# LINUX 内核经典面试题 30 道

1) Linux 中主要有哪几种内核锁？

2) Linux 中的用户模式和内核模式是什么含意？

3) 怎样申请大块内核内存？

4) 用户进程间通信主要哪几种方式？

5) 通过伙伴系统申请内核内存的函数有哪些？

6) 通过 slab 分配器申请内核内存的函数有？

7) Linux 的内核空间和用户空间是如何划分的（以 32 位系统为例）？

8) vmalloc()申请的内存有什么特点？

9) 用户程序使用 malloc()申请到的内存空间在什么范围？

10) 在支持并使能 MMU 的系统中，Linux 内核和用户程序分别运行在物理地址模式还是虚拟地址模式？

11) ARM 处理器是通过几级也表进行存储空间映射的？

12) Linux 是通过什么组件来实现支持多种文件系通的？

13) Linux 虚拟文件系统的关键数据结构有哪些？（至少写出四个）

14) 对文件或设备的操作函数保存在那个数据结构中？

15) Linux 中的文件包括哪些？

16) 创建进程的系统调用有那些？

17) 调用 schedule()进行进程切换的方式有几种？

18) Linux 调度程序是根据进程的动态优先级还是静态优先级来调度进程的？

19) 进程调度的核心数据结构是哪个？

20) 如何加载、卸载一个模块？

21) 模块和应用程序分别运行在什么空间？

22) Linux 中的浮点运算由应用程序实现还是内核实现？

23) 模块程序能否使用可链接的库函数？

24) TLB 中缓存的是什么内容？

25) Linux 中有哪几种设备？

26) 字符设备驱动程序的关键数据结构是哪个？

27) 设备驱动程序包括哪些功能函数？

28) 如何唯一标识一个设备？

29) Linux 通过什么方式实现系统调用？

30) Linux 软中断和工作队列的作用是什么？

**参考解答**

1. 主要有哪几种内核锁？ Linux 内核的同步机制是什么？

Linux 的内核锁主要是自旋锁和信号量。

自旋锁：如果在获取自旋锁时，没有任何执行单元保持该锁，那么将立即得到锁；如果在获取自旋锁时锁已经有保持者，那么获取锁操作将自旋（忙循环，不会引起调用者睡眠）在那里，直到该自旋锁的保持者释放了锁。

自旋锁是专为防止多处理器并发而引入的一种锁，它在内核中大量应用于中断处理等部分(对于单处理器来说，防止中断处理中的并发可简单采用关闭中断的方式，即在标志寄存器中关闭/打开中断标志位，不需要自旋锁)。

spin_lock_init(x)

该宏用于初始化自旋锁 x。自旋锁在真正使用前必须先初始化。

spin_lock();

该宏用于获得自旋锁 lock，只有它获得锁才返回。

spin_trylock(lock)

该宏尽力获得自旋锁 lock，如果能立即获得锁，它获得锁并返回真，否则不能立即获得锁，立即返回假。它不会自旋等待 lock 被释放。

spin_unlock(),

该宏释放自旋锁 lock

Linux 提供两种信号量：

内核信号量，由内核使用。

IPC 信号量， 由用户进程使用。

Linux 内核的信号量在概念和原理上与用户态的 System V 的 IPC 机制信号量是一样的，但是它绝不可能在内核之外使用，因此它与 System V 的 IPC 机制信号量毫不相干。

信号量在创建时需要设置一个初始值，表示同时可以有几个任务可以访问该信号量保护的共享资源，初始值为 1 就变成互斥锁（Mutex），即同时只能有一个任务可以访问信号量保护的共享资源。一个任务要想访问共享资源，首先必须得到信号量，获取信号量的操作将把信号量的值减 1，若当前信号量的值为负数，表明无法获得信号量，该任务必须挂起在该信号量的等待队列等待该信号量可用；若当前信号量的值为非负数，表示可以获得信号量，因而可以立刻访问被该信号量保护的共享资源。当任务访问完被信号量保护的共享资源后，必须释放信号量，释放信号量通过把信号量的值加 1 实现，如果信号量的值为非正数，表明有任务等待当前信号量，因此它也唤醒所有等待该信号量的任务。

void sema_init (struct semaphore *sem, int val);

