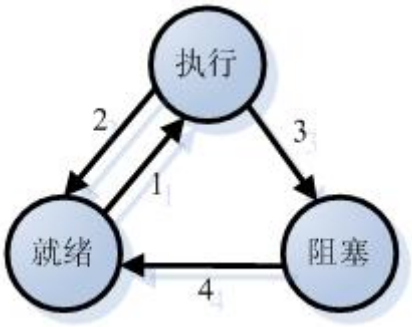
2008-2009 操作系统试题 B 卷

声明：自己写的，答案仅供参考。才疏学浅，出错难免。敬请各位指正。
参考书目：操作系统概念（第七版）高教
by Codefor hk_yuhe@126.com
一、简答题（9 道小题，共 66 分）
1、（6 分）请列出三项你认为最明显是为支持操作系统而设计的硬件特性，并分别说明其作用
硬件中断：现代操作系统是以事件为驱动的，而事件的发生通常是通过硬件（或软件）中断来表示。硬件可以随时通过系统总线向 CPU 发出信号，以触发中断。当 CPU 中断时，它暂停正在做的事并立即转到固定的位置去执行。该固定位置通常是中断服务程序开始位置的地址。中断服务程序开始执行，在执行完后，CPU 重新执行被中断的计算。
硬件同步：用于解决临界区问题。硬件特性能简化编程任务并且提高系统效率。许多系统都拥有简单硬件指令，用来解决临界区问题。这些特殊硬件指令能原子地（不可中断的）检查和修改字的内容或交换两个字的内容。如 TestAndSet 指令。
MMU 内存管理单元：完成从虚拟地址到物理地址的转换。由程序所生成的所有逻辑地址的集合称为逻辑地址空间（或虚拟地址空间），与这些逻辑地址相对应的所有物理地址集合称为物理地址空间。运行时从逻辑地址到物理地址的映射是由被称为内存管理单元的硬件设备完成的。如最简单的重定位寄存器。
还有 TLB 等
2、（8 分）简述进程切换、线程切换和中断处理三者在切换过程和切换内容方面的异动

	切换过程	切换内容
进程切换	中断 CPU，保存当前运行进程的上下文，从就绪进程队列中选择进程交给 CPU 执行	切换进程上下文（用进程控制块 PCB 描述），包括 CUP 寄存器值、进程状态、内存管理信息、打开文件状态等等
线程切换	时间片到，线程让出 CPU，切换到另一个线程。通常采用非抢先式和更简单的规则，也无须用户态和核心态切换	只需保护和设置少量寄存器状态、用户栈、私有存储区域
中断处理	响应中断请求，保护中断现场，开中断，中断服务，关中断，恢复现场，开中断，中断返回	中断的 CPU 状态寄存器，中断返回地址，中断程序使用的 CPU 内部寄存器内容
三者异同点	三者都需要保护现场，执行其他指令，再恢复现场继续执行。线程是独立调度的基本单位，进程是资源拥有的基本单位。创建或撤销进程的开销远大于线程时的开销。现代计算机都是以中断事件为驱动的。	都需要保护切换时机的有关切换对象的状态，如进程的 PCB（进程控制块）、线程的 TEB（线程执行环境块）、CPU 寄存器状态。 三种切换是不同层次。由于切换对象不同，具体切换内容也不一样

3、（8 分）试画出进程的三状态转换图，并对每一条转换路径举出一可能引起这一转换的例子



1 调度算法分配
2 中断，如时间片到等
3I/O 操作或等待事件，如父进程等待子进程结束
4I/O 操作完成或事件发生
4、（8 分）定义一种文件系统（如 FAT、EXT3）通常需要定义哪些部分？各部分的功能及其包含内容如何？试通过举例一实际文件系统说明
操作系统通过文件系统来轻松存储、定位、提取数据。文件系统有两个不同的设计问题。a、定义文件系统对用户的接口（包括定义文件及其属性、文件所允许的操作、组织文件的目录结构）b、创建数据结构和算法将逻辑文件系统映射到物理外存设备上。
通常文件系统是分层的。如应用程序、逻辑文件系统、文件组织系统、基本文件系统、I/O 控制、设备。I/O 控制为最底层，由设备驱动程序和中断处理程序组成，实现内存和磁盘直接的信息传输。基本文件系统向设备驱动发送命令对磁盘物理块进程读写。文件组织系统记录了文件分配类型和文件位置。将逻辑块地址转换成基本文件系统所用的物理地址。逻辑文件系统管理元数据，即文件系统的所有数据结构而非实际数据。通过 FCB 来维护文件结构。

