一、单队列调度方式有优势也有不足。优势是：能够从单 CPU 调度程序很简单地发展而来。不足是：它的**扩展性不好**(为了保证在多 CPU 上正常运行，需要在代码中通过加锁 locking 来保证原子性)，并且**不能很好地保证缓存亲和度**。 扩展性问题没有好的解决办法，亲和性问题可以通过尽量将任务调度在同一个 CPU 上，但可能需要牺牲部分任务的亲和性。

多队列调度方式没有单队列的缺点，但是存在**负载不均衡**问题，即每个 CPU 上运行的任务数量存在差别。解决办法是通过**工作窃取(work stealing)**的方式在 CPU 之间迁移任务。

二、分段是指将虚拟地址空间中代码、堆、栈等段分别映射到物理内存，而不是将整个虚拟地址空间全部映射到物理内存。分页是将虚拟地址空间等分为固定大小的页（如 4KB），以页的方式将虚拟地址空间与物理内存进行映射。分段的优点是硬件支持比较简单；缺点是可能会导致外部碎片、空闲空间的管理比较复杂。分页的优点是空间管理简单、没有外部碎片，缺点是页表存储空间大、地址翻译慢（需要 TLB 加速），因此需要硬件的支持较多。

三、死锁是指两个或多个进程（或线程）在执行的过程中，因为竞争资源而造成互相等待的现象，若无外力作用，它们都无法推进下去。
死锁的四个条件是:
互斥:线程对于需要的资源进行互斥的访问(例如一个线程抢到锁)。
持有并等待:线程持有了资源(例如已将持有的锁)，同时又在等待其他资源(例如，需要获得的锁)。
非抢占:线程获得的资源(例如锁)，不能被抢占。
循环等待:线程之间存在一个环路，环路上每个线程都额外持有一个资源，而这个资源又是下一个线程要申请的。
其它类型的同步问题有：**违反原子性**(atomicity violation)缺陷，即即应该一起执行的指令序列没有一起执行和**违反顺序**(order violation)缺陷，即两个线程所需的顺序没有强制保证。

四、write back 是指数据写入缓存（还未最终写入磁盘）就任务写操作成功，优点是**写性能高**，缺点是可能在系统断电或崩溃时导致**数据丢失**（因为数据仍在缓存中）；write through 是指数据不仅写入缓存，还要保证数据写入到物理磁盘，优点是**不存在数据丢失**，缺点是**写性能不高**。

五、
（1）

```
hello world (pid:17106)
hello, I am child (pid:17107)
      32     123     960 p3.c
hello, I am parent of 17107 (wc:17107) (pid:17106)
```

（2）创建 5 个进程，2 个线程。

六、

（1）地址空间需要 17 位表示；offset 需要 7 位表示；VPN 需要 10 位表示；页表存储空间为 2^{12}（4096 字节）；页目录存储空间为 2^{7}（128 字节）。

（2）

```
1   VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2   (Success, TlbEntry) = TLB_Lookup(VPN)
3   if (Success == True)    // TLB Hit
4       if (CanAccess(TlbEntry.ProtectBits) == True)
5           Offset   = VirtualAddress & OFFSET_MASK
6           PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7           Register = AccessMemory(PhysAddr)
8       else
9           RaiseException(PROTECTION_FAULT)
10  else                    // TLB Miss
11      // first, get page directory entry
12      PDIndex = (VPN & PD_MASK) >> PD_SHIFT
13      PDEAddr = PDBR + (PDIndex * sizeof(PDE))
14      PDE     = AccessMemory(PDEAddr)
15      if (PDE.Valid == False)
16          RaiseException(SEGMENTATION_FAULT)
17      else
18          // PDE is valid: now fetch PTE from page table
19          PTIndex = (VPN & PT_MASK) >> PT_SHIFT
20          PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
21          PTE     = AccessMemory(PTEAddr)
22          if (PTE.Valid == False)
23              RaiseException(SEGMENTATION_FAULT)
24          else if (CanAccess(PTE.ProtectBits) == False)
25              RaiseException(PROTECTION_FAULT)
26          else
27              TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
28              RetryInstruction()
```