该函用于数初始化设置信号量的初值，它设置信号量 sem 的值为 val。

void down(struct semaphore * sem);

该函数用于获得信号量 sem，它会导致睡眠，

int down_interruptible(struct semaphore * sem);

该函数功能与 down 类似，不同之处为，down 不会被信号（signal）打断，但 down_interruptible 能被信号打断，因此该函数有返回值来区分是正常返回还是被信号中断，如果返回 0，表示获得信号量正常返回，如果被信号打断，返回-EINTR。

int down_trylock(struct semaphore * sem);

该函数试着获得信号量 sem，如果能够立刻获得，它就获得该信号量并返回 0，否则，表示不能获得信号量 sem，返回值为非 0 值。因此，它不会导致调用者睡眠，可以在中断上下文使用。

void up(struct semaphore * sem);

该函数释放信号量 sem，即把 sem 的值加 1.

1.自旋锁在申请锁失败后，不断忙循环等待锁可用。

2.信号量在申请失败后，把自己放到等待队列，然后睡眠。

自旋锁可以用在 ISR 中，而 ISR 中不可以使用信号量。但是在中断程序中，就要小心使用，因为当进程拥有一个自旋锁的时候，被中断程序打断，而在中断处理程序中又同样申请该锁，这样就造成了死锁。所以在进程和 中断服务程序都要访问一内核数据的时候，一般只要在进程中或者内核代码中申请自选锁时要禁止中断。就是调用 spin_lock_irqsave,spin_unlock_irqstroe 函数。

Linux 内核中的同步机制：原子操作、信号量、读写信号量和自旋锁的 API，另外一些同步机制，包括大内核锁、读写锁、大读者锁、RCU (Read-Copy Update，顾名思义就是读-拷贝修改)，和顺序锁。

2. Linux 中的用户模式和内核模式是什么含意？

用户模式是种受限模式，它对内存和硬件的访问都必须用过系统调用实现，用户程序运行在用户模式。它用于用户进程。内核模式是一种高特权模式，其中的程序代码能直接访问内存和硬件。内核程序运行在内核模式。

3. 怎样申请大块内核内存？

vmalloc()内核用于申请大块内存，特点是线性地址连续，物理地址不一定连续，不能直接用于DMA。对应的释放函数为 vfree()；

kmalloc()内核用于申请小内存，它基于 slab 实现的,slab 是为分配小内存提供的一种高效机制。最多只能开辟大小为 32XPAGE_SIZE-16 字节的内存,一般的 PAGE_SIZE=4kB,也就是 128kB-16 字节的大小的内存，16 个字节是被页描述符结构占用了。 kmalloc 最大只能开辟 128k-16.特点是线性地址连续，物理地址连续。对于要进行 DMA 的设备十分重要。对应的释放函数为 kfree()；

4. 用户进程间通信主要哪几种方式？

（1）管道（Pipe）：管道可用于具有亲缘关系进程间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。

（2）命名管道（named pipe）：命名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。命名管道在文件系统中有对应的文件名。命名管道通过命令 mkfifo 或系统调用 mkfifo 来创建。

（3）信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外,进程还可以发送信号给进程本身;linux 除了支持 Unix 早期信号语义函数 sigal 外，还支持语义符合 Posix.1 标准的信号函数 sigaction（实际上，该函数是基于 BSD 的，BSD 为了实现可靠信号机制，又能够统一对外接口，用 sigaction 函数重新实现了 signal 函数）。

（4）消息（Message）队列：消息队列是消息的链接表，包括 Posix 消息队列 system V 消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺

（5）共享内存：使得多个进程可以访问同一块内存空间，是最快的可用 IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

（6）信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。

（7）套接字（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的,但现在一般可以移植到其它类Unix系统上:Linux和System V 的变种都支持套接字。

5.　　通过伙伴系统申请内核内存的函数有哪些?