在磁盘上，文件系统可能包括如下信息：如何启动存储的 OS，总的块数、空闲的块数和位置、目录结构各个具体文件。
NTFS 中包括主引导扇区、主控文件表（主控文件表包括卷或分区的详细信息、文件的目录结构等内容）、系统范围内打开的文件表、单个进程打开的文件表、内存缓存的目录结构等。
5、（6 分）试证明：如果为每一资源赋值一优先级，并禁止进程申请优先级小于等于其已占有资源优先级的其他资源，就可以避免死锁。
证明：为每一资源赋值一优先级，并禁止进程申请优先级小于等于其已占有资源优先级的其他资源，也就是规定每个进程只能按递增顺序申请资源。即一个进程开始可以申请任何数量的资源类型 Ri 的实例，之后，当且进当 F（Rj）>F（Ri），该进程可以申请资源类型 Rj 的实例。这种方案可以保证循环等待不存在，进而避免死锁。反证之：
假定有一个循环等待存在，设涉及循环等待的进程集合为{P0,P1,P2...Pn}，其中 Pi 等待一个资源 Ri，而 Ri 又为进程 Pi+1 占有，Pn 等待 P0 的资源。即进程 Pi+1 占有资源 Ri 并等待 Ri+1，对所有的 i 必须有 F（Ri）<F（Ri+1），这意味着 F（R0）<F（R1）<...<F（Rn）<F（R0），根据传递规则，F（R0）<F（R0），这显然矛盾，所以循环等待条件不成立，进而死锁不存在。
6、（6 分）某系统使用请求分页存储管理，如果页在内存中，满足一次内存请求需要 200ns。如果页不在内存中，如果有空闲的页框不需要换出已修改的页，则请求需要 7ms。如果替换

出的页已被修改，则需要15ms，若缺页率为5%，并且60%时间用于换出修改的页，试计算有效访问时间是多少？假定系统只运行一个进程且页交换时CPU空闲。
本题不带TLB，分三种情况：页在内存中，直接访问一次内存，200ns；页不在内存，有空闲页框可用，不需要换出脏块，7ms；换出脏块，15ms
有效访问时间=0.95*200/1000000+0.05（7*0.4+15*0.6）=0.59019ms
7、（8分）大多数操作系统使用高速缓存来改善文件系统性能。首先简述常见的两种缓存组织形式及其适用场合，然后说明使用这类操作系统时，为什么需要先完成系统关机后才能切断电源。
大多数磁盘控制器都有本地内存作为板载高速缓存。
缓冲缓存和页面缓存
缓冲缓存：在系统中开一块独立内存用作缓冲缓存。缓存读入的磁盘块，假设位于其中的块马上需要使用。当从文件系统读入磁盘块时，假设该磁盘块中数据马上需要使用，可以放入缓冲缓存中。
页面缓存：缓存文件数据，页面缓存使用虚拟内存技术，将文件数据作为页面而不是面向文件系统的块来缓存。采用虚拟地址类缓存文件数据，比采用物理磁盘块来缓存相比，更为高效。

不管是缓冲缓存还是页面换存，换存的数据都在内存中，而内存数据是易失的，掉电后数据不能保持。另一方面，缓存的内容可能已被更改变脏，在关机之前必须回写到磁盘中以永久保持。所以使用缓存的系统时，必须先关机再切断电源。
8、（8分）简述I/O软件的分层结构，并说明下列功能可能在哪一层或两层实现?
(a)检查访问权限
(b)调度I/O操作
(c)检验请求信息是否在高速缓存中
(d)处理一个设备中断
(e)分配一个I/O缓冲器
(f)将一个从输入设备中接收到的回车字符转换成为通用的换行符
用户进程、内核I/O子系统、设备驱动器、中断处理程序、设备控制器
(a)检查访问权限 内核I/O子系统
(b)调度I/O操作 内核I/O子系统
(c)检验请求信息是否在高速缓存中 设备驱动器
(d)处理一个设备中断 设备驱动器 设备控制器
(e)分配一个I/O缓冲器 内核I/O子系统 设备驱动器
(f)将一个从输入设备中接收到的回车字符转换成为通用的换行符 设备控制器 中断处理程序
9、（8分）简述什么是动态链接，并进一步说明为什么分段系统比分页系统更容易实现共享
动态链接是相对于静态链接而言的，多用于库文件。所谓静态链接是指把要调用的函数或者过程链接到可执行文件中，成为可执行文件 的一部分。而动态链接所调用的函数代码并没有被拷贝到应用程序的可执行文件中去，而是仅仅在其中加入了所调用函数的描述信息（重定位信息）。仅当可执行文件被装载执行时，在操作系统的帮助下，可执行文件和库文件（如.d11和.so）之间建立链接关系。当要执行所调 用库文件中的函数时，根据链接产生的重定位信息，操作系统才转去执行库中相应的函数代码。一般情况下，库文件镜像在内存中只有一份。动态链接也方便库的更新和共享。

分段与分页
每一段在逻辑上是相对完整的一组信息，分段技术中共享信息是在段一级出现的。因此，任何共享的信息可以单独作一个段，同样段中所有内容就可以用相同的方式进行使用。而页是信息的物理单位，在一个页面中可能存在逻辑上互相独立的两组或更多组信息都各有不同的使用方式。因此，分段系统比分页系统更容易实现共享。
二、算法及实验题（3小题，共34分）
1、（14分）目前大多数的处理都支持TestAndSet原子操作指令，它可有如下定义：

```
boollean TestAndSet (boolean locked)
{
    if(locked)
        return true;
    else
    {
        locked=true;
        return false;
    }
}
```

