Linux 进程管理
一个 task 就是一个 process。Ps 命令-得到进程列表。pstree
TASK_RUNNING：正在运行的进程即系统的当前进程或准备运行的进程即在 Running 队列中的进程。只有处于该状态的进程才实际参与进程调度
TASK_INTERRUPTIBLE：处于等待资源状态中的进程，当等待的 源有效时被唤醒，也可以被其他进程或内核用信号、中断唤醒后进入就绪状态。
TASK_UNINTERRUPTIBLE：处于等待资源状态中的进程，当等待的资源有效时被唤醒，不可以被其它进程或内核通过信号、中断唤醒。
TASK_STOPPED：进程被暂停，一般当进程收到下列信号之一时进入这个状态：SIGSTOP, SIGTSTP, SIGTTIN 或者 SIGTTOU。通过其它进程的信号才能唤醒。
TASK_TRACED：进程被跟踪，一般在调试的时候用到。
EXIT_ZOMBIE：正在终止的进程，等待父进程调用 wait4()或者 waitpid()回收信息。是进程结束运行前的一个过度状态（僵死状态）。 虽然此时已经释放了内存、文件等资源，但是在内核中仍然保留一些这个进程的数据结构（比如 task_struct）等待父进程回收。
EXIT_DEAD：进程消亡前的最后一个状态，表示父进程已经获得了该进程的记账信息，该进程可以被销毁了
进程的 task_struct 是进程存在的唯一标志。
ulimit –u 显示用户可以同时执行的最大进程个数
int pid //进程标识号
struct mm_struct *mm //进程的虚存信息
struct fs_struct *fs //可执行映象所在的文件系统
struct files_struct *files //进程打开的文件
用户标识符 UID 和组标识符 GID
进程控制块和进程内核栈的空间分配在一起(2.4 版内核)，内核为它们分配两个连续的物理页面
在 2.6 后内核 x86 机器上为 thread_info 结构
thread_info: struct task_struct *task;
Current->pid 正在执行的进程的标识符
Init_task：PID=0，为 swapper 空闲进程
进程创建：系统创建的第 1 个真正进程是 init 进程，在启动时运行程序和脚本。pid=1。系统中所有的进程都是由进程使用系统调用 fork()创建的。子进程在创建后执行的是父进程的程序代码。子进程通过调用 exec 系列函数执行真正的任务。
Fork 函数：把父进程 task_struct 拷贝到子进程的 sys_clone(),sys_vfork(),sys_fork()创建子进程，这三个都会调用 do_fork()函数完成主要工作
sys_clone()对应 clone_flags 可能是多个标志位的组合 sys_fork()对应的是 SIGCHILD sys_vfork()对应的是 CLONE_VFORK|CLONE_VM|SIGCHIL
使程序执行的唯一方法是使用系统调用 exec()。系统调用 exec()有多种使用形式，称为 exec()族，
Exec 返回值：该系统调用将引起另一个程序的执行，成功调用后并不需要返回，失败-1
等待队列的两个函数：inline&remove
内核函数 sys_wait4()的功能是使进程进入等待态。当前进程在运行过程中要等待它的子进程终止时，调用该函数使其状态从运行态转换成可中断的等待态(INTERUPTIBLE)，并加入到等待队列中。
state_addr-状态变量的地址；option-控制选项；ru-传递使用资源结构体
唤醒：wake_up&wake_up_interruptible 可中断的
进程终止：do_exit
把线程和进程一视同仁，每个线程拥有唯一的 task_struct 结构。
内核线程：通过系统调用 clone()来实现
由于调用进程是在内核中调用 kernel_thread(),因此当系统调用返回时,子进程也处于内核态中,而子进程随后调用 fn, 当 fn 退出时,子进程调用 exit()退出,所以子进程是在内核态运行的.
例-磁盘缓存刷新，网络连接维护、页面换入换出
swapper（或 idel）进程(init_task)，0 号进程
init(pid=1)进程既是内核线程也是 1 号用户进程
守护进程：kflushd (即 bdflush)：刷新“脏”缓冲区中的内容到磁盘以归还内存 kupdate：刷新旧的“脏”缓冲区中的内容到磁盘以减少文件系统不一致的风险。kpiod：把属于共享内存映射的页面交换出去。 kswapd：执行内存回收功能。
用命令 ps aux 参看所有进程，包括守护进程。
进程调度策略-SEHED_NORMAL 普通进程的时间片轮转 SCHED_FIFO 实时进程先进先出 SCHED_RR 实时进程 RR SCHED_BATCH 后台处理进程

进程调度 Linux 2.6.23-CFS 算法-红黑树
Linux 2.6 的进程 140 个优先级。实时进程 0-99，普通进程 100-139,0 最高
调度程序根据动态优先级选择新进程运行，静态优先级决定基本时间片。

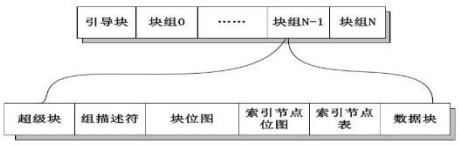
base time quantum （ms）：
- （140-static_priority）*20 if static_priority <120
- （140-static_priority）*5 if static_priority >=120

Sleep_avg，进程平均等待时间 ns，run_list 串连在优先级队列中，time_slice 进程时间片剩余大小。子与父平分
static_prio 静态优先级，和 2.4 的 nice 相同，实时进程的 static_prio 不参与优先级 prio 的计算
prio 动态优先级，bonus 0~10
unsigned long rt_priority 实时进程的优先级
prio_array_t *array 记录当前 CPU 活动就绪队列
runqueue 结构 系统中每个 CPU 都的运行队列，所有 runqueue 结构存放在 runqueues 每 CPU 变量中。
prio_array_t *active, *expired, arrays[2] 活动队列，过期队列
每一类就绪进程都用一个 struct prio_array 表示
调度程序函数：scheduler_tick 维持当前最新的 time_slice 计数器，try_to_wake_up 唤醒睡眠进程
recalc_task_prio，更新进程的动态优先权
schedule，选择要被执行的新进程 load_balance，维持多处理器系统中运行队列的平衡
schedule->policy 进程的调度策略，rt_priority 实时进程优先级。Nice 普通进程的优先级，取值[-20,19], 默认 0;nice 为-20 的进程获得时间片最长。2.6 为 static_prio，静态优先级，不参与优先级计算。counter 进程 CPU 运行时间的计数值
普通进程的权值就是它的 counter 的值，而实时进程的权值 是它的 rt_priority 的值加 1000（2.4）
goodness()函数计算进程的当前权值。该函数的第一个参数是待估进程的控制块。实时进程：1000+rt_priority；普通进程，counter

文件系统：
文件控制信息单独 组成一个称为 inode（索引节点，一个数据结构）。每个文件对应一个 inode，它们有唯一的编号，称为 inode 号（索引节点号）
目录项主要由文件名和 inode 号组成。

```
#define EXT2_NAME_LEN 255
/* The new version of the directory entry.  Since EXT2 structures are  stored in intel byte order, and the
        could never be  bigger than 255 chars, it's safe to reclaim the extra byte for the  file_type field. */
struct ext2_dir_entry_2 {
    __u32          inode;                   /* Inode number */
    __u16          rec_len;                 /* Directory entry length */
    __u8           name_len;                /* Name length */
    __u8           file_type;
    char           name[EXT2_NAME_LEN];     /* File name */
};
```

