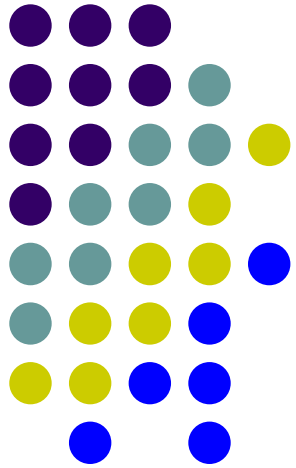
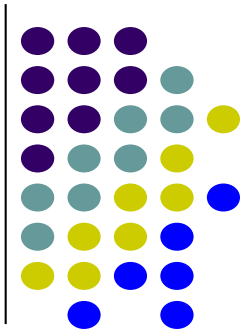
第2章

Linux内核

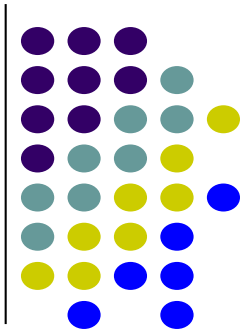


本章内容



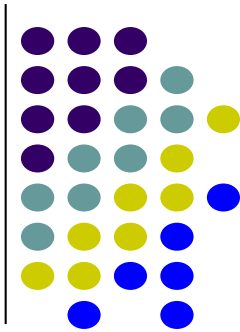
- 认识操作系统
- Linux内核体系结构
- Linux的系统调用

2.1 认识操作系统



- 从使用者的角度看操作系统
  - 打开计算机，首先跳入眼帘的是什么？
  - 要拷贝一个文件，具体的拷贝操作是谁完成的？
    - 需要知道文件存放在何处，在硬盘的那个柱面、磁道、扇区。
    - 数据的搬动过程怎样进行→复杂的I/O操作。
  - 繁琐留给自己，简单留给用户
    - 操作系统穿上华丽的外衣—图形界面
    - 操作系统穿上朴素的外衣—字符界面

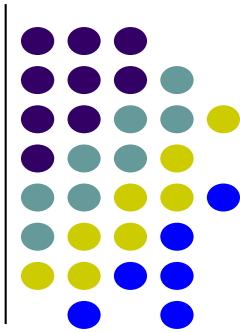
- 从程序开发者的角度看操作系统
  - 拷贝命令的C语言实现片断



```
inf=open(“/mnt/usbdisk/test”,O_RDONLY,0);
outf=open(“/mydir/test”,O_WRONLY,0600);
do{
    read_size=read(inf,buf,4096);
    write(outf,buf, read_size);
} while(read_size);
   close(outf);
   close(inf);
```

- 从操作系统设计者的角度看操作系统
  - 操作系统的设计目标是什么？
    - 让各种软件资源和硬件资源高效而协调地运转起来。
    - 尽可能地方便用户使用计算机。
  - 假设在一台计算机上有三道程序同时运行，并试图在一台打印机上输出运算结果，必须考虑哪些问题 ？
  - 从操作系统设计者的角度考虑，一个操作系统必须包含以下几部分
    - CPU管理
    - 内存管理
    - 设备管理
    - 文件管理

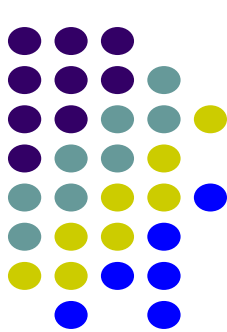
2.2 linux 内核结构



各部分功能介绍:

- 内核是整个操作系统的核心，管理着整个计算机系统的软硬件资源。内核控制整个计算机的运行。提供相应的硬件驱动程序和网络接口程序，并管理所有应用程序的执行，决定着系统的性能和稳定性。
- Linux内核采用模块化的结构，主要模块包括：存储管理、CPU和进程管理、文件系统管理、设备管理和驱动，网络通信、系统调用等。

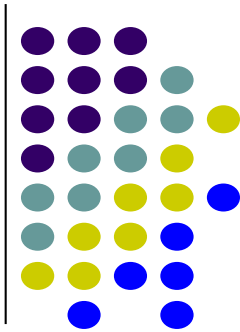
Linux内核的技术特点



1. Linux内核被设计成宏内核（Monolithic）结构。

- 所谓宏内核就是从整体上把内核作为一个大过程来实现，而进程管理、内存管理等是其中的一个个模块。模块之间可以直接调用相关函数。
- 宏内核由于全部功能集中在一块，系统花在内核功能的切换上（例如文件系统到IO驱动系统上的切换上）开销就非常小，提供给用户程序的反应就很快。相对于微内核，宏内核的Linux效率高，紧凑性强。
- 同时，因为全部功能集中在一块，各个功能之间的耦合度就很紧，导致了内核难以修改和增加新功能。

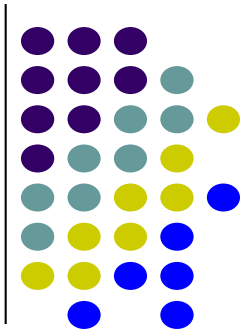
Linux内核的技术特点



2. Linux内核在2.6版本之前是单线程结构。

- 同一时间只允许一个执行线程在内核中执行，不会被调度程序打断而运行其他任务。这种内核称为非抢占式。其好处是内核中没有并发任务（单处理器），避免了许多复杂的同步问题。
- 但非抢占特征延迟了系统的响应速度，新任务必须等到当前任务执行完毕才能获得机会。
- 2.6版本将抢占技术引入了Linux内核，当然，付出的代价是同步变得更复杂。

Linux内核的技术特点

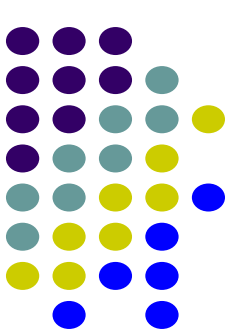


3. Linux内核支持动态加载内核模块。

为保证支持新设备、新功能，又不会无限扩大内核规模，Linux系统对设备驱动或新文件系统等采用了模块化的方式，用户在需要时可以现场动态加载，使用完毕可以动态卸载。

同时，用户可以定制、选择适合自己的功能，将不用的部分剔除内核。这些保证了内核的紧凑性、可扩展性。

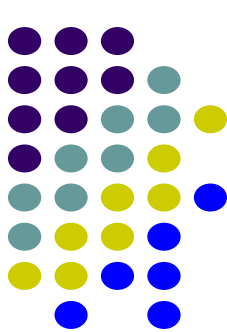
Linux内核的技术特点



4. linux简化了分段机制，使得虚拟地址与线性地址总是一致的。内核采用虚拟内存技术，使得内存空间达到4GB。其中0~3G属于用户空间，这个空间对系统中的其他进程是不可见的，称为用户段，3G~4G属于内核空间，称为内核段。因为每个进程可以通过系统调用进入内核，最高的1GB内核空间则为所有进程以及内核所共享。

