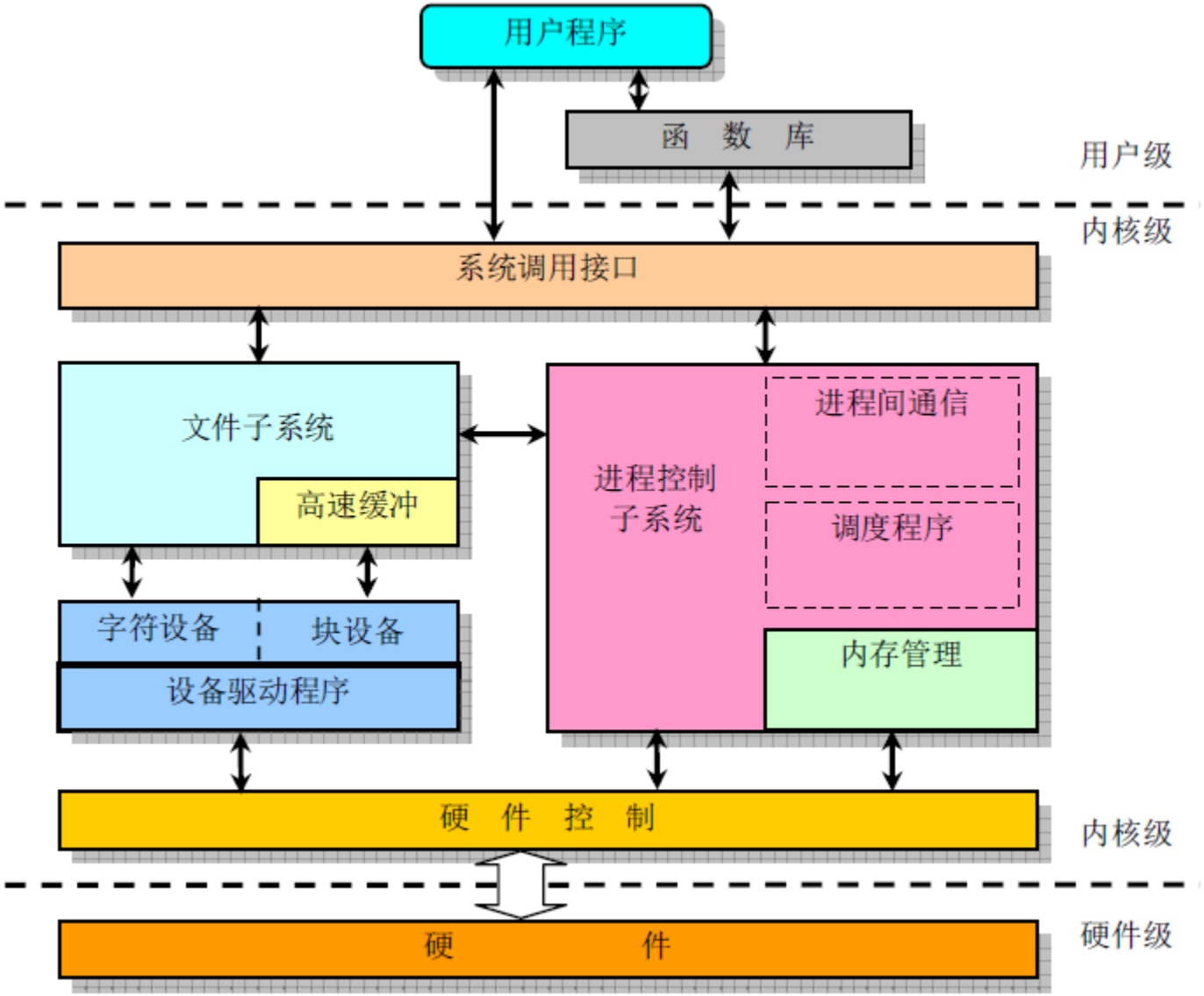
第5章 内核及其配置

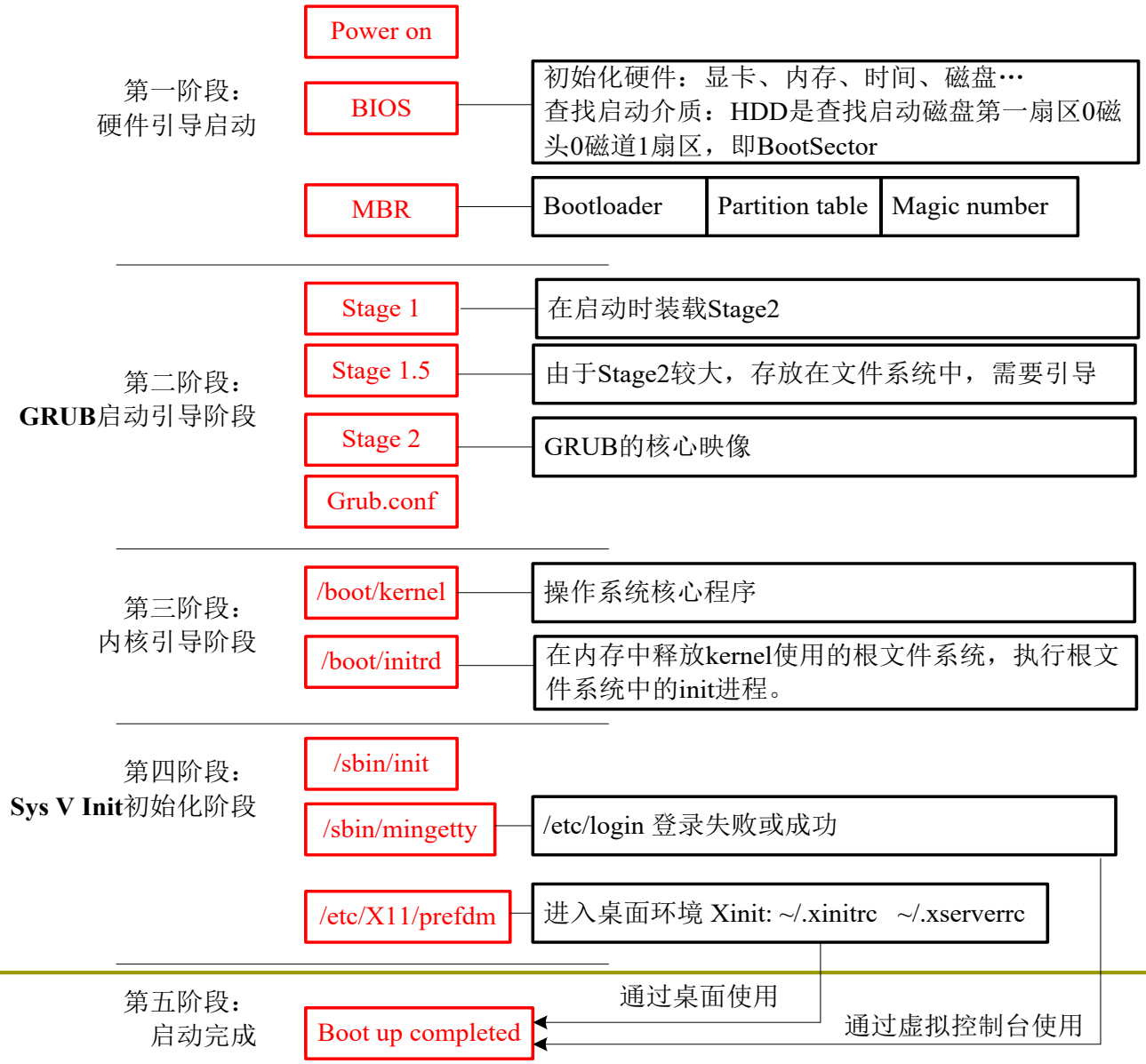
本章内容

- Linux内核体系结构
- 内核引导与设置
- 内核构建
- 系统调用

1. Linux内核体系结构



2. 内核引导与启动


Power on
第一阶段：
硬件引导启动
BIOS
初始化硬件：显卡、内存、时间、磁盘…
查找启动介质：HDD是查找启动磁盘第一扇区0磁头0磁道1扇区，即BootSector
MBR
Bootloader
Partition table
Magic number
第二阶段：
GRUB启动引导阶段
Stage 1
在启动时装载Stage2
Stage 1.5
由于Stage2较大，存放在文件系统中，需要引导
Stage 2
GRUB的核心映像
Grub.conf
第三阶段：
内核引导阶段
/boot/kernel
操作系统核心程序
/boot/initrd
在内存中释放kernel使用的根文件系统，执行根文件系统中的init进程。
第四阶段：
Sys V Init初始化阶段
/sbin/init
/sbin/mingetty
/etc/login 登录失败或成功
/etc/X11/prefdm
进入桌面环境 Xinit: ~/.xinitrc ~/.xserverrc
第五阶段：
启动完成
通过桌面使用
通过虚拟控制台使用
Boot up completed

