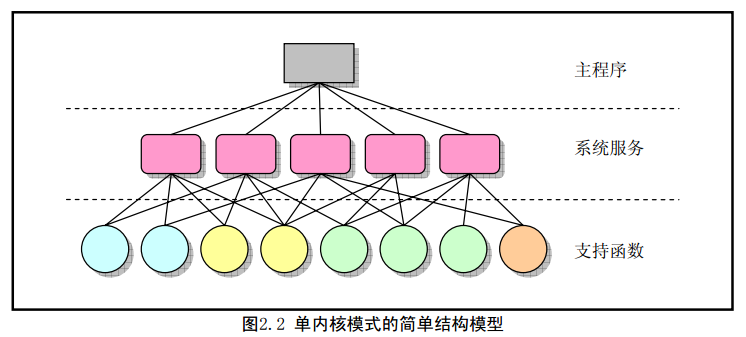
# 1.1内核模式

在单内核模式的系统中，操作系统所提供服务的流程为：应用主程序使用指定的参数值执行**系统调用指令(int x80)**，使 **CPU 从用户态（User Mode）切换到核心态（Kernel Model）**，然后操作系统根据具体的参数值调用特定的系统调用服务程序，而这些服务程序则根据需要再底层的一些支持函数以完成特定的功能。在完成了应用程序所要求的服务后，操作系统又从核心态切换回用户态，返回到应用程序中继续执行后面的指令。因此概要地讲，单内核模式的内核也可粗略地分为三个层次：**调用服务的主程序层、执行系统调用的服务层和支持系统调用的底层函数。**

# 1.2系统体系结构

Linux 内核主要由 5 个模块构成，它们分别是：**进程调度模块、内存管理模块、文件系统模块、进程间通信模块和网络接口模块。**

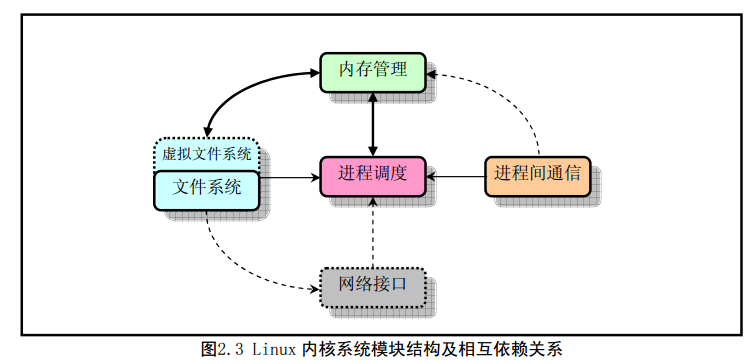
**进程调度模块用来负责控制进程对 CPU 资源的使用**。所采取的调度策略是各进程能够公平合理地访问 CPU，同时保证内核能及时地执行硬件操作。

**内存管理模块**用于确保所有**进程能够安全地共享机器主内存区**，同时，内存管理模块还**支持虚拟内存管理方式**，使得 Linux 支持进程使用比实际内存空间更多大的内存容量。并可以利用文件系统把暂时不用的内存数据块会被交换到外部存储设备上去，当需要时再交换回来。

**文件系统模块**用于**支持对外部设备的驱动和存储**。虚拟文件系统模块通过向所有的外部存储设备提供一个通用的文件接口，隐藏了各种硬件设备的不同细节。从而提供并支持与其它操作系统兼容的多种文件系统格式。

**进程间通信模块**子系统用于**支持多种进程间的信息交换方式**。

**网络接口模块**提供对**多种网络通信标准的访问并支持许多网络硬件**。



所有的模块都与进程调度模块存在依赖关系。因为它们都需要依靠进程调度程序来挂起（暂停）或重新运行它们的进程。通常，一个模块会在等待硬件操作期间被挂起，而在操作完成后才可继续运行。例如，当一个进程试图将一数据块写到软盘上去时，软盘驱动程序就可能在启动软盘旋转期间将该进程置为挂起等待状态，而在软盘进入到正常转速后再使得该进程能继续运行。另外 3 个模块也是由于类似的原因而与进程调度模块存在依赖关系。

其它几个依赖关系有些不太明显，但同样也很重要。进程调度子系统需要使用内存管理器来调整一特定进程所使用的物理内存空间。进程间通信子系统则需要依靠内存管理器来支持共享内存通信机制。这种通信机制允许两个进程访问内存的同一个区域以进行进程间信息的交换。虚拟文件系统也会使用网络接口来支持网络文件系统（NFS），同样也能使用内存管理子系统来提供内存虚拟盘（ramdisk）设备。而内存管理子系统也会使用文件系统来支持内存数据块的交换操作。

