北京郵電大學

Chapter 4 Threads

LI Wensheng, SCS, BUPT

Exercise 1

1. What are two differences between user-level threads and kernel-level threads?
   Under what circumstances is one type better than the other?

Answer:

a. User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.
b. User threads are scheduled by the thread library and the kernel schedules kernel threads.
c. Kernel threads need not be associated with a process whereas every user thread belongs to a process.

Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

Exercise 2

2. Describe the actions taken by a kernel to context switch between kernel level threads.

Answer:

Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

Exercise 3

**3. What resources are used when a thread is created?
How do they differ from those used when a process is created?**

Answer:

Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation.

Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity.

Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

Chapter 5 CPU Scheduling

LI Wensheng, SCS, BUPT

Exercise 1

process	Arrival time	CPU Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

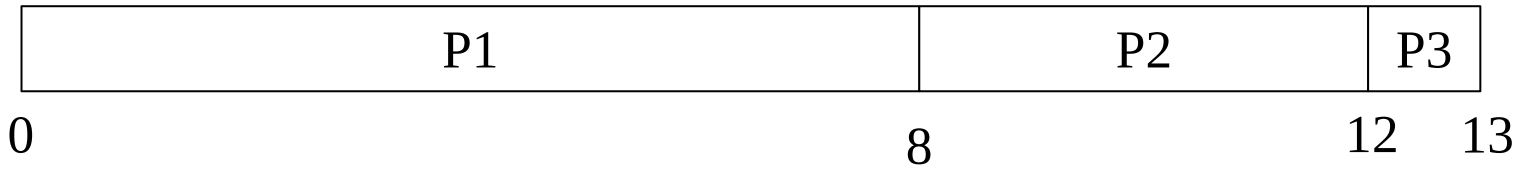
a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process *P*1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P*1 and *P*2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

Answer a:
FCFS scheduling algorithm

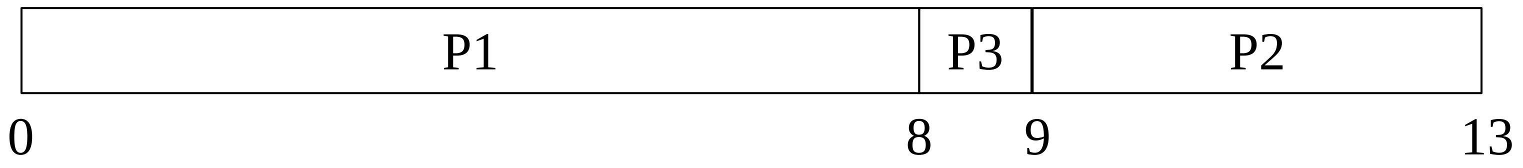
process	Arrival time	CPU Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1



- **turnaround time:**
  T1=8,   T2=12-0.4=11.6,   T3=13-1=12
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(8+11.6+12)/3=31.6/3 ≈10.53

Answer b:
SJF scheduling algorithm

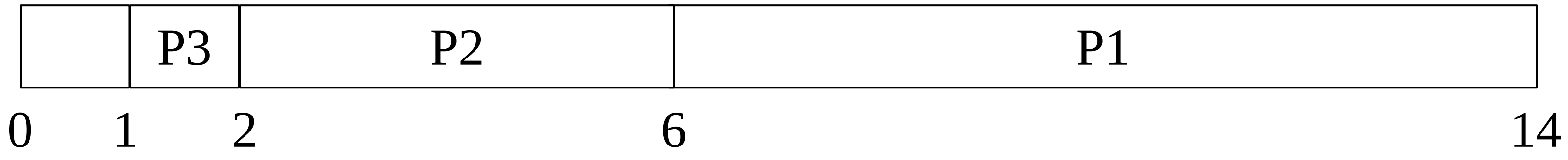
process	Arrival time	CPU Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1



- **turnaround time:**
  T1=8,   T2=13-0.4=12.6,   T3=9-1=8
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(8+12.6+8)/3=28.6/3 ≈9.53

Answer c： SJF future-knowledge scheduling algorithm

process	Arrival time	CPU Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1



- **turnaround time：**
  T1=14, T2=6-0.4=5.6, T3=2-1=1
- **average turnaround time:**
  AT=(T1+T2+T3)/3=(14+5.6+1)/3=20.6/3 ≈6.87