BIOS/OF

- 加电后处理器首先访问通常位于只读内存（一般是Flash ROM或仅仅是Flash）中的某一地址。
- BIOS（基本输入输出）是x86系统加电后最先运行的代码。引导系统并与硬件相关的系统初始化代码。
- Open Firmware是PPC（PowerPC）系统加电后最先运行的代码。

BIOS功能

- 自检及初始化程序；
  - 自检，对CPU，640K基本内存，1M以上的扩展内存，ROM，主板，CMOS存储器等进行测试。
  - 初始化，包括创建中断向量、设置寄存器等。
  - 引导程序 ，引导DOS或Linux等操作系统。
- 硬件中断处理；系统在加电引导机器时，要读取CMOS信息，用来初始化机器各个部件的状态。
- 程序服务请求；程序服务处理程序主要是为应用程序和操作系统服务，这些服务主要与输入输出设备有关，例如读磁盘、文件输出到打印机等。

引导装入程序（Boot Loaders）

- 格式化磁盘时，会创建主引导记录（MBR），该记录存储在引导设备的第一个扇区（0扇区、0磁道、0磁头）。包含：
  - 一个小程序
  - 一张四入口点的分区表
  - 结束标识符（0XAA55），用来做MBR的有效性检测。
- GRUB（Grand Unified Bootloader），基于x86的引导装入程序，用来加载Linux。
- LILO（Linux Loader）x86中Linux的加载程序。
- Yaboot是基于PowerPC及其OF的引导装入程序。

LILO

- LILO （Linux Loader）已成为所有 Linux 发行版的标准组成部分。
- 作为一个较老的Linux 引导加载程序，它那不断壮大的 Linux 社区支持使它能够随时间的推移而发展，并始终能够充当一个可用的现代引导加载程序。

GNU GRUB

- GNU GRUB（GRand Unified Bootloader简称“GRUB”）是一个来自GNU项目的多操作系统启动程序。
- GRUB允许用户可以在计算机内同时拥有多个操作系统，并在计算机启动时选择希望运行的操作系统。
- GRUB可用于选择操作系统分区上的不同内核，也可用于向这些内核传递启动参数。

GRUB2

- GRUB2（GRand Unified Bootloader, Version 2）是GRUB的第二版。
- GRUB目前已经不再继续开发，只是修正存在的错误。
- 官方的说法是GRUB的编码实在太烂，以至于没法进行维护升级了，所以重新从零开发了新的GRUB2，从此以后原来的GRUB就叫做GRUB Legacy了。
- GRUB2对GRUB2的接口进行了完整地重写，并且采用了清晰的架构和模块化的布局。
- 目前大部分都采用GRUB2作为内核引导管理器。

体系结构相关的内存初始化

- X86和PowerPC在硬件方面都具有支持实寻址和虚寻址的内存管理特征。
- Linux的内存管理依赖于底层的硬件结构。
- 现在PPC和x86的代码都集中在init/main.c的start_kernel()中，位于与体系结构无关的代码段中，调用特定体系结构的历程来完成内存初始化。

开始：start_kernel()

- Init/main.c中start-kernel()，执行进程0（即超级用户线程root thread），进程0又产生进程1（即init进程），然后进程0就编程CPU的空闲进程。
- Linux内核只提供了轻量进程的支持，限制了更高效的线程模型的实现。目前最流行的线程机制LinuxThreads所采用的就是线程-进程“一对一”模型，调度交给核心，而在用户级实现一个包括信号处理在内的线程管理机制。

3. 内核构建

- Linux集成套件包括多种内核，能够处理各种机器。通过编译内核，选择符合硬件类型的驱动等，可以调整Linux系统，使其更合理地安装到计算机中。
- 重新编译内核以便实现一些新功能，如将Linux系统设置为一个临时路由器。
- 使得全世界内核设计者提供的各种为改进性能而设计的内核得到充分利用。