文件类型--字符设备文件(charcater device)和块设备文件(block device)。管道(FIFO)文件：用于在进程间通信。链接文件，提供了共享文件的方法。
socket 文件
用 ls -l 或 ls -ld 命令显示文件的访问权限：
VFS 结构：超级块对象 superblock:存储已安装文件系统的信息，通常对应磁盘文件系统的文件系统超级块或控制块。
索引节点对象 inode object ：存储某个文件的信息。通常对应磁盘文件系统的文件控制块
目录项对象 dentry object：dentry 对象主要是描述一个目录项，是路径的组成部分
文件对象 file object：存储一个打开文件和一个进程的信息。只要文件一直打开，就一直存在于内存。
VFS 超级块是各种具体文件系统在安装时建立的，并在卸载时被自动删除。super_block 结构内容：
s_type：指向文件系统的类型的指针。
s_op： 指向 super_operations 的指针
s_root：指向目录的 dentry 项。
super_operations 内容：put_super()：由于当前的文件系统的卸载而释放当前的超级块对象。statfs()：返回当前 mount 的文件系统的一些统计信息。remount_fs()：按照一定的选项重新 mount 文件系统。umount_begin()：开始 umount 操作，并中断其它的 mount 操作，用于网络文件系统。
inode：物理文件系统的 inode 在外存中并且是长期存在的，VFS 的 inode 对象在内存中。inode 结构定义在文件<include/linux/fs.h>中
struct inode_operations *i_op; 指向 inode 操作函数入口表的指针
VFS 的 inode 组成一个双向链表，全局变量 first_inode 指向链表的表头。
管理 inode 的全局变量：nr_inodes 表示当前使用的 inode 数量 nr_free_inodes 表示空闲的 inode 数量。
目录项对象 dentry object：每个文件还有一个 dentry 结构。每个 dentry 代表路径中的一个特定部分，目录项也可包括安装点。 struct dentry_operations *d_op;/* 操作目录项的函数*/
VFS 文件系统维护着一个目录项的缓存。
File 对象：该对象由 open()创建，由 close()销毁。
定义在<include/linux/fs.h>
struct file_operations *f_op; 文件对象操作集合
Linux 支持的文件系统必须注册后才能使用，文件系统不再使用时则予以注销。两种方式：
[1]系统引导时在 VFS 中注册（/etc/fstab），在系统关闭时注销。[2]把文件系统做为可装卸模块，在安装时在 VFS 中注册，并在模块卸载时注销。
每种文件系统登记在 file_system_type 结构体中，file_system_type 结构体组成一个链表，称为注册链表， 链表的表头由全局变量 file_system 给出。
file_system_type 结构-> fs_flags：mount 的文件系统的参数；get_sb：当这种类型的文件系统要被 mount 的时候用以得到相应文件系统的超级块；kill_sb：当这种类型的文件系统被 umount 的时候。
list_head fs_supers：文件系统的超级块的双向链表。owner：THIS_MODULE。
mount [-t fstype] [-o options] device dirname
device：设备文件，格式：/dev/xxyN dirname：挂载目录 options- loop：把一个文件当成块设备挂接（环回设备）
用 vfsmount 结构 <include/linux/mount.h>描述
系统打开文件表：双向表，结点是 file 结构，全局变量 first_file 指向系统打开文件表的表头。
对于一个进程打开的所有文件，由进程的两个私有结构进行管理。fs_struct 结构记录着进程所在文件系统根目录和当前目录，files_struct 结构包含着进程的打开文件表。
files_struct: fd(_array)[]每个元素是一个指向 file 结构体的指针，该数组称为进程打开文件表。
进程打开一个文件时，建立一个 file 结构体，并加入到系统打开文件表中，然后把该 file 结构体的首地址写入 fd[]数 组的第一个空闲元素中，一个进程所有打开的文件都记载在 fd[]数组中
fd[]数组的下标称为文件描述符。0 标准输入设备，1 标准输出设备，2 标准错误输出设备。
open()系统调用内核函数是 sys_open( )。
sys_open()->1. Invokes getname( ) to read the file pathname from the process address space.2.用 get_unused_fd( )在 current->files->fd 所指向的文件对象指针数组中查找一个未使用的文件描述符（号），存储在局部变量 fd 中。3. 调用 filp_open（）函数。Open_namei 得到 dentry，path_walk 路径解析。4. 调用 fd_install （）函数，将文件对象装入当前进程的打开文件表：
Read()内核为 sys_read。读操作若页面不在 pagecache ,i_mapping->a_ops>readpage( )读入
Ext2：符号连接”（symbolic links）的方式，使得连接文件只需要存放 inode 的空间。ext2 分区的第一个磁盘块用于引导， include\linux\ext2.fs



每个块组中包含的超级块内容是相同的。
__u16 s_magic; /* 文件系统标识 0x38-0x39 */
所有 inode 的大小相同，即 128 字节
__u32 i_block[EXT2_N_BLOCKS];/*磁盘块指针 */
dd /dev/zero：零设备“0” /dev/loop：loopback device 指是用文件来模拟块设备
if = 输入文件（或设备名称）
of = 输出文件（或设备名称）
bs = bytes 同时设置读/写缓冲区的字节数（等于设置 ibs 和 obs
count=blocks 只拷贝输入的 blocks 块
conv = ucase/lcase 把字母由小写转换为大(小)写

dd if=/dev/zero of=myfs bs=1M count=1
cp –R ext2 myext2　make clean 删除.o
查看一下 myext2 文件系统是否加载成功:
cat /proc/filesystem |grep myext2
dmesg|tail 查看内核输出

内存管理:
虚拟内存－共 4G 字节，内核空间（高 1G） 用户空间（低 3G），每个进程最大拥有 3G 字节私有虚存空间。内核空间和用户空间大小的划分由 PAGE_OFFSET 决定，在 src/include/asmi386/page.h
内核空间由所有进程共享，其中存放的是内核代码和数据， 即"内核映象"。 "内核映象"不可回收和换出。进程的用户空间中存放的是用户程序的代码和数据。内核空间映射到物理内存总是从最低地址（0x00000000）开始，当系统启动时，Linux 内核映像被装入在物理地址 0x00100000 开始处，即 1MB 开始的区间(第 1M 留作它用)。
进程用户空间： 一个进程的用户地址空间主要由 **mm_struct** 结构和 vm_area_structs 结构来描述。
mm_struct 对进程整个用户空间进行描述
vm_area_structs 对用户空间中各个区间进行描述，mm_struct 首地址在 task_struct 成员 mm 中
vm_area_struct 结构是虚存空间中一个连续区域，在这个区域中的信息具有相同的操作和访问特性。
VMA：vm_mm 指针指向进程的 mm_struct 结构体。vm_start 和 vm_end 虚拟区域的开始和终止地址。vm_flags 指出了虚存区域的操作特性：vm_page_prot 虚存区域的页面的保护特性。
所有 vm_area_struct 结构体按地址递增链接成一个单向链表，vm_next 指向下一个 vm_area_struct 结构体。链表首地址由 mm_struct 中成员项 mmap 指出。
vm_ops：是指向 vm_operations_struct 结构体的指针。该结构体中包含着指向各种操作的函数的指针，在 include/linux/mm.h 文件中定义。
vm_next_share 和 vm_prev_share ， 把有关 vm_area_struct 合成一个共享内存时使用双向链表。所有的 vm_area_struct 虚存段都作为树的一个节点。节点中 vm_rb 的左指针 rb_left 指向相邻的低地址虚存段，右指针 rb_right 指向相邻的高地址虚存段。
Linux 使用 **do_mmap()**函数完成可执行映像向虚存段的映射，由它建立有关的虚存段。unsigned long do_mmap(struct file * file, unsigned long addr, unsigned long len,unsigned long prot, unsigned long flags, unsigned long off)
创建进程用户空间：内核调用 copy_mm()函数，为新进程建立所有页表和 mm_struct 结构
分页内存管理机制:→页目录 PGD（Page Directory）页中间目录 PMD（Page Middle Directory）页表 PTE（Page Table）
2.6.11 以后四级页表 Page Global Directory(pgd_t),PageUpperDirectory(pud_t),Page Middle Directory(pmd_t) 和 Page Table(pte_t)。
对 IA32，三级分页管理转换成两级分页机制。其中一个重要的方面就是把 PGD 与 PMD 合二为一
MMU 地址映射：CR3 指向该页目录的基址
页面异常的处理程序是 **do_page_fault()**函数，首先得到导致异常发生的线性地址（虚拟地址），对于 IA32 该地址放在 CR2 寄存器中。
页帧交换守护进程 kswapd，它有自己的进程控制块 task_struct 结构，与其它进程一样受内核调度，但没有独立的地址空间
物理内存的管理:，PAGE_SIZE 宏定义。IA32 4KB
结点：访问速度相同的一个内存区域称为一个结点（Node），每个结点关联到一个处理器。
区：每个结点的物理内存因为用途不同又分成不同的区(zone)。例如 x86， 分成如下三个区:
DMA ZONE 低于 16MB 的内存，是 DMA 方式能够访问的物理内存。尽可能这部分内存供 DMA。
NORMAL ZONE 16MB~896MB，直接被内核映射。
HIGHMEM ZONE 高端内存不能被内核直接射。
Linux 设置了一个 mem_map[]数组管理内存页帧，元素是一个个的 page 结构体->struct list_head lru; //Contains pointers to the least recently used doubly linked list of pages.
两组连续页帧被认为是一对"**伙伴**"必须满足如下条件:大小相同,;物理地址连续;第 2 个页帧块(组)的后面页帧号必须是 2×n 的倍数。
Linux 把物理内存划分成了 1、2、4、8、…、1024 共十一种页块
Linux 把空闲的页帧按照页块大小分组进行管理，数组 free_area[]来管理各个空闲页块组 static struct free_area_struct free_area[NR_MEM_LISTS];
函数 alloc_pages( )用于分配物理页帧
函数 free_pages()用于页块的回收
slab 分配器的基本思想：为经常使用的小对象建立缓冲，小对象的申请与释放都通过 slab 分配器来管理。slab 分配 器再与伙伴系统打交道。
Slab 相关操作：(1)**kmem_cache_create**( ) 该函数创建一种特定对象 kmem_cache_t 结构，加入 cache_cache 所管理的队列(2)**kmem_cache_alloc( ) 与 kmem_cache_free( )** 当需要分配一个拥有专用 slab 队列的对象时,应该通过 kmem_cache_alloc( )函数，相反的动作则是 kmem_cache_free( )函数 (3)**kmem_cache_grow()** 与 **kmem_cache_reap()**kmem_cache_create()函数只是建立了所需的专用缓冲区队列的 基础设施,所形成的 slab 队列是个空队列。而具体 slab 的创建则要等需要分配缓冲区时,却发现队列中并无空闲的缓冲区可供分配时，再通过 kmem_cache_grow()来进行，kmem_cache_grow()再向伙伴系统申请空间。kswapd 会定时调用 kmem_cache_reap()来"收割" 缓冲区队列。(4)**kmalloc( )**函数与 **kfree( )**函数，从通用的缓冲区队列中申请和释放空间。

