清華大學
Tsinghua University

《嵌入式系统》

4-2 嵌入式操作系统概述

清華大學
Tsinghua University

本章提纲

- *嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

清華大學
Tsinghua University

本章提纲

- *嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

清華大學 Tsinghua University

嵌入式 Linux 简介

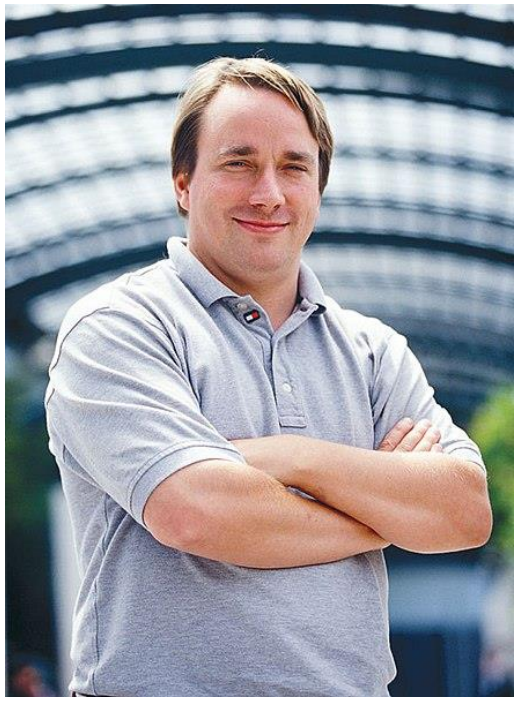
- Linux 的诞生与发展
  - Linux 的诞生
  - Linux 的发展
- Linux 在嵌入式领域的延伸

Linux 的诞生

- 1991年，Linus·Torvalds 在赫尔辛基大学上学时，对操作系统很好奇。由于当时386BSD还没有出来，同时他又不喜欢他的386电脑上的MS-DOS操作系统，所以就安装了 Minix，可对 MINIX 只允许在教育上使用很不满（在当时 MINIX 不允许被用作任何商业使用），于是他便开始写他自己的操作系统。

- Linux 的第一个版本在1991年9月被大学 FTP server 管理员 Ari Lemmke 发布在 Internet 上，最初 Torvalds 称这个内核的名称为“Freax”，意思是自由(“free”)和奇异(“freak”)的结合字，并且附上“X”这个常用的字母，以配合所谓的类 Unix 的系统。但是 FTP 服务器管理员嫌原来的命名“Freax”的名称不好听，把内核的称呼改成“Linux”，当时仅有10000行程序码，仍必须运行于 Minix 操作系统之上，并且必须使用硬盘开机；随后在10月份第二个版本(0.02版)发布。

- 1994年3月，Linux1.0 版正式发布。为了让 Linux 可以在商业上使用，Linus·Torvalds 决定更改他原来的协议(这个协议会限制商业使用)，以 GNU GPL 协议来代替。之后许多开发者致力融合 GNU 元素到 Linux 中，做出一个有完整功能的、自由的操作系统。



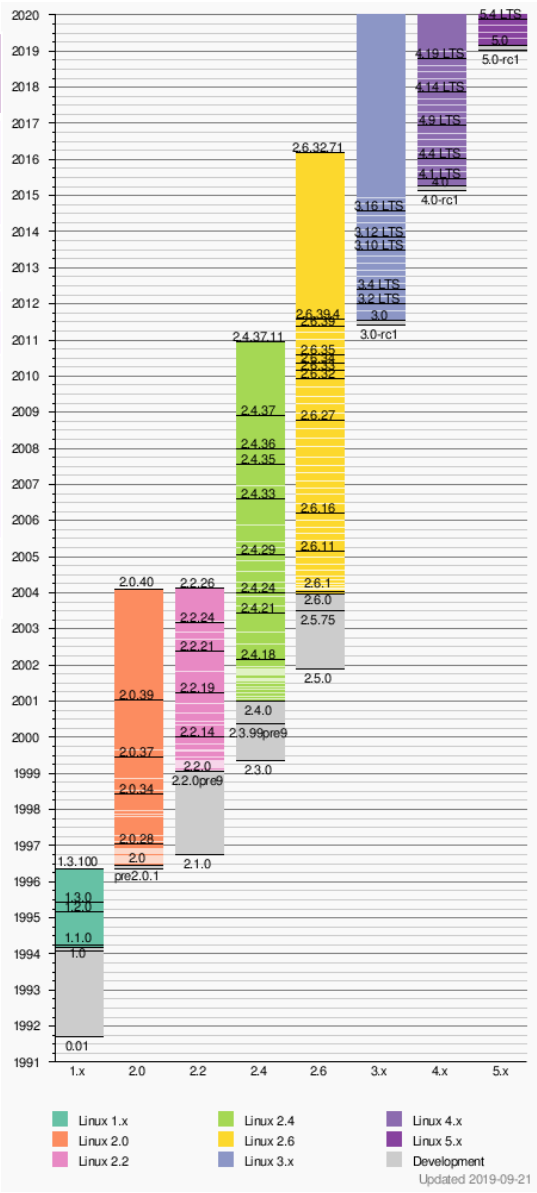
Linux 的诞生

- LINUX 诞生和成长的五大重要支柱
  - UNIX 操作系统 -- UNIX 于 1969 年诞生在 Bell 实验室。Linux 就是 UNIX 的一种克隆系统。
  - MINIX 操作系统 -- Minix 操作系统也是 UNIX 的一种克隆系统，它于 1987 年由著名计算机教授 Andrew S. Tanenbaum（AST） 开发完成。由于 MINIX 系统的出现并且提供源代码(只能免费用 于大学内)在全世界的大学中刮起了学习 UNIX 系统旋风。Linux 刚开始就是参照 Minix 系统于 1991 年才开始开发。
  - GNU 计划（项目、工程）-- 开发 Linux 操作系统，以及 Linux 上所用大多数软件基本上都出自 GNU 计划。Linux 只是操作系统的一个内核，没有 GNU 软件环境(比 如说 bash shell)，则 Linux 将寸步难行。
  - INTERNET -- 如果没有Internet，没有遍布全世界的无数计算机骇客的无私奉献，那么 Linux 最多只能发展到 0.13(0.95)版的水平。
  - POSIX 标准 -- 该标准在推动 Linux 操作系统以后朝着正规路上发展，起着重要的作用，是 Linux 前进的灯塔。

Linux 的发展

版本	初始发布日期	当前版本	维护者	维护支持情况
5.0	3 Mar 2019	5.0.21	Greg Kroah-Hartman	EOL (Mar 2019 to Jun 2019)
5.1	5 May 2019	5.1.21	Greg Kroah-Hartman	EOL (May 2019 to Jul 2019)
5.2	7 Jul 2019	5.2.20	Greg Kroah-Hartman	EOL (Jul 2019 to Oct 2019)
5.3	15 Sep 2019	5.3.18	Greg Kroah-Hartman	EOL (Sep 2019 to Dec 2019)
5.4	24 Nov 2019	5.4.17	Greg Kroah-Hartman & Sasha Levin	20th LTS (Nov 2019 to Dec 2021)
5.5	19 Jan 2020	5.5.1	Linus Torvalds	Preview release

- 从 Linux 诞生开始，Linux 内核就从来没有停止过升级，从 0.02 版本到 1999 年具有里程碑意义的 2.2 版本，一直到我们现在看到的最新的 5.5.1 版本。



Linux 在嵌入式领域的延伸

- 由于 Linux 具有对各种设备的广泛支持性，因此，能方便地应用于机顶盒、信息化家电设备、PDA、掌上电脑、WAP 手机、寻呼机、车载盒以及工业控制等智能信息产品中。
- 与 PC 相比，手持设备以及信息家电的市场容量要高得多，而 Linux 嵌入式系统的强大的生命力和利用价值，使越来越多的企业和高校表现出对它极大的研发热情。
- Linux 嵌入式操作系统所具有的技术优势和独特的开发模式给业界以新意，有理由相信，它能成为Internet时代嵌入式操作系统中的最强音。

嵌入式 Linux 操作系统

- μCLinux
  - μ: Micro. C: Control.

    μCLinux: Micro-Control-Linux
  - 针对微控制领域而设计的 Linux 系统。
  - μCLinux 是 Lineo 公司的主打产品，同时也是开放源码的嵌入式 Linux 的典范之作。
  - μCLinux 秉承了标准 Linux 的优良特性，经过各方面的小型化改造，形成了一个高度优化的、代码紧凑的嵌入式 Linux。虽然它的体积很小，没有MMU，却仍然保留了Linux的大多数优点：
    - 稳定、良好的移植性
    - 优秀的网络功能
    - 对各种文件系统完备的支持和标准丰富的 API

- μCLinux
  - 最初的 μCLinux 仅支持 Palm 硬件系统，基于 Linux 2.0 内核。随着系统的日益改进，支持的内核版本从2.0、2.2、2.4一直到最后的2.6（EOL版本）。
  - 最新版本: (2016-09-19)
    https://sourceforge.net/projects/uclinux/files/
  - 官网: www.uclinux.org (已经无法访问)
    - WebArchive:
      http://web.archive.org/web/20181113230737/http://www.uclinux.org/

嵌入式 Linux 操作系统

- RT-Linux
  - Real-time Linux
  - RT-Linux 是源代码开放的具有硬实时特性的多任务操作系统，它部分支持POSIX. 1b标准。
  - RT-Linux 是美国新墨西哥州大学计算机科学系 Victor Yodaiken 和 Micae Brannanov 开发的嵌入式Linux操作系统。
  - RTLinux **通过硬件和操作系统间的中断控制来支持硬实时（确定性）操作**。进行确定性处理所需要的中断由实时核心管理，其他中断被送往非实时操作系统。**操作系统运行为低优先级线程**。先进先出管道（FIFOs）或共享内存可以被用来在操作系统和实时核心之间共享数据。

- RT-Linux
  - RT-Linux 被 Victor Yodaiken 和 Micae Brannanov 开发后，成为 FSMLabs 的商业产品。
  - 2007 年 2 月， Wind River 收购了 FSMLabs 的嵌入式技术，并发布了 Wind River Real-Time Core for Wind River Linux。
  - 2011 年 8 月， Wind River 停止了此产品系列。
  - 官网: http://www.rtlinuxfree.com/ （仅保留域名）

嵌入式 Linux 操作系统

- 红旗嵌入式 Linux
  - 由北京中科红旗软件技术有限公司推出，是国内做得较好的一款嵌入式Linux操作系统。
    - 精简内核，适用于多种常见的嵌入式CPU；
    - 提供完善的嵌入式GUI和嵌入式X-Windows；
    - 提供嵌入式浏览器、邮件程序和多媒体播放程序；
    - 提供完善的开发工具和平台。

嵌入式 Linux 操作系统

❑红旗嵌入式 Linux

- ❑1980年代末，个人电脑开始进入中国，当时包括中国政府部门的在内的所有个人电脑几乎全部是安装MS-DOS操作系统。1992年海湾战争和1999年北约入侵南斯拉夫联盟科索沃地区时，成功运用信息战瘫痪了对方几乎所有通讯系统，这使得中国政府很多人认为，由于伊拉克和南联盟各部门使用的电脑操作系统百分之百是微软和其它外国公司的操作系统，虽然没有证据说明美国的计算机软件公司和通讯公司在这场信息战中向美国军方提供了某些后门或计算机病毒，但如果有自己独立的电脑操作系统及相应的软件，在信息战中将比较不容易受到攻击。于是中国科学院软件研究所奉命研制基于Linux的自主操作系统，并于1999年8月发布了红旗Linux 1.0版。红旗Linux最初主要用于关系国家安全的重要政府部门。
- ❑至2018年全大陆有7万台以上ATM机采用红旗Linux。
- ❑官网: http://www.redflag-linux.com/

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

概念回顾

- **内存管理**，是指软件运行时对计算机内存资源的分配和使用。其最主要的目的是如何高效，快速的分配，并且在适当的时候释放和回收内存资源。
- 通常包括以下功能机制：
  - 地址映射
  - 内存空间分配
  - 地址访问限制（/保护机制）
  - 其他……

进程虚存空间的管理

Linux 的存储管理主要是管理进程虚拟内存的用户区

- 进程运行时能访问的存储空间只是它的虚拟内存空间。对当前该进程而言只有属于它的虚拟内存是可见的。
- Linux 操作系统采用了请求式分页存储管理方法。系统为每个进程提供 **4GB 的虚拟内存空间**。各个进程的虚拟内存彼此独立。
- 每一个进程，用一个 mm_struct 结构体来定义它的虚存用户区 mm_struct 结构体首地址在任务结构体 task_struct 成员项 mm 中：struct mm_struct *mm。

mm_struct 结构定义在 /include/linux/schedul.h 中

虚拟存储技术

- 虚拟存储技术——用户的代码和数据（可执行映像）等并不是完整地装入物理内存，而是全部映射到虚拟内存空间。在进程需要访问内存时，在虚拟内存中“找到”要访问的程序代码和数据等。
- 虚拟存储技术的设计动机
  - 程序访问的局部性原理（时间上，空间上）
  - 运行比内存还要大的程序
  - 便于实现内存共享
  - 缩短程序加载、启动的时间
  - 多程序同时驻留内存，提高CPU利用率
  - 减轻程序员负担，简化其内存管理负担
  - ……

物理内存空间的管理

物理内存的页面管理

- Linux 对物理内存空间按照分页方式进行管理，把物理内存划分成大小相同的物理页面（如4KB）。
- Linux设置了一个mem_map[] 数组管理内存页面。mem_map[] 在系统初始化时由free_area_init()函数创建，它存放在物理内存的底部（低地址部分）