在物理页面管理上实现了基于区的伙伴系统（zone based buddy system）。对不同区的内存使用单独的伙伴系统(buddy system)管理,而且独立地监控空闲页。相应接口 alloc_pages(gfp_mask, order)，_ _get_free_pages(gfp_mask, order)等。

伙伴系统算法

在实际应用中，经常需要分配一组连续的页框，而频繁地申请和释放不同大小的连续页框，必然导致在已分配页框的内存块中分散了许多小块的 空闲页框。这样，即使这些页框是空闲的，其他需要分配连续页框的应用也很难得到满足。

为了避免出现这种情况，Linux 内核中引入了伙伴系统算法(buddy system)。把所有的空闲页框分组为 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 个连续页框的页框块。最大可以申请 1024 个连 续页框，对应 4MB 大小的连续内存。每个页框块的第一个页框的物理地址是该块大小的整数倍。

假设要申请一个 256 个页框的块，先从 256 个页框的链表中查找空闲块，如果没有，就去 512 个 页框的链表中找，找到了则将页框块分为 2 个 256 个 页框的块，一个分配给应用，另外一个移到 256 个页框的链表中。如果 512 个页框的链表中仍没有空闲块，继续向 1024 个页 框的链表查找，如果仍然没有，则返回错误。

页框块在释放时，会主动将两个连续的页框块合并为一个较大的页框块。

6)　　通过 slab 分配器申请内核内存的函数有?

kmem_cache_create/ kmem_cache_alloc 是基于 slab 分配器的一种内存分配方式，适用于反复分配释放同一大小内存块的场合。首先用 kmem_cache_create 创建一个高速缓存区域，然后用 kmem_cache_alloc 从 该高速缓存区域中获取新的内存块。kmem_cache_alloc 一次能分配的最大内存由 mm/slab.c 文件中的 MAX_OBJ_ORDER 宏定义，在默认的 2.6.18 内核版本中，该宏定义为 5， 于是一次最多能申请 1<<5 * 4KB 也就是 128KB 的 连续物理内存。分析内核源码发现，kmem_cache_create 函数的 size 参数大于 128KB 时会调用 BUG()。测试结果验证了分析结果，用 kmem_cache_create 分 配超过 128KB 的内存时使内核崩溃。kmalloc 是内核中最常用的一种内存分配方式，它通过调用 kmem_cache_alloc 函数来实现

slab 分配器

slab 分配器源于 Solaris 2.4 的 分配算法，工作于物理内存页框分配器之上，管理特定大小对象的缓存，进行快速而高效的内存分配。

slab 分配器为每种使用的内核对象建立单独的缓冲区。Linux 内核已经采用了伙伴系统管理物理内存页框，因此 slab 分配器直接工作于伙伴系 统之上。每种缓冲区由多个 slab 组成，每个 slab 就是一组连续的物理内存页框，被划分成了固定数目的对象。根据对象大小的不同，缺省情况下一个 slab 最多可以由 1024 个页框构成。出于对齐 等其它方面的要求，slab 中分配给对象的内存可能大于用户要求的对象实际大小，这会造成一定的 内存浪费。

7）Linux 的内核空间和用户空间是如何划分的（以 32 位系统为例）？

Linux 内核将这 4G 字节的空间分为两部分。将最高的 1G 字节（从虚拟地址 0xC0000000 到 0xFFFFFFFF），供内核使用，称为“内核空 间”。而将较低的 3G 字节（从虚拟地址 0x00000000 到 0xBFFFFFFF），供各个进程使用，称为“用户空间）。因为每个进程可以通过系统调用进入内核，因此，Linux 内核由系统 内的所有进程共享。于是，从具体进程的角度来看，每个进程可以拥有 4G 字节的虚拟空间。虽然内核空间占据了每个虚拟空间中的最高 1GB 字节，但映射到物理内存却总是从最低地址（0x00000000）开始。对内核空间来说，其地址映射是很简单的线性映射，0xC0000000 就是物理地址与线性地址之间的位移量。