系统调用:
运行模式：Linux 使用了其中的两个：特权级 0 和特权级 3 ，即内核模式(kernel mode) 和用户模式(user mode)
上下文（context）。一个进程的上下文可以分为三个部分：用户级上下文、寄存器上下文以及系统级上下文。用户级上下文：正文（代码）、数据、用户栈以及共享存储区；寄存器上下文：通用寄存器、程序寄存器（eip）、处理机状态寄存器 （eflags）、栈指针 （esp）；系统级上下文：进程控制块 task_struct 、内存管理信息(mm_struct 、vm_area_struct、 pgd、pmd、pte 等)、核心栈等。
通过 strace 命令可以查看操作系统命令所调用的系统调用→ strace ls strace -o log.txt hostname
system_call()函数实现了系统调用中断处理程序 ：根据 eax 中所包含的系统调用号调用对应的特定服务例程

内核:
内核映像：Linux 系统引导过程使用内核映像，/boot 目录下文件名称 如：vmlinuz-2.6.15.5；
普通内核映像：zImage（Image compressed with gzip），大小不能超过 512k 大内核映像：bzImage（big Image compressed with gzip ），包含了大部分系统核心组件：系统初始化、进程调度、内核管理模块。
内核构建步骤：一、ELF 格式的 vmlinux 二、裸二进制格式的 vmlinux.bin 三、setup.bin 四、kbuild 调用内核自带的程序 build，将 vmlinux.bin 和 setup.bin 合并为 bzImage。
用 uname –r 命令查看当前的内核版本号，
make menuconfig // 同时生成.config 文件
编译内核 make 编译模块 make modules
安装模块 sudo make modules_install
安装内核 sudo make install
查看启动选项 gedit /boot/grub/grub.cfg

内核模块:
int init_module(void) void cleanup_module(void)
使用 lsmod 命令查看模块
显示内核符号和模块符号表的信息，可以读取 /proc/kallsyms 文件。 cat /proc/kallsyms
modprobe 是自动根据模块之间的依赖性插入模块的程序

Linux 概述
系统结构：bin：该目录存放最常用的基本命令；boot:该目录包含了系统启动需要的配置文件、内核（vmliuxz）和系统镜像(initrd…img)等；dev：外部设备文件；etc：该目录下包含了所有系统服务和系统管理使用的配置文件；比如系统日志服务的配置文件 syslog.conf，系统用户密码文件 passwd 等；host+found:该目录包含了磁盘扫描检测到的文件碎片；proc：该目录属于内存影射的一个虚拟目录，不在磁盘中存储；sbin:该目录下包含系统管理员使用的系统管理命令,比如防火墙设置命令 iptable，系统停机命令 halt 等；usr：该目录下一般来说包含系统发布时自带的程序，三个子目录 1./usr/src：Linux 内核代码 2./usr/man：帮助文件 3./usr/local：新安装的应用；var：该目录中存放着在不断扩充着的信息，比如日志文件。
.a .so .sa —库文件
gcc -o mypro1 mypro1.c 生成 mypro1
缺省头文件目录是/usr/include 目录及其子目录下。
调用 C 语言编译器时，我们可以使用-I 标志来包含保存在子目录或非标准位置 中的 include 文件，例如：gcc –I /usr/openwin/include power.c
数学库 libm.a 链接到 power.o ：
gcc –o power power.o -lm
gcc –o power power.o /usr/lib/libm.a
Linux 的汇编语言，采用的是 AT&T386 汇编语言
pid_t fork(void); 返回值：子进程中为 0，父进程中为子进程号 PID，出错为-1
当进程调用一种 exec 函数时，该进程完全由新程序代换，而新程序则从其 main 函数开始执行。调用 exec 并不创建新进程，所以前后的进程 PID 并未改变。exec 只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。
管道通信：管道是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称为 pipe 文件。管道通信是基于文件系统形式的一种通信方式。
线程：pthread_t 线程 ID；pthread_mutex_t 互斥锁对象；pthread_cond_t 条件变量。pthread_join():用于将当前线程挂起来等待线程的结束。
线程创建：int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void *(*start_routine)(void*), void * arg);
线程终止：void pthread_exit(void *value_ptr);
等待线程结束 int pthread_join(pthread_t thread, void **value_ptr);
pthread_cond_wait(&cond, &mutex);
sem_wait()和 sem_trywait()都相当于 P 操作，，两者的区别在于若信号量小于零时，sem_wait()将会阻塞进程。而 sem_trywait()则会立即返回。sem_post()相当于 V 操作，它将信号量的值加一同时发出信号来唤醒等待的进程。
超级用户的用户 ID 是 0

文件用户分为 user\group\others；文件类型和访问权限 链接计数 所有者 所在组 文件大小 日期 时间 文件名
Crond 守护进程定期检查是否有要执行的作业

---

cal 显示日历、**uptime** 显示系统运行时间 **date** 显示日期和时间、**su** 改变用户身份、**passwd** 修改密码、**pwd** 显示当前工作目录、**mkdir** 创建目录、**rmdir** 删除目录、**touch** 建立文件、**more** 分页显示文本文件内容、**less** 分页显示文件文本内容、**head** -c N 显示文件的前 N 个字节内容 -N 显示开始的 N 行 -5 file.txt 显示文件的前面 5 行、**tail** 默认显示 10 行，+2 foo 显示第 2 行开始的所有行-2 foo 显示文件最后 2 行、**cp** [options] source desti 复制文件、**mv** [options] file1 file2 转移文件 file1 到 file2 ，mv a.txt b.c 将文件 a 重命名为 b，mv ~dir1/* . 将..移动到当前目录、**rm** 删除文件、**wc** 统计文件大小（-c 字节数，-m 字符数，-l 行数，-L 最长行长度，-w 单词数）、删除重复行 **uniq**、**diff** 比较文件（-a 当做文本文件一行行比较-b 忽略空格符-B 忽略空白行-c 列出所有行，标出不同，-i 忽略大小写-q 不显示文件不同的行，只显示差异性，-s 当两个文件相同时提出报告）、**id**[opt][username]查看用户和组 id（-g 显示用户所属组 ID，-G 显示用户所有群组 ID，-n 显示用户名，-r 显示实际 ID，-u 显示用户 ID）、**chmod** 改变文件存取权限，chmod u=rwx file 将可读、可写可执行的权限给 file 所有者、**chown** 改变文件或目录的所有者，chown -c root acmd 将 acmd 文件的所有者改成 root、**umask** 设置缺省文件权限，默认值 022、**ln** 建立文件链接（-f 强制建立-i 询问是否覆盖-n 若存在则不-s 符号链接）[硬链接有相同索引号，符号链接不同，且可跨文件系统]，ln file1 file1.hard 建立 file1.hard 链接文件、**grep** 搜索指定文件中的匹配行（-c 输出数量-i 忽略大小写-l 仅输出有匹配行的文件名-n 同时输出行号-v 打印不匹配的行）、**find** 查找符合表达式的目录列表、**gzip** 压缩文件.gz、**tar**[optinos][filelist]打包文件、**cat** 显示文本文件内容(-b 在每一个非空白行开头编号-n 每一行都编号-s 合并连续空白行为一行)、**fg** 把后台的进程移到前台执行（fg[%jobID] 当前作业、%-前一个、%N 作业号为 N、%Name 开