Exercise 2

- 某系统采用基于动态优先级的剥夺式调度算法，并且优先数越大的进程其优先级越高。

  系统为所有新建进程赋予优先数0，当一个进程在就绪队列中等待CPU时，其优先数的变化速率为 α；进程获得CPU后开始执行，执行过程中，其优先数的变化速率为β。

  为参数 α 和 β 设置不同的值，则导致不同的调度算法。

问题：

a. 如果 β>α>0，则调度原则是什么？

b. 如果 α<β<0，则调度原则是什么？

解答a：如果 β>α>0，调度原则是FCFS

分析：

- 由于新建进程的优先数是0， 具有最低优先级。
- 当它在就绪队列中等待时，进程的优先数以速率 α 增加，即优先级不断提高，所以最早进入就绪队列的进程，其优先级也最高。
- 优先级最高的进程获得调度。
- 当进程在CPU上执行时，它的优先数以速率 β 增加，β>α，任何ready状态的进程不可能具有比正在 running 的进程更高的优先级，所以，running进程将一直占有CPU，直到它执行结束。
- 然后，在就绪队列中等待时间最长的进程被调度到CPU上执行。

所以，调度原则是 first-come first-served。

解答b：如果 $\alpha<\beta<0$，调度原则是LCFS

分析：

- 新建进程具有最高优先级。由于新建进程的优先数是0， 一旦它进入就绪队列，它的优先数就开始以速率 α 减少，就绪时间越长，其优先数越小，即优先级越低。
- 新建进程优先级最高，被调度到CPU上执行。
- **running** 状态的进程，其优先数以速率 β减少，$\alpha<\beta<0$，故任何**ready** 进程不可能具有比 **running** 进程更高的优先级。
- 新建进程抢占CPU。原来的执行进程进入就绪队列。
- 如果进程执行过程中，没有新建进程，则将一直占有CPU，直到它执行结束。然后，在就绪队列中等待时间最短的进程被调度到CPU上执行。

所以，调度原则是 **Last-Come First-Served**。

Exercise 3

- Consider the following set of processes P1, P2, P3 and P4. For 1≤ i ≤4, the arrival time of each Pi , the length of the CPU burst time of each process Pi, and the priority number for each Pi are given as below, and a smaller priority number implies a higher priority.

Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3

(1) Suppose that HRRN scheduling is employed,

Draw a Gantt chart illustrating the execution of these processes.

What are the average waiting time and the average turnaround time.

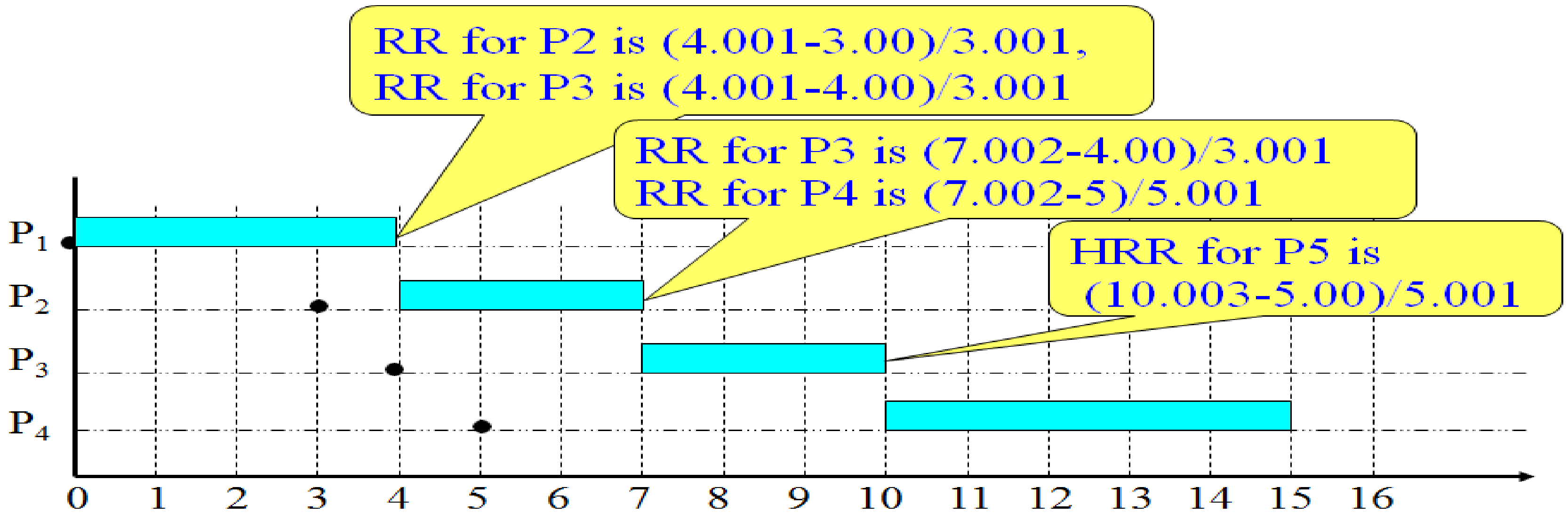
(2) Suppose that nonpreemptive priority scheduling is employed,