空闲页面的管理

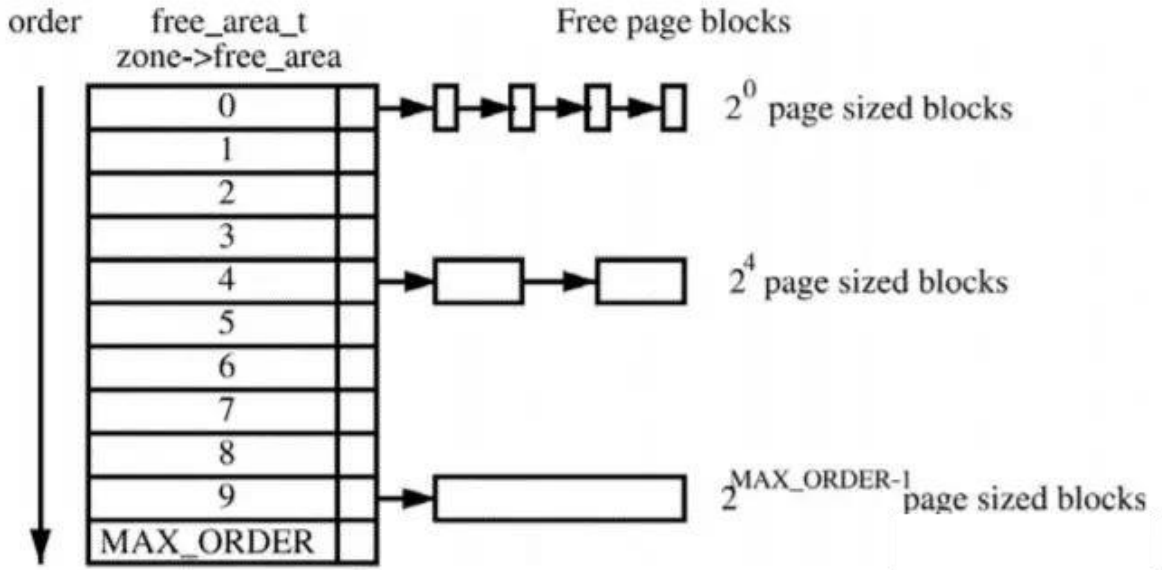
- Buddy 算法

内存空闲空间的管理采用Buddy 算法（接下页）

Buddy 算法

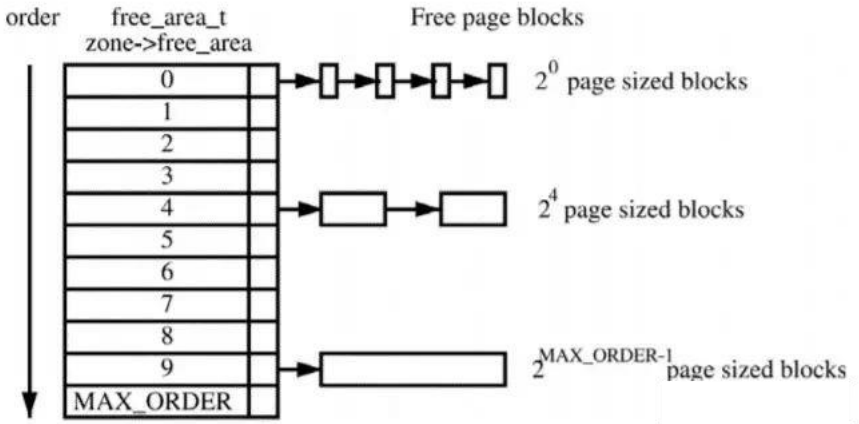
- 假设这是一段连续的页（称为块），阴影部分表示已经被使用的页，现需要申请一个连续的5个页
  - 问题：这个时候，在这段内存上不能找到连续的5个空闲的页，就会去另一段内存上去寻找5个连续的页，这样子，久而久之就形成了页的浪费
- 为了避免出现这种情况，Linux内核中引入了Buddy算法

Buddy 算法



- 把所有的空闲页分组为11个块链表，每个块链表分别包含大小为1，2，4，8，16，32，64，128，256，512和1024个连续页的块
- 最大可以申请1024个连续页的块，对应4MB大小的连续内存。每个块的第一个页的物理地址是该块大小的整数倍

Buddy 算法



- 假设要申请一个256个页的块，先从256个页框的链表中查找空闲块，如果没有，就去512个页的链表中找，找到了则将该块分为2个256个页的块，一个分配给应用，另外一个移到256个页的链表中。如果512个页的链表中仍没有空闲块，继续向1024个页的链表查找，如果仍然没有，则返回错误。块在释放时，会主动将两个连续的块合并为一个较大的块
- 算法核心：世界上任何正整数都可以由2^n的和组成

ARM-Linux 的内存管理和映射模型

- ARM体系结构MMU
  - ARM7及以下不支持MMU
  - ARM9及以上支持MMU
- **虚拟地址到物理地址的地址映射模型**有两种：
  - 单层的**段映射** / 二层的**页面映射**
- 段的大小是1MB
- 页的大小可以是以下几种之一：
  - 64KB的大页面/4KB的小页面/1KB的细小页面
  - 64KB和4KB的页面映射称为粗页面映射
  - 1KB的页面映射称为细页面映射

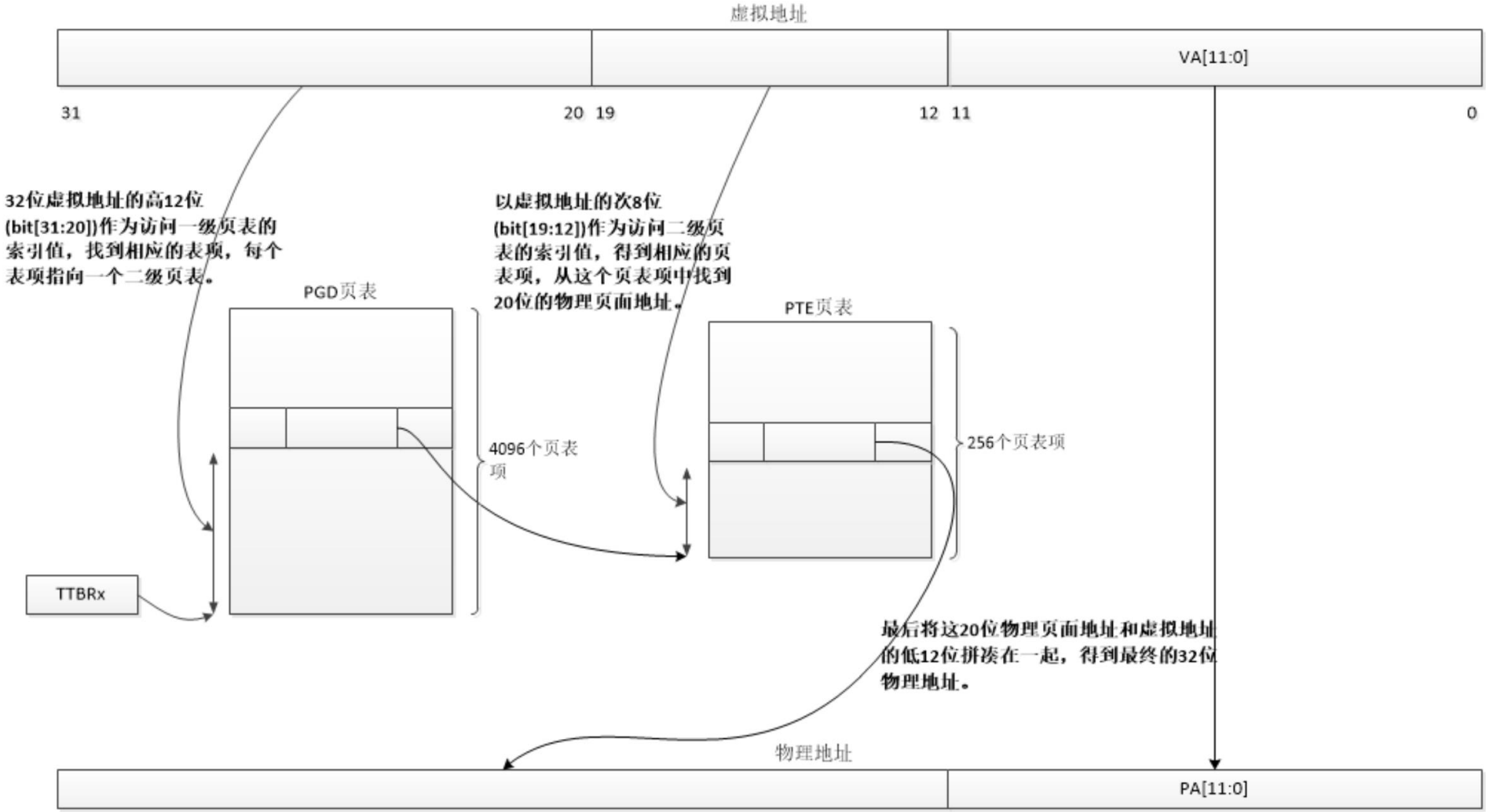
单层映射——段映射

- 内存中的段映射表共有4096个表项，每个描述项长度为4B，所以整个映射表，大小为16KB，而且，位置必须与16KB边界对齐。
- 当CPU访问内存时，其32位虚拟地址的高12位被用作访问段映射表的下标，从表中找到相应的表项。每个表项提供了12位的物理段地址，以及对这个段的访问许可标志。将这12位物理段地址与虚拟地址的低20位拼接在一起，就得到32位的物理地址。
- 整个过程由MMU硬件完成，无需CPU介入。

两层映射——页面映射

- 如果采用页面映射的方式，段映射表就变成一级映射表(Linux中称为PGD)，其页表项提供的不再是物理地址，而是二级页表的基地址。如果是4KB的页面，则二层映射表中有256个表项。
- 32位虚拟地址的**高12位**(bit[31:20])作为访问一级页表的索引值，找到相应的表项，每个表项指向一个二级页表。
- 以虚拟地址的**次8位**(bit[19:12])作为访问二级页表的索引值，得到相应的页表项，从这个页表项中找到**20位**的物理页面地址。
- 最后将这**20位**物理页面地址和虚拟地址的**低12位**拼凑在一起，得到最终的32位物理地址。
- 同样，整个过程由MMU硬件完成，CPU并不介入。

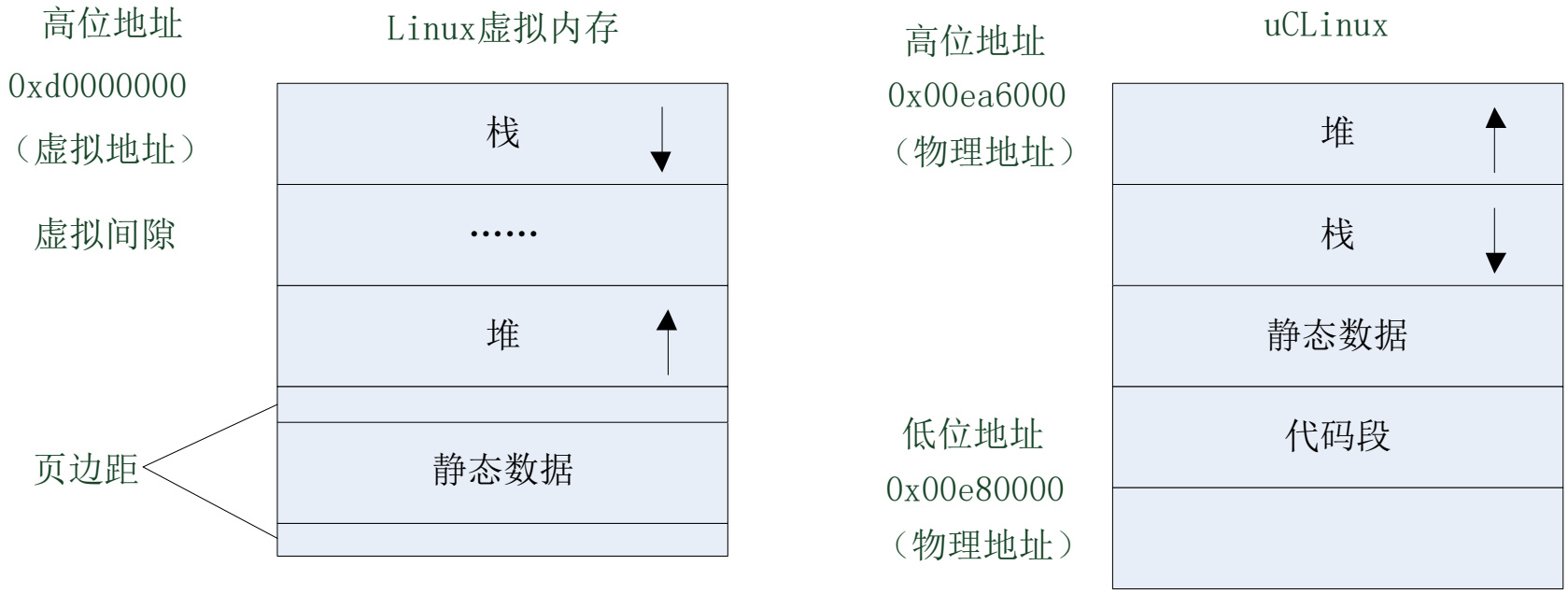
两层映射——页面映射图示


虚拟地址
VA[11:0]
31
20
19
12
11
0
32位虚拟地址的高12位(bit[31:20])作为访问一级页表的索引值，找到相应的表项，每个表项指向一个二级页表。
以虚拟地址的次8位(bit[19:12])作为访问二级页表的索引值，得到相应的页表项，从这个页表项中找到20位的物理页面地址。
PGD页表
PTE页表
4096个页表项
256个页表项
TTBRx
最后将这20位物理页面地址和虚拟地址的低12位拼凑在一起，得到最终的32位物理地址。
物理地址
PA[11:0]

μCLinux 的内存管理

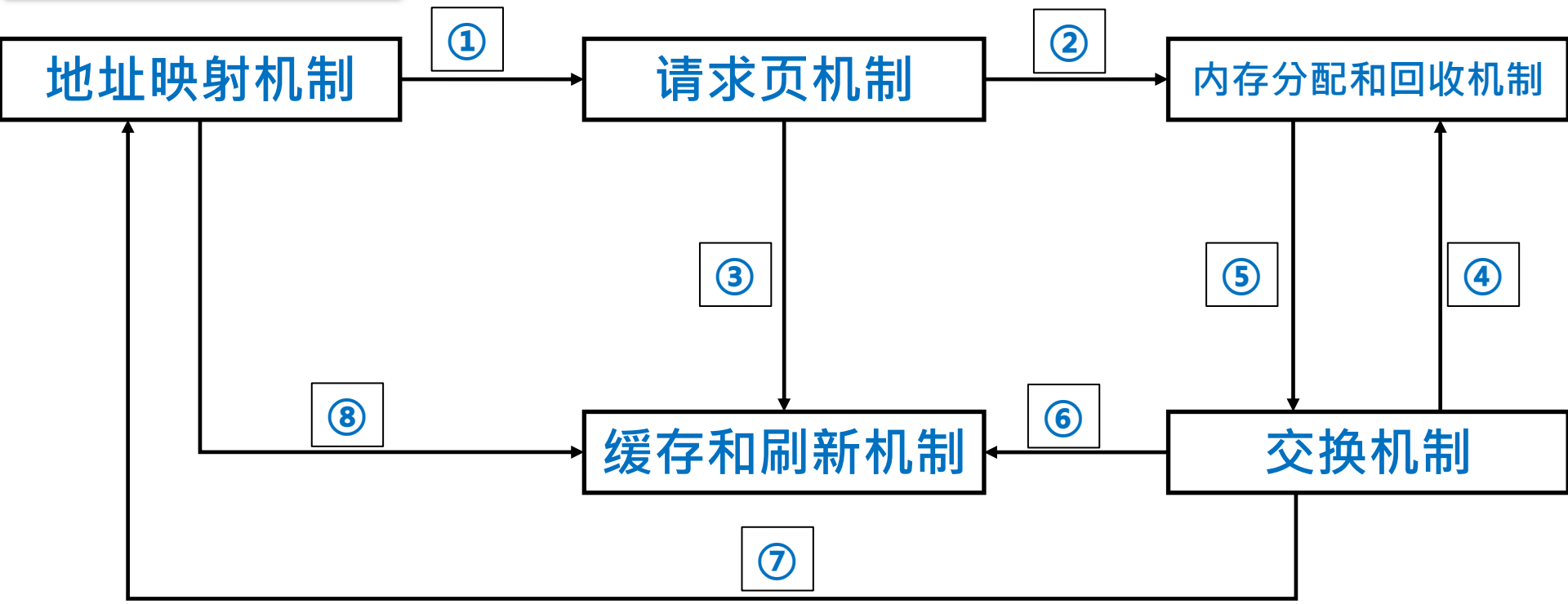
- μCLinux 专门针对没有 MMU 的处理器。
- μCLinux 中，系统为进程分配的内存区域是连续的，代码段、数据段和栈段间没空隙。为节省内存，进程的私有堆被取消，所有进程共享一个由操作系统管理的堆空间。
- μCLinux 不能使用处理器的虚拟内存管理技术，它仍然采用存储器的分页管理：实存储器管理（Real Memory Management）。