头名字为..%?Name 名字中有..)、**crontab** 创建、编辑或删除 crontab 文件、**at** 向 crond 守护进程提交作业、cat < create.c > stdout.c：cat 的输入来自 create.c，再将数据传输到 stdout.c、cat ch1 ch2 ch3 1>ch.out 2>ch.error 输入来自 ch123、**pip** 连接进程

L-1 Introduction

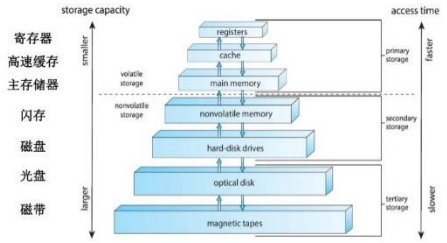
Computer system can be divided into four components: hardware, operating system, application programs, users

A trap（陷入） is a software-generated interrupt caused either by an error or a user request.

An operating system is interrupt driven.

中断的分类：1.外部中断(interrupt)，异步中断->外部设备所发出的 I/O 请求、分为可屏蔽的和不可屏蔽的两类，由一些硬件设备产生,可以在指令执行的任意时刻产生。2. 异常(exception)，内部中断，同步中断：由 CPU 产生，一条指令终止执行后 CPU 才会发出中断。常见的异常有除零、溢出及页面异常(fault 出错)等。另一种情况是使用 int 指令(trap 陷入)，Linux 使用该指令来实现系统调用。

Storage structure： main memory, secondary storage, hard disk drives, non-volatile memory（非易失存储器）faster than hard disks, nonvolatile



IO：Synchronous I/O（同步 I/O ）:After I/O starts, control returns to user program only upon I/O completion. Asynchronous I/O （异步 I/O ） : After I/O starts, control returns to user program without waiting for I/O completion.

多处理器系统：1. Asymmetric Multiprocessing 主处理器调度从处理器 2. Symmetric Multiprocessing 每个处理器都要完成所有任务

Clustered Systems 是由一组互联的主机（节点）构成统一的计算机资源，通过相应软件协调工作的计算机机群，给人以一台机器的感觉。(SAN)

Asymmetric clustering（非对称集群） has one machine in hot-standby mode. Symmetric clustering （对称集群） has multiple nodes running applications, monitoring each other

Software error or request creates exception（异常）or trap（陷入）陷阱或异常是一种软件中断，源于出错或源于用户程序的一个特别请求

特权指令: 不允许用户程序中直接使用的指令。转换到用户模式、IO 控制、定时器管理、中断 set value of timer, clear memory, trun off interrupts, modify entries in device-status table, access io dev

Multiprogramming（多道程序）：Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.

Time-Sharing Systems 分时系统，多个用户分时共享使用同一台计算机，CTSS（第一个） Linux

相应时间：S=n*q（每个进程的运行时间片为 q）

L-2 Operating-System Structures

系统调用的实现是在内核完成的，而用户态的函数是在函数库中实现的

向操作系统传参的三种方法：1. in registers2. address of block in a register3. Parameters placed, or pushed, onto the stack

系统调用类型：进程控制、文件管理、设备管理、信息维护、通信

Policy 策略： What will be done?

Mechanism 机制： How to do it?

操作系统结构：

Simple Structure： Not divided into modules

MS-DOS Layered Approach：效率低

Monolithic Structure（单/宏内核结构）：the entire code of the kernel — including all its subsystems such as memory management, filesystems, or device drivers — is packed into a single file. Each function has access to all other parts of the kernel;

UNIX, OS/360, VMS and Linux 优势：高效；缺点：难以隔离错误来源和其他错误。难以修改和维护。随着操作系统的发展，内核变得越来越大。

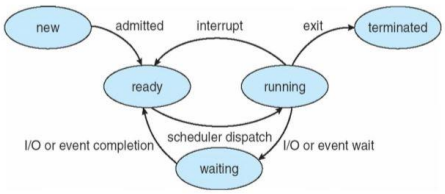
Microkernel System Structure （微内核结构）：将所有非基本部分从内核移走，实现为系统程序或用户程序。便于扩展，安全性、可靠性。Windows NT Windows 8、Windows 10、Mac OS、L4、Mach

模块：Solaris, Linux, 混合：Mac OS X

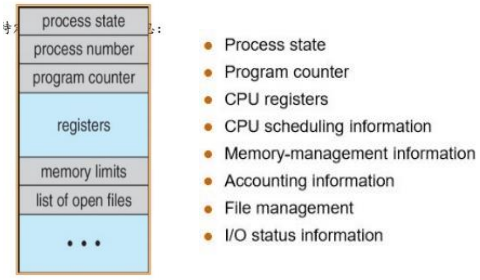
L-3 Process

jobs=user programs = tasks = process 进程

A process includes: The program code, also called text section Program counter (PC) Registers Data section (global data) Stack (temporary data) Heap (dynamically allocated memory)



Windows NT/2000 线程有 7 种状态；Linux 进程有 6 种状态；Windows 2003 server 有 9 种状态.



Job queue – set of all processes in the system.

Ready queue – set of all processes residing in main memory, ready and waiting to execute. Device queues – set of processes waiting for an I/O device 长程调度（作业调度）从缓冲池中选择进程装入内存。短期调度（CPU）从准备执行的进程选择

Most operating systems have no long-term scheduler (e.g. Windows, UNIX,Linux)

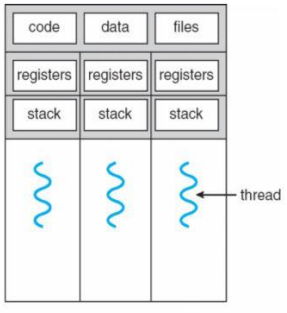
进程通信：shared memory, message passing

Interprocess Communication (IPC) 进程间通信

消息传递易实现，共享内存速度快

L-4 Threads & Concurrency

Process-> Unit of Resource ownership; Unit of Dispatching;



many-to-one->OS is not aware of user-level threads. Solaris Green Threads, GNU Portable Threads. Entire process blocks when one thread blocks, 效率高，不需要 OS 支持，但不是真正的并行；one-to-one->需要 OS 支持，Windows NT/XP/2000 , Linux, Solaris 9 and later。When one thread blocks, other threads from process can be scheduled。开销大并发性好；

many-to-many-> Solaris prior to version 9 Windows NT/2000 with the ThreadFiber package

Two-level Model-> IRIX HP-UX Tru64 UNIX Solaris 8 and earlier

L-5 CPU Scheduling

周转时间 Turnaround time 进程从提交到完成所经历的时间。

响应时间 Response time 从进程提出请求到首次被响应

Waiting time = turnaround time – burst time

First-Come, First-Served Scheduling 先来先服务

Shortest-Job-First (SJF) Scheduling 短作业优先

Priority Scheduling 优先权调度

Round Robin (RR) 时间片轮转调度

Multilevel Queue Scheduling 多级队列调度（foreground – RR-80% background – FCFS-20%）

Multilevel Feedback Queue Scheduling 多级反馈队列调度

允许进程在队列间移动，CPU 时间高，优先级低（时间片=8，队列 0；时间片 16，队列 1；FCFS 队列 2；0->1->2）

SJF 变型：***最短剩余时间优先*** SRT(Shortest Remaining Time)- 允许比当前进程剩余时间更短的进程来抢占

高响应比优先调度算法 Highest Response Ratio Next(HRRN) **响应比 R** = (等待时间 + 要求执行时间) / 要求执行时间 WIN XP 优先级

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

L-6 Process Synchronization

Solution to Critical-Section Problem must satisfy: Mutual Exclusion; Progress; Bounded Waiting.

