《操作系统》课堂测试题 1-参考答案

一、选择题(每小题 2 分，共 20 分)

1. 临界区是指并发进程中访问临界资源的(　　)。
A. 管理信息块　B. 信息存储块　　C. 数据块　　D. 代码块

2. 磁盘 I/O 调度算法中先来先服务和(　　)算法可能会随时改变移动臂的运动方向。
A. CLOCK 算法　B. SSTF 算法　C. SCAN 算法 D. CSCAN 算法

3. UNIX 系统内核提供了多种块设备的读写操作,其中不包括如下(　)写操作。
A. 延迟写　　B. 预先写　　C. 同步写　　D. 异步写

4、进程从运行状态进入就绪状态的原因可能是(　　)。
A.等待某一事件　B.被选中占有处理器　C.时间片用完　D.等待的事件已发生

5. 实时系统中的进程调度，通常采用（　）算法。
A.响应比高者优先　　B.短作业优先　C.时间片轮转　D.抢占式的优先数高者优先

6. 一个进程从等待状态进入就绪状态可能是由于当前进程（ ）。
A. 执行了 V 操作　　B. 执行了 P 操作　　C. 运行结束　　D. 时间片用完

7. 在以下的存储管理方案中，能实现虚拟存储管理的是（ ）。
A.固定式分区分配　　B.可变式分区分配　C.页式存储管理　D.请求页式存储管理

8. UNIX 文件系统对盘空间的管理采用（ ）。
A. FAT 表法　　B. 位示图法　　C. 空闲块链接法　　D. 空闲块成组链接法

9. 在段页式存贮管理中一条内存访问指令实际需要访问内存(　　) 次。
A. 1　　B. 2　　C. 3　　D. 4

10. 在可变式分区分配方案中，某一作业完成后，系统收回其主存空间，并与相邻空闲区合并，为此需修改空闲分区说明表，会造成空闲区数减 1 的情况是(　)。
A.无上邻也无下邻空闲区　　B.有上邻空闲区，但无下邻空闲区
C.有下邻空闲区，但无上邻空闲区　　D.有上邻也有下邻空闲区

二、简答如下问题（每小题 5 分，共 30 分）

（1）分析程序、进程、线程三者之间的区别与联系。为什么要引入线程概念？

	程序	进程	线程
概念	求解问题的处理过程描述	是并发程序在一个处理机上的执行过程，是 OS 分配资源的单位	是进程中可并发执行的“小进程”，是分配处理机的单位
特性	静态	动态	动态
生存期	永存	有生有灭，有生存期	有生有灭，有生存期
组成	代码与数据	代码、数据+ PCB	代码、数据+ TCB
对应关系	多对多	多对多	一个进程可包含多个线程
相互关系	程序是进程的基础和实体	进程是程序在处理机上的执行过程	进程中线程共享属于进程的资源

引入线程：是为了在提高多道系统的并发性的同时，不增加太大的系统管理（进程上下文切换）开销。

（2）什么是作业调度？什么是进程调度？试列举出至少两种作业调度算法和三种进程调度算法。

作业调度：亦称为高级调度，用于从提交到外存的作业中选择一个作业，并为其创建进程、调入内存；

进程调度：亦称为低级调度，用于从内存中就绪状态的进程中选择一个进程，为其分配处理机，并使其运行；

调度算法：先来先服务（FCFS）、短者优先；响应比高者优先；优先级高者优先、基于时间片的轮转法（RR）、反馈式多队列

（3）何谓缓冲技术？给出操作系统中应用缓冲技术的两个例子，并说明其中解决了什么问题。

缓冲技术：是一种用空间换时间的技术，即在两个有速度矛盾的系统之间设置一定规模的缓存区（空间），通过预先读、延迟写等方式缓解其速度矛盾；

例如，页式存储管理中通过放置在高速缓存中的**“快表”（TLB）**就能较好解决地址重定位时查内存页表**“费时”**的问题；通过放置在内存中的**“磁盘缓冲区”**就能较好解决磁盘 I/ O（比如文件读写）**“太慢”**的问题；

（4）何谓存储管理中的内、外碎片问题?如何解决?