μCLinux 和标准 Linux 的内存映射比较



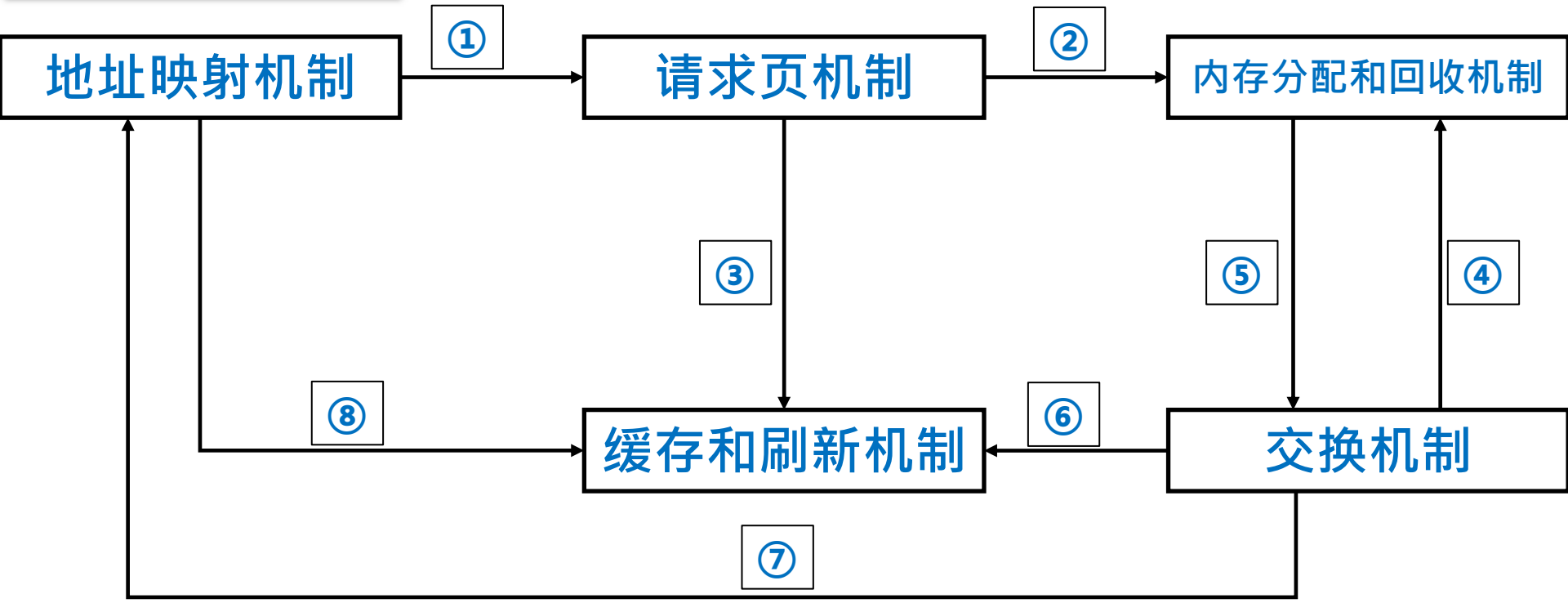
- μCLinux操作系统对内存空间没有保护，各个进程没有独立的地址转换表，共享一个运行空间。

ARM-Linux 的虚拟内存管理



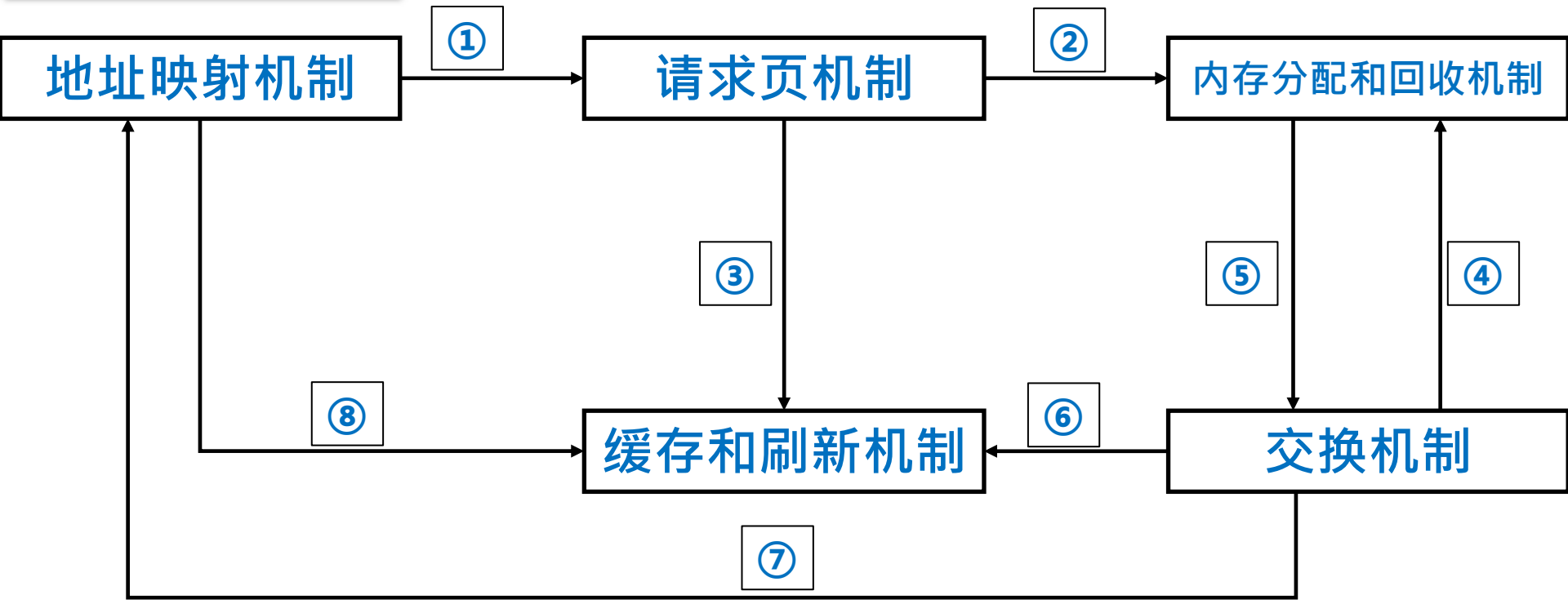
当程序运行时，如果程序发现要用的虚拟地址没有对应的物理内存，就会发出**请求页**要求①

ARM-Linux 的内存管理



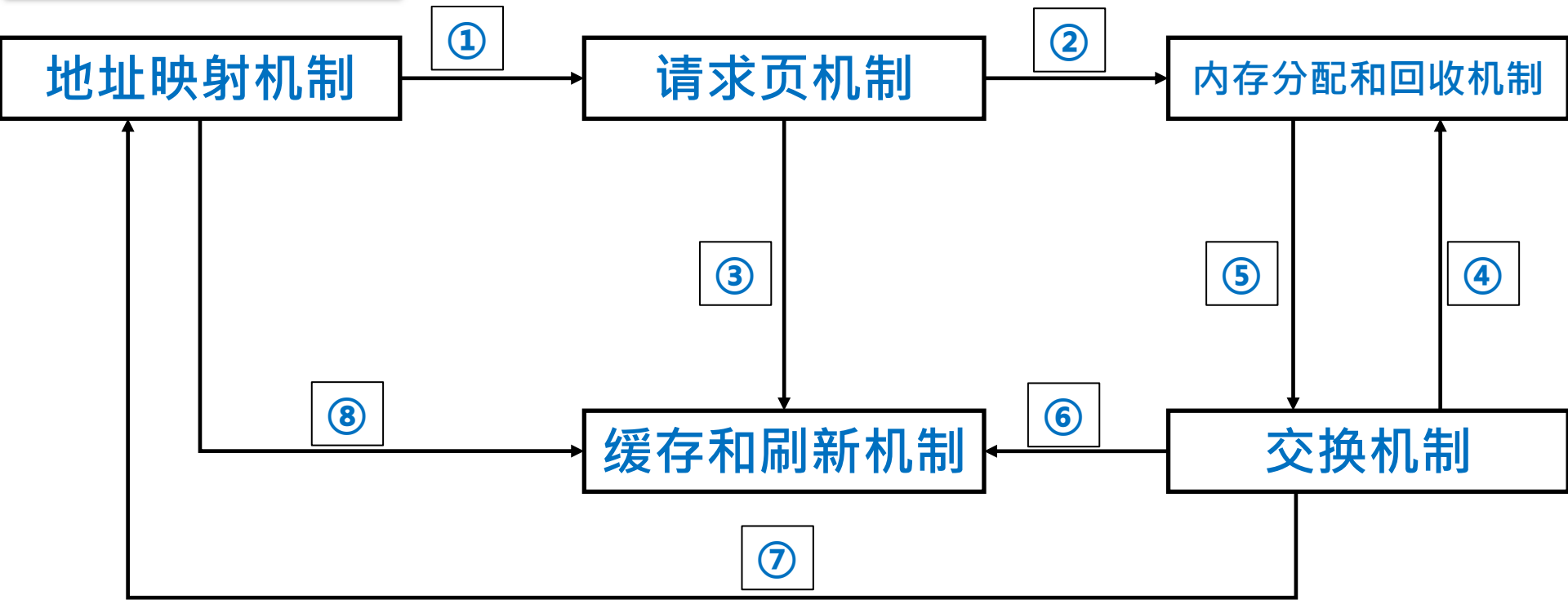
如果有空闲的内存可供分配，就请求分配内存②（**内存分配回收机制**），并把正在使用的物理页记录在**缓存**中③

ARM-Linux 的内存管理



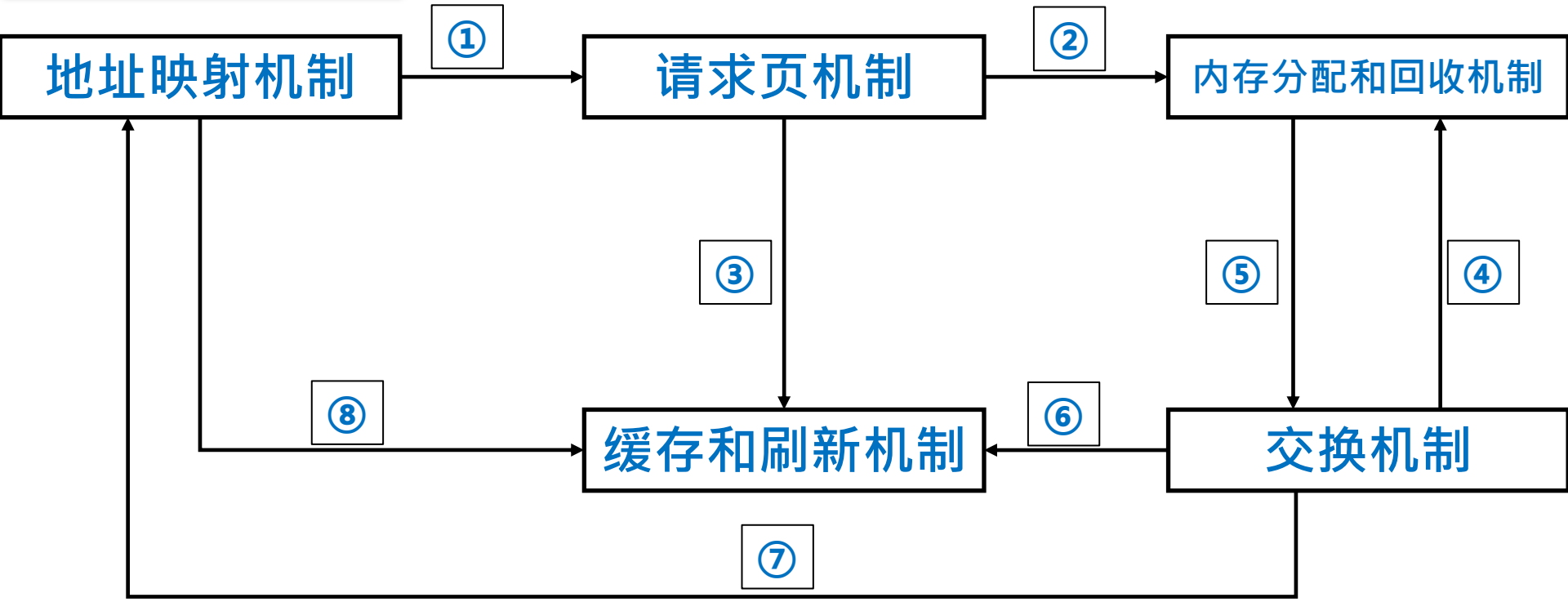
如果没有足够的内存可供分配，那么就要调用**交换机制**，这是为了腾出一部分内存以供分配④⑤