让权等待：当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入"忙等"状态。(受惠的是其他进程)。

Peterson
```
do {
        flag [i]:= true;
        turn = 1-i;
        while (flag [1-i] and turn == 1-i) ;
                Critical Section
        flag [i] = false;
                Remainder Section
} while (1);
```

分析：满足了互斥、有空让进和有限等待。这个算法能够解决临界区问题。这里，turn 的含义是"当前的'进入权'给了谁"。而 Flag[i]==ture 的含义为，Pi 已经准备好进入自己临界区了。

面包房算法
```
do {
        choosing[i] = true;
        number[i] = max(number[0], number[1], ..., number [n – 1])+1;
        choosing[i] = false;
        for (j = 0; j < n; j++) {
            while (choosing[j]) ;
                while ((number[j] != 0)&&((number[j], j) < (number[i], i))) ;
        }
                //critical section
        number[i] = 0;
                //remainder section

} while (1);
```

choosing[n]为真，表示进程 i 正在获取它的排队登记号；number[i]的值，是进程 i 的当前排队登记号。如果值为 0，表示进程 i 未参加排队，不想获得该资源。初始化：False, 0

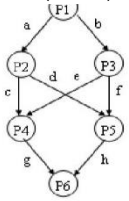
```
boolean TestAndSet(boolean *target)  {
    boolean rv = *target;
        *target = true;
        return rv;
}
```

信号量：wait->P; signal->V

```
wait (S) { while S <= 0 ; // no-op S--; }
signal (S) { S++; }
wait(S){        申请资源
        S.value--;
        if (S.value < 0){
                //add this process to S.List;
                block;
        }
}
signal(S){      释放资源
        S.value++;
        if (S.value <= 0){
                //remove a process P from S.List;
                wakeup(P);
        }
}
```

如果两个 wait 操作相邻，那么它们的顺序至关重要，而两个相邻的 signal 操作的顺序无关紧要。一个同步 wait 操作与一个互斥 wait 操作在一起时，同步 wait(empty/full)在互斥 wait(mutex)前。

```
main( )
cobegin{
    P1( ) { …; V(a); V(b); }
    P2( ){ P(a); …; V(c); V(d); }
    P3( ){ P(b); …; V(e); V(f); }
    P4( ){ P(c); P(e); …; V(g); }
    P5( ){ P(d); P(f); …; V(h); }
    P6( ){ P(g); P(h); …; }
```

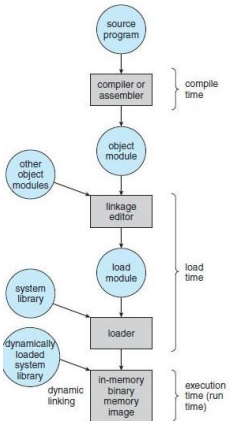


Mutex=1; full=0; empty=N producer->

wait(empty);wait(mutex);signal(mutex);signal(full);
The first readers-writers problem-> multiple readers to read at the same time.
The second readers-writers problem-> Once a writer is ready the writer performs write as soon as possible , readers may starve

L-7 Deadlocks
产生死锁的四个必要条件：mutual exclusion, hold and wait, no preemption, circular wait
请求边-p->r；分配边-r->p
死锁定理：S 为死锁状态的充分条件是：当且仅当 S 状态的资源分配图是不可完全简化的。
Prevention→针对四个条件（占有等待：执行前获得所有资源/没有资源时才能申请；循环等待：每个进程按照递增顺序申请资源）
Avoidance→不安全状态可能产生死锁
银行家算法：不从头开始（？）
Detection→单个实例情况：Maintain wait-for graph 检查图有无环(n^2)；多个实例：m*n^2(m资源，n 进程)

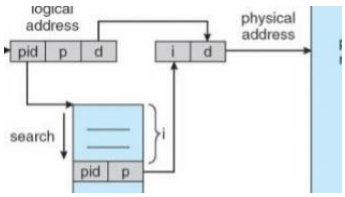
L-8 Main Memory
CPU 能直接访问的只有内存和寄存器
Logical address=virtual address physical->实地址
A pair of base and limit registers （基址寄存器和限长寄存器）define the logical address space
Address binding [1]compile time->absolute code; [2]load time->relocatable code 编译时不知道地址；[3]execution time-> 进程在执行时可能换内存段。



Memory-Management Unit (MMU)：物理地址->虚拟地址
Dynamic Loading（动态加载）子程序在调用时才被加载。
Dynamic Linking（动态链接）将链接延迟到运行时，external libraries can be preloaded into (shared) memory
Multiple-partition allocation （多分区分配）分区式管理的基本思想是将内存划分成若干个连续区域，称为分区。每个分区只能存放一个进程。Fix or dynamic(first, best, worst，首次适应和最佳适应会产生外部碎片)next fit，从上次结束处开始
内部碎片(fragmentation)：分区内不能使用
compaction or defragmentation （紧缩，拼接）
分页：frame->物理内存； page->逻辑内存

■ Effective Access Time (EAT)
$EAT = (t+\varepsilon)\,\alpha + (t + t + \varepsilon)\,(1 - \alpha)$

Linux4 级页表，Windows2 级
反向页表：按照物理地址排序→



分段 segmentation：段名称+段偏移

某计算机采用二级页表的分页存储管理方式，按字节编址，页大小为2^{10}字节，页表项大小为2字节，逻辑地址结构为：

页目录号	页号	页内偏移量

逻辑地址空间大小为2^{16}页，则表示整个逻辑地址空间的页目录表中包含表项的个数至少是

2^10/2=2^9 2^16/2^9=2^7

L-9 Virtual Memory
按需调页：Bring a page into memory only when it is needed
Three major components of the page-fault service time [1]Service the page-fault interrupt [2]Read in the page(将缺页读入) [3]Restart the process
Copy-on-Write（写时拷贝）父进程与子进程共享页面，若写则创建副本

Page Replacement Algorithms ***NRU[最近未使用]***
OPT&LRU 都没有 belady's anomaly
First-In-First-Out Algorithm (FIFO，先进先出算法)
Optimal Algorithm （OPT 最佳页面置换算法）
Least Recently Used (LRU) Algorithm (最近最少使用算法)
->实现方式 Counter implementation, Stack implementation
LRU Approximation Algorithms （近似 LRU 算法）：
Additional-Reference-Bits Algorithm
（附加引用位，被访问时左边最高位置 1，定期右移并且最高位补 0，于是寄存器数值最小的是最久未使用页面。）
Second-Chance（clock）Algorithm
Enhanced Second-Chance Algorithm
(Reference bit, modified bit)：（增强二次机会，macintosh）
- (0,0): best page to replace
- (0,1): not quite good for replacement
- (1,0): will be used soon
- (1,1): worst page to replace.

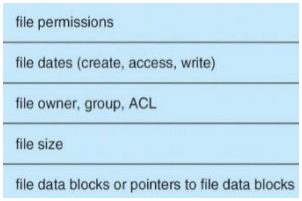
淘汰次序:(0,0) ⇒(0,1)⇒(1,0)⇒(1,1)

Counting-Base Page Replacement:
Least Frequently Used Algorithm (LFU 最不经常使用算法)
Most Frequently Used Algorithm (MFU 引用最多算法)
Allocation of Frames(帧分配) Fix or priority
置换策略：Global vs Local
组合成三种策略：[1]固定分配局部置换策略[2]可变分配全局置换策略[3]可变分配局部置换
Thrashing（颠簸、抖动）
a process is busy swapping pages in and out
通过局部置换（优先置换）限制颠簸，如果一个进程开始颠簸，那么它不能从其他进程拿到帧，且不能使后者颠簸。
working set (WS)工作集：The set of pages in the most recent △page references
△->working-set window(工作集窗口)-> a fixed number of
WSS_i (working set size of Process P_i 工作集大小) =
total number of pages referenced in the most recent Δ (varies in time)

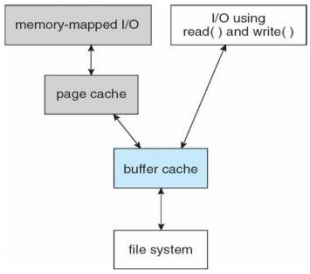
Page-Fault Frequency（PFF）页错误频率
内存映射文件 Memory-Mapped Files：使用虚拟内存将文件 I/O 作为普通内存访问，Simplifies file access by treating file I/O through memory rather than read() write() system calls