内碎片：存在于固定分区存储管理中，即分配给申请者被其浪费的空间，解决办法是采用可变分区方法；

外碎片：存在于可变分区存储管理中，即按需分配给申请者之后留在系统中而被浪费的空间（太小无法利用），解决办法是：内存紧缩、分配时加阈值、页式存储管理。

（5）什么要引入设备独立性？如何实现设备独立性？

设备独立性：是指应用程序与物理设备无关性，实现设备独立性的方法是，要求在应用程序中使用逻辑设备名（即设备类型），操作系统维持一张“逻辑设备-物理设备映射表”，并实现设备的动态分配和动态绑定。

（6）文件的物理结构（组织）有哪几种？ UNIX 系统采用什么样的文件物理结构？

文件的物理结构：连续文件、串联（链式）文件、索引文件；UNIX 系统的文件物理结构采用**多重索引**（亦称“**混合索引**”）的索引文件。

三、综合题（每小题 10 分，共 50 分）

1．一条东西方向的河上有一座桥，桥面仅能容纳一辆汽车通过。允许同一时刻若干汽车沿同一方向过桥，但若桥两边的车同时上桥、向对面行驶就会发生阻塞。试用信号量机制设计一个避免阻塞发生、同时还能防止某个方向的汽车因对面源源不断的汽车过桥而无休止等待的算法，简述算法设计思想并用流程图或伪代码描述算法处理流程。

解：这本质上是一个变形的**读者-写者问题**。

解法一（“读者-写者”问题解决方案）：将一个方向的车辆看做“读者”，另一个方向的车辆看做“写者”，则可直接使用“读者-写者”问题解决方案

（1）**读者优先**的“读者-写者”问题解决方案：

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true)
    {
     semWait (x);
     readcount++;
     if (readcount == 1)
          semWait (wsem);
     semSignal (x);
     READUNIT();
     semWait (x);
     readcount--;
     if (readcount == 0)
          semSignal (wsem);
     semSignal (x);
    }
 }
void writer()
{
    while (true)
    {
     semWait (wsem);
     WRITEUNIT();
     semSignal (wsem);
    }
}

void main()
{
    readcount = 0;
    parbegin (reader, writer);
    }
```

（2）**写者优先**的“读者-写者”问题解决方案：

```
/*program readersandwriters*/
int  readcount, writecount;
semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
     while (true)
     {
       semWait (z);
            semWait (rsem);
                 semWait (x);
                      readcount++;
                      if (readcount == 1)
                        semWait (wsem);
                 semSignal (x);
            semSignal (rsem);
       semSignal (z);
       READUNIT();
       semWait (x);
            readcount--;
            if (readcount == 0)
                 semSignal (wsem);
       semSignal (x);
     }
}
void writer ()
{
     while (true)
     {
       semWait (y);
            writecount++;
            if (writecount == 1)
                 semWait (rsem);
       semSignal (y);
       semWait (wsem);
       WRITEUNIT();
       semSignal (wsem);
       semWait (y);
            writecount--;
            if (writecount == 0)
                 semSignal (rsem);
       semSignal (y);
     }
}
void main()
{
     readcount = writecount = 0;
     parbegin (reader, writer);
}
```

【存在什么问题呢？】

解法二（改进的“读者-写者”问题解决方案）:

设整型变量 countS=0，countN=0(分别表示两个方向的车辆数目)；互斥信号量 x=1,y=1,mutex=1(分别用于对 countS 和 countN，以及对“桥”的互斥使用)；

（1）对于“南-北”方向的车辆：

```
Wait (x);
countS++;
If countS==1 wait (mutex);
Signal (x);
Passing;
Wait (x);
countS--;
If countS==0 Signal (mutex);
Signal (x);
```

（2）对于“北-南”方向的车辆：