这样，每个进程可以拥有4GB的虚拟地址空间（也叫虚拟内存）。应用程序就可以使用远远大于实际物理内存的存储空间了。

Linux内核的技术特点

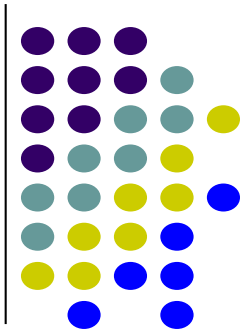


- 用户空间不是被进程共享，而是被进程隔离的。每个进程最大可以有3GB用户空间。所以说一个进程对一个地址的访问，与另一个进程对同一个地址的访问不冲突，因为尽管是同一个地址但因为，进程的用户空间不共享 导致他们其实并没有指向同一个地址。而对于cpu来讲，在任意的时刻，整个系统都只有4GB的虚拟地址空间，这个虚拟空间是面向进程的，所以当进程切换的时候，虚拟地址空间也会切换。所以只有此进程运行的时候，其虚拟地址空间才被CPU所知。其他时刻，其虚拟空间不被CPU所知。

Linux内核的技术特点

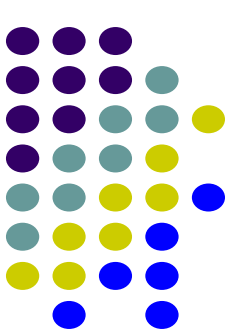
- 一个程序编译链接后形成的是虚拟地址空间，虚拟地址空间必须被映射到物理内存空间中，这个映射关系需要通过硬件体系结构所规定的数据结构来建立。即段描述符表和页表，而Linux主要通过页表来进行映射。
- 如果给出的页表不同，那么CPU将某一虚拟地址空间中的地址转化成的物理地址也不同，所以每个进程都建立了页表，将每个进程的虚拟地址空间根据自己的需要映射到物理地址空间上。既然在一个时刻CPU上只能有一个进程在运行，那么当进程发生切换时，将页表也更换为相应进程的页表，这就可以实现每个进程都有自己的虚拟地址空间而互不影响

Linux内核的技术特点



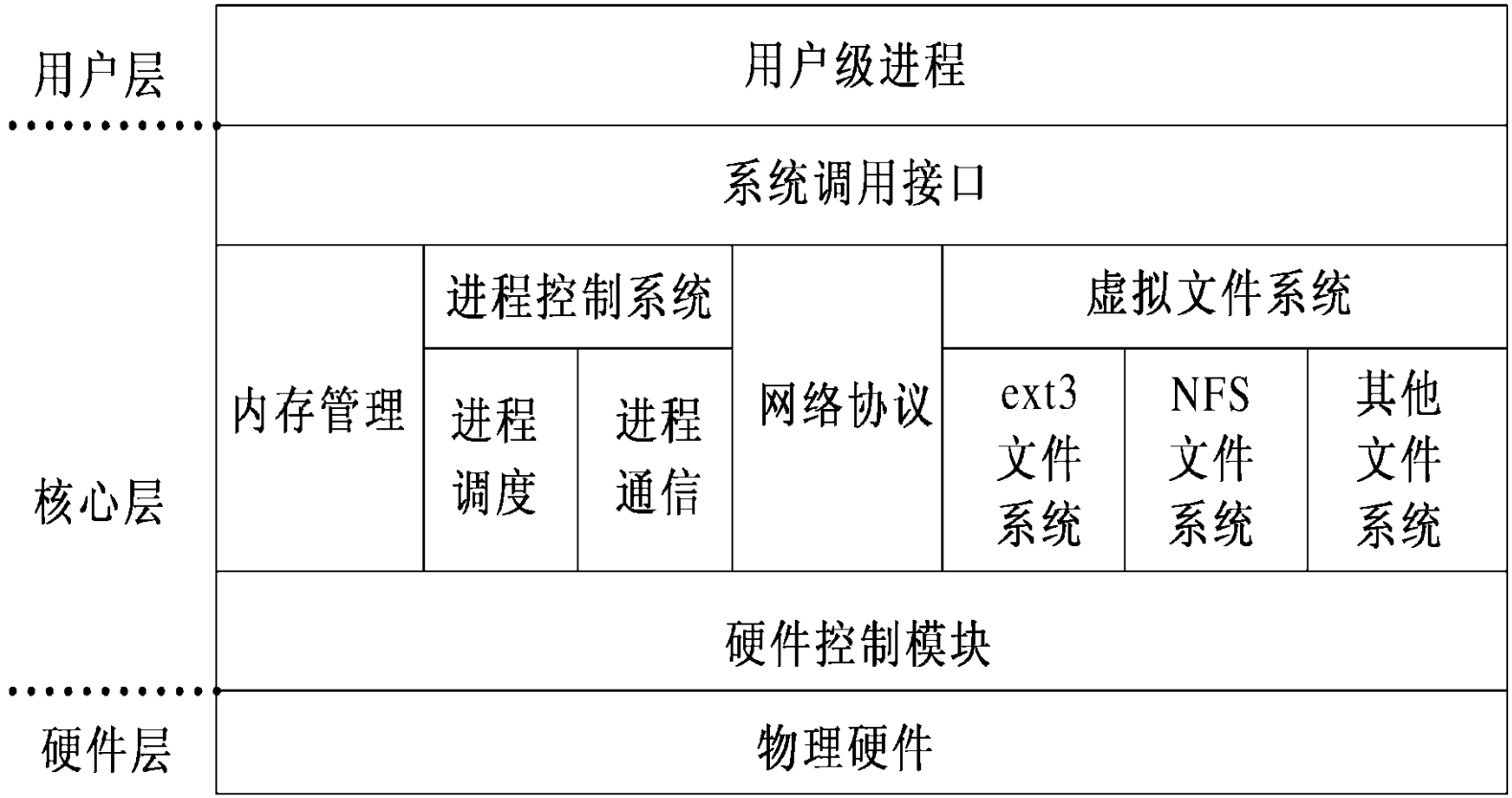
- 内核空间到物理空间内存的映射：
  - 内核空间占据了每个虚拟空间中的最高1GB，但映射到物理内存却总是从最低的地址开始的,所以3GB就是物理地址与虚拟地址之间的位移量，而在Linux代码中就叫做PAGE_OFFSET。
  - 对于内核空间而言，给定一个虚地址x，其物理地址为x-PAGE_OFFSET,给定一个物理地址x，其虚地址为x+PAGE_OFFSET。
  - 这只适合内核空间的虚地址映射到物理地址，而绝不适用于用户空间。

Linux内核的技术特点

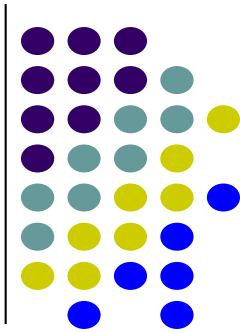


5. Linux的文件系统实现了一种抽象文件模型----虚拟文件系统（Virtual Filesystem Switch, VFS），VFS是Linux的特色之一。通过使用虚拟文件系统，内核屏蔽了各种文件系统的内在差别，使得用户可以通过统一的界面访问各种不同格式的文件系统。