8） vmalloc()申请的内存有什么特点？

vmalloc()内核用于申请大块内存，特点是线性地址连续，物理地址不一定连续，不能直接用于 DMA。对应的释放函数为 vfree()。

9）用户程序使用 malloc()申请到的内存空间在什么范围？

从虚拟地址 0x00000000 到 0xBFFFFFFF 的用户空间

10） 在支持并使能 MMU 的系统中，Linux 内核和用户程序分别运行在物理地址模式还是虚拟地址模式？

Linux 内核和用户程序都用行在虚拟地址模式。

对于没有 MMU 的系统实际上用户空间和内核空间是不做区分的，如果一定要分用户空间和内核空间也只是形式上的。

uClinux 同标准 Linux 的最大区别就在于内存管理。标准 Linux 是针对有 MMU 的处理器设计的。在这种处理器上，虚拟地址被送到 MMU，MMU 把虚拟地址映射为物理地址。通过赋予每个任务不同的虚拟—物理地址转换映射，支持不同任务之间的保护。对于 uCLinux 来说，其设计针对没有 MMU 的处 理器，不能使用处理器的虚拟内存管理技术。

11) ARM 处理器是通过几级也表进行存储空间映射的？

ARM 处理器采用两级页表实现地址映射：

1）一级页表中包含以段为单位的地址变换条目或者指向二级页表的指针，一级页表实现的地址映射粒度较大。

2）二级页表中分类有大页、小页和极小页为单位的地址变换条目。

12) Linux 是通过什么组件来实现支持多种文件系通的？

虚拟文件系统（Virtual File System, 简称 VFS）， 是 Linux 内核中的一个软件层，用于给用户空间的程序提供文件系统接口；同时，它也提供了内核中的一个 抽象功能，允许不同的文件系统共存。系统中所有的文件系统不但依赖 VFS 共存，而且也依靠 VFS 协同工作。

13) Linux 虚拟文件系统的关键数据结构有哪些？（至少写出四个）Linux 虚拟文件系统的关键数据结构有哪些？（至少写出四个）

超级块对象存储一个已安装的文件系统的控制信息，代表一个已安装的文件系统；

```
struct super_block;
```

索引节点对象存储了文件的相关信息，代表了存储设备上的一个实际的物理文件。。当一个 文件首次被访问时，内核会在内存中组装相应的索引节点对象，以便向内核提供对一个文件进行

操 作时所必需的全部信息；

struct inode;

目录项的概念主要是出于方便查找文件的目的。一个路径的各个组成部分，不管是目录还是 普通的文件，都是一个目录项对象。如，在路径/home/source/test.c 中，目录 /, home, source 和文件 test.c 都对应一个目录项对象。不同于前面的两个对象，目录项对象没有对应的磁盘数据结构，VFS 在遍 历路径名的过程中现场将它们逐个地解析成目录项对象。目录项的概念主要是出于方便查找文件的目的。一个路径的各个组成部分，不管是目录还是 普通的文件，都是一个目录项对象。

文件对象是已打开的文件在内存中的表示，主要用于建立进程和磁盘上的文件的对应关系。它由 sys_open() 现场创建，由 sys_close()销毁。

struct file

14) 对文件或设备的操作函数保存在那个数据结构中？

struct file_operations;

15) Linux 中的文件包括哪些？

有执行文件，普通文件，目录文件，链接文件和设备文件，管道文件。

16)　创建进程的系统调用有那些?