ARM-Linux 的内存管理



另外，在地址映射中要通过TLB（Translation Lookaside Buffer）来寻找物理页⑧

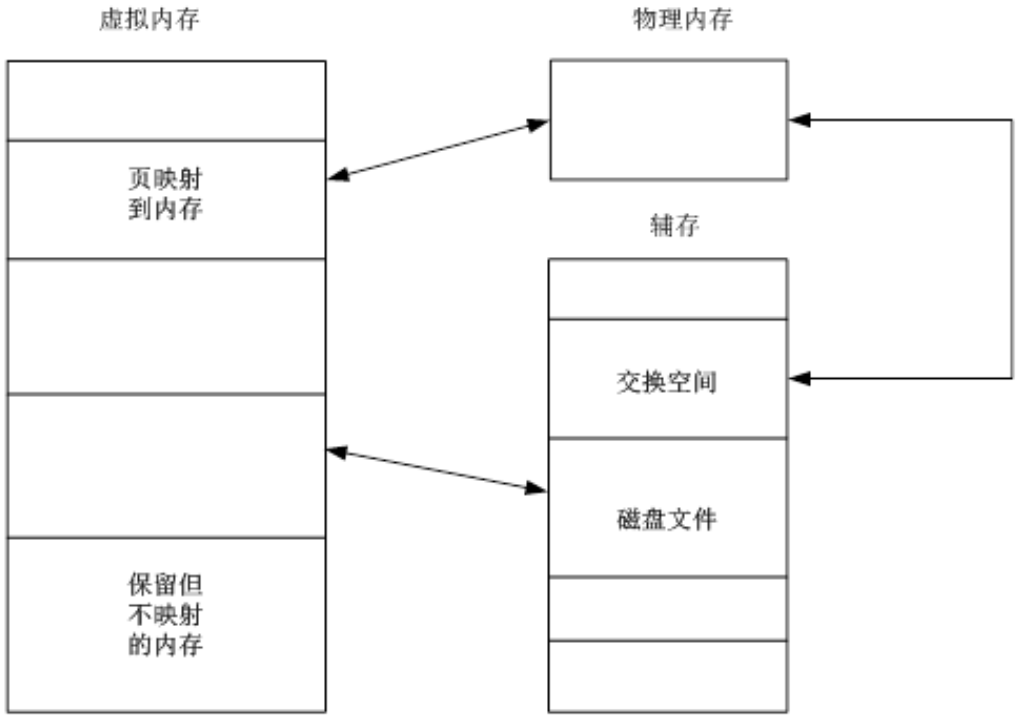
ARM-Linux 的内存管理



交换机制中也要用到交换缓存⑥，并且把物理页的内容交换到交换文件中，也要修改页表来映射文件地址⑦

地址映射机制

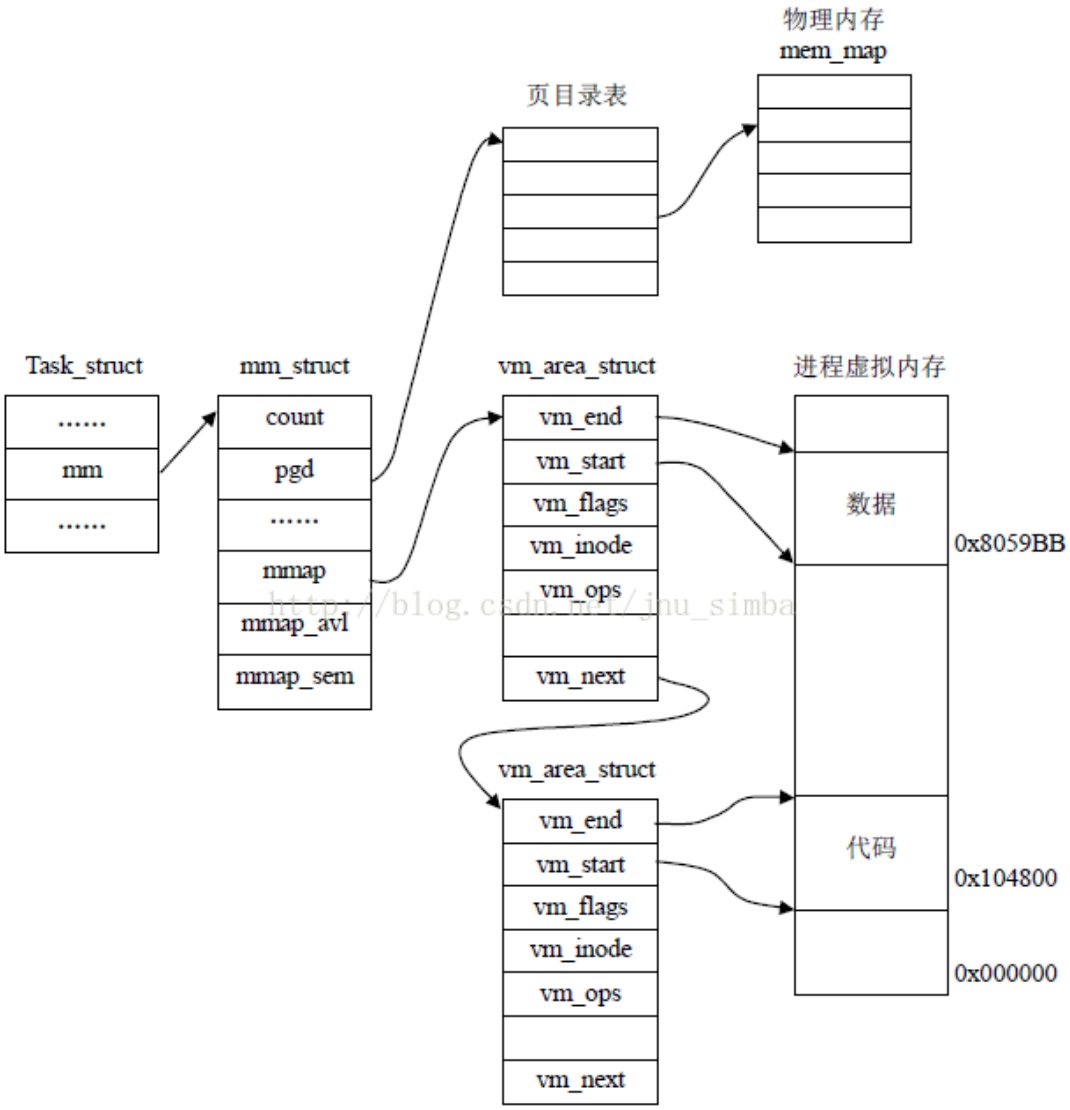
- 地址映射就是建立几种存储媒介（内存，辅存，虚存）间的关联，完成地址间的相互转换，它既包括磁盘文件到虚拟内存的映射，也包括虚拟内存到物理内存的映射，如图所示



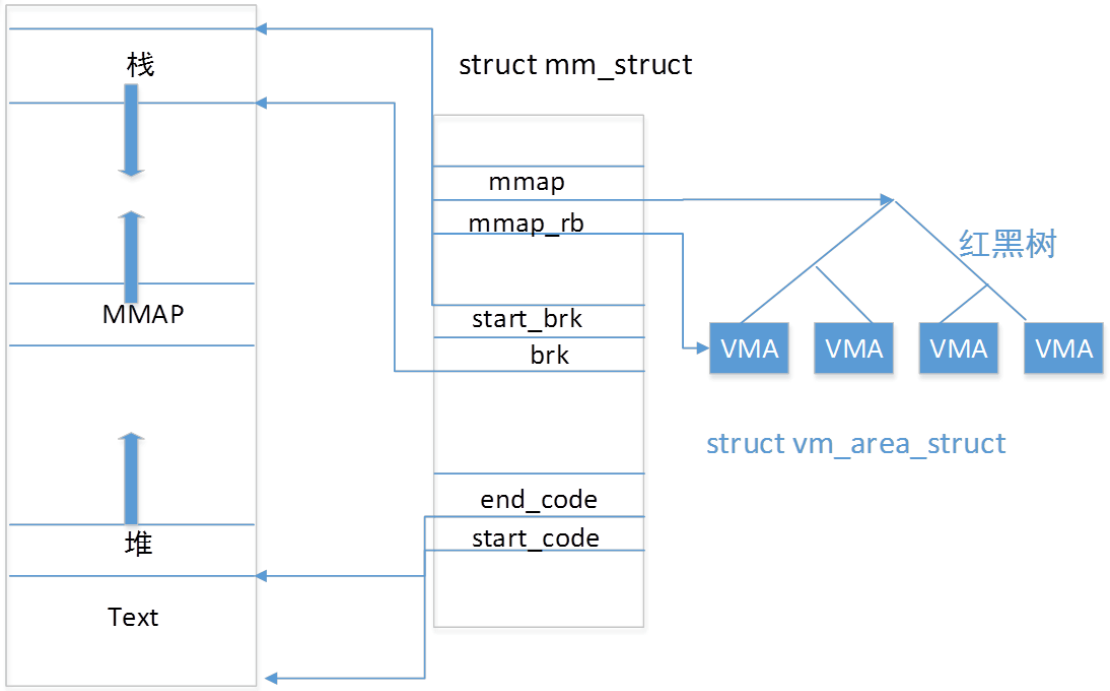
地址映射机制

- 一个**进程的虚拟地址空间**主要由两个数据结来描述。一个是最高层次的：*mm_struct*，一个是较高层次的：*vm_area_structs*。最高层次的*mm_struct*结构描述了一个进程的整个虚拟地址空间。较高层次的结构*vm_area_struct*描述了虚拟地址空间的一个区间（简称虚拟区）。

地址映射机制


物理内存
mem_map
页目录表
Task_struct
mm_struct
vm_area_struct
进程虚拟内存
.....
mm
.....
count
pgd
.....
mmap
mmap_avl
mmap_sem
vm_end
vm_start
vm_flags
vm_inode
vm_ops
vm_next
数据
0x8059BB
vm_area_struct
vm_end
vm_start
vm_flags
vm_inode
vm_ops
vm_next
代码
0x104800
0x000000

地址映射机制



- 红黑树是平衡二叉树的变种，有很好的性质，树中的结点都是有序的，查找不会出现非常恶劣的情况，基于红黑树的操作的时间复杂度是O(log(N))。
- Linux内核在管理vm_area_struct时就是采用了红黑树来维护内存块的。 每个虚拟内存空间都对应红黑树的一个节点，左指针指向相邻的低地址虚拟内存空间，右指针指向相邻的高地址虚拟内存空间
- 在创建新的虚拟内存区或处理页面不在物理内存中的情况时，缩短查找时间。

请求页机制

- 进程的虚拟内存包括可执行代码和多个资源数据。**任何时候进程都不同时使用包含在其虚拟内存中的所有代码和数据。**如果将这些使用频率比较低的代码和数据，如初始化或者处理特殊事件的代码、一些共享库的部分子程序等，全部加载到物理内存中，就会引起极大的浪费。

- **Linux使用请求调页技术来把那些进程需要访问的虚拟内存载入物理内存中。**内核将进程页表中这些虚拟地址标记成存在但不在内存中的状态，而无须将所有代码和数据直接调入物理内存。当进程试图访问这些代码和数据时，系统硬件将产生页面错误，并将控制转移到Linux内核来处理。这样对于处理器地址空间中的每个虚拟内存区域，内核都必须知道这些地址的虚拟内存从何处而来以及如何将其载入内存，以便于处理页面错误。

内存分配回收机制

- **当进程请求分配虚拟内存时，Linux并不直接分配物理内存。**它只是创建一个vm_area_struct结构来描述此虚拟内存，此结构被连接到进程的虚拟内存链表中。
- 当进程试图对新分配的虚拟内存进行写操作时，系统将产生页面访问错误。处理器会尝试解析此虚拟地址，但是如果找不到对应此虚拟地址的页表入口时，处理器将放弃解析并产生页面错误异常，由Linux内核来处理，Linux查看此虚拟地址是否在当前进程的虚拟地址空间中。如果存在，内核会创建正确的PTE（页表项Page table entry）并为此进程分配物理页面，包含在此页面中的代码或数据可能需要从文件系统或者交换磁盘上读出，然后进程将从页面错误处开始继续执行。此时，由于物理内存已经存在，所以不会再产生页面异常。

缓存机制

- **缓存区高速缓存**：包含由块设备使用的数据缓冲区。在这些缓冲区中包含从设备中读取的数据块或写入设备的数据块，并通过设备标识号和块标号来进行索引，因此可以快速找出数据块。
- **页面高速缓存**：页面I/O操作访问数据所使用的磁盘高速缓存。在文件系统中常见的read()、write()和mmap()等对常规文件的访问都是通过页面高速缓存来实现的。
- **交换高速缓存**：包含一个页面表项链表，每个页面表项对应了系统的一个物理页面。修改后的（脏）页面会保存在交换文件中，页面表项包含保存该页面的交换文件信息，以及该页面在交换文件中的位置信息。如果某个交换页面表项非零，则表明保存在交换文件中的对应的物理页面没有被修改；如果被修改，则处于交换缓存中的页面表项就会被清零。
- **硬件高速缓存**：是对页面表项的缓存，由处理器完成，操作和具体的处理器架构有关。

快表技术 TLB 刷新机制

- TLB（Translation Lookaside Buffer，旁路转换缓冲，或页表缓冲）
- 快表就是存放在Cache中的部分页表。作为页表的Cache，它的作用与页表相似，但是提高了访问速率。由于采用页表做地址转换，读写内存数据时CPU要访问2-3次内存。有了快表，有时只要访问一次高速缓冲存储器和一次内存，这样可加速查找并提高指令执行速度。
- 刷新机制的作用是为了保持TLB和其他缓存中的内容的同步性。 Linux刷新机制，包括TLB的刷新、缓存的刷新等

交换机制

- 交换的基本原理：当物理内存量无法满足要求时，在Linux中，会把磁盘空间作为内存使用，这部分磁盘空间叫做**交换文件或交换区**。
- 交换的单位选择：以往的Unix交换以进程为单位（Swapping），在 Linux中，交换的单位是页面而不是进程。
- 置换算法：在页面置换中，要考虑到哪种页面要换出、如何在交换区中存放页面、如何选择被交换出的页面以及何时执行页面换出操作4个会影响交换性能的关键性指标。
- 置换算法有很多：Linux基于LRU改进。其他如NRU、LRU、FIFO、CLOCK等等，同学们可以课后扩展了解。

内存共享机制

- 共享内存是UNIX/ Linux中最快速的进程间通信（IPC）方法。
- Linux的进程拥有各自独立的地址空间，当多个进程要共享同一内存段时，就会通过系统提供的共享内存机制进行，同一块物理内存会被映射到进程A、B各自的进程地址空间。
- 共享区域内的任何进程都可以读写内存。由于多个进程共享同一块内存区域，所以必然需要**同步机制的保障（在后续章节会专门学习）**。

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

概念回顾

- **进程管理**，是操作系统的功能之一，特别是多任务处理的状况下，这是必要的功能。
- 操作系统将资源分配给各个进程，让进程间可以分享与交换信息，保护每个进程拥有的资源，不会被其他进程抢走，以及使进程间能够同步。
- 为了达到这些要求，操作系统为每个进程分配了一个数据结构（在Linux中是task_struct）用来描述进程的状态，以及进程拥有的资源。操作系统可以通过这个数据结构，来控制每个进程的运作。

清華大學 Tsinghua University

Linux 进程管理介绍

- Linux 是一个多用户多任务的操作系统。
  - 多用户：指多个用户可以在同一时间使用计算机系统
  - 多任务：多任务是指 Linux 可以同时执行几个任务
- 操作系统监控着一个等待执行的任务队列
- 操作系统根据每个任务的优先级为每个任务分配合适的时间片

Linux进程常见的5种状态

1. TASK_RUNNING：运行态和就绪态的合并，表示进程正在运行或准备运行（包括Running队列中等待被安排到CPU的进程）
2. TASK_INTERRUPTIBLE：浅度睡眠态（被阻塞），等待资源到来时唤醒，也可以通过其他进程信号或时钟中断唤醒，进入运行队列。

- 处于这个状态的进程因为等待某某事件的发生（比如等待socket连接、等待信号量），而被挂起。这些进程的task_struct结构被放入对应事件的等待队列中。当这些事件发生时（由外部中断触发、或由其他进程触发），对应的等待队列中的一个或多个进程将被唤醒。
- 一般情况下，进程列表中的绝大多数进程都处于该状态。

