Chapter 3 exercises

1.多线程 Web 服务器希望跟踪其服务的请求数（称为命中）。考虑以下两种策略来防止变量命中的竞争条件。第一个策略是在更新匹配时使用基本互斥锁：

```
int hits;
mutex_lock hit_lock;
hit_lock.acquire();
hits++;
hit_lock.release();
```

第二种策略是使用原子整数：

```
atomic_t hits; atomic_inc(&hits);
```

解释这两种策略中的哪一种更有效。

2. 请考虑用于分配和释放下图所示流程的代码示例。

```
#define MAX_PROCESSES 255 int number_of_processes = 0;
    /* the implementation of fork() calls this function */
    int allocate process() {
      int new_pid;
      if (number_of_processes == MAX PROCESSES)
        return -1;
      else {
        /* allocate necessary process resources */
        ++number_of_processes;
      }
    }
    return new_pid;
    /* the implementation of exit() calls this function */
    void release_process() {
      /* release process resources */
      --number_of_processes;
    }
```

a. Identify the race condition(s). 确定竞争条件。

b.假设您有一个名为 mutex 的互斥锁，其操作为 acquire（）和 release（）。指示需要锁定的位置以防止竞争条件。

c.我们可以替换整数变量吗？

```
int number of processes = 0
```

用原子整数

```
Atomic_t number_of_processes = 0
```

来防止竞争条件？

3. 可以将服务器设计为限制打开的连接数。例如，服务器可能希望在任何时间点只有 N 个套接字连接。一旦建立了 N 个连接，服务器就不会接受另一个传入连接，直到释放现有连接。说明服务器如何使用信号量来限制并发连接的数量。

   回答：信号量被初始化为允许的开放套接字连接的数量。当接受连接时，将调用 acquire（）方法，当释放连接时，将调用 release（）方法。如果系统达到允许的套接字连接数，则后续对 acquire（）的调用将阻塞，直到现有连接终止并调用 release 方法。

4. 演示如何使用 test 和 set（）指令在多处理器环境中实现 wait（）和 signal（）信号量操作。解决方案应该表现出最少的繁忙等待

回答：这是用于实现操作的伪代码：

```
int guard = 0;
 int semaphore value = 0;
wait()
{ while (TestAndSet(&guard) == 1);
 if (semaphore value == 0)
 { atomically add process to a queue of processes waiting for the semaphore and set guard to 0; }
 else { semaphore value--; guard = 0; }
}
 signal()
{ while (TestAndSet(&guard) == 1);
if (semaphore value == 0 && there is a process on the wait queue)
wake up the first process in the queue of waiting processes;
 else semaphore value++; guard = 0; }
```

5. 解释抢先和非抢先调度之间的区别。

假设以下过程在指定的时间到达执行。每个进程都将运行所列的时间。在回答问题时，使用非抢先式调度，并根据您必须做出决定时所拥有的信息做出所有决策。

过程	到达时间	爆发时间
Process	Arrival Time	Burst Time
P 1	0.0	8
P 2	0.4	4
P 3	1.0	1

a. 使用 FCFS 调度算法的这些进程的平均周转时间是多少？

b. 使用 SJF 调度算法，这些过程的平均周转时间是多少？

7. SJF 算法应该可以提高性能，但请注意我们选择在时间 0 运行进程 P 1，因为我们不知道两个较短的进程很快就会到达。计算如果 CPU 在前 1 个单元处于空闲状态然后使用 SJF 调度时的平均周转时间。请记住，进程 P 1 和 P 2 在此空闲时间内正在等待，因此它们的等待时间可能会增加。该算法可以称为未来知识调度。

8. 在多级排队系统的不同级别上有不同的时间 - 量子大小有什么优势？

9. 假设调度算法（在短期 CPU 调度级别）支持那些在最近使用最少处理器时间的进程。为什么这个算法会支持受 I / O 约束的程序，却又不会永久地饿死 CPU 绑定的程序？