- Linux系统的核心框图如图所示

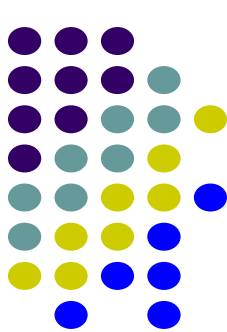


Linux内核子系统



- 进程调度－控制着进程对CPU的访问。
- 内存管理－允许多个进程安全地共享主内存区域
- 虚拟文件系统－隐藏各种不同硬供统一的接口。
- 网络接口－提供了对各种网络标准的存取和各种网络硬件的支持。
- 进程间通信(IPC)

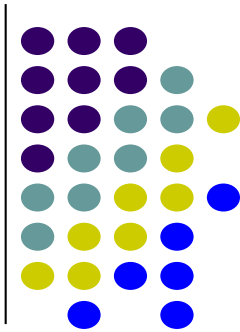
Linux内核组成



1. 进程调度

- 负责控制进程访问C P U。调度程序所使用的策略，可以保证进程能够公平地访问C P U，同时保证内核可以准时执行一些必需的硬件操作。
- 当需要选择下一个进程运行时，由调度程序选择最值得运行的进程。
- 可运行进程实际上是仅等待CPU资源的进程，如果某个进程在等待其它资源，则该进程是不可运行进程。
- Linux使用了基于优先级的进程调度算法选择新的进程。

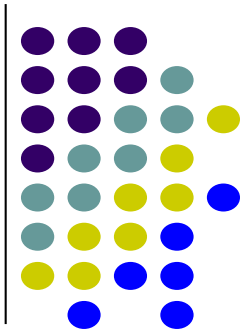
Linux内核组成



2. 内存管理

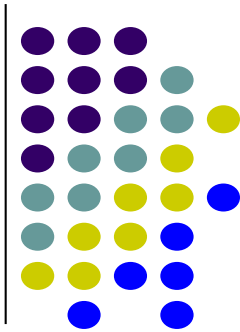
- 物理内存的管理：内存分配、回收等
- 使多个进程可以安全地共享机器的主存系统。此外，内核管理程序支持虚拟内存。
- Linux内核采用虚拟内存技术，使得Linux可以支持进程使用超过系统中的内存数量的内存。

Linux内核组成



3. 虚拟文件系统VFS，通过提供一个所有设备的公共文件接口，ＶＦＳ抽象了不同硬件设备的细节。此外，ＶＦＳ支持与其他操作系统兼容的不同的文件系统格式。实现了一种通用文件模型，为用户访问不同的文件系统提供统一的通用的接口。

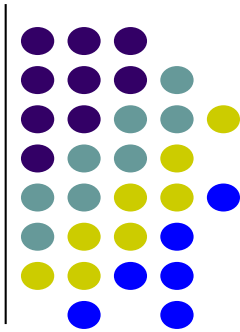
Linux内核组成



4. 网络接口

- 提供了对各种网络标准的存取和各种网络硬件的支持。
- 网络接口可分为网络协议和网络驱动程序。
  - 网络协议部分负责实现每一种可能的网络传输协议。
  - 网络设备驱动程序负责与硬件设备通讯，每一种可能的硬件设备都有相应的设备驱动程序。