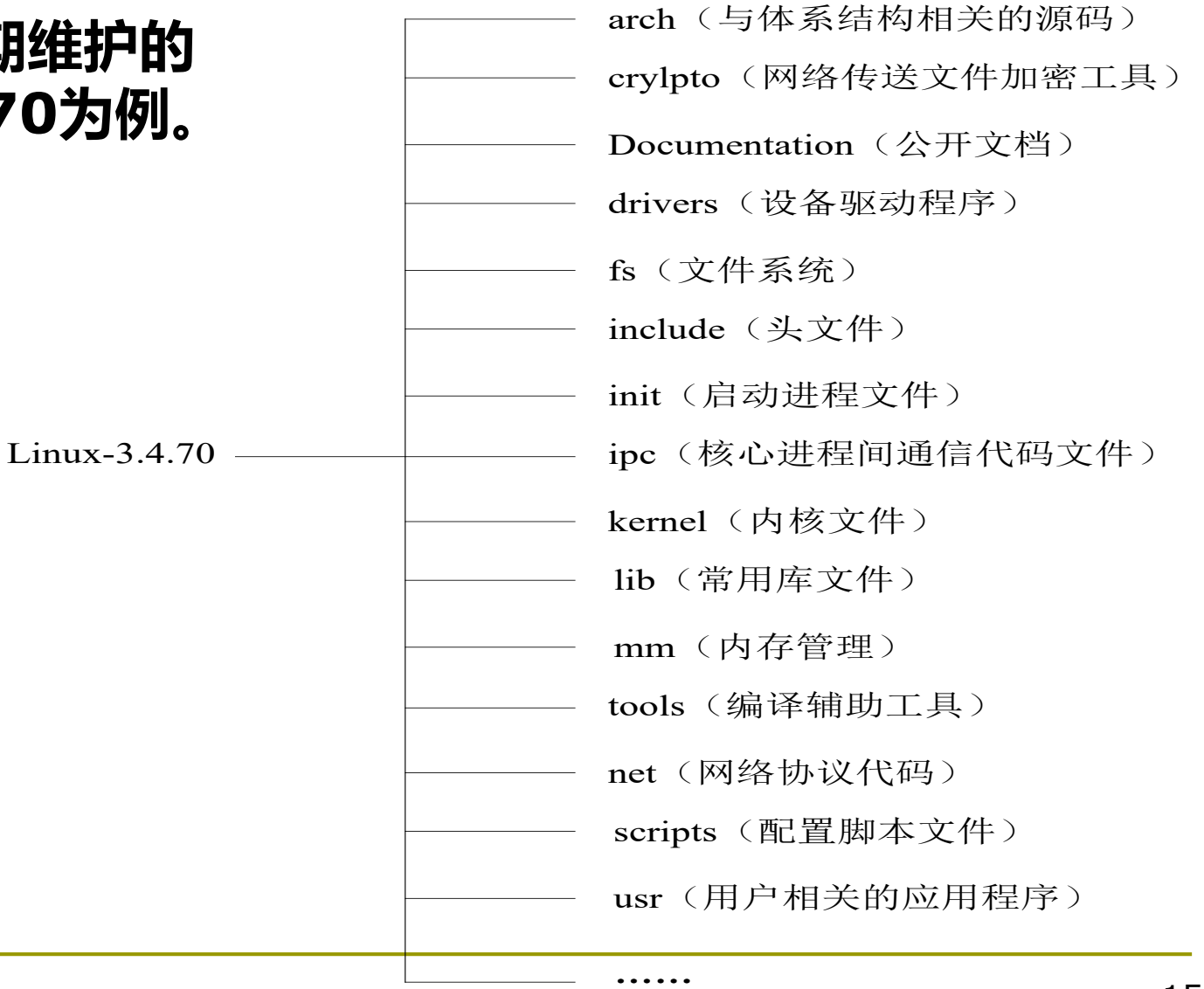
构建Linux内核

- Linux官方源代码发布网址：www.kernel.org
- gzip压缩的.tar.gz包，bzip2压缩的.tar.bz2。

- Linux源代码分为：
  - 与系统结构相关的部分
  - 与系统结构无关的部分
  - 文档和工具

Linux内核文件组织结构

- **以内核长期维护的版本3.4.70为例。**



Linux内核文件说明

- arch：包含了所有和体系结构相关的核心代码，它的每一个子目录都代表一种被支持的体系结构。
- include：包含编译核心所需要的大部分头文件，与平台无关的头文件放在include/linux子目录中。
- init：包含核心的初始化代码。
- mm：包含所有独立于CPU体系结构的内存管理代码。
- kernel：主要的核心代码，实现大多数Linux系统的内核函数，包括进程调度、系统调用等。