回答：I / O 绑定程序具有仅执行少量计算的属性

在执行 IO 之前。此类程序通常不会耗尽其整个 CPU 量。 CPU 绑定

另一方面，程序使用它们的整个量程而不执行任何阻塞 IO 操作。

因此，人们可以通过给予更高的优先级来更好地利用计算机资源

I / O 绑定程序，允许它们在 CPU 绑定程序之前执行

10. 为什么调度程序将 I / O 绑定程序与 CPU 绑定程序区分开来很重要？

11. 讨论以下几对调度标准在某些设置中是如何冲突的。

a. CPU 利用率和响应时间

b. 平均周转时间和最长等待时间

c. I / O 设备利用率和 CPU 利用率

11.考虑以下一组进程，以毫秒为单位给出 CPU 突发的长度：

Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

假设过程在时间 0 处都以 P1，P2，P3，P4，P5 的顺序到达。

a. 绘制四个甘特图，使用以下调度算法说明这些过程的执行：FCFS，SJF，非抢先优先级（较大的优先级数意味着较高的优先级）和 RR（量子= 2）。

b. a 部分中每个调度算法的每个进程的周转时间是多少？

c. 每个调度算法的每个进程的等待时间是多少？

d. 哪种算法导致最小平均等待时间（在所有过程中）？

12.使用抢占式循环调度算法正在调度以下过程。为每个进程分配一个数字优先级，数字越大表示相对优先级越高。除了下面列出的进程外，系统还有一个空闲任务（它不占用 CPU 资源，并被标识为 P idle）。此任务的优先级为 0，并且只要系统没有其他可用的进程运行，就会安排该任务。时间量的长度是 10 个单位。如果进程被更高优先级的进程抢占，则抢占的进程将放置在队列的末尾。

Thread	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

a. 使用甘特图显示流程的计划顺序。

b. 每个流程的周转时间是多少？

c. 每个流程的等待时间是多少？

d. 什么是 CPU 利用率？

13.以下哪种调度算法可能导致饥饿？

a. 先到先得

b. 最短的工作

c. 循环赛

d. 优先

14.考虑一个运行十个 I / O 绑定任务和一个 CPU 绑定任务的系统。假设 I / O 绑定任务为每毫秒 CPU 计算发出一次 I / O 操作，并且每个 I / O 操作需要 10 毫秒才能完成。还假设上下文切换开销为 0.1 毫秒，并且所有进程都是长时间运行的任务。在以下情况下描述循环调度程序的 CPU 利用率：

a. 时间量是 1 毫秒

b. 时间量是 10 毫秒

15.解释以下调度算法在有利于短流程时的差异：

a. FCFS
b. RR
c. 多级反馈队列

16. 假设系统处于不安全状态。表明进程可以在不进入死锁状态的情况下完成执行。
17.请考虑以下系统快照：

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

使用银行家的算法回答以下问题：
a. 矩阵需要什么内容？
b. 系统是否处于安全状态？

c. 如果进程 P1 的请求到达（0,4,2,0），请求是否可以立即授予？

回答：
a. Need 矩阵的内容是 P0（0 0 0 0） P1（0 7 5 0） P2（1 0 0 2） P3（0 0 2 0） P4（0 6 4 0）。
b. 系统处于安全状态，因为 Available 矩阵等于（1 5 2 0），进程 P0 和 P3 都可以运行，当进程 P3 运行完时，它释放它的资源，而允许其它进程运行。
c. 可以被满足，满足以后，Available 矩阵等于（1 1 0 0），当以次序 P0，P2, P3, P1 ,P4 运行时候，可以完成运行。

18. 考虑一个由四个相同类型的资源组成的系统，这三个资源由三个进程共享，每个资源最多需要两个资源。显示系统无死锁。
回答：
19.考虑由 n 个进程共享的相同类型的 m 个资源组成的系统。进程一次只能请求或释放一个资源。如果满足以下两个条件，则表明系统无死锁：
a. 每个进程的最大需求是在一个资源和 m 个资源之间。
b. 所有最大需求的总和小于 m + n。
证明：使用 Section7.6.2 的术语,可以有：
a. $\sum_{i=1}^{n} Max_i$ <m+n

b. $Max_i \geqslant 1$ for all i
Proof: $Need_i = Max_i - Allocation$
If there exists a deadlock state then:
c. $\sum_{i=1}^{n} Allocation_i = m$
Use a. to get: $\sum Need_i + \sum Allocation_i = \sum Max_i < m + n$

Use c. to get: $\sum Need_i + m < m + n$

Rewrite to get: $\sum_{i=1}^{n} Need_i$

这意味着存在一个Pi的进程，其 $Need_i$ =0. 如果 Max_i >=1，那么Pi进程至少有一个资源可以释放。从而系统就不会进入死锁状态。