请利用TestAndSet并采用C语言实现一个通用信号量。队列可使用QueueList queue形式的定义。将当前进程阻塞到队列可表示为queue.Block,将队列的第一个进程唤醒可表示为queue.Wakeup.

```
typedef struct
{
    int value;
}semaphore;

boolean lock = false;

do
{
    while(TestAndSet (&lock))
    {}
//critical section
    S->value--;
    if(S->value<0)
    {
        queue.Block;
    }

    lock=false;
}while(TRUE)

do
{
```

```
    while(TestAndSet (&lock))
    {}
//critical section
    S->value++;
    if(S->value<=0)
    {
        queue.Wakeup;
    }
    lock = false;
}while(TRUE)
```

2、（14分）用C语言编写一个VtoP，实现基于分页的虚存系统中的虚拟地址到物理地址的转换，需要考虑缺页中断以及物理内存短缺时的换出

系统采用两级页表机制，两级页表项结构同为pageEntry，页目录的入口是pageTable。页号和页框号的位数均按其定义类型的C语言长度计算。假设已经实现了以下函数可直接调用。

Free：在空闲表中查找一个空闲页。如找到，返回页框号并从空闲表中摘取，否则返回-1.

Replace：使用局部页替换策略选择一个替换页，并返回其页号，错误返回-1.

PageMap：以页号和页框号为参数，将页面从可执行文件中读写到页框中。正确返回页框号，错误返回-1

SwapIn：以页号和页框号为参数，将页面从交换设备上读写到页框中。正确返回页框号，错误返回-1.

SwapOut:以页号和页框号为参数，将页框写到交换设备的页存储区中，正确返回页框号，错误返回-1.

相关定义如下：

```
    struct pageEntry
    {
        int frameNum;//frame number
        boolean isOut;//in (false) or swapped out(true)
        boolean dirty;//not modified(false)or modified(true)
        boolean present;//not mapped(false)or mapped (true)
    }
    struct virtAddress
    {
        short gPage;//portion of address specifying page directory
        short sPage;//portion of address specifying page table
        short offset;//portion of address specifying offset
    }

    struct pageEntry * pageTable
    int Free();
    int Replace();
    int PageMap(int page,int frame);
```

```
    int SwapIn(int page,int frame);
    int SwapOut(int page,int frame);


/*
整体思路：如果页在内存 Present 为真，直接返回其帧号；
如果页被换出，需要换进来。首先使用 Free 获得空白帧，如果获取成功，使用 SwapIn 函数换进原先被换出的帧，并返回此帧号，如果 Free 不成功，说明物理内存不足，需要换出牺牲页，使用 RePlace 获得牺牲的页号，然后使用 SwapOut 函数换出该页，并取得该页对应的页帧，使用此页帧换进目标页，并返回此帧号。
类似，如果页未映射，首先 Free，Free 成功就 PageMap，返回帧号；Free 的话就 Replace 换出牺牲页，再 Pagmap 没有映射的页
*/
    int VtoP(virtAddress* vadd,int frameNum)
    {
         pageEntry* p= pageTable+vadd->gPage;//第 gPage 个目录的地址
         //*p 是第 gPage 条目录对应页表的首地址
         pageEntry* s=*p+vadd->sPage;//s 就是我们要找的页

         if(s->present)
         return s->frameNum;
         else if(s->isOut)
        {
            int fr=Free();//返回物理帧号，空白帧，需要页映射
            if(fr!=-1)
            {
                 if(SwapIn(vadd->sPage,fr)!=-1)
                 return fr;
                 else
                 return -1;
            }
            else//没有剩余物理帧，需要换出
            {
                 int re=Replace();//返回待牺牲页号
                 if(re==-1)
                 return -1;

                  pageEntry* temp = *p+re;

                 if(temp->isDirty)//脏块才换出
                 {
                     SwapOut(re,temp->frameNum);
                }
                if(SwapIn(vadd->sPage,temp->frameNum)!=-1)
```

```
            return temp->frameNum;
            else
            return -1;
    }

}
 else//页没有映射
 {
    int fr=Free();//返回物理帧号，空白帧，需要页映射
    if(fr!=-1)
    {
         if(PageMap(vadd->sPage,fr)!=-1)
         return fr;
         else
         return -1;
    }
    else//没有剩余物理帧，需要换出
    {
         int re=Replace();//返回页号
         if(re==-1)
         return -1;

         pageEntry* temp = *p+re;
         if(temp->isDirty)
         {
           SwapOut(re,temp->frameNum);
         }
        if(PageMap(vadd->sPage,temp->frameNum)!=-1)
        return temp->frameNum;
        else
        return -1;
    }
 }
}
```