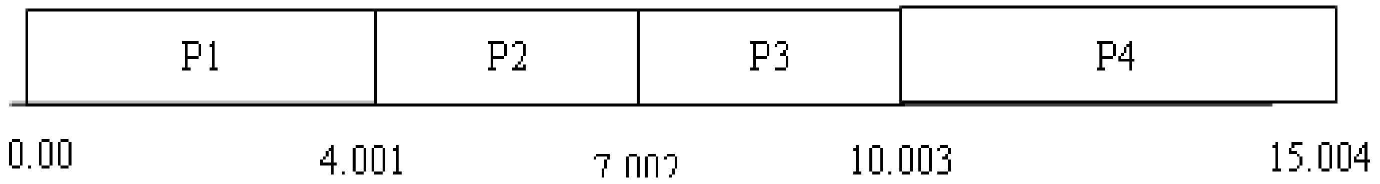
Draw a Gantt chart illustrating the execution of these processes.

What are the average waiting time and the average turnaround time.

Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3

Answer (1) HRRN

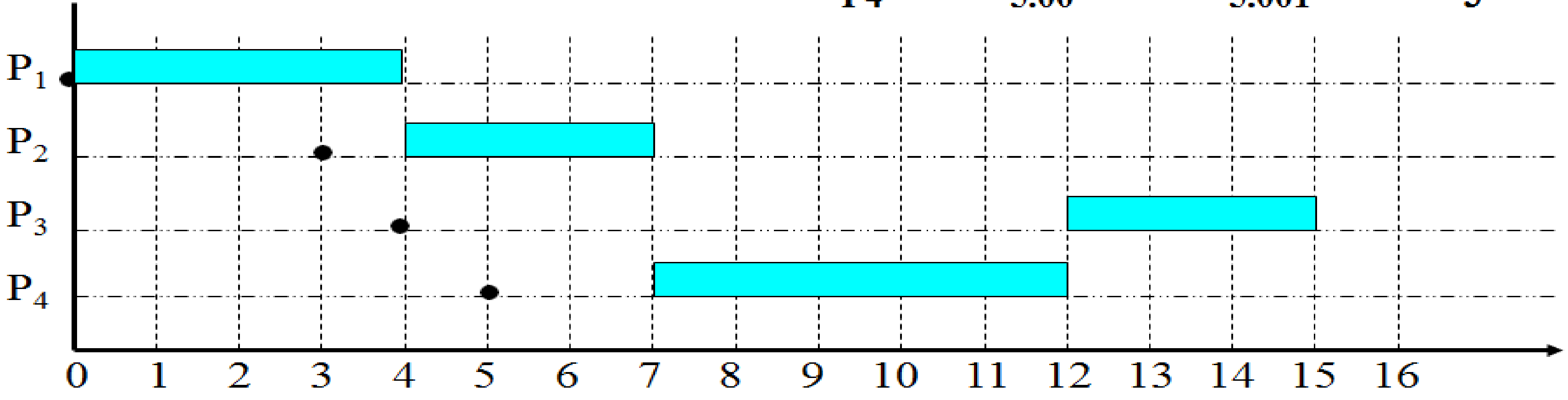


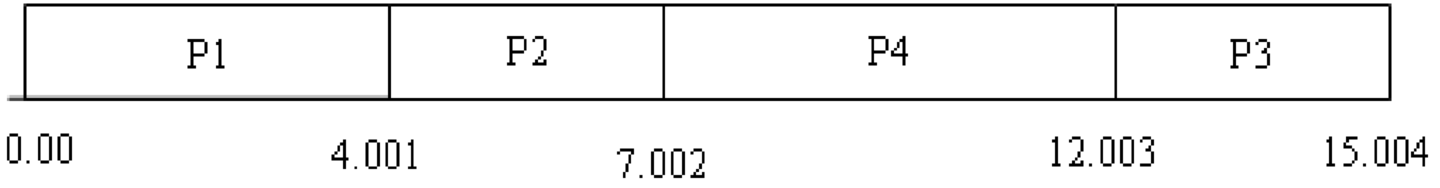


	Waiting time	Turnaround time
P1	0.00-0.00 = 0.00	4.001-0.00 = 4.001
P2	4.001-3.00 = 1.001	7.002-3.00 = 4.002
P3	7.002-4.00 = 3.002	10.003-4.00 = 6.003
P4	10.003-5.00 = 5.003	15.004-5.00 = 10.004
Average time	2.2515	6.0025

（2）Priority

Process	Arrival Time	Burst Time	Priority Number
P1	0.00	4.001	2
P2	3.00	3.001	1
P3	4.00	3.001	4
P4	5.00	5.001	3





	Waiting time	Turnaround time
P1	0.00-0.00 = 0.00	4.001-0.00 = 4.001
P2	4.001-3.00 = 1.001	7.002-3.00 = 4.002
P3	12.003-4.00 = 8.003	15.004-4.00 = 11.004
P4	7.002-5.00 = 2.002	125.003-5.00 = 7.003
Average time	2.7515	6.5025

Chapter 6   Process Synchronization

LI Wensheng, SCS, BUPT

Exercise 1

- 用信号量机制描述4*100米接力赛。

1）定义信号量如下：

S12=0;　S23=0　S34=0

2）四个运动员进程的代码结构如下：

```
P1:
    Begin
      跑100米;
      signal (S12)
    End
P2:
    Begin
      wait (S12);
      跑100米;
      signal (S23)
    End
P3:
    Begin
      wait (S23);
      跑100米;
      signal (S34)
    End
P4:
    Begin
      wait (S34);
      跑100米;
    End
```

Exercise 2

- 图书馆阅览室管理问题：

  某阅览室共有100个座位。

  读者进入阅览室之前，先看还有没有座位。

  若有，则先在登记本上登记(check in)，然后进入阅览室，离开阅览室之前还要注销登记(check out)。

  若阅览室已经没有空位，则读者等待。

  注意：每次只能有一个读者check in 或者check out。
- 要求：用信号量机制解决该问题。

假定：没有空位，则读者等待。

answer

1．定义如下两个信号量：
   Seat=100，用于描述图书馆中当前空闲的座位数量
   Mutex=1，用于读者互斥地使用登记表
2．读者进程的代码结构如下：

```
Begin
   wait(seat);
   wait(mutex);
   登记姓名、座位号；
   signal(mutex);
   阅读；
   wait(mutex);
   注销登记内容；
   signal(mutex);
   signal(seat)
End.
```

answer

如到来的读者发现没有空位了，则离开。

1．定义如下变量和信号量量：
 int Seat=100; // 用于描述图书馆中当前空闲的座位数量
 semaphore Mutex1=1; // 用于读者互斥地访问seat
 semaphore Mutex2=1; // 用于读者互斥地使用登记表

2．读者进程的代码结构如下：

```
Begin
   wait(mutex1);
   if (seat>0) { seat--;
                  signal(mutex1)};
   else {signal(mutex1);
         exit() };
   wait(mutex2);
   check in;
   signal(mutex2);
   阅读；
   wait(mutex2);
   check out;
   signal(mutex2);
   wait(mutex1);
   seat++;
   signal(mutex1);
End.
```