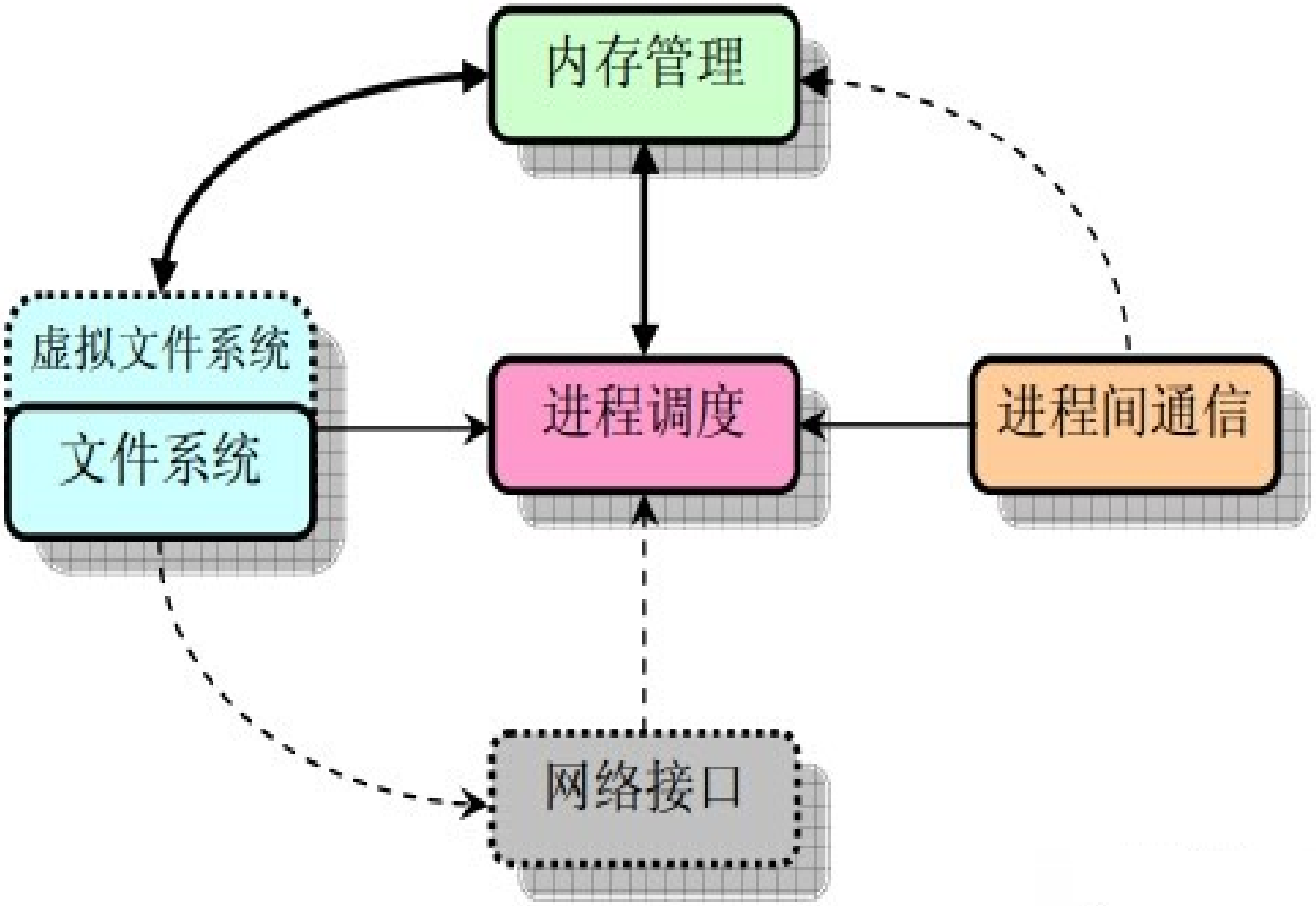
Linux内核组成



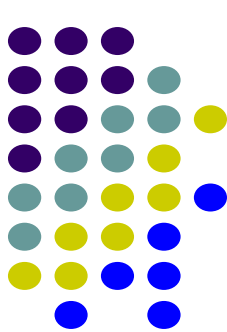
5. 进程间通信（IPC）

- 支持进程间各种通信机制，包括共享内存、消息队列及管道等。
- Linux的IPC是从Unix系统的进程间通讯机制移植过来的。

子系统之间关系



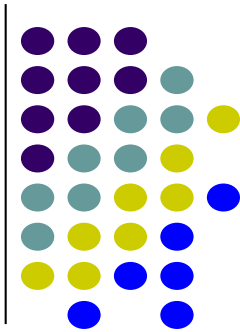
子系统之间关系



各个子系统之间的依赖关系

- 处于中心位置的进程调度，其它的子系统都依赖它，因为每个子系统都需要挂起或恢复进程。一般情况下，当一个进程等待硬件操作完成时，它被挂起；当操作真正完成时，进程被恢复执行。例如，当一个进程通过网络发送一条消息时，网络接口需要挂起发送进程，直到硬件成功地完成消息的发送，当消息被成功的发送出去以后，网络接口给进程返回一个代码，表示操作的成功或失败。其他子系统以相似的理由依赖于进程调度。

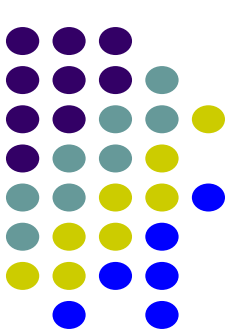
子系统之间关系



各个子系统之间的依赖关系

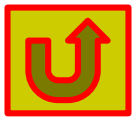
- 进程调度与内存管理之间的关系：这两个子系统互相依赖。在多道程序环境下，程序要运行必须为之创建进程，而创建进程的第一件事情，就是将程序和数据装入内存。
- 进程间通信与内存管理的关系：进程间通信子系统要依赖内存管理支持共享内存通信机制，这种机制允许两个进程除了拥有自己的私有空间，还可以存取共同的内存区域。

子系统之间关系

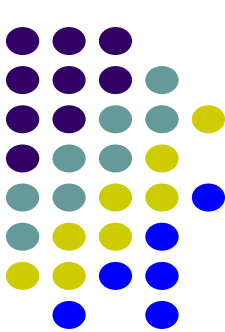


各个子系统之间的依赖关系

- 虚拟文件系统与网络接口之间的关系：虚拟文件系统利用网络接口支持网络文件系统(NFS)，也利用内存管理支持RAMDISK设备。
- 内存管理与虚拟文件系统之间的关系：内存管理利用虚拟文件系统支持交换，交换进程(swapd)定期由调度程序调度，这也是内存管理依赖于进程调度的唯一原因。当一个进程存取的内存映射被换出时，内存管理向文件系统发出请求，请求文件系统从永久性存储设备中去取该内存同时，挂起当前正在运行的进程。

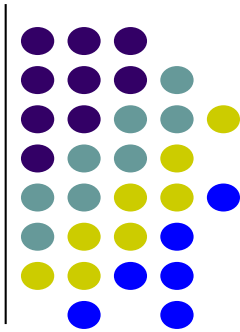


2.3 Linux的系统调用



- OS内核中都有一组实现系统功能的过程，系统调用就是对上述过程的调用。用户程序利用系统调用，向OS提出服务请求，由OS代为完成。
- 系统调用是操作系统与应用程序之间的一组“特殊”接口，是为用户程序或其它系统程序在执行过程中访问系统资源，调用系统功能建立的，是用户程序获得操作系统服务的唯一途径。

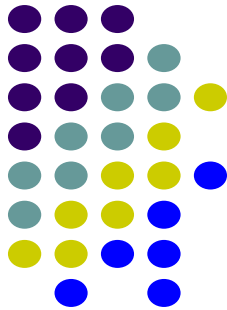
为什么需要系统调用



出于安全的考虑。

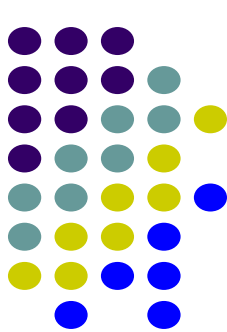
- 禁止用户程序和底层硬件直接打交道。
- 禁止用户程序任意访问物理内存。
- 用户进程不能随便的进入内核空间，访问内核变量和内核函数。
- 用户进程只能通过系统调用获取系统内核服务，规定了用户进程进入内核的具体位置，也就是用户访问内核的路径是事先规定好的，只能从规定位置进入内核，执行规定的内核函数。

系统调用



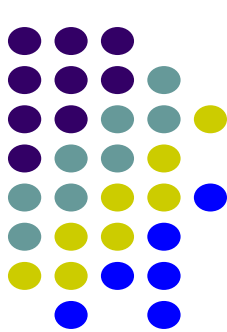
- 因此，系统调用像一个黑箱子那样，对用户屏蔽了操作系统的具体动作而只提供有关的功能。
- 系统调用本质上是通过特殊硬件指令和中断系统实现的，但它不是一条简单的硬件指令，而是带有一定功能号的“访管指令”。
- 它的功能的实现并非由硬件直接提供，而是由操作系统中的一段例行子程序完成的，即由软件方法实现的。

系统态与用户态



- 在计算机系统中存在两类不同的程序：一类是用户程序，一类是系统程序。而且用户程序必需在系统程序的控制和管理下运行。为了使计算机有条不紊地工作，在运行过程中对这两类不同的程序应该予以区分。
- 我们把用户工作的状态称为用户态或算态；将系统程序工作的状态称为管态或系统态。

访管指令



- 用户程序在算态下运行，只能使用算态指令，而操作系统是系统程序，在管态下运行，它既可使用算态指令，也能使用特权指令。而用户要使用外设，必须在管态下完成，因而引入访管指令。

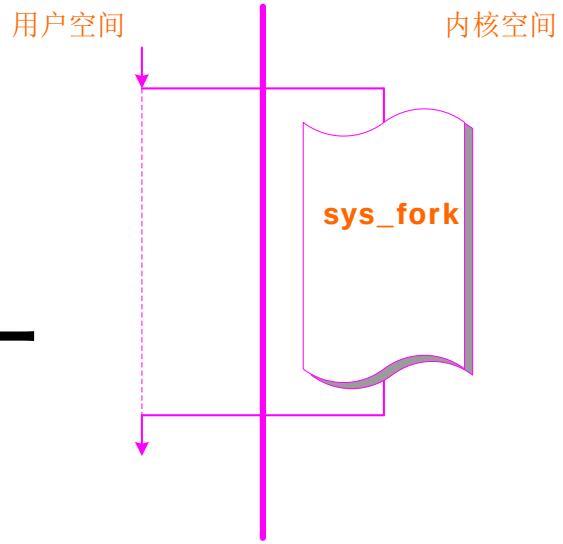
系统调用

用户程序如何同设备打交道？例如，用户需通过网卡发送数据

- 硬件被linux 内核隔离，只能通过内核实现。
- 不可能直接调用操作系统的函数：不可行，也不安全。
- Linux提供的解决方法：**系统调用**
- 从用户态切换到内核态，