所有这些模块还会依赖于内核中的通用资源。这些资源包括

内核所有子系统都会调用的内存分配和收回函数、打印警告或出错信息函数以及一些系统调试函数。

# 1.3 Linux内核进程控制

程序是一个可执行的文件，而进程（process）是一个执行中的程序实例。在 Linux 操作系统上同时可以执行多个进程。 对于 linux 0.11 内核来讲， 系统最多可有 64 个进程同时存在。 系统除了第一个进程是“手工”建立以外， 其余的都是进程使用系统调用 fork 创建的新进程， 被创建的进程称为子进程（child process），创建者，则称为父进程（parent process）。内核程序使用进程标识号（process ID， pid）来标识每个进程。

**进程由可执行的指令代码、数据和堆栈区组成。**进程中的代码和数据部分分别对应一个执行文件中的代码段、数据段。每个进程只能执行自己的代码和访问自己的数据及堆栈区。进程之间相互之间的通信需要通过系统调用进行。对于只有一个 CPU 的系统，在某一时刻只能有一个进程正在运行。内核通过调度程序分时调度各个进程运行。

Linux 系统中，一个进程可以在内核态（kernel mode）或用户态（user mode）下执行，因此， linux 内核栈和用户栈是分开的。

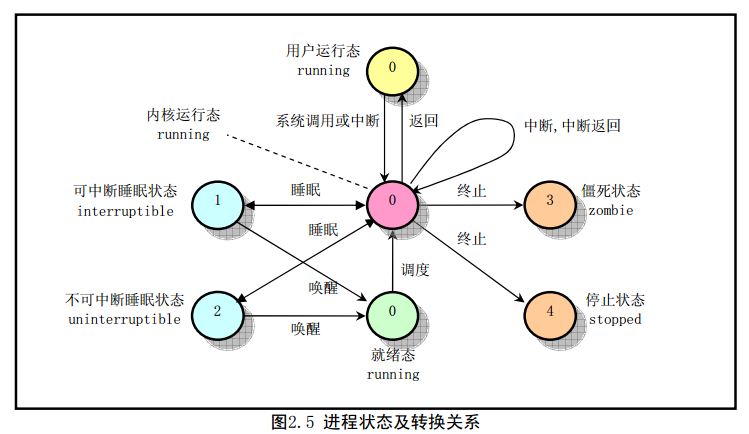
用户栈用于进程在用户态下临时保存调用函数的参数、局部变量等数据。

内核栈则含有内核程序执行函数调用时的信息。

内核程序是通过进程表对进程进行管理的，每个进程在进程表中占有一项。在 linux 系统中，进程表项是一个 task 结构。当一个进程在执行时， CPU 的所有寄存器中的值、进程的状态以及堆栈中的内容被称为该进程的上下文。

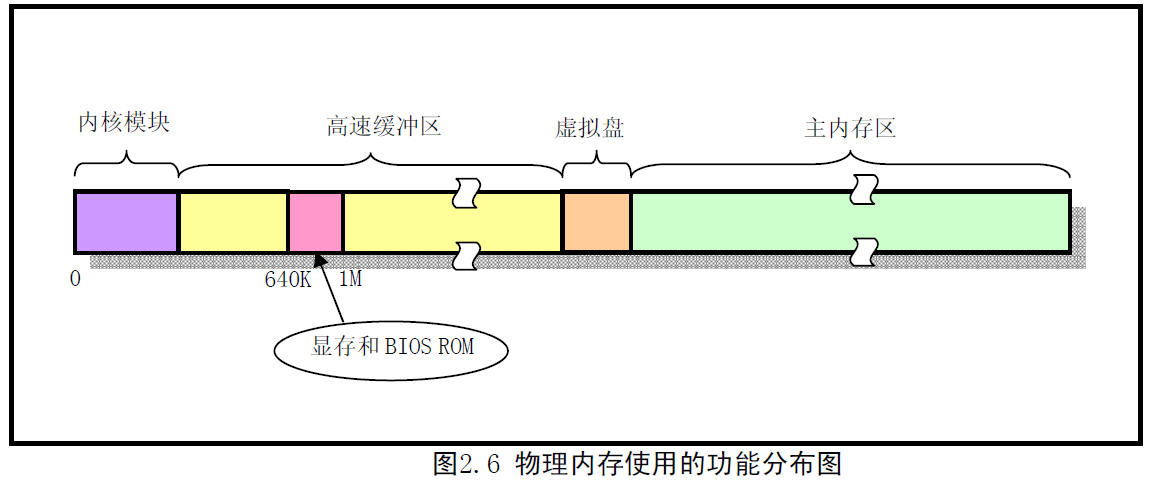
当内核需要切换（switch）至另一个进程时，它就需要保存当前进程的所有状态，也即保存当前进程的上下文，以便在再次执行该进程时，能够恢复到切换时的状态执行下去。在发生中断时，内核就在被中断进程的上下文中，在内核态下执行中断服务例程。但同时会保留所有需要用到的资源，以便中断服务结束时能恢复被中断进程的执行。