```
Wait (y);
countN++;
If countN==1 wait (mutex);
Signal (y);
Passing;
Wait (y);
countN--;
If countN==0 Signal (mutex);
Signal (y);
```

【还有问题吗？】

解法二（既公平又无饿死的“读者-写者”问题解决方案）:

新增一个信号量 Q=1（“过桥”许可证（无论哪个方向!），申请到了才能通过！未申请成功者被阻塞并依次在 Q 的阻塞队列排队，随后将被依次唤醒）；整型变量 countS=0，countN=0(分别表示两个方向正在过桥的车辆数目)；互斥信号量 x=1,y=1,mutex=1(分别用于对 countS 和 countN，以及对“桥”的互斥使用)；

（1）对于“南-北”方向的车辆：

```
Wait (Q);
Wait (x);
countS++;
If countS==1 wait (mutex);
Signal (x);
Signal (Q);

Passing;

Wait (x);
countS--;
If countS==0 Signal (mutex);
Signal (x);
```

（2）对于“北-南”方向的车辆：

```
Wait (Q);
Wait (y);
countN++;
If countN==1 wait (mutex);
Signal (y);
Signal (Q);

Passing;

Wait (y);
countN--;
If countN==0 Signal (mutex);
Signal (y);
```

2、分析死锁产生的原因和发生死锁的必要条件。试给出一个实现死锁检测的算法：（1）基本思想；（2）数据结构；（3）伪代码/流程图。

死锁原因：竞争资源（资源受限）；推进次序不当。

死锁条件：互斥、保持等待、非剥夺、循环等待

死锁检测算法描述一：

已知，资源需求矩阵 Claim、资源分配矩阵 Allocation、可用资源向量 Available；记 Q 为进程还需请求的资源矩阵，Q=C-A。

算法思想：标记没有死锁的进程

1) 所有进程均未标记；

2) 标记 Q 中行向量为 0 的进程；

3) 初始化临时向量 W，令 W=Available；

4) 查找下标 i，使进程 i 当前未被标记，且 Q[i,*] <W(向量比较)；若找不到这样的 i，则算法终止；

5) 若找到这样的 i，则标记进程 i，并 W=W+Allocation [i,*] ，GOTO 4)；

6) 结论：存在未标记进程↔(当且仅当)存在死锁,未标记的进程即死锁进程；即所有进程均被标记↔(当且仅当)不存在死锁

=====

死锁检测算法描述二：

```
struct state
{
        int resource[m];
        int available[m];
        int claim[n][m];
        int alloc[n][m];
}
```

(a) global data structures

```
if (alloc [i,*] + request [*] > claim [i,*])
     < error >;                                  /* total request > claim*/
else if (request [*] > available [*])
     < suspend process >;
else                                                  /* simulate alloc */
{
     < define newstate by:
     alloc [i,*] = alloc [i,*] + request [*];
     available [*] = available [*] - request [*] >;
}
if (safe (newstate))
     < carry out allocation >;
else
{
     < restore original state >;
     < suspend process >;
}
```

(b) resource alloc algorithm

```
boolean safe (state S)
{
    int currentavail[m];
    process rest[<number of processes>];
    currentavail = available;
    rest = {all processes};
    possible = true;
    while (possible)
    {
        <find a process Pk in rest such that
            claim [k,*] - alloc [k,*] <= currentavail;>
        if (found)                              /* simulate execution of Pk */
        {
            currentavail = currentavail + alloc [k,*];
            rest = rest - {Pk};
        }
        else
            possible = false;
    }
    return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

3、段式与页式存储管理的主要区别是什么？（1）分析二者相结合为基础的段页式虚拟存储管理方法的优缺点；（2）描述其地址重定位的具体实施方案（主要数据结构和处理流程等）。

分区式、页式与段式存储管理的主要区别：

	分区	分页	分段
大小	大小不同，个数不定	所有页的大小固定、相同、较小	各个段的大小不一定相同
由谁划分	系统自动分页，对用户透明	系统自动分页，对用户透明	分段对用户可见，通常由程序员或编译程序进行
逻辑地址空间	单一线性地址空间	单一线性地址空间	多个线性地址空间
地址形式	一维	一维	二维
地址映射方法	基地址寄存器	控制寄存器+页表	控制寄存器+段表
优缺点	地址映射简单高效；存在内、外碎片问题；	内存利用率高；但地址映射耗费时间	有利于大型软件开发和内存保护；存在内、外碎片问题；地址映射耗费时间