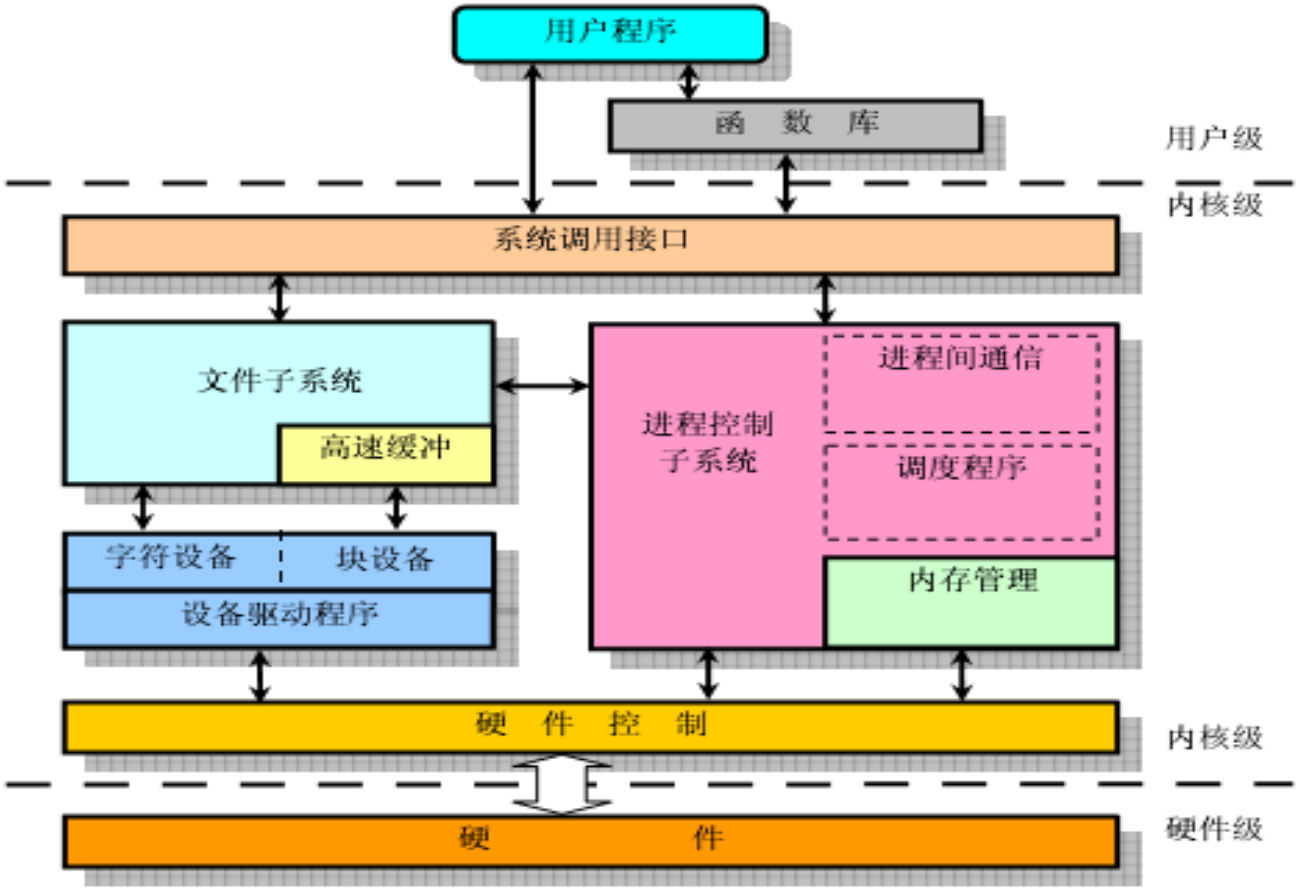
在内核态完成任务后再返回用户态

- 系统调用是用户态进入内核态的唯一入口：**一夫当关，万夫莫开**。

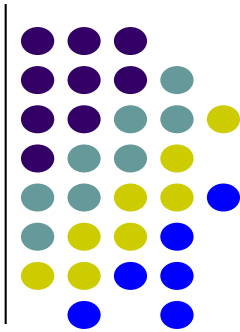


Linux的系统调用

- 程序员或系统管理员并非直接与系统调用打交道，在实际使用中程序员调用的是应用程序接口——API。
- Linux的API遵循POSIX标准，定义了一系列API，通过C库（glibc）实现。

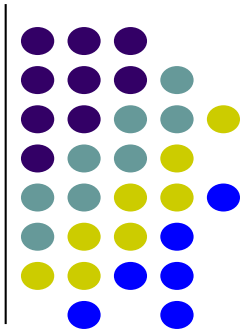


API和系统调用



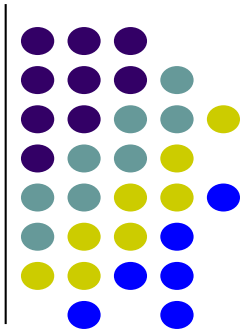
- API和系统调用是不同的
  - API是一个函数定义
  - 系统调用通过软中断向内核发出一个明确的请求
- Libc库除了定义标准的C函数外，还提供了一套封装例程(wrapper routine），将系统调用在用户空间封装后供用户编程使用，唯一目的就是发布系统调用。
  - 一般每个系统调用对应一个封装例程
  - 库再用这些封装例程定义出给用户的API

API和系统调用



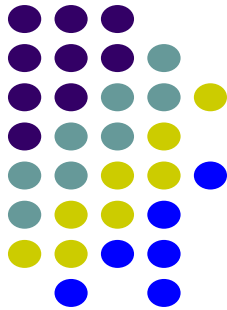
- 不是每个API都对应一个特定的系统调用。
  - 首先，API可能直接提供用户态的服务(比如一些数学函数)
  - 其次，一个单独的API可能调用几个系统调用
  - 不同的API可能调用了同一个系统调用

系统调用实现



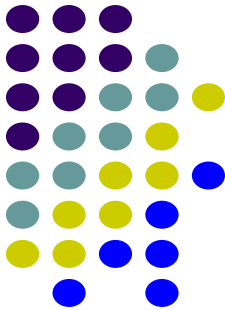
- linux系统有几百个系统调用，为了唯一的标识每一个系统调用，linux为每一个系统调用定义了一个唯一的编号，此编号就是系统调用号。
- 每一个系统调用bar()在内核都有一个对应的内核函数sys_bar()，这个内核函数就是系统调用bar()的实现，也就是说在用户态调用bar()，最终会有内核函数sys_bar()为用户服务，这里的sys_bar()就是系统调用的服务例程。