Linux内核文件说明（续）

- drivers：系统所有的设备驱动程序，每种驱动程序各占用一个子目录。
- 其它：lib放置核心的库代码；net放置核心与网络相关的代码；Ipc包含核心的进程间通信的代码；fs包含文件系统代码；scripts包含用于配置核心的脚本文件。

Linux内核的makefile文件

- 源代码树的每个子目录下都有一个makefile文件。
- 在源代码树的根目录下执行make，则调用顶层makefile文件，它定义了随后要输出到其他makefile的变量，以及向子目录中的每个makefile发出make调用。
- Script/makefile.build中定义了makefile向下级子目录递归并编译的规则。

编译内核过程

- 第1步，预处理
- 第2步，配置内核
- 第3步，生成内核
- 第4步，安装内核
- 第5步，建立模块

第1步，预处理

- Linux内核源文件缺省位置：/usr/src/linux
- 从Internet下载最新版本到你创建的主目录。
  - 如~yan
- 清除以前试图建立内核过程遗留下的多余文件。
  - Make mrproper

第2步，配置内核

- make config：手工逐项配置
- make menuconfig：菜单选项配置
- make xconfig： XWindow配置
- 修改配置文件/linux/.config
  - 注意，make mrproper命令要删除这个文件，可以从/linux/arch/i386/defconfig拷贝复制一个。

第3步，生成内核

- 有三步：
  - 1、make dep：生成相关性
  - 例如：如果激活“Set Version Information For All Symbols On Modules”选项，那么它为所建立的模块确定其版本信息。
  - 2、make clean：清除一些目录中现有文件，将存储创建的新文件。
  - 3、make bzImage：编译内核本身，花费时间长。对于新内核规模小，可以使用make zImage，如果不确定，最好还是使用bzImage。
  - 建立/linux/arch/i386/boot/bzImage

第4步，安装内核

- 有些集成套件使用LILO作为引导装入程序。
  - /etc/lilo.conf文件中的“image＝”
- su命令成为超级用户登录，把刚创建的bzImage拷贝到/boot中。
  - cp ~yan/linux/arch/i386/boot/bzImage /boot/vmLinuz
  - 修改lilo.conf文件中“image”行。
  - 告诉LILO更新其配置信息:/sbin/lilo

第5步，建立模块

- 配置Linux内核时，可将许多选项配置为模块而不是直接放进内核。
- 每个模块可以分别装入和卸载。
- /linux目录下
  - make modules：创建在配置过程中要求的模块，但是并不安装。
  - Make modules_install：将已经完成的模块拷贝到对应该内核版本的/lib/modules/子目录中。

管理多内核

- 不同的情况使用不同的内核，lilo.conf文件：

```
 ……
delay=15   #15-second delay
……
image=/boot/vnlinux
label=Linux
……
image=/home/yan/bzImage
Label=TestKernel
```

最后，执行/sbin/lilo

Added Linux * (表示Linux标记为缺省内核)