Linux进程常见的5种状态

3. TASK_UNINTERRUPTIBLE：深度睡眠态。不可中断，指的并不是不响应外部硬件的中断，而是指进程不响应异步信号。直接等待硬件条件，资源有效时才唤醒。

- 内核的某些处理流程是不能被打断的。如果响应异步信号，程序的执行流程中就会被插入一段用于处理异步信号的流程于是原有的流程就被中断了。例如，在进程对某些硬件进行操作时（比如进程对某个设备文件进行读操作,与对应的物理设备进行交互），可能需要使用该状态对进程进行保护，以避免进程与设备交互的过程被打断，造成设备陷入不可控的状态。
- 这种情况下的该状态通常是非常短暂的。如果时间过长，则意味着IO/外设可能出了问题。

Linux进程常见的5种状态

4. TASK_STOPPED：进程被暂停，等待其他进程的信号才能唤醒。在调试期间进程都会进入这个状态。向进程发送一个SIG_CONT信号，可以让其从TASK_STOPPED状态恢复到TASK_RUNNING状态。

5. TASK_ZOMBIE：僵死状态，进程已经结束但未释放PCB，等待其父进程收集相关信息。

- 子进程在退出的过程中，内核会给其父进程发送一个信号，通知父进程来“收尸”。父进程可以通过wait系列的系统调用（如wait4、waitid）来等待某个或某些子进程的退出，并获取它的退出信息。然后wait系列的系统调用会顺便将子进程的尸体（task_struct）也释放掉。

关于“僵尸进程”

处于TASK_ZOMBIE状态的僵尸进程并不做任何事情，不会使用任何资源也不会影响其它进程，因此一个僵尸进程并不会对系统有害。

但是，由于进程表中的退出状态以及其它一些进程信息也是存储在内存中的，存在太多僵尸进程（同学们可以自行搜索，进一步了解这种情况为什么可能出现）有可能导致系统崩溃。

> 你是一家建筑公司的老板。你每天根据工人们的工作量来支付工资。有一个工人每天来到施工现场，就坐在那里，你不用付钱， 他也不做任何工作。 他只是每天都来然后呆坐在那，仅此而已！
>
> 这个工人就是僵尸进程的一个活生生的例子。如果你有很多僵尸工人，你的建设工地就会很拥堵从而让那些正常的工人难以工作。

Linux 进程调度算法

- 分时调度策略（SCHED_OTHER）

  面向普通进程的时间片轮转

- 先到先服务的实时调度策略（SCHED_FIFO）

  一旦占用CPU就一直运行，直到主动放弃或被抢占

- 时间片轮转的实时调度策略（SCHED_RR）

  实时进程轮转获得CPU，比FIFO更加公平

//参考课本P62，了解进程调度策略的实现方法。

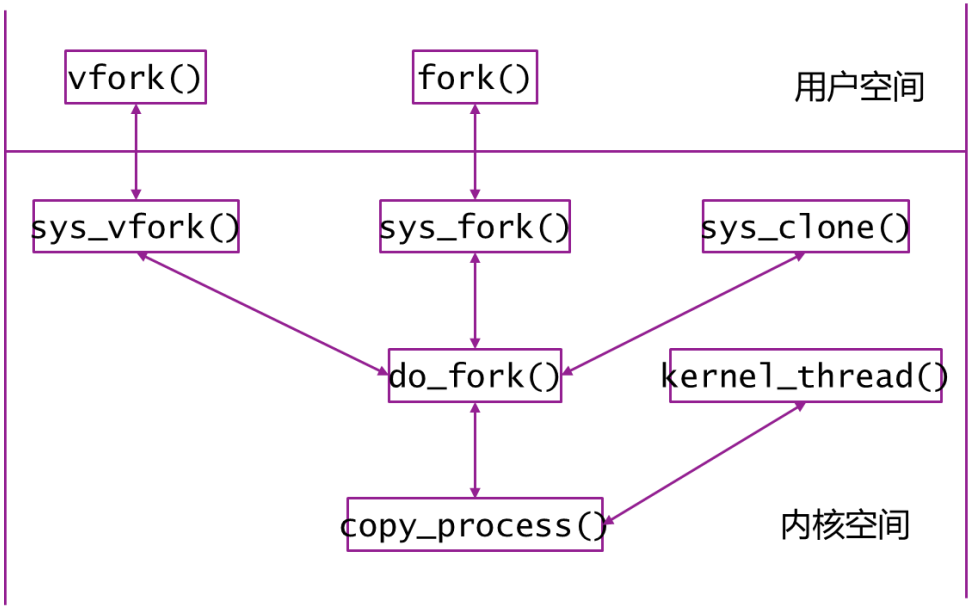
进程管理Linux指令

- 由于 Linux 是个多用户系统，同时也是一个多进程系统，经常需要对这些进程进行一些调配和管理，就要知道当前的进程情况。

Who 命令	该命令主要用于查看当前线上的用户情况
W 命令	可以显示出当前用户当前正在进行的工作
Ps 命令	是非常强大的进程查看命令
Top 命令	top 命令和 ps 命令的基本作用是相同的
Kill 命令	该命令可以终止后台进程
Nohup 命令	no hang up, 使进程在用户退出后仍继续执行

创建Linux进程的三种系统调用

- sys_fork：从父进程完整派生一个子进程
- sys_clone：可以通过参数决定复制给子进程的资源
- sys_vfork：复制task_struct，共享父进程资源，本质上只是产生了一个线程，在其运行结束前会一直阻塞父进程

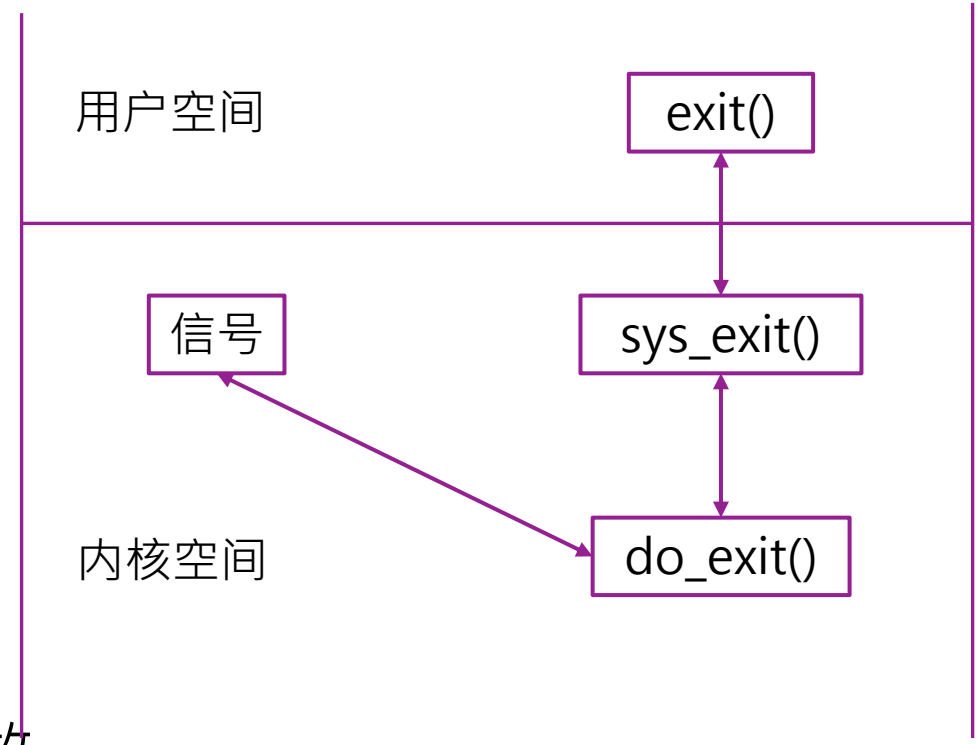


- 三种都会使用do_fork创建进程

- 复制进程虚拟空间的技术：
  - Copy-On-Write (P151)：当且仅当共享虚拟内存的两个进程之一试图进行写操作时，才复制此虚拟内存块。
  - 思考：为什么这样设计？

销毁Linux进程的系统调用

- 进程销毁由以下三类事件之一驱动
  - 正常的进程结束
  - 信号
  - exit函数的调用
- 都通过调用内核函数do_exit()实现



RT-Linux 的进程管理

- RT-Linux有两种中断：硬中断和软中断
  - 软中断是常规Linux内核中断。
  - 硬中断的延迟低于15μs。
- RT-Linux通过一个高效的、可抢占的实时调度核心来全面接管中断，并把Linux作为此实时核心的一个优先级最低的进程运行。当有实时任务需要处理时，RT-Linux运行实时任务；无实时任务时，RT-Linux运行Linux的非实时进程。
- 上述设计的出发点：不全面重构Linux的进程管理的前提下，使之具备响应实时任务的能力。

μCLinux的进程管理

- 在实现多个进程时需要实现数据保护。μCLinux 下，由于 μCLinux 没有 MMU 管理存储器，系统虽然支持 fork 系统调用，但其实质上就是 vfork 。
- μClinux 系统 fork 调用完成后，要么子进程代替父进程执行（此时父进程已经 sleep），直到子进程调用 exit 退出；要么调用 exec 执行一个新的进程（替换父进程），这个时候产生可执行文件的加载，即使这个进程只是父进程的拷贝，这个过程也不可避免。
- 当子进程执行 exit 或 exec 后，子进程使用 wakeup 把父进程唤醒，使父进程继续往下执行。
- 编程实现多进程时，程序员需要考虑数据保护的问题。

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

概念回顾

- **文件系统**，是一种存储和组织计算机数据的方法，它使得对其访问和查找变得容易，文件系统使用**文件**和**树形目录**的抽象逻辑概念代替了硬盘和光盘等物理设备使用数据块的概念。
- 用户使用文件系统来保存数据不必关心数据实际保存在硬盘（或者光盘）的地址为多少的数据块上，只需要记住这个文件的所属目录和文件名。在写入新数据之前，用户不必关心硬盘上的那个块地址没有被使用，硬盘上的存储空间管理（分配和释放）功能由文件系统自动完成，用户只需要记住数据被写入到了哪个文件中。
- 严格地说，**文件系统是一套实现了数据的存储、分级组织、访问和获取等操作的抽象数据类型**（Abstract data type）。

文件系统的一个描述性定义

- 包含在磁盘驱动器或者磁盘分区的目录结构，整个磁盘空间可以给一个或者多个文件系统使用。在对某个文件系统做在某一个挂载点的挂载（Mount）操作后，就可以使用该文件系统了。

嵌入式文件系统的特点

- 嵌入式设备
  - 很少使用大容量的 IDE 硬盘等
  - 往往选用 ROM、Flash Memory 等
- 普通文件系统
  - 管理文件，提供文件系统API
  - 管理各种设备，支持对设备和文件的操作
- 嵌入式文件系统
  - 为嵌入式系统的设计目的服务
  - 设计目标各不相同

嵌入式文件系统设计目标

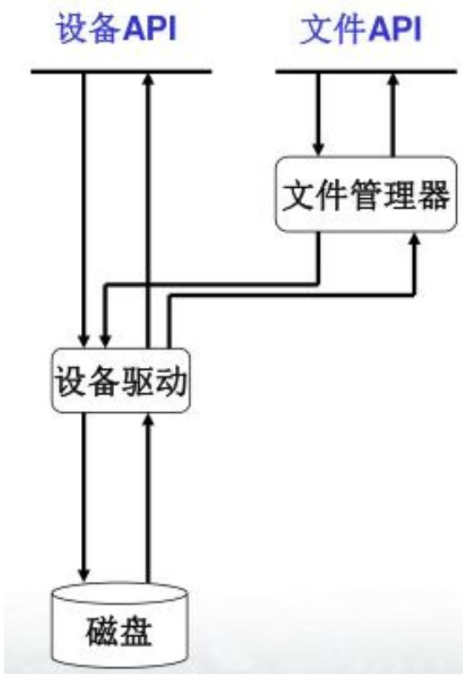
- 使用简单方便：普遍需求
- 安全可靠：嵌入式系统应用大多强调高可靠性
- 实时响应：嵌入式系统的典型特性
- 接口标准的开放性和可移植性：适配多样的硬件和OS
- 可伸缩性和可配置性：“可剪裁”特性
- 开放的体系结构：方便使用和开发
- 资源有效性：嵌入式系统往往资源非常有限
- 功能完整性：普遍要求
- 支持热插拔：更新升级不影响正在使用文件系统的操作
- 支持多种文件类型：嵌入式应用的差异性、多样性

Linux文件系统

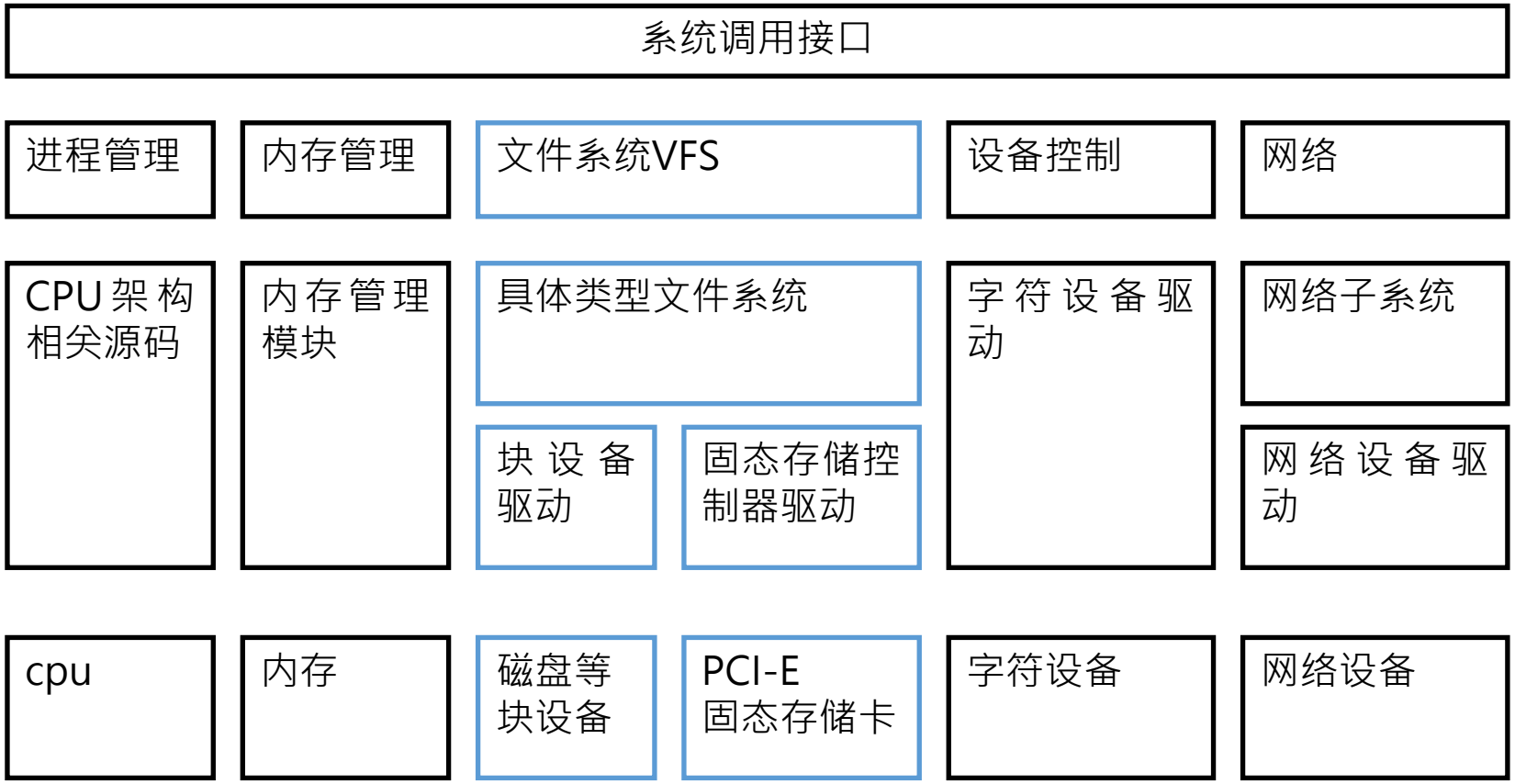
- Linux 支持许多种文件系统。Linux 初期的基本文件系统是 Minix，但其适用范围和功能都很有限。其文件名最长不能超过 14 个字符并且最大的文件不超过 64MB。
- 因此于 1992 年开发了 Linux 专用的文件系统 ext（Extended File System），解决了很多的问题。但 ext 的功能也并不是非常优秀，最终于 1993 年增加了 ext2 。
- Linux 还支持 ext3、JFS2、XFS 和 ReiserFS 等新的日志型文件系统。另外，Linux 支持加密文件系统（如 CFS）和伪文件系统（如 /proc）。