Exercise 3

- 有一个盘子，可容纳三个水果的。
  父亲向盘子中放苹果，母亲向盘子中放桔子，女儿从盘子中取苹果吃，儿子从盘子中取桔子吃。
  他们只能以互斥的方式操作盘子，并且每次只能向盘子中放一个水果，或从盘子中取一个水果。
- 请设计父亲、母亲、儿子、女儿4个进程，用信号量机制实现他们对盘子的正确操作。

要求：

1. 定义信号量，为信号量赋初值、并说明每个信号量的作用
2. 描述父亲、母亲、儿子、女儿4个进程代码结构

answer

1) Semaphores definition:

```
    empty=3;   mutex=1;
    apple=0;   orange=0;
```

2) Synchronous processes:

```
father: while (1) {
            wait (empty);
            wait(mutex);
            puts apple into plate;
            signal(mutex);
            signal (apple);
            };
mather: while (1) {
             wait (empty);
             wait(mutex);
             puts orange into plate;
             signal(mutex);
             signal (orange);
            };
```

```
son:  while (1) {
        wait (orange);
        wait(mutex);
        takes orange from plate;
        signal(mutex);
        signal (empty);
        eats orange;
        };
daughter: while (1) {
              wait (apple);
              wait(mutex);
              takes apple from
    plate;
              signal(mutex);
              signal (empty);
              eats apple;
              };
```

Ch6 练习2：

- 某应用有三个进程，数据采集进程C、数据处理进程M、和数据输出进程P。
  - C把采集到的数据放到buf1中
  - M从buf1中取数据进行处理，并把结果放入buf2中
  - P从buf2中取出结果打印输出

  考虑以下两种情况，用信号量机制实现进程C、M、和P之间的同步，分别给出进程C、M、和P的代码结构。（假设各进程对缓冲区的操作需要互斥）

  (a) buf1、buf2都只能保存一个数据

  (b) buf1可以保存m个数据、buf2可以保存n个数据

(a) (buf1、buf2都只能存放一个数据)

Semaphore： cm=1; mc=0; mp=1; pm=0;

Process C	Process M	Process P

Process C
{
  While (1) {
    采集数据;
    wait(cm);
    数据存入 buf1;
    signal(mc)
  }
}
	Process M
	{
	  While (1) {
	    wait(mc);
	    从buf1中取出数据;
	    signal(cm);
	    对数据进行处理;
	    wait(mp);
	    处理结果存入buf2;
	    signal(pm)
	  }
	}
		Process P
		{
		  While (1) {
		    wait(pm);
		    从buf2中取出结果;
		    signal(mp);
		    打印结果
		  }
		}

(b) buf1可以存放m个, buf2能存放n个

Semaphore： empty1=m; full1=0; empty2=n; full2=0; mutex1=1; mutex2=1

```
Process C
{
 While (1) {
   采集数据;
   wait(empty1);
   wait(mutex1);
   把数据存入Buf1;
   signal(mutex1);
   signal(full1)
 }
}
```

```
Process M
{
 While (1) {
   wait(full1);
   wait(mutex1);
   从Buf1中取出数据;
   signal(mutex1);
   signal(empty1);
   对数据进行处理;
   wait(empty2);
   wait(mutex2);
   把处理结果存入Buf2;
   signal(mutex2);
   signal(full2)
 }
}
```

```
Process P
{
 While (1) {
   wait(full2);
   wait(mutex2);
   从Buf2中取出结果;
   signal(mutex2);
   signal(empty2);
   打印结果
 }
}
```

exercise 6

- 某系统中有m个进程并发执行，分为A、B两组，他们共享文件F，A组进程对文件F进行写操作，B组进程对文件F进行只读操作。写操作需要互斥进行，读操作可以同时发生，即当有B组的某进程正在对文件F进行读操作时，若没有A组的进程提出写请求，则B组的其他进程可以同时对文件F进行读操作，如果有A组的进程提出写请求，则阻止B组的其他进程对文件F进行读操作的后续请求，等当前正在读文件的B组进程全部执行完毕，则立即让A组提出写请求的进程执行对文件F的写操作。