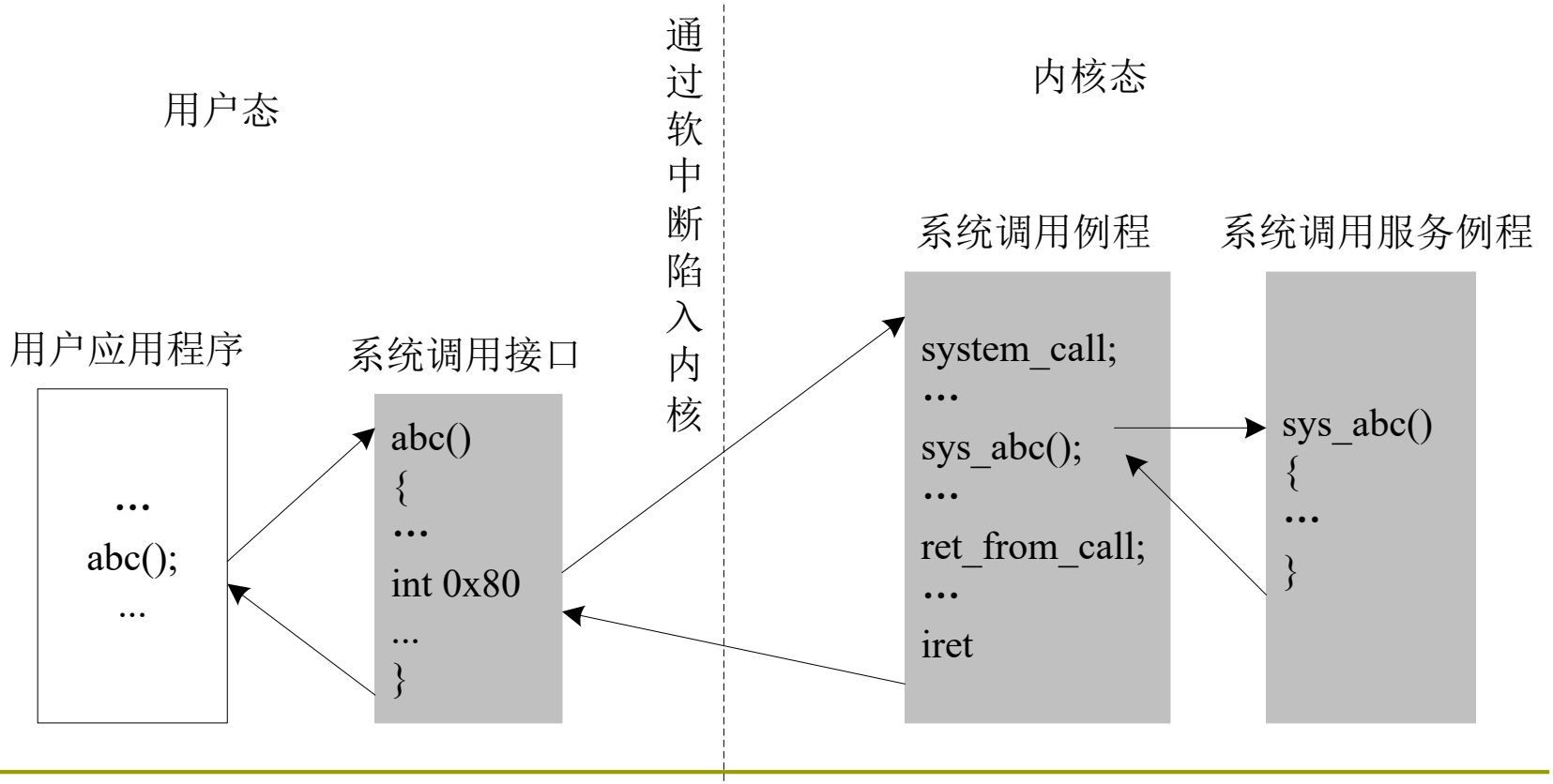
Added TestKernel（表示添加新内核）

4. 系统调用

- 每个系统调用都是通过一个单一的入口点多路传入内核。eax 寄存器用来标识应当调用的某个系统调用，这在 C 库中做了指定（来自用户空间应用程序的每个调用）。
- 当加载了系统的 C 库调用索引和参数时，就会调用一个软件中断（0x80 中断），它将执行 system_call 函数（通过中断处理程序），这个函数会按照 eax 内容中的标识处理所有的系统调用。
- 在经过几个简单测试之后，使用 system_call_table 和 eax 中包含的索引来执行真正的系统调用了。从系统调用中返回后，最终执行 syscall_exit，并调用 resume_userspace 返回用户空间。然后继续在 C 库中执行，它将返回到用户应用程序中。

系统调用的实现

- **系统调用是操作系统向用户提供的程序一级的服务，用户程序借助于系统调用命令向操作系统提出各种资源要求和服务请求。**



系统调用处理

系统调用过程描述

（1）用户程序调用C库中的API。

（2）API里面有软中断int 0x80语句，这条指令的执行会让系统跳转到一个预设的内核空间地址，它指向系统调用处理程序，即system_call函数，同时把系统调用号放入eax寄存器中。

（3）system_call系统调用处理程序是在执行系统调用服务例程之前的一个引导过程，针对int 0x80这条指令，面向所有的系统调用。system_call读取eax寄存器，获取系统调用号，将其乘以4，生成偏移地址，并以sys_call_table为基址，转到执行具体的系统调用服务例程。

