 8E BB 6E BF 8D 19 D3 A6 32 76
CC 68 1E 78 F8 21 34 5D E7 83 FF 7D C4 9A 85 EB D8 82 75 28
37 5D 77 1D 80 7A F6 24 3E 3E 8E EE 5D 88 86 F8 1F C9 FC A0
69 35 E4 71 8E 50 20 E8 09 18 07 85 ED 71 87 3E 44 99 55 05
35 5C 6E 02 1F AF 0F 05 EA 31 76 85 69 80 6D E1 F0 29 4A E3
FB 08 8D 71 78 31 5B 53 EF 37 67 DC 48 07 90 57 54 BA 89 EF
A9 BF 83 63 07 6E B2 32 02 C9 CA 48 27 2F BF 80 39 4F FE 90
F2 CA CA 16 08 4F 83 25 28 87 BB EC E1 D4 29 53 70 39 5D 3C
F8 B7 47 F9 F2 EB 6F 28 2F 2F E7 F6 98 6F C2 66 83 35 DE B9
46 7D 09 76 21 84 E8 48 0C C3 A0 7F BF 8E 9C 7F EE 39 BC F1
D6 08 DC F7 D0 C3 75 02 D9 6D D8 77 00 F0 C3 BC 1F F9 61 DE
8F 75 FA EE CE CF A7 58 07 AE E1 F7 C0 60 30 88 CF E7 08 87
9F 7E EA 29 6C F8 E5 11 E6 C0 9F CF A2 C5 48 C8 CA CC 64 F6
09 C7 F3 FC 8B 2F B1 76 FD 7A 96 FD FC 33 C3 87 00 25 28 B3
13 1F 7E FC 31 00 83 06 0C 08 8F D1 AF 5F 5F 00 76 EC CC A8
53 5E 80 7F FD 68 ED F6 D2 02 8D 6C D4 19 80 59 C7 1F 27