Linux文件系统框架

- Linux文件系统具有良好的扩展性
  - 支持ext、ext2、ext3、ext4、xia、vfat、minix、msdos、umsdos、 proc、smb、ncp、iso9660、sysv、hpfs、affs、ufs
- 现代操作系统都提供多种访问存储设备的方法，Linux文件系统有两条独立的途径控制设备驱动
  - 通过设备驱动的接口：
    - 继承了文件API，直接访问存储设备
  - 通过文件管理器接口：
    - 通过VFS，同样使用文件API，访问存储设备



文件系统在Linux中位置



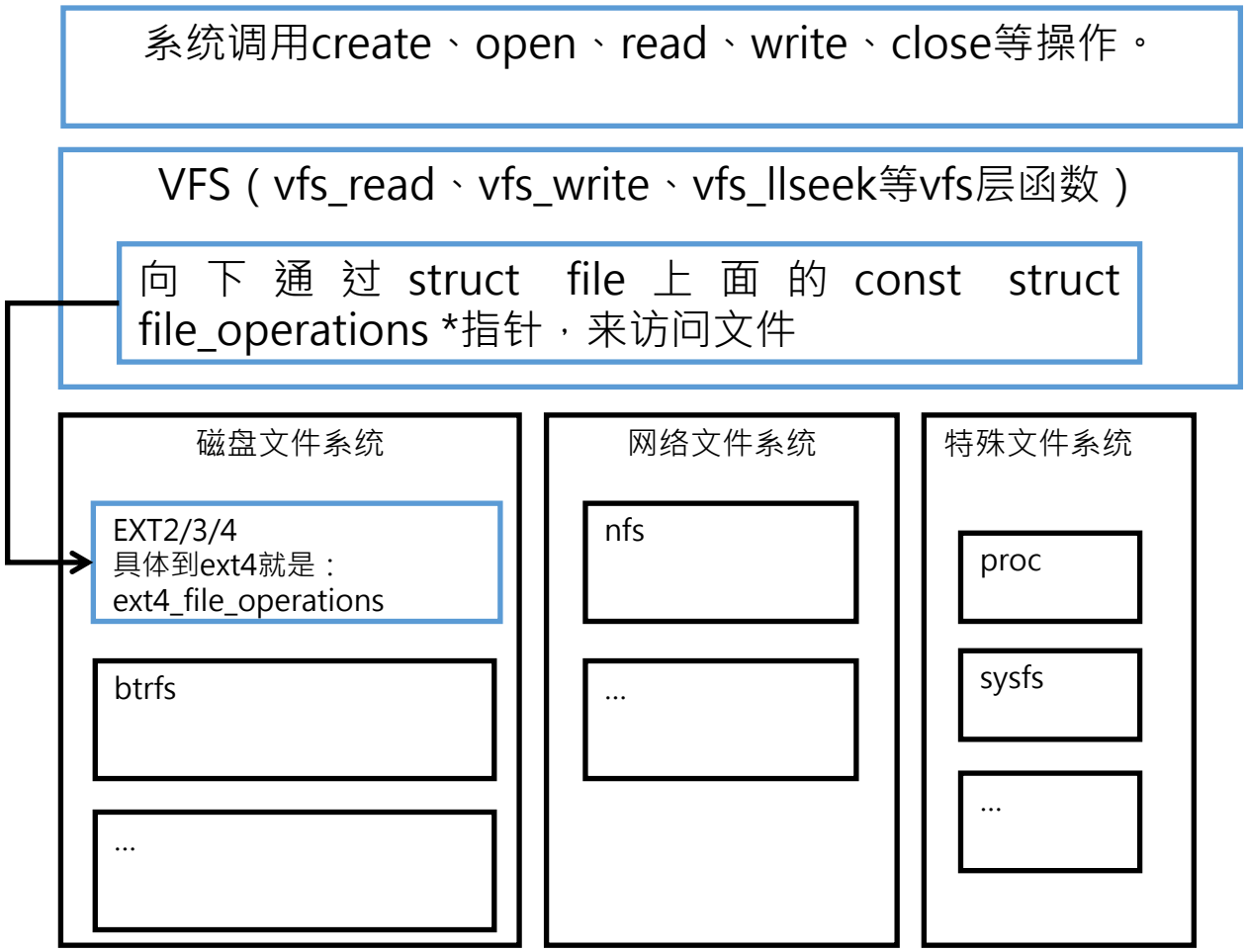
虚拟文件系统 VFS

- Linux内核支持装载不同的文件系统类型，不同的文件系统有各自管理文件的方式。
- 为支持各种文件系统，Linux内核在用户进程和具体的文件系统之间引入了一个抽象层，该抽象层称之为虚拟文件系统VFS (Virtual File System)。
- VFS一方面提供一种操作文件、目录及其他对象的统一方法（文件API），使用户进程不必知道文件系统的细节。另一方面，VFS提供的各种方法必须和具体文件系统的实现达成一种妥协， 为此，VFS中定义了一个通用文件模型，以支持文件系统中文件的统一视图。
- VFS主要由一组标准的、抽象的文件操作构成，以系统调用的形式提供于用户程序，如read()、write()、lseek()等等。

文件系统模块层次图

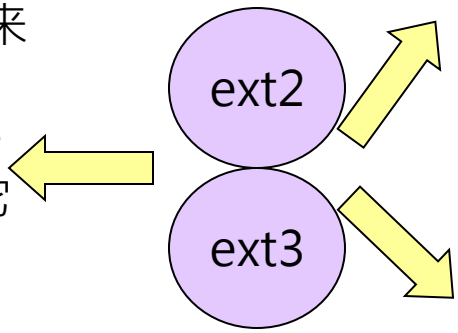
文件系统模块层次图

每个文件系统实现（比如ext2/ext3/ext4、btrfs 等等）导出一组通用接口，供 VFS 使用，和上层VFS对接。



ext2/ext3 文件系统

◆Linux ext2/ext3 文件系统使用索引节点(inode)来记录文件信息，作用像windows的文件分配表。索引节点是一个结构，它包含了一个文件的长度、创建及修改时间、权限、所属关系、磁盘中的位置等信息。



◆Linux 缺省情况下使用的文件系统为 ext2，ext2 文件系统高效稳定

◆ext3 文件系统是直接从 ext2文件系统发展而来，目前 ext3文件系统已经非常稳定可靠

◆一个文件系统维护了一个索引节点的数组，每个文件或目录都与索引节点数组中的唯一一个元素对应

ext2 vs. ext 文件系统

- ext2 是 Linux 事实上的标准文件系统，它已经取代了它的前任——ext 文件系统。
- ext 文件系统（Extended file system）支持的文件大小最大为 2GB，支持的最大文件名称大小为255个字符，而且它不支持索引节点（包括数据修改时间标记）。

ext2文件系统 — 优点

- ext2支持达4TB容量（ext是2G）
- ext2文件名称最长可以到1012个字符
- 当创建文件系统时，可以选择逻辑块大小（1024/2048/4096字节）
- ext2实现快速符号链接，无需分配数据块，并提升性能
  - **符号链接：**是一类特殊的文件，包含一条以绝对路径或者相对路径的形式指向其它文件或者目录的引用。可以跨文件系统引用。
  - 符号链接vs快捷方式？

ext2文件系统 — 缺点

- ext2是针对IDE设备设计，逻辑块是512B的倍数，不太适合扇区大小各异的闪存设备
- ext2没有提供对基于扇区的擦除/写操作的良好管理，比如擦除一个字节需复制整扇区到RAM中
- ext2在出现电源故障时不防崩溃
- ext2不支持损耗平衡（轮流擦写存储设备的不同区域），缩短闪存寿命

ext3 文件系统

- 在 ext2 文件系统尚未关闭之前就关机的话，很可能会造成文件系统的异常，在系统重新开机后，就能检测到内容的不一致性，此时，文件系统需要做检查工作，将不一致和错误的地方进行修复。这个检查和整理工作是比较耗时的，特别在文件系统容量比较大的时候，而且不能保证 100% 的内容能完全修复。为了克服这个问题，**日志式文件系统（Journal File System）**应运而生。这种文件系统的做法是：**它会将整个磁盘的写入动作完整记录在磁盘的某个区域上，以便于有需要时可以回溯追踪。**
- ext3 便是一种日志式文件系统，是对 ext2 系统的扩展，它兼容 ext2。

ext3 文件系统

- ext3 相比较 ext2 文件系统而言，具有以下特点：
  - **高可用性**: 系统使用了 ext3 文件系统后，即使在非正常关机后，文件系统的恢复时间只需要数十秒钟
  - **很好的数据完整性**: ext3 文件系统能够极大地提高文件系统的完整性，避免了意外关机对文件系统的破坏
  - **访问速度快**: 尽管使用 ext3 文件系统时，有时在存储数据时可能需要多次写数据，但是 ext3 的日志功能对磁盘的驱动器读写头进行了优化，因此从总体上看来，ext3 比 ext2 的性能还要好一些
  - **兼容性好**: 由于 ext3 兼容ext2，因此将 ext2 文件系统转换成 ext3 文件系统非常容易；同时，ext3 文件系统可以不经修改直接加载为 ext2 文件系统
  - **多种日志模式**: 两种日志模式：Journal，对文件数据和 metadata 进行记录；Ordered/Writeback，只记录 metadata

加密文件系统 CFS

- CFS(Crypt File System)是一个Unix下的加密文件系统，工作在用户层，以NFS服务器的形式工作。
- CFS需要用户在本地或远程文件系统中创建一个目录，用来存取加密数据。密钥和加密算法是在这个目录创建时指定的。用户需要通过一个挂载命令向CFS代理发送加密数据访问请求。代理负责验证用户ID和密钥是否合法，然后在挂载点目录创建一个目录供用户访问。挂载成功后，用户将看到解密后的数据，并可以像操作普通目录一样在这个目录里存取数据，唯一的不同在于所有的数据都会被透明的加密。
- CFS的特点：
  - 设计合理、可移植、内置多种加密算法
  - 工作在用户模式，大量时间被花费在用户空间和内核空间之间的上下文切换，因此效率较低。

/proc

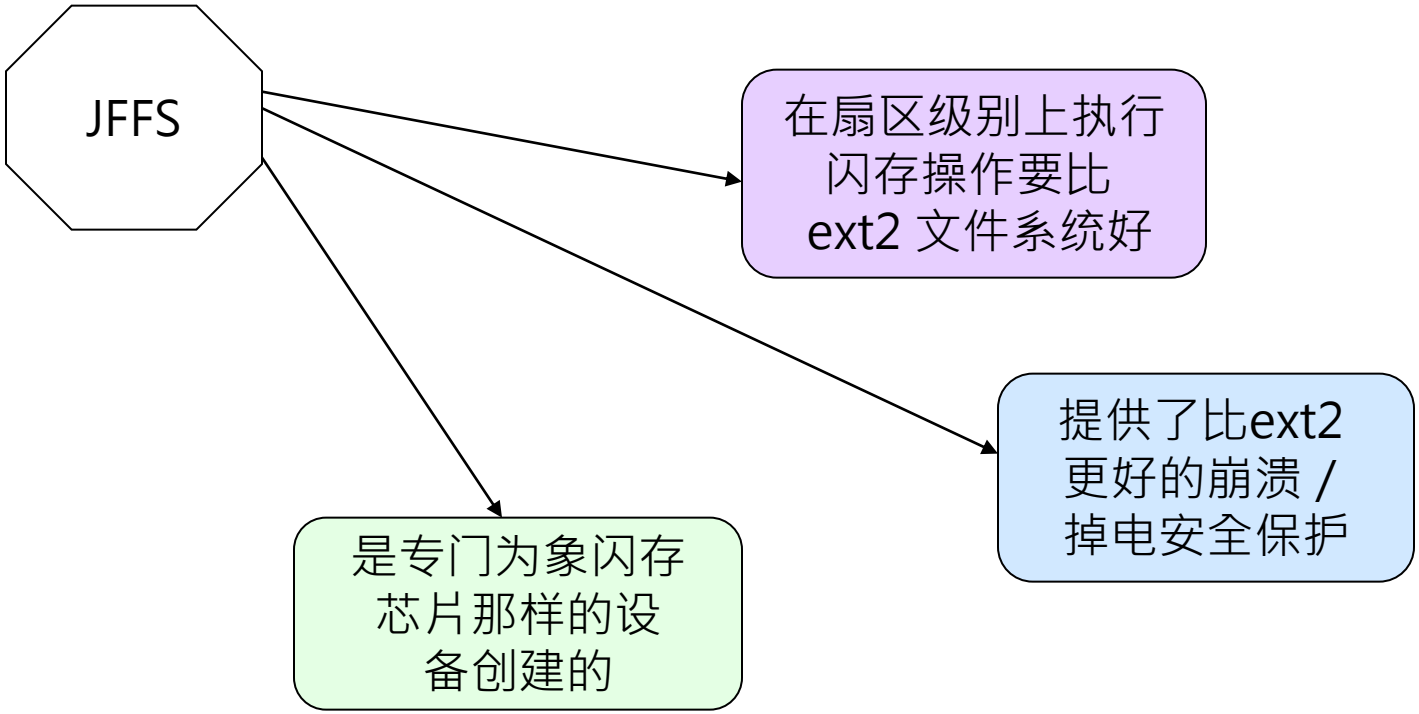
- /proc被称为进程信息伪文件系统(process information pseudo-file system)。它不包含“真正的”文件，而是存储运行时的系统信息（如系统内存、设备挂载、硬件配置等）。因此/proc可以看作是内核的控制和信息中心。
- 事实上，很多系统实用程序都只是对这个目录中的文件进行了调用。例如，'lsmod'等价于'cat /proc/modules'，'lspci'等价于'cat /proc/pci'。通过修改这个目录下的文件，甚至可以在系统运行时读取/改变内核参数（sysctl）。
- 这个目录下的文件最特别的地方是，除了kcore、mtrr和self之外，所有的文件大小都是0。一个目录列表看起来类似于下面这个样子：

```
› ls -l /proc
total 0
dr-xr-xr-x  9 root             root                          0 Mar  7 23:04 1
dr-xr-xr-x  9 root             root                          0 Mar  7 15:04 10
dr-xr-xr-x  9 root             root                          0 Apr  6 13:04 10609
dr-xr-xr-x  9 root             root                          0 Apr  6 13:04 10614
dr-xr-xr-x  9 root             root                          0 Apr  6 13:05 10620
dr-xr-xr-x  9 root             root                          0 Apr  6 13:05 10621
dr-xr-xr-x  9 root             root                          0 Apr  6 13:05 10622
```