系统调用



- 那么为什么不直接调用内核函数呢？这是因为用户空间无法直接执行内核代码，因为内核驻留在受保护的地址空间上，不允许用户进程在内核地址空间上进行读写。所以，应用程序应该以某种方式通知系统，告诉内核自己需要执行一个系统调用，而这种机制是通过软中断实现的，通过引发一个异常促使系统切换到内核态去执行异常处理程序。此时的异常处理程序就是所谓的系统调用处理程序（中断处理程序）。

系统调用

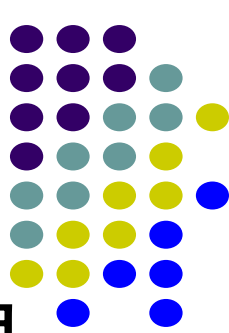


- 从程序调用的角度来看，用户应用程序中要调用一些子程序。子程序可以分为用户自己编写的子程序和软件提供的子程序，对后者的访问即被称为访问系统程序（访管）指令，又称为陷阱（TRAP）指令。
- 访管指令并不是特权指令。所谓特权指令，是指用于操作系统或其他系统软件的指令，一般不提供给用户使用。

系统调用

- 访管指令是让程序拥有“自愿进管”的手段，从而引起访管中断，主要功能为：□
  - 实现从算态到管态的改变；□
  - 在管态下由操作系统代替用户完成其请求；
  - 操作系统工作完成后由管态返回到算态。
- 自愿性中断事件（软中断）是正在运行的程序所期待的事件。这种事件是由于执行了一条访管指令而引起的，它表示正在运行的程序对操作系统有某种需求。一旦机器执行这一中断时，便自愿停止现行程序而转入访管中断处理程序处理。

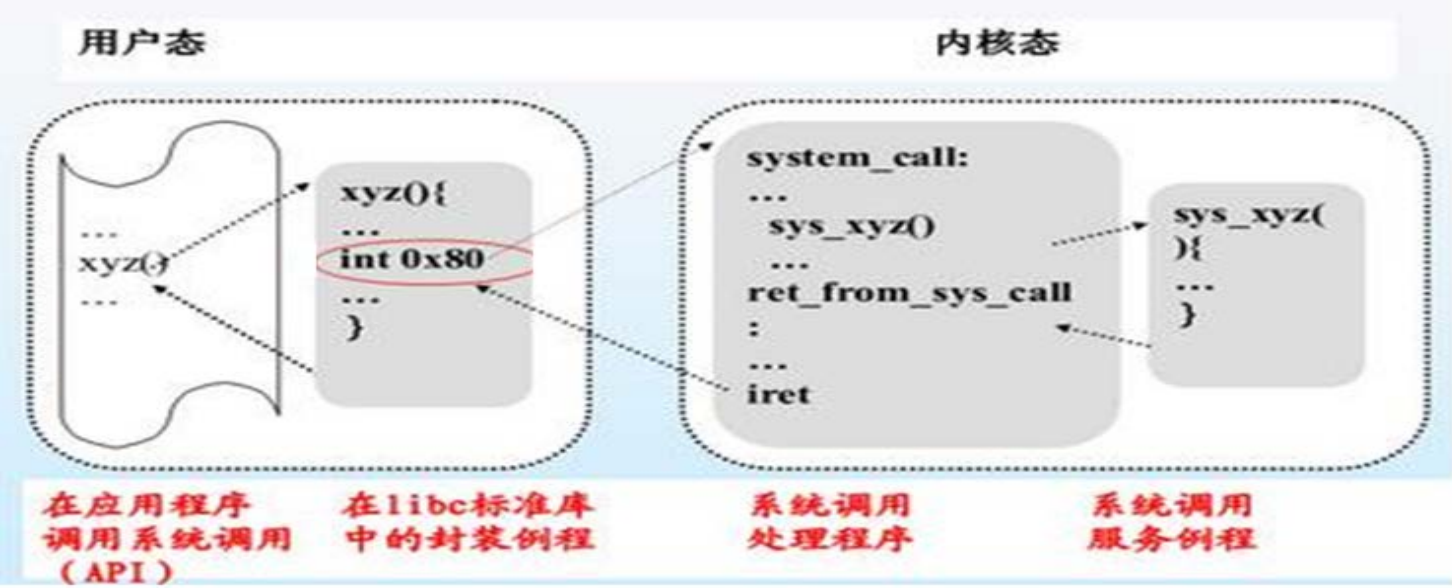
系统调用实现



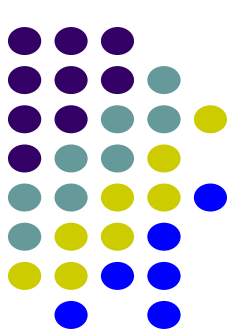
- 应用程序能直接调用的是系统提供的API，这个在用户态下就可做到。然后相应的API就会将相应的系统调用号保存到EAX寄存器中。
- 当用户态进程调用一个系统调用时，CPU切换到内核态并开始执行一个内核函数。
  - 在Linux中是通过执行int $0x80这条汇编语言来执行系统调用的，这条汇编指令产生向量为128的编程异常，CPU便切换到内核态执行内核函数，并将控制权交给系统调用过程的起点：
  - system_call（）。

system_call()函数

- system_cal()检查系统调用号，该号码告诉内核进程请求哪种服务。
- 内核进程查看系统调用表(sys_call_table)找到所调用的内核函数入口地址。
- 接着调用相应的函数，在返回后做一些系统检查，最后返回到进程。