clone(), fork(), vfork()

fork（）创造的子进程复制了父亲进程的资源，使用写时复制技术，子进程与父进程使用相同物理页，只有子进程试图写一个物理页时，才 copy 这个物理页到一个新的物理页，子进程使用新的物理页，子进程与物理地址的内存地址空间分开，开始独立运行。

vfork()创建的子进程完全运行在父进程的地址空间上，子进程对虚拟地址空间任何数据的修改都为父进程所见。这与 fork 是完全不同的，fork 进程是独立的空间。为了防止父进程重写子进程需要的数据，父进程会被阻塞，直到子进程执行 exec（）和 exit（）。

clone()可以有选择性的继承父进程的资源，可以选择像 vfork 一样和父进程共享一个虚存空间，从而使创造的是线程，你也可以不和父进程共享，你甚至可以选择创造出来的进程和父进程不再是父子关系，而是兄弟关系

系统调用服务例程 sys_clone, sys_fork, sys_vfork 三者最终都是调用 do_fork 函数完成.

17) 调用 schedule()进行进程切换的方式有几种?

一 系统调用 do_fork()

二 定时钟断 do_timer()

三 唤醒进程 wake_up_process()

四 改变进程的调度策略 setscheduler()

五 系统调用礼让 sys_sched_yield()

参考：

一 系统调用 do_fork()

1 当前进程调用 fork（）创建子进程，进入 kernel

2 当前进程分一半多时间片给子进程，

3 如果当前进程时间片剩余为 0，设定当前进程 need_sched=1,

4 从系统调用退出

5 到达 ret_from_sys_call

6 到达 ret_with_reschedule

7 发现当前进程要求调度，跳转到 reschedule

8 调用 schedule()

9 schedule()处理当前进程的调度要求，

10 如果有其他进程可运行，将在 schedule()内发生切换。

二 定时钟断 do_timer()

11 当定时钟断发生时 8235->irq0->do_timer_interrupt()->do_timer()

12 ->update_process_times()递减当前进程的时间片，

13 如果当前进程时间片为 0，设定当前进程 need_sched=1,

14 从中断调用退出，

15 到达 ret_from_intr

16 到达 ret_with_reschedule ，

17 发现当前进程要求调度，跳转到 reschedule

18 调用 schedule()

19 schedule()处理当前进程的调度要求，

20 如果有其他进程可运行，将在 schedule()内发生切换。

三 唤醒进程 wake_up_process()

21 当前进程调用 fork（）创建子进程，进入 kernel

22 当前进程调用了 wake_up_process 来唤醒进程 x

23 使进程 x 状态为 RUNNING,并加入 runqueue 队列，

24 调用 reschedule_idle（）

25 发现进程 x 比当前进程更有资格运行，设定当前进程 need_sched=1,

26 从系统调用退出

27 到达 ret_from_sys_call

28 到达 ret_with_reschedule

29 发现当前进程要求调度，跳转到 reschedule

30 调用 schedule()

31 schedule()处理当前进程的调度要求，

32 如果有其他进程可运行，将在 schedule()内发生切换。这次大多数可能切换到进程 x

四 改变进程的调度策略 setscheduler()

33 进入系统调用 setscheduler（）

34 改变进程 x 的调度策略

35 提前进程 x 在 runqueue 队列的位置

36 设定当前进程 need_sched=1,

37 从系统调用退出

38 到达 ret_from_sys_call

39 到达 ret_with_reschedule

40 发现当前进程要求调度，跳转到 reschedule

41 调用 schedule()

42 schedule()处理当前进程的调度要求，

43 如果有其他进程可运行，将在 schedule()内发生切换。

五 系统调用礼让 sys_sched_yield()

44 进入系统调用 sys_sched_yield（）

45 如果有其他的进程，进行礼让，

46 设定当前进程 need_sched=1,

47 从系统调用退出

48 到达 ret_from_sys_call

49 到达 ret_with_reschedule

50 发现当前进程要求调度，跳转到 reschedule

51 调用 schedule()

52 schedule()处理当前进程的调度要求，

53 如果有其他进程可运行，将在 schedule()内发生切换。

need_sched 表示 CPU 从系统空间返回到用户空间前夕要进行一次调度。

18) Linux 调度程序是根据进程的动态优先级还是静态优先级来调度进程的？

Linux 调度程序是根据进程的动态优先级来调度进程的,但动态优先级又是依据静态优先级通过算法得到的，两者是两个相关连的值。

因为高优先级的进程总比低优先级的进程先被调度,为防止有多个高优先级且一直占用 CPU 资源,导致其它进程不能占用 CPU,所以引用动态优先级概念.