常见嵌入式Linux文件系统一览

JFFS 文件系统

◆JFFS，日志闪存文件系统最初由瑞典的 Axis Communications 研发，Red Hat的DavidWoodhouse 对他进行了改进。



JFFS2 文件系统

- JFFS2全名是 Journalling Flash File System Version2，是 Redhat 公司开发的闪存文件系统，其前身是 JFFS，最早只支持 NOR Flash，自 2.6 版以后开始支持 NAND Flash，适合使用于嵌入式系统。
  - 可读写、支持数据压缩
  - 基于哈希表的日志型文件系统
  - 提供了崩溃/掉电安全保护
  - 提供“写平衡”支持
- 缺点
  - 挂载时间过长
  - 磨损平衡的随意性
  - 很差的扩展性

JFFS2 特性

- 垃圾收集
  - JFFS2使用了多个级别的待回收块队列。
  - 垃圾收集操作的主要工作是将数据块里面的有效数据移动到空闲块中，然后清除脏数据块，最后将数据块从dirtylist链表中摘除，并且放入空闲块链表。
  - 由于JFFS2中使用了多种节点，所以在进行垃圾收集的时候也必须对不同的节点进行不同的操作。JFFS2进行垃圾收集时也对闪存文件系统中的不连续数据块进行整理。
  - 垃圾收集的详细代码同学们可以参考参考JFFS2源代码中的文件gc.c，这里就不再详细描述了

JFFS2 特性

- 写平衡
  - 写平衡策略是在垃圾收集中实现的，垃圾收集的时候会读取系统时间，使用这个系统时间产生一个伪随机数。利用这个伪随机数结合不同的待回收链表选择要进行回收的链表。使用了这个平衡策略以后能提供较好的写平衡效果。

*其他常见文件系统--基于 Flash

- 基于 Flash Memory 的文件系统
  - Flash 作为嵌入式系统的主要存储媒介，有其自身的特性。
    - Flash 的写入操作只能把对应位置的 1 修改为 0，而不能把 0 修改为 1（擦除 Flash 就是把对应存储块的内容恢复为 1），因此写内容时，需要先进行擦除操作。
    - Flash 的擦写次数是有限的

NAND Flash v.s. NOR Flash

- 共性:
  - 都是非易失存储介质。即掉电都不会丢失内容。
  - 在写入前都需要擦除。NOR Flash的一个bit可以从1变成0，而要从0变1就要擦除整块。NAND flash都需要擦除。
- 特性：

属性	NOR	NAND
容量	较小	较大
XIP(eXecute in Place)	可以	不可以
读取速度	很快	快
写入速度	慢	快
擦除速度	很慢	快
可擦除次数	10,000-100,000	100,000-1,000,000
可靠性	高	较低
访问方式	可随机访问	512字节块
价格	高	低

*其他常见文件系统

- YAFFS
  - YAFFS针对NAND FLASH设计，和JFFS相比它减少了一些功能，所以速度更快，挂载时间很短，对内存的占用较小。自带NAND芯片驱动，并为嵌入式系统提供了直接访问文件系统的API
  - YAFFS2是YAFFS的改进版本
    - YAFFS仅支持小页(512B) NAND闪存， YAFFS2则可支持大页(2KB) NAND闪存
    - YAFFS2在内存空间占用、垃圾回收速度、读/写速度等方面均有大幅提升

*其他常见文件系统--基于 Flash

- Cramfs
  - 压缩只读闪存文件系统（compressed ROM file system）
  - cramfs是Linus Torvalds任职于全美达（Transmeta）时，所参与开发的文件系统，cramfs当前以zlib方式压缩，不须加载到RAM中，因此可以节省许多RAM的空间。
  - 与传统压缩文件系统不同的是，可以直接使用cramfs映像，而无须先解压。出于这个原因，一些Linux发行版使用cramfs作为initrd映像（特别是Debian 3.1）与安装映像（特别是SUSE Linux），在一些对内存和映像大小有限制的地方。

*其他常见文件系统--基于 Flash

- Romfs
  - ROM filesystem 是一个缺乏许多功能的极为简单的文件系统，用途是将重要的文件存入 EEPROM。
  - 该文件系统非常适合用作初始化ROM内核驻留模块，具有体积小、可靠性好、读取速度快等优点，可以在需要时才加载。

其他常见文件系统--基于 RAM

- 基于 RAM 的文件系统
  - 随机存取存储器（英语：Random Access Memory，缩写：RAM；也叫主存）是与CPU直接交换数据的内部存储器。它可以随时读写（刷新时除外，见下文），而且速度很快，通常作为操作系统或其他正在运行中的程序的临时数据存储介质。

其他常见文件系统--基于 RAM

- RamDisk
  - RamDisk 是通过使用软件将 RAM 模拟当做硬盘来使用的一种技术。
  - 相对于传统的硬盘文件访问来说，这种技术可以极大的提高在其上进行的文件访问的速度。但是RAM的易失性也意味着当关闭电源后的数据将会丢失。某些时候这不是问题，比如说对于一个加密文档的明文来说。但是在大多数情况下，传递到RAM盘上的数据都是其他在别处有永久性存贮文件的一个拷贝，当系统重启后可以重新建立。
  - 典型的用法：内存载入OS内核可执行映像、根文件系统

Ramdisk 文件系统

What is Ramdisk???

RamDisk就是将内存中的一块区域作为物理磁盘来使用的一种技术。

- 对于用户来说，可以把 RAM disk 与通常的硬盘分区（如/dev/hda1）同等对待来使用。
- RAM disk 不适合作为长期保存文件的介质，掉电后 Ramdisk 的内容会随内存内容的消失而消失。
- RAM disk 的其中一个优势是它的读写速度高，内存盘的存取速度要远快于目前的物理硬盘，可以被用作需要高速读写的文件

*其他常见文件系统--基于 RAM

- Ramfs / Tmpfs
  - Tmpfs 是类 Unix 系统上暂存档存储空间的常见名称，通常以挂载文件系统方式实现，并将数据存储在易失性存储器而非永久存储设备中。和 RAM disk 的概念近似，但后者会呈现出具有完整文件系统的虚拟磁盘。
  - 所有在 tmpfs 上存储的数据在理论上都是暂时借放的，那也表示说，文件不会创建在硬盘上面。一旦重启，所有在 tmpfs 里面的数据都会消失不见。理论上，存储器使用量会随着 tmpfs 的使用而时有增长或消减。当前有许多 Unix 的发行版都有激活 tmpfs，默认是把它以共享存储器的方式用在系统的 /tmp 目录底下。

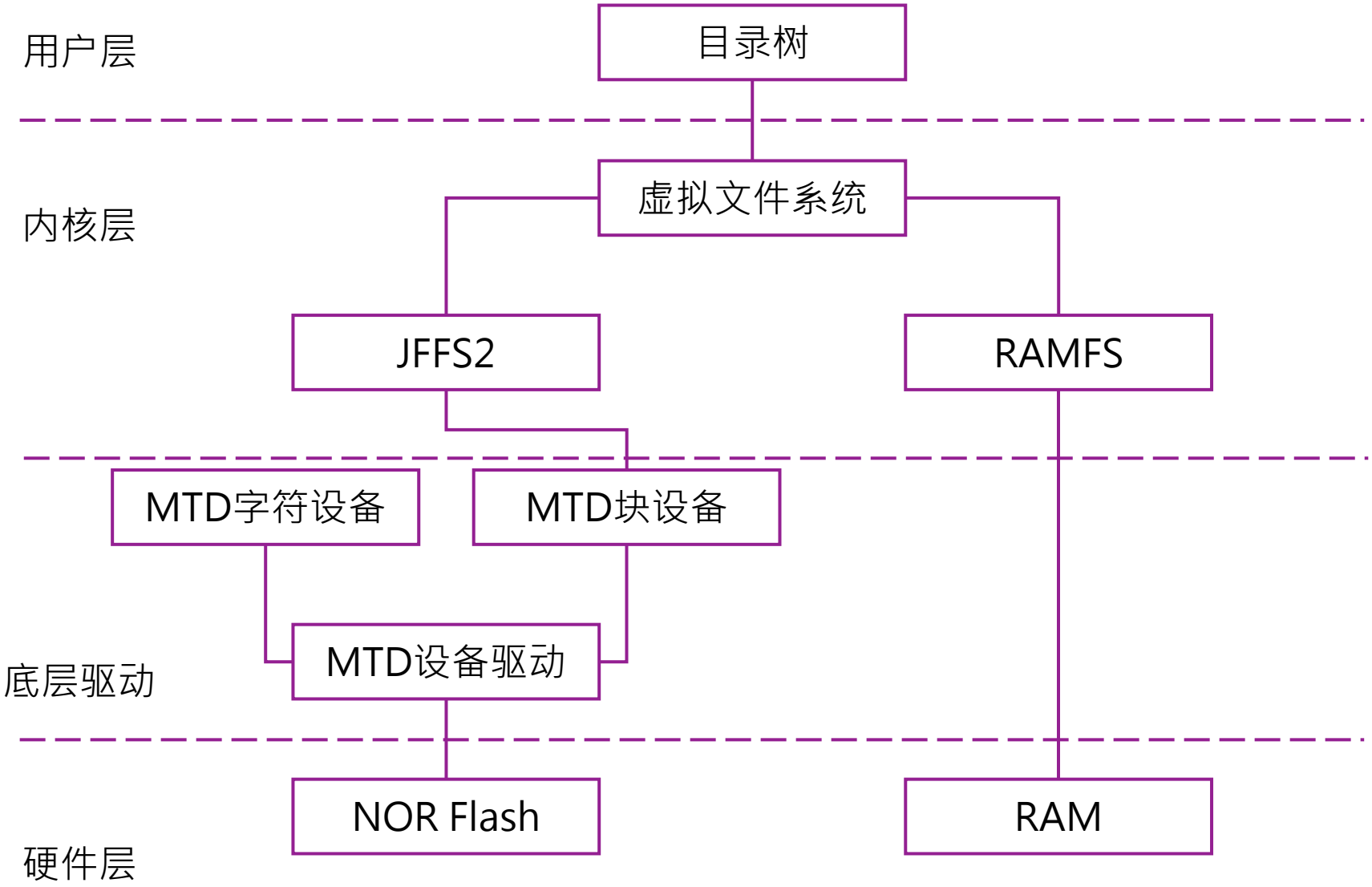
其他常见文件系统--NFS

- 网络文件系统 NFS
  - NFS 是一种分布式文件系统，力求客户端主机可以访问服务器端文件，并且其过程与访问本地存储时一样，于1984年发布。
  - 特点：
    - 提供透明文件访问以及文件传输
    - 容易扩充新的资源或软件，不需要改变现有的工作环境
    - 高性能，可灵活配置

根文件系统

- Linux启动时，第一个必须挂载的是根文件系统；若系统不能从指定设备上挂载根文件系统，则系统会出错而退出启动。之后可以自动或手动挂载其他的文件系统。
- 根文件系统是系统挂载的第一个文件系统。根文件系统和普通的文件系统的区别在于，根文件系统要包括 Linux启动时所必需的目录和关键性的文件，例如，Linux启动时都需要有init目录下的相关文件，在挂载分区时 Linux一定会找/etc/fstab这个挂载文件等，根文件系统中还包括许多的应用程序bin目录等，任何包括这些 Linux系统启动所必需的文件都可以成为根文件系统。

Linux下常用文件系统结构



*建立JFFS2根文件系统

- JFFS2在 Linux中有两种使用方式，一种是作为根文件系统，另一种是作为普通文件系统在系统启动后被挂载。
- 考虑到实际应用中需要动态保存的数据并不多，且在Linux系统目录树中，根目录和/usr等目录主要是读操作，只有少量的写操作，但是大量的读写操作又发生在/var和/tmp目录（这是因为在系统运行过程中产生大量log文件和临时文件都放在这两个目录中），因此，通常选用后一种方式。
- 根文件指的是 Romfs、var和/tmp，目录采用 Ramfs，当系统断电后，该目录所有的数据都会丢失。

*建立 JFFS2 文件系统的步骤

- 准备制作JFFS2根文件系统的工具mkfs.ffs2
- 建立目录；
- 编译busyBox；
- 复制动态链接库到lib目录中；
- 创建/etc/init.d/rcS、/etc/profile、/etc/fstab、/etcc/inittab文件，并且复制主机中的etc/passwd、etc/shadow、/etc/group文件到相应的目录中；
- 移植bash，将其复制到/bin目录中；
- 执行mkfs.jffs2-r. rootfs- rootfs.jffs2-n-e0x20000，生成FFS2根文件系统镜像。

* Busybox

- Busybox 是 Debian GNU/Linux 的大名鼎鼎的 Bruce Perens首先开发。后来有许多 Debian developers贡献力量，这其中尤推 busybox目前的维护者 Erik Andersen，身患癌症，却是一名优秀的自由软件开发者
- Busybox包括一个迷你的vi编辑器，系统不可或缺的 /sbin/init程序，以及其他诸如 sed, ifconfig, halt, reboot, mkdir, mount, ln, ls, echo, cat ... 等等，大小也不过 100K 左右。用户还可根据自己的需要，决定到底要在busybox 中编译进哪几个应用程序的功能，busybox的体积可以进一步缩小。
- Busybox支持多种体系结构，可以静态或动态链接glic或者uclibc库，以满足不同的需要，也可以修改Busybox默认的编译配置以移除不想使用的命令的支持

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

内核

- 内核：OS的核心部分，提供硬件抽象层、磁盘及文件系统控制、多任务等OS的基本功能。内核不是一套完整的操作系统。一套基于Linux内核的完整操作系统叫作Linux操作系统。