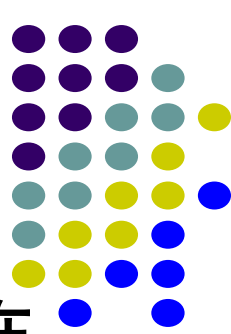


系统命令、内核函数



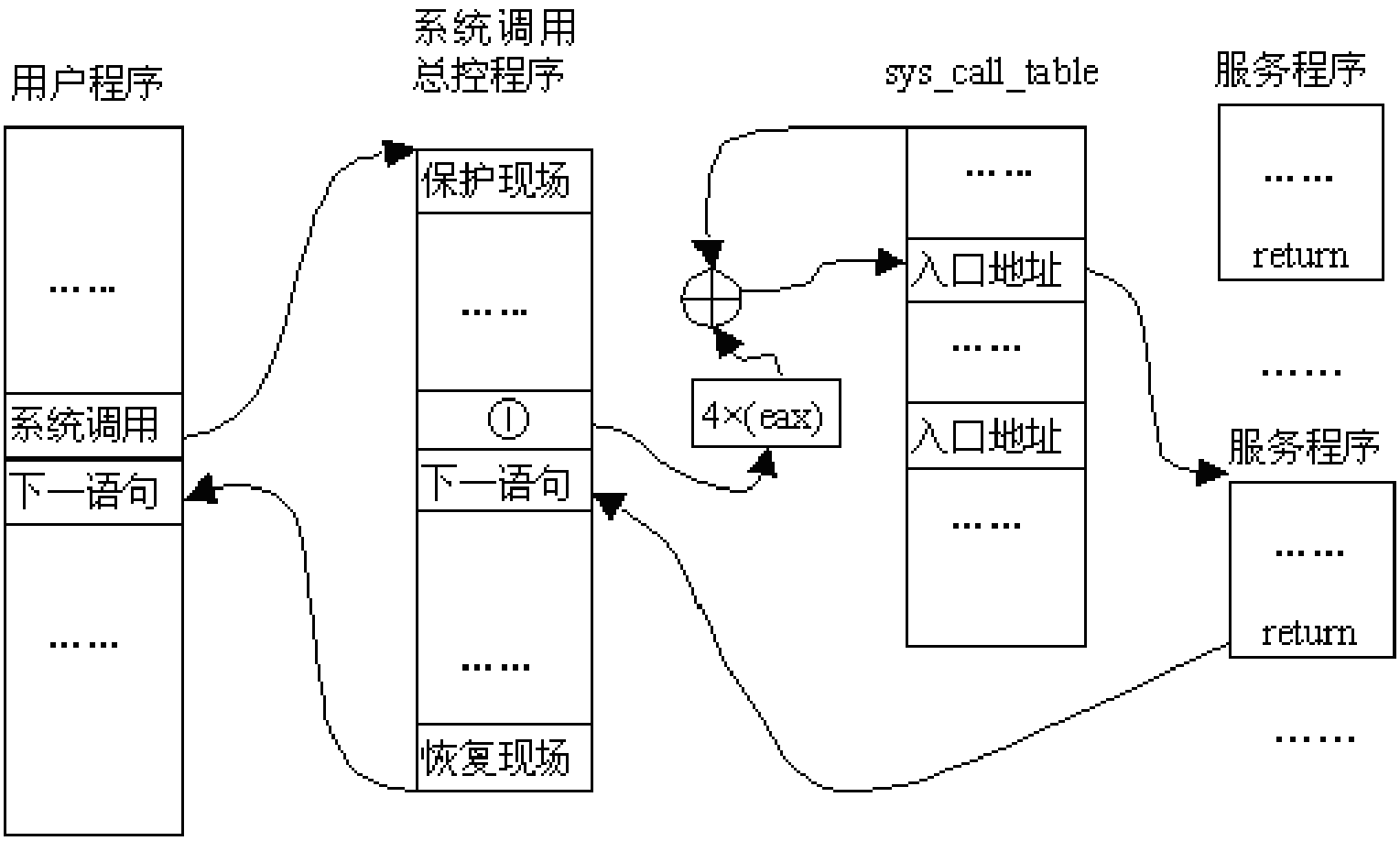
- 系统调用与系统命令
  - 系统命令相对API来说，更高一层。每个系统命令都是一个执行程序，如ls命令等。这些命令的实现调用了系统调用。
- 系统调用与内核函数
  - 系统调用是用户进入内核的接口层，它本身并非内核函数，但是它由内核函数实现。
  - 进入内核后，不同的系统调用会找到各自对应的内核函数，这些内核函数被称为系统调用的“服务例程”。如系统调用getpid实际调用的服务例程为sys_getpid()，或者说系统调用getpid()是服务例程sys_getpid()的封装例程。

系统调用表与调用号



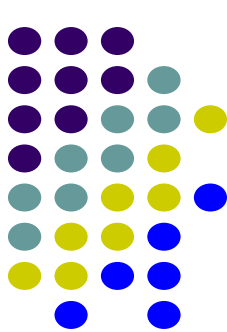
- 核心中为每个系统调用定义了一个唯一的编号，同时在内核中保存了一张系统调用表（ sys_call_table），该表中保存了系统调用编号和其对应的服务例程地址。第n个表项包含系统调用号为n的服务例程的地址。
- 系统调用陷入内核前，需要把系统调用号一起传入内核。这个传递动作是通过在执行int $0x80前把调用号装入eax寄存器实现。
- 这样系统调用处理程序一旦运行，就可以从eax中得到系统调用号，然后再去系统调用表中寻找相应服务例程。

系统调用过程



注①：此处语句为：　call * SYMBOL_NAME(sys_call_table)(,%eax,4)；eax 中为系统调用号

Linux的系统调用



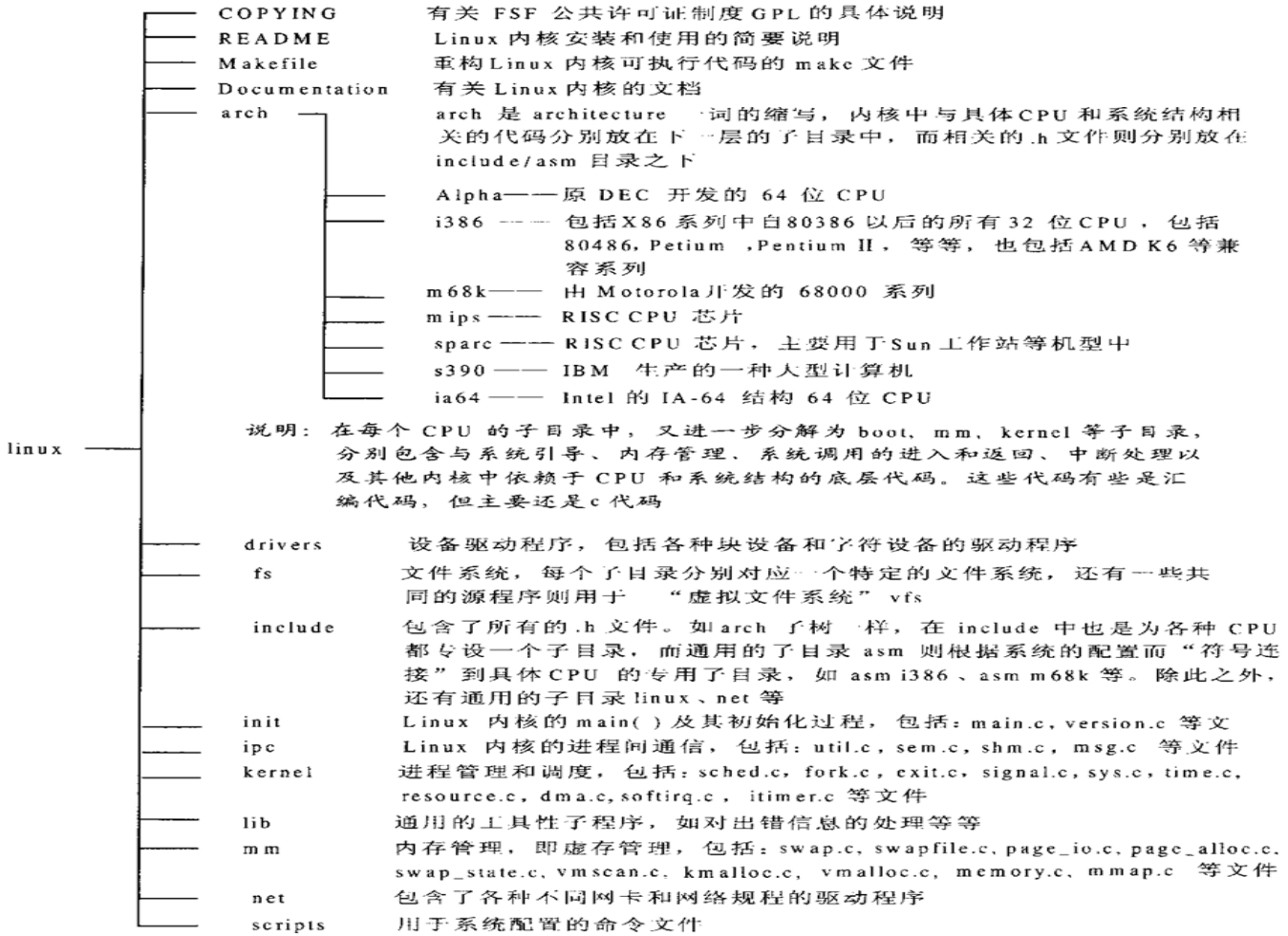
- Linux的系统调用跟很多Unix和windows系统相比，简洁和高效。（Linux设计精髓）
- Linux系统调用继承Unix的部分系统调用（最基本和最有用的系统调用），所以Linux全部系统调用只有300个左右。
- 按照功能大致可分为