C1 F8 51 E0 A8 6F BE 05 E0 C3 73 CE AB 88 FD D8 6F 19 36 74
```


Linux ABI

ABI是一系列约定的集合，可以说调用惯例(calling convention)就是ABI。因此，ABI是和具体CPU架构和OS相关的。

具体而言，ABI包含以下内容：

1. 一个特定的处理器指令集
2. 函数调用惯例
3. 系统调用方式
4. 可执行文件的格式(ELF, PE)

那么，究竟我们为什么要纠结于ABI这个概念呢？答案是为了兼容，只要OS遵守相同的ABI规范，那么不同的应用就可以实现向前兼容，再也不用担心版本升级后，旧版本的应用不能运行了。

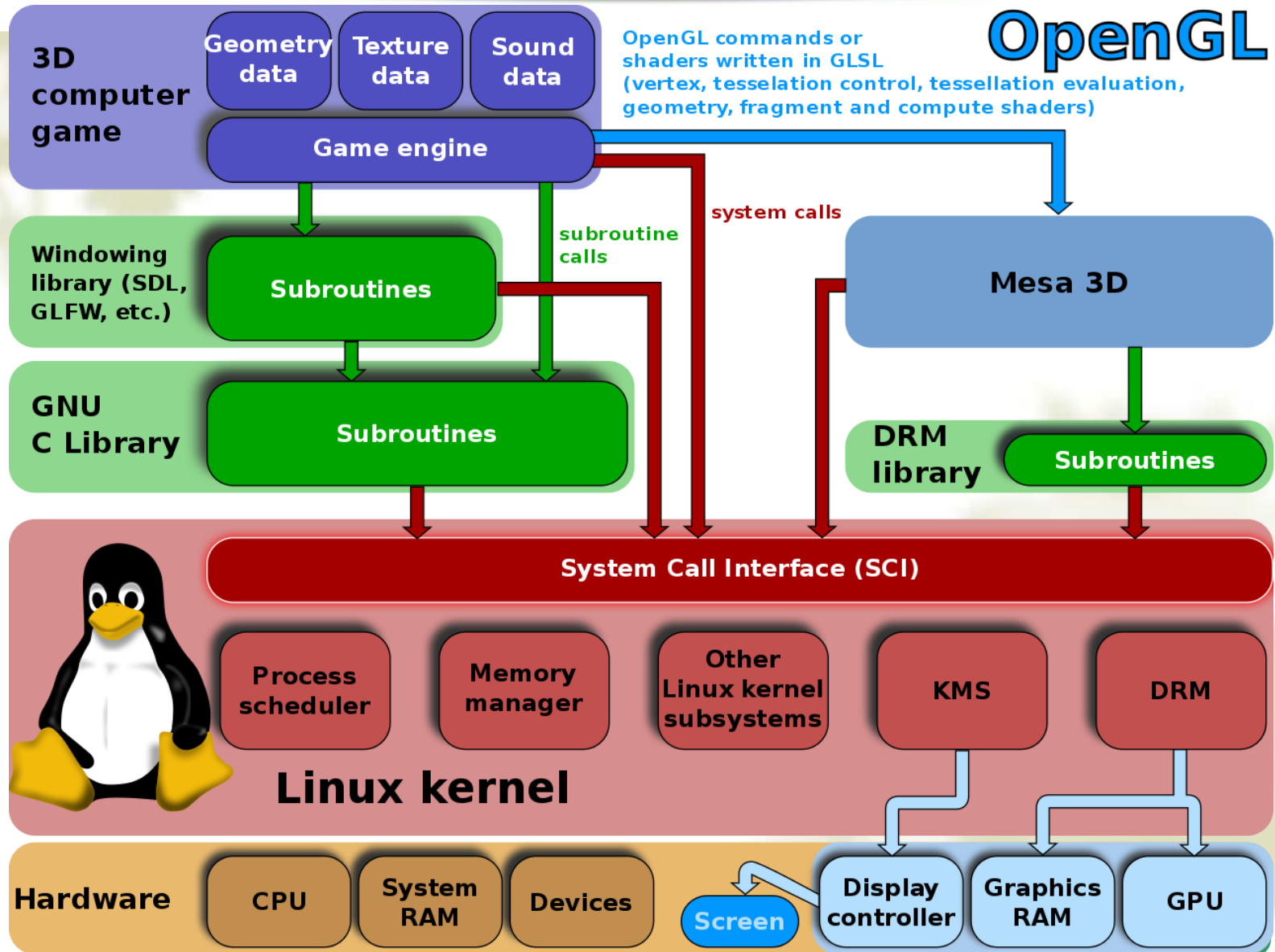
内核API

```
void add_wait_queue(wait_queue_head_t *q,  
wait_queue_t *wait)  
{  
    unsigned long flags;  
    wait->flags &= ~WQ_FLAG_EXCLUSIVE;  
    spin_lock_irqsave(&q->lock, flags);  
    __add_wait_queue(q, wait);  
    spin_unlock_irqrestore(&q->lock, flags);  
}  
  
EXPORT_SYMBOL (add_wait_queue)
```

内核API主要是内核中标记为

“EXPORT_SYMBOL”的函数。这些函数主要是为了内核模块的编写而提供的。受到内核版本迭代的影响，内核API并不稳定。3.x版本内核的模块可能在4.x版本上就无法使用。

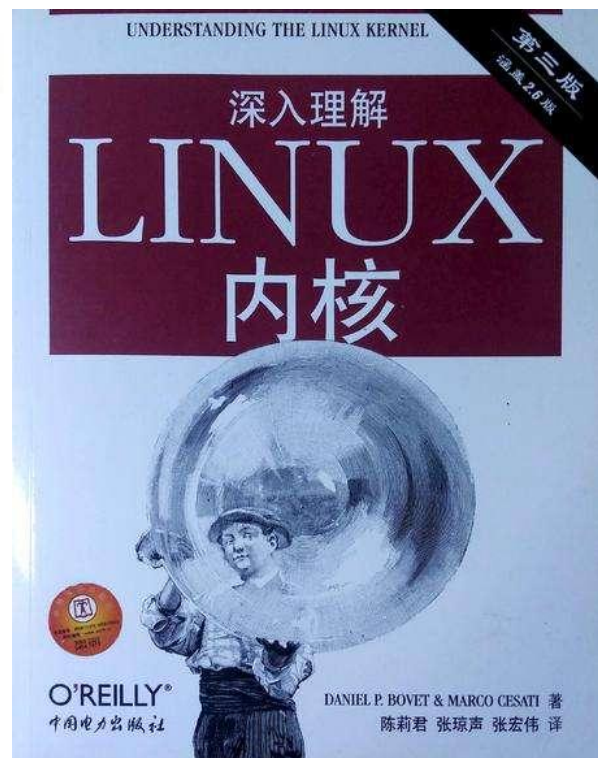
抽象API



抽象API

在某些情况下，内核过于底层，开发者需要更高一层的抽象。于是出现了类似Mesa 3D的为图形驱动开发而生的API。

参考资料



深入理解Linux内核 第三版第十章

带着思考离开



保持一个稳定的 ABI 和保持一个稳定的 API 相比，谁更困难，为什么？

谢谢大家！



THANK YOU