图 20.4　多级页表控制流

（3）

七、生产者和消费者访问共享的数据（buffer 数组），没有相应的同步控制，会产生竞态条件。图中代码没有正确解决生产者消费者问题。有两种解决方法：使用条件变量或信号量。

```
1   cond_t empty, fill;
2   mutex_t mutex;
3
4   void *producer(void *arg) {
5       int i;
6       for (i = 0; i < loops; i++) {
7           Pthread_mutex_lock(&mutex);            // p1
8           while (count == MAX)                   // p2
9               Pthread_cond_wait(&empty, &mutex); // p3
10          put(i);                                // p4
11          Pthread_cond_signal(&fill);            // p5
12          Pthread_mutex_unlock(&mutex);          // p6
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          Pthread_mutex_lock(&mutex);            // c1
20          while (count == 0)                     // c2
21              Pthread_cond_wait(&fill, &mutex);  // c3
22          int tmp = get();                       // c4
23          Pthread_cond_signal(&empty);           // c5
24          Pthread_mutex_unlock(&mutex);          // c6
25          printf("%d\n", tmp);
26      }
27  }
```

Figure 30.14: **The Correct Producer/Consumer Synchronization**

```
1   sem_t empty;
2   sem_t full;
3   sem_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           sem_wait(&empty);           // line p1
9           sem_wait(&mutex);           // line p1.5 (MOVED MUTEX HERE...)
10          put(i);                     // line p2
11          sem_post(&mutex);           // line p2.5 (... AND HERE)
12          sem_post(&full);            // line p3
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          sem_wait(&full);            // line c1
20          sem_wait(&mutex);           // line c1.5 (MOVED MUTEX HERE...)
21          int tmp = get();            // line c2
22          sem_post(&mutex);           // line c2.5 (... AND HERE)
23          sem_post(&empty);           // line c3
24          printf("%d\n", tmp);
25      }
26  }
27
28  int main(int argc, char *argv[]) {
29      // ...
30      sem_init(&empty, 0, MAX);  // MAX buffers are empty to begin with...
31      sem_init(&full, 0, 0);     // ... and 0 are full
32      sem_init(&mutex, 0, 1);    // mutex=1 because it is a lock
33      // ...
34  }
```

图 31.8　增加互斥量（正确的）

八、

（1）

```
REQUESTS ['10', '11', '12', '13']

Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate:320  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 40  Rotate:425  Transfer:120  Total: 585
```

（2）在处理完 11 号扇区后，要切换到 12 号扇区所在的磁道，但是在寻道的同时磁盘也在旋转，因此当寻道完成后，磁头已经不在 12 号扇区的起始位置，要再旋转一圈后才能开始读取 12 号扇区的数据。偏斜为 2。

（3） skew = track-distance / seek-speed / (rotational-space-degrees * rotation-speed)，本题假设寻道速率 seek-speed 为 1，旋转速率 rotation-speed 为 1，因此 skew = track-distance / rotational-space-degrees，因此本题中的 skew 应该是 skew = 40 / (360/12) = 40 / 30 = 1.3，skew = 2。

九、

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	N	$N/2$	$N-1$	$N-1$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4}R$

由要求“**在某些情况下，多于 1 个磁盘失效不会影响其可靠性**”可知 RAID-A 要使用 RAID-1，由要求“**在保证可靠性以及顺序读写性能的情况下，随机写性能要求尽可能高**”可知 RAID-B 要使用 RAID-5，以此为依据进行计算。

由条件可知分配给 RAID-A 的工作负载为：100R*20%的随机写，200S*80%的顺序读。那么 RAID-A 上执行时间由两部分组成：20R/((N/2)*R)=40/N=40/4=10s，和，160S/((N/2)*S)=320/N=320/4=80s。总时间是 80+10=90s。

由条件可知分配给 RAID-B 的工作负载为：100R*80%的随机写，200S*20%的顺序读。那么 RAID-B 上的执行时间有两部分组成：80R/((N/4)*R)=320/N=320/4=80s，和，40S/((N-1)*S)=40/3=13.3s，总时间是 80+13.3=93.3s。

因为 RAID-A 和 RAID-B 是同时执行，那么总时间应该是 93.3s。