一个进程在其生存期内，可处于一组不同的状态下，称为进程状态。



# 1.4 内核对内存的使用

为了有效地使用系统的物理内存，内存被划分成几个功能区域。



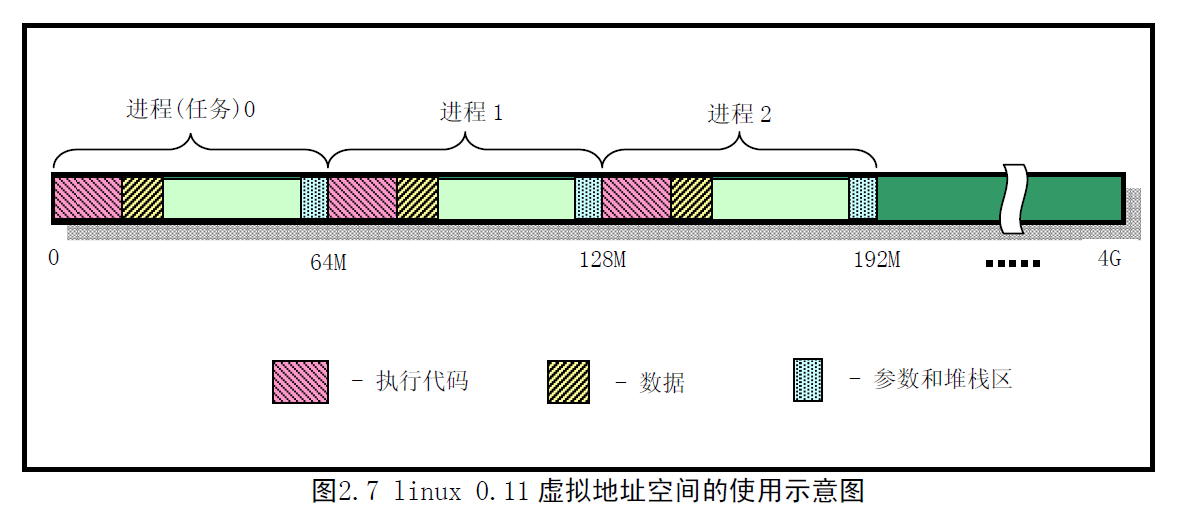
1. linux 内核程序占据在物理内存的开始部分
2. 供硬盘或软盘等块设备使用的高速缓冲区部分

进程需要读取块设备中的数据时，系统会首先将数据读到高速缓冲区中；当有数据需要写到块设备上去时，系统也是先将数据放到高速缓冲区中，然后由块设备驱动程序写到设备。

1. 所有程序可以随时申请使用的主内存区部分。内核程序在使用主内存区时，也同样要首先向内核的内存管理模块提出申请，在申请成功后方能使用。对于含有RAM 虚拟盘的系统，主内存区头部还要划去一部分，共虚拟盘存放数据。

计算机系统中所含的实际物理内存容量是有限制的。为了能有效地使用这些物理内存，Linux 采用了Intel CPU 的内存分页管理机制，使用虚拟线性地址与实际物理内存地址映射的方法让所有同时执行的程序共同使用有限的内存。内存分页管理的基本原理是将整个主内存区域划分成4096 字节为一页的内存页面。程序申请使用内存时，就以内存页为单位进行分配。

对于Intel 80386 系统，其CPU 可以提供多达4G 的线性地址空间。对于linux 0.11 内核，系统设置全局描述符表GDT 中的段描述符项数最大为256，其中2 项空闲、2 项系统使用，每个进程使用两项。因此，此时系统可以最多容纳(256-4)/2 + 1=127 个任务，并且虚拟地址范围是 ((256-4)/2)\* 64MB 约等于8G。但0.11 内核中人工定义最大任务数NR\_TASKS = 64 个，每个进程虚拟地址（或线性地址）范围是64M，并且各个进程的虚拟地址起始位置是(任务号-1)\*64MB。因此所使用的虚拟地址空间范围是64MB\*64 =4G，见图2.7 所示。4G 正好与CPU 的线性地址空间范围或物理地址空间范围相同



1. 进程虚拟地址，是从虚拟地址0 开始计，最大64M；

进程的虚拟地址需要首先通过其局部段描述符变换为CPU 整个线性地址空间中的地址，然后再使用页目录表PDT（一级页表）和页表PT（二级页表）映射到实际物理地址页上。

b. CPU 的线性地址空间（0--4G）；

c. 实际物理内存地址。

为了使用实际物理内存，每个进程的线性地址通过二级内存页表动态地映射到主内存区域的不同内存页上。因此每个进程最大可用的虚拟内存空间是64MB。每个进程的逻辑地址通过加上任务号\*64M，即可转换为线性地址。