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**;**

**}**

1. 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？

1. 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。

1. 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; }

1. 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

1. 使用FCFS调度算法的这些进程的平均周转时间是多少？
2. 使用SJF调度算法，这些过程的平均周转时间是多少？
3. 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。
4. 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？
5. 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

1. 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？
2. 讨论以下几对调度标准在某些设置中是如何冲突的。
3. CPU利用率和响应时间
4. 平均周转时间和最长等待时间
5. 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

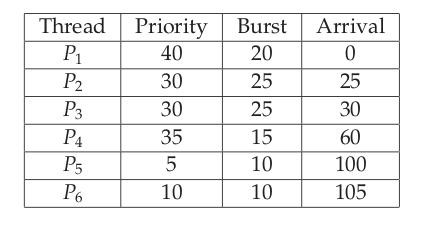
P4 4 2

P5 5 3

假设过程在时间0处都以P1，P2，P3，P4，P5的顺序到达。

1. 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）。
2. a部分中每个调度算法的每个进程的周转时间是多少？
3. 每个调度算法的每个进程的等待时间是多少？
4. 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。



1. 使用甘特图显示流程的计划顺序。
2. 每个流程的周转时间是多少？
3. 每个流程的等待时间是多少？
4. 什么是CPU利用率？

13.以下哪种调度算法可能导致饥饿？

1. 先到先得
2. 最短的工作
3. 循环赛
4. 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

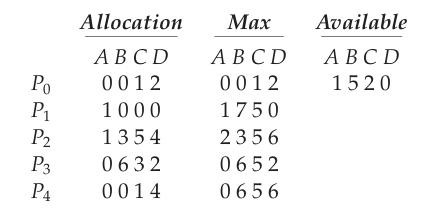
b. 时间量是10毫秒

15.解释以下调度算法在有利于短流程时的差异：

a. FCFS

1. RR
2. 多级反馈队列
3. 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：



使用银行家的算法回答以下问题：

1. 矩阵需要什么内容？
2. 系统是否处于安全状态？
3. 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。

1. 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：

1. 每个进程的最大需求是在一个资源和m个资源之间。
2. 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.<m+n

b. ≥ 1 for all *i*

Proof: = − 

If there exists a deadlock state then:

c. = *m*

Use a. to get:+ = *< m* + *n*

Use c. to get:+ *m < m* + *n*

Rewrite to get:**

这意味着存在一个Pi的进程，其=0.如果 >=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。