L-10 File-System Interface
一个磁盘可以有多个分区，多个磁盘可组成一个分区
Acyclic-Graph Directories 无环图结构目录 Have shared subdirectories and files
File Sharing->NFS

L-11 File System Implementation
File system resides on secondary storage (disks)



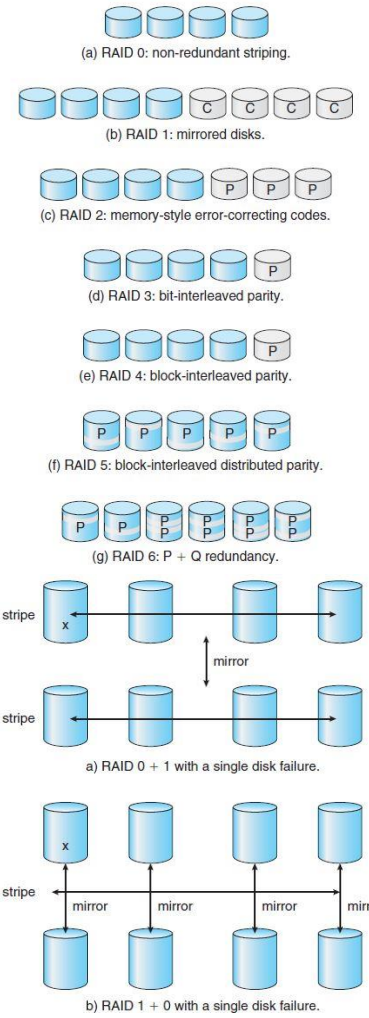
Boot control block, Volume control block 卷的详细信息，Directory structure(organizes the files), ***FCB***
Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
Allocation Methods 分配方法
[1]Contiguous allocation（连续分配）浪费空间、文件大小不可变，外部碎片；Extent-Based Systems （基于长度系统） [2]Linked allocation （链接分配） FAT32[3]Indexed allocation（索引分配）
Free-Space Management [1] Bit vector[2] Linked list[3] Grouping 分组，将 n 个空闲块的地址存在第一个块 [4] Counting 记录第一块地址和之后连续空间数



←I/O Without a Unified Buffer Cache

L-12 Mass-Storage Systems
磁盘附属 [1]Host-attached storage（主机附属）通过 IO 端口，IDE, ATA, SCSI16 个设备(1initiator15gtargets) 每个 targets8 个 logical units SAN FC-AL 126 个设备

[2] Network-Attached Storage NAS
定位时间=寻道时间+旋转等待时间
调度：FCFS, SSTF, SCAN(到头), C-SCAN, LOOK, C-LOOK
格式化：[1]低级格式化（物理）为磁盘每个扇区采用数据结构[2]将磁盘由柱面组成分区[3]逻辑格式化，创建文件系统
Swap->计数器 0，页槽可用；计数 3，映射到 3 进程
RAID→ Disk striping（条带化） uses a group of disks as one storage unit



2-HAMMING 3-位交织奇偶校验 4-块交织 5-块交织分布，将数据、奇偶分布在 N+1 6-差错纠正码 R-S

0+1 分散成条，然后镜像；一个出错，整条不能
1+0 先镜像，再分散；单个不可用，其镜像可用
三级存储设备：可移动，软盘、磁带，CD，DVD

L-13 I/O Systems

CPU 作用	等待设备	数据传送
采用**轮询**方式的设备控制器 • CPU需要等待设备就绪，且参与数据传送		
轮询方式	需要	参与
采用**中断**方式的设备控制器 • CPU无需等待设备就绪，但响应中断后参与数据传送		
中断方式	不需要	参与
通过**DMA**直接控制存储器 • CPU在数据传送开始和结束时参与，与主存进行数据交换时不参与		
DMA 方式	不需要	不参与

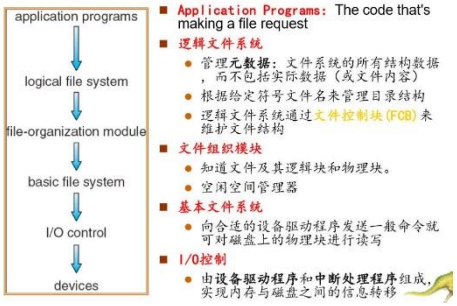
Blocking->进程挂起直到 I/O 完成为止；Nonblocking->I/O 调用立刻返回[User interface, data copy (buffered I/O) Implemented via multi-threading] Asynchronous（异步）->进程与 I/O 同时运行
非阻塞与异步系统调用的差别是：非阻塞 read 调用会马上返回，其所读取的数据可以等于或少于所要求的，或为零；异步 read 调用所要求的传输应完整地执行，其具体执行可以是将来某个特定时间。
SPOOLing（Simultaneous Peripheral Operation On Line）虚拟设备
Buffer->维持"copy semantics"

semaphore chopstick[5] = 1。
对于第 i 哲学家：
do {
wait(chopstick[i]);
wait(chopstick[(i+1) % 5]);
eat;
signal(chopstick[i]);
signal(chopstick[(i+1) % 5]);
think;
} while (1);
其他解决方法：
1.allow at most four philosophers to be sitting simultaneously at the table--a spare resource.
2.allow a philosopher to pick up chopsticks only if both are

available, and pick up them simultaneously.
3.use a asymmetric solution: an odd philosopher picks up first the left chopstick and then the right chopstick, whereas an even one picks up first the right and then the left.
Type of access-> read, write, execute, append, delete, list; owner group public RWX
Swap-space — Virtual memory uses disk space as an extension of main memory
内核与 I/O 有关服务：I/O scheduling、buffering、caching、spooling(虚拟化)、device reservation 提供对设备的独占访问 and error handling.

mm_struct

count	对mm_struct结构的引用进行计数。为了在Linux中实现线程，内核调用clone派生一个线程，线程和调用进程共享用户空间，即mm_struct结构，派生后系统会累加mm_struct中的引用计数。
pgd	进程的页目录基地址，当调度程序调度一个进程运行时，就将这个地址转成物理地址，并写入控制寄存器（CR3）
map_count	在进程的整个用户空间中虚存区的个数
semaphore	对mm_struct结构进行互斥访问所使用的信号量
start_code, end_code, start_data, end_data	进程的代码段和数据段的起始地址和终止地址
start_brk, brk, start_stack;	每个进程都有一个特殊的地址区间，这个区间就是所谓的堆，也就是前图中的空洞。前两个域分别描述堆的起始地址和终止的地址，最后一个域描述堆栈段的起始地址。
arg_start, arg_end, env_start, env_end	命令行参数所在的堆栈部分的起始地址和终止地址； 环境串所在的堆栈部分的起始地址和终止地址
rss, total_vm, locked_vm	进程驻留在物理内存中的页面数，进程所需的总页数，被锁定在物理内存中的页数。
mmap	vm_area_struct虚存区结构形成一个单链表，其基址由小到大排列
mmap_avl	vm_area_struct虚存区结构形成一个颗AVL平衡树
mmap_cache	最近一次用到的虚存区很可能下一次还要用到，因此，把最近用到的虚存区结构放入高速缓存，这个虚存区就由mmap_cache指向。