进程控制	文件系统控制
系统控制	存储管理
网络管理	socket控制
用户管理	进程间通信

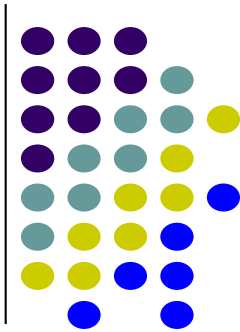
- 使用man 2 syscalls 命令查看系统调用的说明

Linux内核源码分析

- 内核源程序代码安装在/usr/src/linux目录下，该目录下还有几个其它目录，每一个都代表一个特定的内核功能性子集。

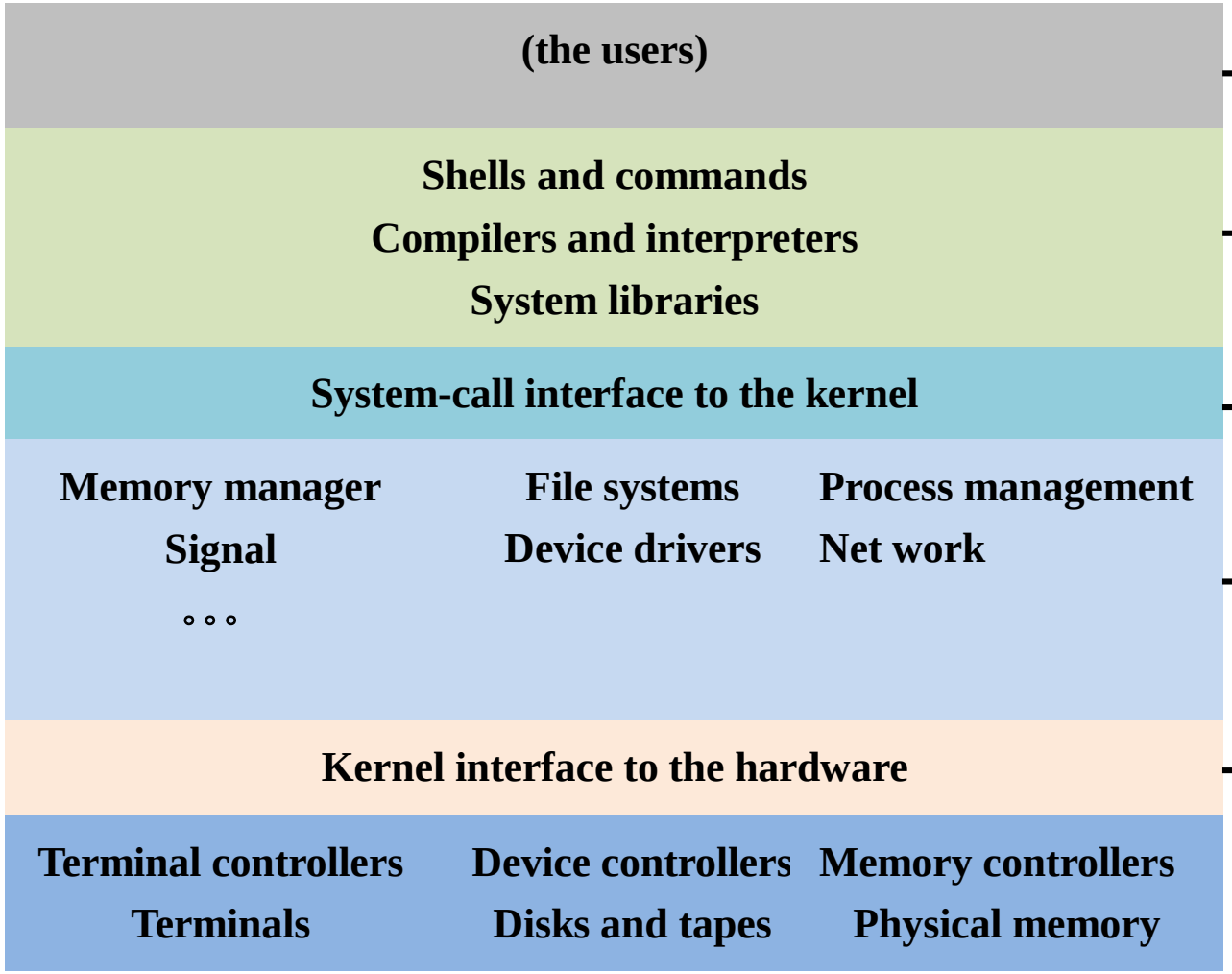


内核源代码结构



- Linux 内核源代码的结构
  - Linux内核源代码位于/usr/src/linux目录下
  - /include子目录包含了建立内核代码时所需的大部分包含文件
  - /init 子目录包含了内核的初始化代码
  - /arch子目录包含了所有硬件结构特定的内核代码
  - /drivers子目录包含了内核中所有的设备驱动程序
  - /fs子目录包含了所有的文件系统的代码
  - /net子目录包含了内核的网络连接代码
  - /mm子目录包含了所有内存管理代码
  - /ipc子目录包含了进程间通信代码
  - /kernel子目录包含了主内核代码

一个典型的Linux操作系统的结构



用户应用程序

Shell，lib

System call

Kernel implementation

对硬件资源的管理

最简单也是最复杂的操作