系统调用过程描述（续）

（4）系统调用服务例程将从堆栈里获取参数并执行。由于system_call执行前，会先将参数存放在寄存器中，system_call执行时会首先将这些寄存器压入堆栈。system_call退出后，用户可以从寄存器中获得（被修改过的）参数。

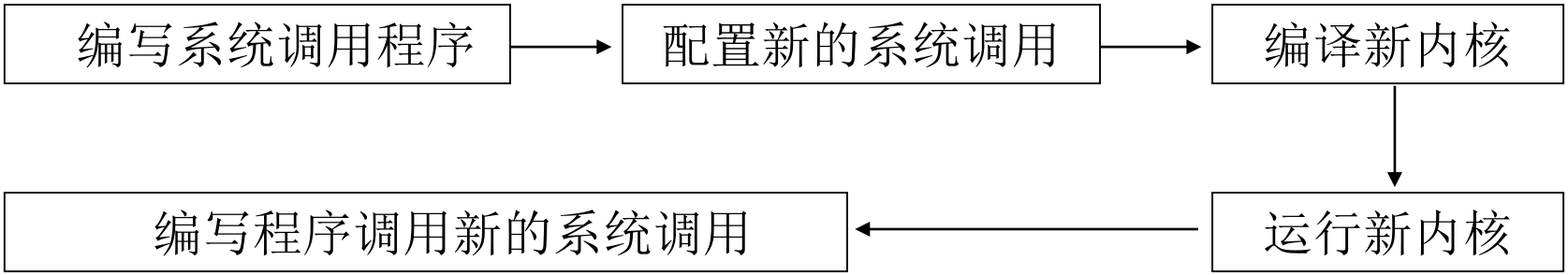
注意：系统调用通过软中断INT 0x80陷入内核，跳转到系统调用处理程序system_call函数，然后执行相应的服务例程。但是由于是代表用户进程，所以这个执行过程并不属于中断上下文，而是进程上下文。因此，系统调用执行过程中，可以访问用户进程的许多信息，可以被其它进程抢占，可以休眠。

系统调用过程描述（续）

（5）系统调用完成后，把控制权交回到发起调用的用户进程前，内核会有一次调度。如果发现有优先级更高的进程或当前进程的时间片用完，那么会选择优先级更高的进程或重新选择进程执行。

增加系统调用

- **如果用户在Linux中添加新的系统调用，应该遵循几个步骤才能添加成功。**



第1步，添加新的系统调用函数源代码

- 第一个任务是编写加到内核中的源程序，即将要加到一个内核文件中的一个函数，该函数的名称应该是新的系统调用名称前面加上sys_标志。
- 假设新加的系统调用为mycall(int number)，在/usr/src/Linux/kernel/sys.c文件中添加源代码，它对应的函数名称为sys_mycall。

第2步，连接新的系统调用

- 添加新的系统调用后，下一个任务是使Linux内核的其余部分知道该程序的存在。为了从已有的内核程序中增加到新的函数的连接，需要编辑两个文件：
  - /usr/src/Linux/include/asm-i386/unistd.h：该文件中包含了系统调用清单，用来给每个系统调用分配一个唯一的号码。
  - /usr/src/Linux/arch/i386/kernel/entry.S：该清单用来对sys_call_table[]数组进行初始化。该数组包含指向内核中每个系统调用的指针。这样就在数组中增加了新的内核函数的指针。

第3步，重建新的Linux内核

- 为使新的系统调用生效，需要重建Linux的内核。
- 这需要以超级用户身份登录。
- 具体方法可以参考内核编译步骤。

第4步，使用新的系统调用

- 在应用程序中使用新添加的系统调用。
- 由于使用了系统调用，编译和执行程序时，用户都应该是超级用户身份。

本章小结

- 虽然Linux 内核是一个庞大而复杂的操作系统核心，但是由于采用子系统和分层的概念，具有很好的扩展性和可读性。
- Linux的优势就在于用户能够定制内核，内核编译可以使得全世界内核设计者提供的各种为改进性能而设计的内核得到充分利用。
- 系统调用是操作系统向用户提供的程序一级的服务，用户在Linux中添加新的系统调用后，需要重新编译内核、运行新的内核之后才能使用。