段页式虚拟存储管理的优缺点：（1）优点：内存管理（页式）最高效；为用户提供了比实际内存大的“虚拟存储器”。（2）缺点：管理信息（段表、页表）需要耗费存储空间；地址映射和缺页中断处理耗费时间。

地址映射的软硬件：（1）控制寄存器；（2）中断处理机构；（3）段表、页表；（4）虚存管理软件。

地址映射过程：

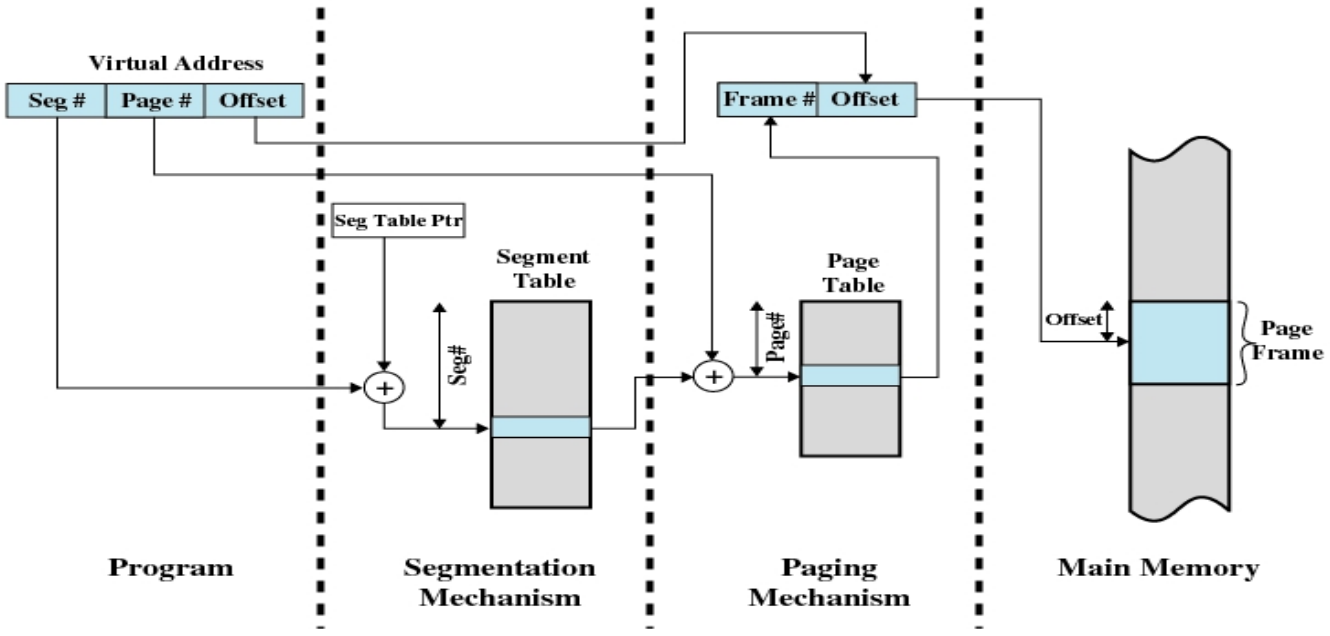
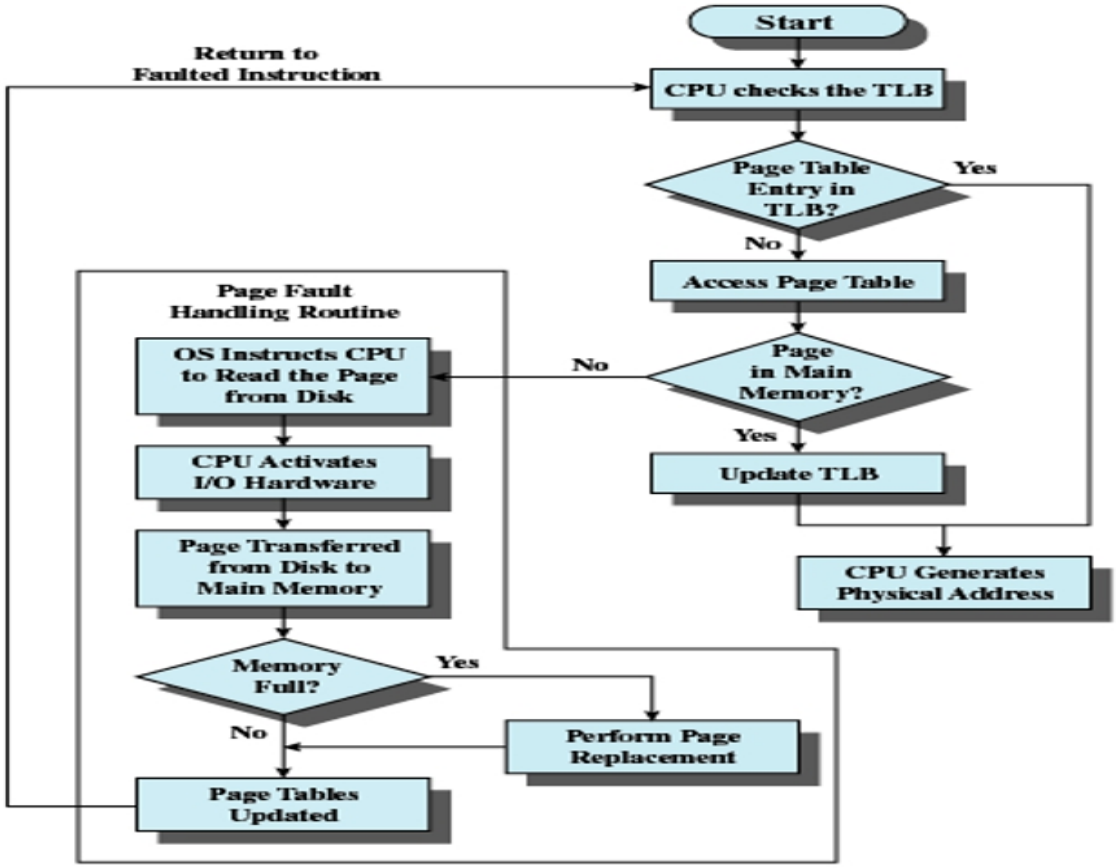