→The context-switch causes overhead by OS. The action affects many objects, but ？ is not included. global variable
→Which process state transition is valid only on a preemptive multitasking system?
A. Running → Blocked B. Blocked → Running
C. Ready → Running **D. Running → Ready**
→The mutual exclusion semaphore of two concurrent processes has the value 0 (zero) at thismoment. It indicates that a process has entered the critical-section, and no process is being blocked
→若某单处理器多进程系统中有多个就绪态进程，则下列关于处理机调度的叙述中**错误**的是在进程处于临界区时不能进行处理机调度
→下列选项中，降低进程优先级的合理时机是？
A. 进程从就绪态转为运行态
B. 进程的时间片用完
C. 进程刚完成 I/O，进入就绪队列
D. 进程长期处于就绪队列中
→As to semaphores, we can think an execution of signal operation as applying for a resource.**FALSE**
→While a process is blocked on a semaphore's queue, it is engaged in busy waiting.**FALSE**
→ Critical section can be enforced with a general semaphore whose initial value is greater than 1.**FALSE**
→ Which of the following Critical Section problem solutions results in busy-waiting?
A. Special machine instruction
B. critical region
C. Semaphore
D. Monitor
→ Assume that a system has 9 instances of 1 resource type shared by 4 processes. How many resource instances can a process be allowed to request in order to avoid deadlock? **3**
→ ？ memory allocation scheme may produce external fragmentation.
A. None of above
B. Multiple-partition
C. Demand
D. system halts
→首次适应算法的空闲区是____。
A. 始端指针表指向最大空闲区
B. 寻找从最大空闲区开始
C. 按大小递增顺序连在一起
D. 按地址递增顺序连在一起
→ Implementing LRU precisely in an OS is expensive, so practical implementations often use an approximation called .
A. LFU **B. NRU**
C. MRU D. MFU
→Which of the following memory management is not suitable for a multi-programming environment?
A. single contiguous memory allocation
B. variable-sized partitions allocation
C. segmentation with paging
D. fix-sized partitions allocation
→ Assume that you have a page-reference string for a process with m frames (initially all empty). The page-reference string has length p; and n distinct page numbers occur in it. Answer these questions for any page-replacement algorithms:
a. What is a lower bound on the number of page faults? **n**
b. What is an upper bound on the number of page faults? **P**
→UNIX treats I/O devices as _____.
A. directory files
B. indexed files
C. special files
D. regular files
→The I/O control of disk devices mainly adopt following method. **DMA**
→Disk access time does not include ____ .
A. seek time
B. CPU scheduling time
C. read/write time
D. rotational latency time
→下面哪条命令可以把 f1.txt 复制到 f2.txt?
A. cp f2.txt | f1.txt
B. cp f1.txt | f2.txt
C. cat f1.txt | f2.txt
D. cat f1.txt > f2.txt
→若在超级权限下，对当前目录的 linux-3.17.1 子目录建立一个符号链接文件，该符号链接文件名为/usr/src/linux，应该执行哪个指令？
A. ln /usr/src/linux ./linux-3.17.1/
B. ln –s ./linux-3.17.1/ /usr/src/linux
C. ln ./linux-3.17.1/ /usr/src/linux
D. ln –s /usr/src/linux ./linux-3.17.1/
→CPU 调度算法____没有在 LINUX 的 2.6 以后的版中使用。
A.ROUND ROBIN
B. SJF
C. PRIORITY
D. FCFS
→在 ext2 文件系统中，下面的说法，　　是错误的。
A. 目录文件有 inode 节点
B. 磁盘设备有 inode 节点
C. 打印机设备文件没有 inode 节点
D. 每个文件都有一个 inode 节点
→在系统调用处理过程中，下面的哪一个是最合适的流程?
A. do_fork -> sys_fork -> fork
B. sys_fork -> do_fork -> fork
C. sys_fork -> fork -> do_fork
D. fork -> sys_fork -> do_fork
→下面的哪一个结构是描述 VFS 的一个文件对象?
struct file
→Linux 内核为了管理物理空间，采用了_____。
Buddy 算法（伙伴算法）
上述三者都是
struct page
Slab 技术
→Commonly, File control block is created on disk when the ____ system call （i.e., Linux/Unix） is invoked. **Open**
→查找并显示/var/log/message 文件中最后 10 行内容的命令是?
dmesg /var/log/message | tail 10
cat /var/log/message | head
cat /var/log/message
dmesg |tail
→ Paging is a virtual memory-management scheme. **F**
→ If the size of disk space is large enough in the virtual memory-management system, then a process can have unlimited address space. **F**
→ File directory is stored in a fixed zone in main memory. **F**
→Thread can be a basic scheduling unit, but it is not a basic unit of resource allocation.**T**
→ 14. The size of address space of a virtual memory is equal to the sum of main memory and secondary memory. **F**
→ 15. Virtual address is the memory address that a running program want to access. **F**
→ 19. The main purpose to introduce buffer technology is to improve the I/O efficiency. **F**
→ Which of following memory management schemes is fit for program's dynamic linking?
A. Segmentation B. Paging
C. Dynamic-sized partitions
D. Fixed-sized partitions
→ 27. An I/O channel is a special ___。
A．I/O device B.I/O controller
C. processor D. memory
→ 28. Virtual devices are implemented with _____ technology。
A. channel B. buffer
C. SPOOLing D. controller
→ 4. Consider a file system on a disk that has both logical and physical block size of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, indexed), answer these questions:
(1). How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)
(2). If we are currently at logical block 10 (the last block accessed was 20) and want to access logical block 5, how many physical blocks must be read from the disk?
令 Z 是文件开始物理地址（块号）。
(1). 若使用连续分配策略时。用 512 去除逻辑地址，则 X 和 Y 分别表示得到的整数和余数。
（a）将 X 加上 Z 得到物理块号，Y 为块内的位移（b）1
(2). 若使用链接分配策略。用 511 去除逻辑地址，则 X 和 Y 分别表示得到的整数和余数。
（a）查找链表到第 X+1 块，Y+1 位该块内的位移量。
（b）5
(3). 若使用索引分配策略。用 512 去除逻辑地址，则 X 和 Y 分别表示得到的整数和余数。
（a）把索引块读入内存中，则物理块地址存放在索引块在第 X 位置中，Y 为块内的位移量。（b）2
实时系统&分时系统
分时操作系统按照相等的时间片调度进程轮流运行，分时操作系统由调度程序自动计算进程的优先级，而不是由用户控制进程的优先级。这样的系统无法实时响应外部异步事件。[使一台计算机同时为几个、几十个甚至几百个用户服务的一种操作系统。]
实时操作系统能够在限定的时间内执行完所规定的功能，并能在限定的时间内对外部的异步事件作出响应。
→ 1. The sequential file is good for sequential storage devices, however not fit for disks. **F**
→ 3. Directories are usually kept in memory. **F**
→ 5. In the tree-like directory organization, files are accessed by means of paths which consist of directories followed by the file name. **T**
→ 7. Most of devices with lower speed are of shared devices. **F**
→ 9. The cache technology takes a portion of external storage as the cache pool. **F**

→ 11. In terms of response time, the requirements to time-sharing system are the same as that to real-time system. **F**

→ 19. In the paging system, the addresses of a program that has been loaded into memory are physical addresses. **F**

→7 . The cache technology____

A. is to solve the speed mismatch between CPU and I/O devices, but is not dedicated for that

B. is dedicated to solve the speed mismatch between CPU and I/O devices

C. employs a portion of memory to served as buffering area

D. defines buffer pool consists of buffer units, each unit consists of a header and a body

哲学家进餐解决办法：

```
1.  semaphore chopstick[5]={1,1,1,1,1};
2.  semaphore count=4; // 设置一个 count，最多有四个哲学家可以进来
3.  void philosopher(int i)
4.  {
5.      while(true)
6.      {
7.          think();
8.          wait(count);
9.          wait(chopstick[i]);
10.         wait(chopstick[(i+1)%5]);
11.         eat();
12.         signal(chopstick[i]);
13.         signal(chopstick[(i+1)%5]);
14.         signal(count);
15.     }
16. }
```

```
1.  semaphore mutex = 1; // 这个过程需要判断两根筷子是否可用，并保护起来
2.  semaphore chopstick[5]={1,1,1,1,1};
3.  void philosopher(int i)
4.
5.  {
6.      while(true)
7.      {
8.          think();
9.          wait(mutex); // 保护信号量
10.         wait(chopstick[(i+1)%5]);
11.         wait(chopstick[i]);
12.         signal(mutex);
13.         eat();
14.         signal(chopstick[(i+1)%5]);
15.         signal(chopstick[i]);
16.     }
17. }
```

```
if(i%2 == 0) //偶数哲学家，先右后左。
{
        wait (chopstick[(i + 1)%5]) ;
        wait (chopstick[i]) ;
        eat();
        signal (chopstick[(i + 1)%5]) ;
        signal (chopstick[i]) ;
}
else //奇数哲学家，先左后右。
{
        wait (chopstick[i]) ;
        wait (chopstick[(i + 1)%5]) ;
        eat();
        signal (chopstick[i]) ;
        signal (chopstick[(i + 1)%5]) ;
}
```

→ 4. The evolution of operating systems for mainframes is roughly like from _____.