- 当前常见的OS内核结构
  - 单内核
  - 微内核

单内核（monolithic-kernel，宏内核）

- 单内核操作系统核心组件都在同一进程实现，内核是一个大的二进制映像，传统Unix是单内核。
  - OS各模块间通信：直接调用模块的函数
- 优点
  - 运行效率较高
  - 组织方式比较简单
- 缺点
  - 对开发者要求高
  - 可扩展性和可维护性较差

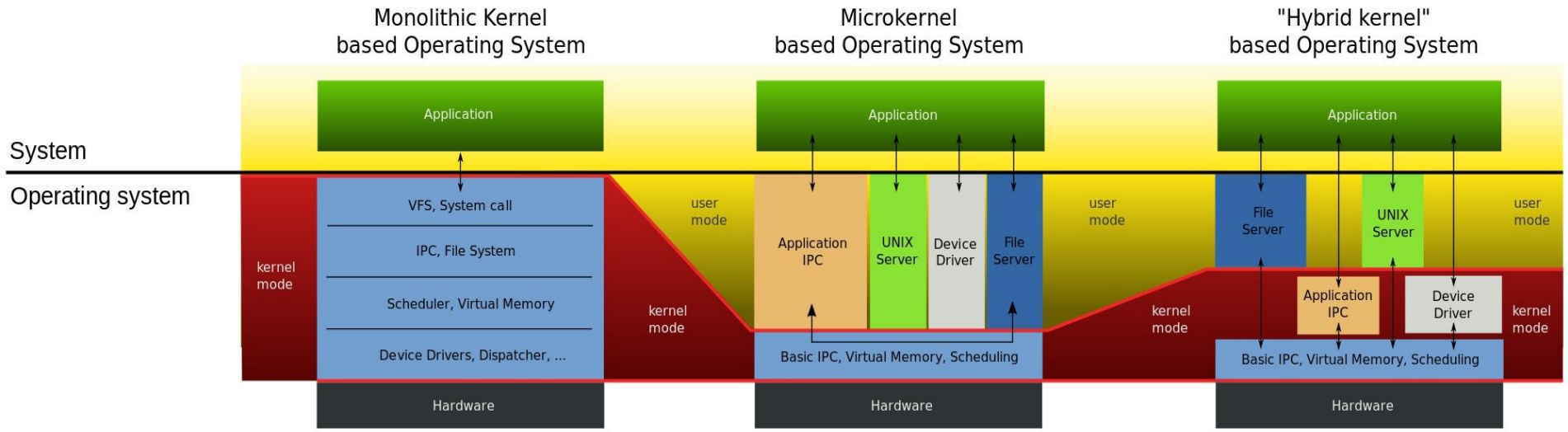
微内核（Micro-kernel）

- 尽量小型化（/简化）内核
  - 非常简单的硬件抽象层和必需的原语或系统调用组成，如进程管理、最基本的内存管理等。其他OS主要组件(如文件系统,设备驱动,I/O管理等)运行为独立进程
  - 由此，将系统服务的实现和系统的基本操作规则分离
  - 如Minix，Mach（后来成为XNU-Mac OS X核心的基础）
- 组件之间用消息传递的方式通信
- 优缺点
  - 易于扩展
  - 可靠性和安全性更容易保障
  - 消息传递机制导致运行效率偏低

“混合内核”

不少OS同时借鉴了单内核和微内核各自有优势的设计理念和方法，我们可以理解为是“混合内核”。

- 允许组件动态加载进内核，即工作在内核态
- 组件间通信混合采用函数调用和消息传递机制



嵌入式 Linux 模块机制

- Linux支持动态可加载内核模块（Loadable Kernel Module，LKM）
- 模块的编译需要配置过的内核源码，生成的内核模块后缀为.ko
- 可根据需要动态加载/卸载，载入内核后，便为内核的一部分，与其他部分地位一致
  - 内核充分模块化
  - 可控的内核代码规模
  - 配置高度灵活
  - 内核修改后不需重新编译内核和启动系统

思考：为什么嵌入式Linux要引入模块机制？模块机制体现了嵌入式系统的哪些特点？

嵌入式 Linux 模块机制

- Linux的**内核模块（Module）**，是**动态可加载内核模块（LKM）**的简称，是一种目标文件（object file），在其中包含了能在操作系统内核空间运行的代码。它们运行在核心基底（base kernel），通常是用来支持新的硬件（设备驱动），新的文件系统，或是新增的系统调用（system calls）。当不需要时，它们也能从存储器中被卸载。
- 内核模块是Linux内核的一部分，但是没有编译到内核里面去。
- 整体上说，Linux是单内核结构，通过模块机制加强了内核的可扩展性和可移植性。

嵌入式 Linux 内核模块相关的命令

- lsmod: 显示已载入系统的模块
- insmod: 载入模块
- rmmod: 删除模块
- modprobe: 自动处理可载入模块。可载入指定的个别模块，或是载入一组相依的模块。modprobe会根据depmod所产生的相依关系，决定要载入哪些模块。
- depmod: 可检测模块的相依性，供modprobe在安装模块时使用。

模块加载卸载示例

```c
#include<linux/module.h>
#include<linux/init.h>

static int __init init(void)
{
    printk("Hi module!\n");
    return 0;
}

static void __exit exit(void)
{
    printk("Bye module!\n");
}

module_init(init);
module_exit(exit);
```

- 编写一个简单的模块为例
- init为模块入口函数，在模块加载时被调用执行
- exit为模块出口函数，在模块卸载被调用执行

模块加载卸载示例

```
obj-m += main.o

#generate the path

CURRENT_PATH:=$(shell pwd)

#the current kernel version number

LINUX_KERNEL:=$(shell uname -r)

#the absolute path

LINUX_KERNEL_PATH:=/usr/src/linux-headers-$(LINUX_KERNEL)

#complie object

all:

    make -C (LINUX_KERNEL_PATH)M=(CURRENT_PATH) modules

#clean

clean:

    make -C (LINUX_KERNEL_PATH)M=(CURRENT_PATH) clean
```

- 使用make命令编译模块，得到模块文件main.ko
- 然后可以使用insmod来安装模块
- 使用rmmod来卸载模块

```
› sudo insmod main.ko;dmesg | tail -1
```

```
› sudo rmmod main.ko;dmesg | tail -1
```

- 可以分别看到输出
- Hi module!
- Bye module!

模块加载卸载示例

- 模块加载由内核的系统调用init_module完成
  - 系统调用init_module由SYSCALL_DEFINE3(init_module...)实现，其中有两个关键的函数调用。load_module用于模块加载，do_one_initcall用于回调模块的init函数。
    - 函数load_module内有四个关键的函数调用。copy_and_check将模块从用户空间拷贝到内核空间，layout_and_allocate为模块进行地址空间分配，simplify_symbols为模块进行符号解析，apply_relocations为模块进行重定位。
- 模块卸载由内核的系统调用delete_module完成
  - 通过回调exit完成模块的出口函数功能，最后调用free_module将模块卸载

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

概念回顾

- **中断**（Interrupt）是指处理器接收到来自硬件或软件的信号，提示发生了某个事件，应该被注意，这种情况就称为中断。
- 在接收到来自外围硬件的异步信号，或来自软件的同步信号之后，处理器将会进行相应的硬件/软件处理。发出这样的信号称为进行中断请求（interrupt request，IRQ）。
- 硬件中断导致处理器通过一个上下文切换（context switch）来保存执行状态。
- 软件中断则通常作为CPU指令集中的一个指令，以可编程的方式直接指示这种上下文切换，并将处理导向一段中断处理代码。

中断：同步vs异步

- 中断可分为同步（synchronous）中断和异步（asynchronous）中断
- 同步中断
  - 当指令执行时由CPU控制单元产生，之所以称为同步，是因为只有在一条指令执行完毕后 CPU 才会发出中断，而不是发生在指令代码执行期间，比如系统调用。
- 异步中断
  - 由其他硬件设备依照 CPU 时钟信号随机产生，即意味着中断能够在指令之间发生，例如键盘中断。

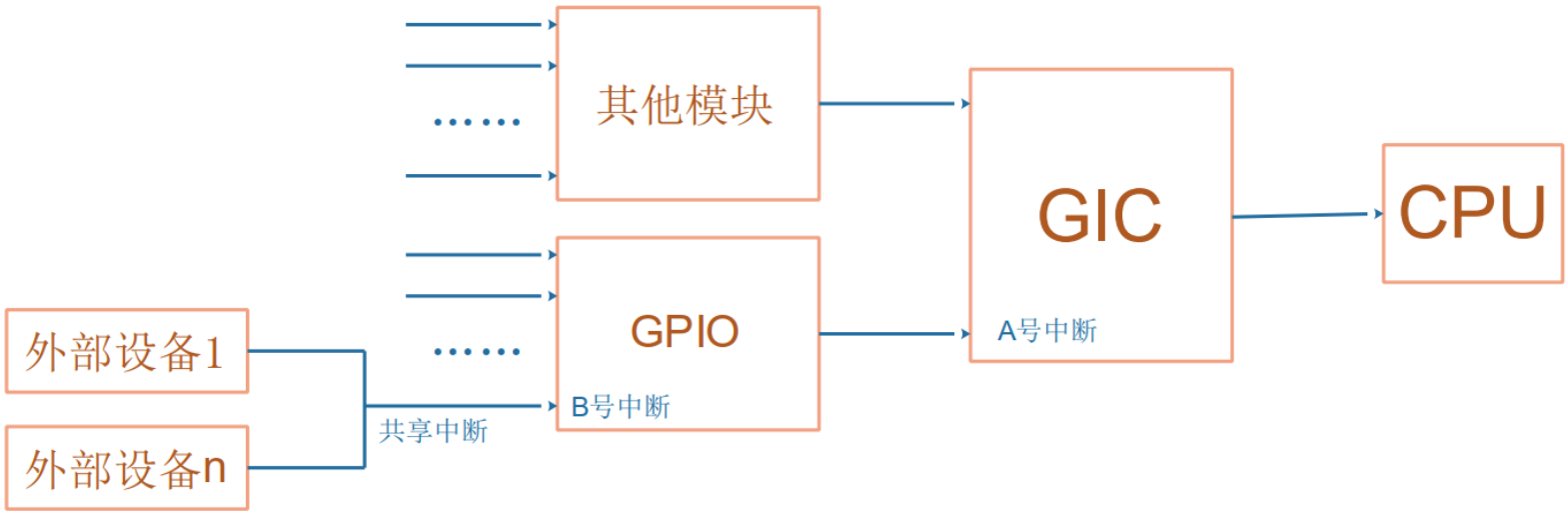
嵌入式 Linux 中断管理

- 中断处理是一个过程，一般分三个阶段
  - 中断响应：主要任务是确定中断源
  - 中断处理：进入中断服务程序，完成对应的处理操作
  - 中断返回

中断响应

- 中断响应的设计要素
  - 快速确定中断源、使用尽可能少的芯片引脚
- ARM的通用中断控制器（GIC，集成在CPU芯片内）
  - 多个外设可共享一条中断请求线, 多条中断请求线（相**或**）合一
  - 将外部中断汇总成一个IRQ中断请求
  - 包含1个中断请求寄存器和1个中断控制寄存器
  - 中断请求寄存器：每一位代表一个中断来源
  - 中断控制寄存器：通过各控制位的设置屏蔽或允许特定的中断源
- 中断向量：CPU在响应中断时向外发出“中断响应”（ACK）信号，中断源在接收ACK时，通过数据总线提供给CPU的一个代表具体设备的数值

中断处理架构示意图



嵌入式 Linux 中断管理

- ARM-Linux 中断处理前 CPU 动作
  - 将进入中断响应前的内容装入 r14_irq，即中断模式的 lr，使其指向中断点。
  - 将 cpsr (程序状态寄存器) 原来的内容装入 spsr_irq，即中断模式的 spsr (程序状态保存寄存器)；同时改变 cpsr 的内容使 CPU 运行于中断模式，并关闭中断。
  - 将堆栈指针 sp 切换成中断模式的 sp_irq。
  - 将 pc 指向 0x18（中断处理入口）。

GPIO (General-Purpose Input/Output)

- 在嵌入式系统中，经常需要控制许多结构简单的外部设备或者电路，这些设备有的需要通过CPU控制，有的需要CPU提供输入信号。
- GPIO是一个**通用的可编程的**I/O接口，每一位都可在程序的控制下设置用于输入或者输出 ；用于输入时，可引发中断请求。
- 一个GPIO端口至少需要两个寄存器
  - 通用I/O端口(GPIO)控制寄存器
  - 通用I/O端口(GPIO)数据寄存器: 每一位和GPIO的硬件引脚对应，数据的传递方向是通过控制寄存器设置的，通过控制寄存器可以设置每一位引脚的数据流向。
- 推荐阅读：**《GPIO就是芯片上的一根干啥都行的引脚》**
  http://www.elecfans.com/d/1127954.html

清華大學
Tsinghua University

本章提纲

- 嵌入式 Linux 简介
- 嵌入式 Linux 内存管理
- 嵌入式 Linux 进程管理
- 嵌入式 Linux 文件系统
- 嵌入式 Linux 模块机制
- 嵌入式 Linux 中断管理
- 嵌入式 Linux 系统调用

概念回顾

- **系统调用**（system call），指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务。系统调用提供用户程序与操作系统之间的接口。大多数系统交互式操作需求在内核态运行。如设备I/O操作或者进程间通信。
- 系统调用是应用程序从用户空间主动进入内核空间的唯一手段和途径。

嵌入式 Linux 系统调用

- Unix实现系统调用2种方式
  - 封装C库调用

    通常情况，每个特定的系统调用对应了至少一个 glibc 封装的库函数，如系统提供的打开文件系统调用 sys_open 对应的是 glibc 中的 open 函数；
  - 直接调用

    long int syscall (long int sysno, ...)

    sysno 是系统调用号，每个系统调用都有唯一的系统调用号来标识。在 sys/syscall.h 中有所有系统调用号的宏定义。
- Linux的第三种方式——自陷指令：产生软中断，将用户程序交给内核接管

自陷指令

- 在x86处理器上，Linux系统调用是通过自陷指令“INT 0x80”实现的。
- 系统调用号由eax传递，参数则通过寄存器传递，而不是通过堆栈传递。
  - 一共可以具有5个参数，顺序存放在ebx，ecx，edx，esi，edi。
  - 返回值通过eax返回。
- ARM处理器也具有自陷指令，SWI。
- 系统调用号就是SWI的操作数，参数结构和x86系统结构类似，一样通过寄存器传递
  - 一共具有5个参数，顺序存放在r0~r4
  - 返回值通过r0返回

x86体系系统调用（ARM体系类似）

- x86体系操作系统实现系统调用的基本过程是：
  - 应用程序调用库函数（API）；
  - API 将系统调用号存入 EAX，然后通过中断调用使系统进入内核态；
  - 内核中的中断处理函数根据系统调用号，调用对应的内核函数（系统调用）；
  - 系统调用完成相应功能，将返回值存入 EAX，返回到中断处理函数；
  - 中断处理函数返回到 API 中；
  - API 将 EAX 返回给应用程序。

特别提醒

- 下一章《嵌入式软件开发与调试》，需要提前配置好你们自己电脑上的环境。