Figure 8.13 Address Translation in a Segmentation/Paging System

内存扩充：通过请求页式（部分装入、缺页中断和页面淘汰）实现虚拟内存



4、在一个多用户文件系统中，要实现文件共享需要解决哪些主要问题？简述在 UNIX 系统中是如何解决这些问题的。

主要问题：（1）共享访问权限管理；（2）并发访问控制。

UNIX 系统解决方案：

（1）将文件使用者分为 3 类：创建者、同组用户和其它用户，将文件访问权限也规定为 3 类：只可执行“e”、只读“r”和写“w”(可读可写可执行)，因此对每个文件，仅使用 9bit 即实现了共享文件的访问权限属性描述；在文件操作时，通过检查使用者对该文件的权限属性，便可实现共享文件的访问权限控制；

（2）并发访问控制：通过粗粒度的文件锁和细粒度的记录锁实现并发访问控制。

5、某磁盘系统有 200 个柱面，由外向里升序编号，假定当前磁头向里移动后停在 100 号柱面。现有如下访问磁盘请求序列：190、10、160、80、90、125、30、20、140、25，试给出分别采用 SSTF 和 CSCAN 磁盘调度算法时响应上述请求的次序，并分别计算两种算法平均移动磁头时间（距离）。

SSTF：（100）->90 ->80-> 125->140-> 160->190 ->30-> 25->20->10
平均移动磁头时间（距离）=(10+10+45+15+20+30+160+5+5+10)/10 =31
CSCAN：（100）-> 125->140-> 160->190 ->10->20-> 25->30 ->80->90
平均移动磁头时间（距离）=(25+15+20+30+180+10+5+5+50+10)/10 =35