A.) no software multi-programming multi-tasking

B.) no software multi-tasking multi-programming

C.) no software resident monitors multi-tasking multi-programming

D.) no software resident monitors multi-programming multi-tasking

→6. _____ is to keep multiple jobs in memory simultaneously in order to keep the CPU busy.

A.) batch processing

B.) real-time processing

C.) multiprogramming

D.) parallel execution

→7. What is the purpose of system calls?

A.) System calls allow us to write assembly language programs.

B.) System calls are the standard interface between a user process and a kernel process.

C.) System calls allow user-level processes to request services of the operating system.

D.) There is no real purpose to system calls

→17. A message-passing system for an OS is _____.

A.) A kind of direct communication

B.) A kind of low-level communication

C.) A kind of inter-process communication

D.) A kind of symmetrical communication

→18. We will have a rendezvous between the sender and the receiver if _____.

A.) The sender is non-blocking and the receiver is non-blocking.

B.) The sender is non-blocking and the receiver is blocking.

C.) The sender is blocking and the receiver is non-blocking.

D.) The sender is blocking and the receiver is blocking.

→19. The threads of a single process can not share _____.

A.) code B.) files

C.) stacks D.) priority

→ 22. Which of the following is *incorrect* for the CPU long-term scheduler?

A.) It controls the degree of multi-programming.

B.) It runs as often as short-term scheduler.

C.) It selects a good process mix of I/O-bound and CPU-bound processes.

D.) It can be a user rather than a program.

→ 20. Which of the following is *incorrect*?

A.) The system call fork may just duplicate the thread that invoked it.

B.) The system call fork may duplicate all the threads of a process.

C.) The system call exec may just replace the thread that invoked it.

D.) The system call exec may replace the entire process.

→ 24. Which of the following Operating systems use preemptive scheduling?

A.) Mac OS 8

B.) Windows 3.x

C.) Windows 2000

D.) DOS 6.0

→ 38. Which of the following methods is to prevent the deadlock from the beginning?

A.) Banker's algorithm

B.) Deadlock detection

C.) Resource allocation in an increasing order of enumeration

D.) Simplification of resource allocation graph

→ 42. The critical section for process synchronization is _____.

A.) a buffer B.) a data section

C.) a synchronization mechanism

D.) a segment of code

→43. The semaphores can be used to solve ____ mutual exclusive problem(s).

A.) one B.) some

C.) all the D.) None of the above

→ 57. Dynamic relocation relies on _____.

A.) a relocation register B.) object code

C.) relocation program D.) None of the above

→ 65. With paging memory management, paging is usually done by _____.

A.) Programmer B.) User

C.) Compiler D.) hardware

→ 67. For a system with a 64-bit logical address space, which page table structure is inappropriate?

A.)6-level page table B.) 3-level page table

C.)hashed page table D.)inverted page table

→85. The cache for hard disks is usually implemented by _____.

A.) registers B.) primary memory

C.) secondary memory D.) tertiary storage

→ 100.The UNIX system call for creating a file is **creat**

tar -xvf linux-4.8.tar.xz xz -d patch-4.8.xz | patch –p1 make mrproper 清除目录下所有配置文件和先前生成核心时产生的.o 文件； cp /boot/config-`uname -r` .config 复制配置文件 **mkfs [-V] [-t fstype] [fs-options] filesys [blocks]** 在特定的分区上建立 linux 文件系统 （device 预备检查的硬盘分区，例如 /dev/sda1-V 详细显示模式-t 给定档案系统的型式，Linux 的预设值为 ext2-c 在制做档案系统前，检查该 partition 是否有坏轨-l bad_blocks_file 将有坏轨的 block 资料加到 bad_blocks_file 里面 block 给 block 的大小）将 sda6 分区格式化为 ext3 格式 mfks -t ext3 /dev/sda6 **mke2fs（mkfs.ext2）** [-cFMqrSvV][-b <区块大小>][-f <不连续区段大小>][-i <字节>][-N <inode 数>][-l <文件>][-L <标签>][-m <百分比值>][-R=<区块数>][设备名称][区块数] **losetup** [-d <卸载>][-e <加密方式>][-o <平移数目>][循环设备代号][文件] **umount** [-ahnrvV][-t <文件系统类型>][文件系统] 可以使用 man 和 info 命令来获得每个 Linux 命令的帮助手册，使用 whoami 命令找到用户名 uname（显示操作系统的名称），uname -n（显示系统域名），uname -p（显示系统的 CPU 名称）;使用 uptime 命令判断系统已启动运行的时间和当前系统中有多少登录用户; 11. Linux 系统规定，隐含文件是首字符为"."的文件，如.profile。查找主目录下隐含文件：find . -name '.*' stat temp 查看~/temp 目录 inode 号 file lab1 查看文件内容类型 tail -n 5 file 显示后 5 行 cp 如果要保留修改时间不变，则可加上-a 选项 。搜索主目录，显示创建时间在~/smallFile 之后的文件及其路径→find ~ -newer smallFile -print ； ls [-alrtAFR] [name...]（-a 显示所有文件及目录 (ls 内定将文件名或目录名称开头为"."的视为隐藏档，不会列出)-l 除文件名称外，亦将文件型态、权限、拥有者、文件大小等资讯详细列-r 将文件以相反次序显示(原定依英文字母次序)-t 将文件依建立时间之先后次序列出-A 同 -a ，但不列出"."(目前目录) 及 ".."(父目录)-F 在列出的文件名称后加一符号; 例如可执行档则加 "*", 目录则加 "/"-R 若目录下有文件，则以下之文件亦皆依序列出）

中断机制不属于操作系统提供给用户的可使用资源。

实时操作系统必须在被控对象规定时间内处理完实践，考虑实时性和可靠性。

批处理作业必须提交作业控制信息，分时系统提供全部人机交互。

下面选项 ，不可能发生在用户态的是 A 系统调用 B 外部中断 C 进程切换 D 缺页。

多道程序并发，分区法分配内存代价最小。

只有分区法要求作业占用连续的存储空间。

段页式（3 次访存）中段管理逻辑地址空间，页管理物理地址空间。 分页提供一维的地址结构。

页表修改位->是否被写过。引用位->是否用到过。

产生颠簸的主要原因是页面置换算法不合理。

下列关于虚拟存储的叙述中，正确的是（）。

虚拟存储只能基于连续分配技术

虚拟存储只能基于非连续分配技术

虚拟存储容量只受外存容量的限制

虚拟存储容量只受内存容量的限制

分页式不能采用静态重定位，因为运行中可能改变程序位置

多级页表优点->减少占用的连续空间。

```
1.  unsigned short  uid, gid//用户标识号组标识号
2.  unsigned short euid, egid //用户有效标识号，组有效标识号
3.  unsigned short suid, sgid //用户备份标识号，组备份标识号
4.  unsigned short fsuid, fsgid //用户文件标识号，组文件标识号
5.  struct task_struct *p_opptr //指向祖先进程 PCB 的指针
6.  structtask_struct *p_pptr//指向父 PCB
7.  structtask_struct *p_cptr //指向子 PCB
8.  struct task_struct *p_ysptr //弟 PCB
9.  struct task_struct *p_osptr //兄 PCB
10. struct task_struct *next_task//下一个 PCB
11. struct task_struct *prev_task//上一个
12. struct task_struct *next_run //指向可运行队列的下一个 PCB 的指针
13. struct task_struct *prev_run //指向可运行队列的上一个 PCB 的指针
14. long  counter   //时间片计数器
15. long  nice      //进程优先级
16. unsigned long rt_priority//实时进程的优先级
17. unsigned long policy  //进程调度策略
18. long start_time //进程创建的时间
19. long utime //进程在用户态下耗费的时间
20. long stime //进程在核心态下耗费的时间
21. long cutime //所有子进程在用户态下耗费的时间
22. long cstime//所有子进程在核心态下耗费的时间
23. unsigned longtimeout //进程申请延时
24. struct desc_struct *ldt //进程局部描述符表指针
25. unsigned long saved_kernel_stack //核心态下堆栈的指针
26. unsigned long kernel_stack_page //核心态下堆栈的页表指针
27. unsigned longsignal //进程接收到信号
28. unsigned longblocked //阻塞信号掩码
29. struct signal_struct*sig//信号处理函数表指针
30. int exit_signal //进程终止的信号
31. struct sem_undo *semundo//进程要释放的信号量
32. struct sem_queue *semsleeping //与信号量操作相关的等待队列
```