19) 进程调度的核心数据结构是哪个？

struct runqueue;

20) 如何加载、卸载一个模块？

通过命令 Insmod 加载一个模块， 通过命令 rmmod 卸载一个模块。

21) 模块和应用程序分别运行在什么空间？

模块运行在内核空间，应用程序运行在用户空间。

22) Linux 中的浮点运算由应用程序实现还是内核实现？

由应用程序实现，Linux 中的浮点运算由数学库函数实现，库函数能被应用程序链接后调用，不能被内核链接调用。

这些运算是在应用程序中进行的，然后再将运算结果反馈给系统。 linux 内核如果一定要进行浮点运算，需要在建立内核时选上 math－emu，使用软件模拟浮点运算。据说这样作的代价有两个：

1.用户在安装驱动时需要重建内核；

2.可能会影响到其他应用程序，使得这些应用程序在进行浮点运算时也使用 math－emu，会严重将低效率。

23) 模块程序能否使用可链接的库函数？

模块程序运行在内核空间，不能链接库函数。

24) TLB 中缓存的是什么内容？

TLB(Translation lookaside buffer)即旁路转换缓冲，称为页表缓冲,当线性地址第一次被转换为物理地址时，将线性地址与物理地址存放到 TLB 中，用于下次访问这个线性地址时，加快转换速度。

25) Linux 中有哪几种设备？

可以分为字符设备和块设备。网卡是例外，它不直接与设备文件对应。mknod()系统调用用来创建设备文件。

26) 字符设备驱动程序的关键数据结构是哪个？

字符设备描述符 struct cdev; cdev_alloc()用于动态分配 cdev 描述符。cdev_add()注册一个 cdev 描述符。Cdev 包含一个 struct kobject 类型的数据结构，它是设备驱动程序的核心数据结构。

27) 设备驱动程序包括哪些功能函数？

open(), read(),wirte(),ioctl(), release(),llseek()

28) 如何唯一标识一个设备？

linux 使用设备编号来唯一标识一个设备，设备编号分为两部分：主设备号和次设备号。一般主设备号标识设备对应的驱动程序，次设备号用于确定设备文件指向的设备。在内核中使用数据结构 dev_t 表示设备编号，一般他是 32 位长度，其中 12 位用于表示主设备号，后 20 位用于表示次设备号。函数 MKDEV(int major, int minor)用于生成一个 dev_t 类型的对象。

29) Linux 通过什么方式实现系统调用？

靠软件中断实现的。首先，用户程序为系统调用设置参数。其中一个参数是系统调用编号。参数设置完成后，程序执行“系统调用”指令。x86 系统上的软中断由 int 产生。这个指令会导致一个异常：产生一个事件，这个事件会致使处理器切换到内核态并跳转到一个新的地址，并开始执行那里的异常处理程序。此时的异常处理程序实际上就是系统调用处理程序。

30) Linux 软中断和工作队列的作用是什么？

Linux 软中断和工作队列的作用是中断处理.

1 软中断一般是“可延迟函数”的总称,它不能睡眠、不能阻塞。它处于中断上下文，不能进程切换， 软中断不能被自己打断，只能被硬件中断打断（上半部），可以并发运行在多个 CPU 上（即使同一类型的也可以）。所以软中断必须设计为可重入的函数（允许多个 CPU 同时操作），因此也需要使用自旋锁来保护其数据结构。

2 工作队列中的函数处于进程上下文中，它可以睡眠，能被阻塞。能够在不同的进程间切换，以完成不同的工作。

可延迟函数和工作队列中的函数都不能访问进程的用户空间。可延迟函数执行时不可能有任何正在运行的进程。工作队列的函数由内核进程执行，它不能访问用户空间地址。