（1）定义信号量及变量，给出其初值，说明其作用

（2）写出A组和B组进程的代码结构

Answer:

```
int reader_count=0;
Semaphore
    mutex=1;
    RW_mutex=1;
    W_first=1;
writers(){
   while(1){
       wait(W_first);
       wait(RW_mutex);
       writing file F;
       signal(RW_mutex);
       signal(W_first);
   }
}
```

```
Readers(){
   While(1){
       wait(W_first);
       wait(mutex);
       reader_count++;
       if (reader_count==1)
           wait(RW_mutex);
       signal(mutex);
       signal(W_first);
       reading file F;
       wait(mutex);
       reader_count--;
       if (reader_count==0)
           signal(RW_mutex);
       signal(mutex);
   }
}
```

Answer 2:

Int R-num=0; W_num=0;
Semaphore wrt=1; W_first=1; mutex_w=1; mutex_R=1;

```
Writer(){
    Wait(mutex_w);
    W_num++;
    If (W_num==1)  wait(W_first);
    signal(mutex_w);
    Wait(wrt);
    写文件;
    signal(wrt);
    Wait(mutex_w);
    W_num--;
    If (W-num==0) signal(W_first);
    signal(mutex_w);
}
```

```
Reader(){
    Wait(W_first);
    Wait(mutex_R);
     R_num++;
     if (R_num==1)  wait(wrt);
     signal(mutex_R);
     signal(W_first);
     读文件;
     Wait(mutex_R);
     R_num--;
     If (R_num==0)  signal(wrt);
     signal(mutex_R);
}
```

Exercise 6.11

- The Sleeping-Barber Problem.
- A barbershop consists of a waiting room with n chairs and a barber room with one barber chair.
  If there are no customers to be served, the barber goes to sleep.
  If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop.
  If the barber is busy but chairs are available, then the customer sits in one of the free chairs.
  If the barber is asleep, the customer wakes up the barber.
- Write a program to coordinate the barber and customers.

Answer 1 to The Sleeping-Barber Problem

```
Semaphore
    customers=0; // 等待的顾客数，不包括正在理发的顾客
    barber=1;  // 理发师状态，1-闲，
                          用于理发师进程和顾客进程之间的同步
    mutex=1;   // 控制对wait的互斥访问
int waiing=0; // 等待的顾客数，包括正在理发的顾客
```

```
void barber ( ) {
    while (1)  {
        wait (customers);
        givehaircut ();
        signal (barber);
    }
}
```

Answer 1 to The Sleeping-Barber Problem(cont.)

```
void customer () {
    wait (mutex);
    if (waiting==n+1) {  signal (mutex);  exit();  }
    waiting++;
    signal (customers);
    signal (mutex);
    wait (barber);
    receiveHaircut();
    wait (mutex);
    waiting--;
    signal (mutex);
}
```

Answer 2 to The Sleeping-Barber Problem

```
Semaphore
    customers=0; // 用于理发师进程和顾客进程之间的同步
    barber=0;  // 用于顾客进程之间的同步
    mutex=1;   // 控制对wait的互斥访问
int waiing=0; // 坐在椅子上等待的顾客数
```

```
void barber ( ) {
    while (1)  {
        wait (customers);
        givehaircut ();
    }
}
```

Answer 2 to The Sleeping-Barber Problem(cont.)

```
void customer () {
    wait (mutex);
    if (waiting==n+1) { signal (mutex);  exit(); }
    waiting++;
    if (waiting>1) { take a chair;  signal(customers);
                          signal (mutex);   wait(barber); }
    else {  signal (mutex);
            signal (customers); }  // 唤醒理发师
    receiveHaircut();
    wait (mutex);
    waiting--;
    if (waiting>0) { signal(barber); }
    signal (mutex);
}
```

Answer 3 to The Sleeping-Barber Problem

```
Semaphore customers=0;
        barber=0; mutex=1;
int  waiing=0;
```

```
void barber ( ) {
    while (1)    {
        wait(customers);
        wait(mutex);
        waiting--;
        signal(mutex);
        signal(barber);
        givehaircut ();
    }
}
```

```
void customer ( ) {
    while (1)    {
        wait(mutex);
        if (waiting<n) {
            waiting++;
            signal(mutex);
            signal(customers);
            wait(barber);
            gethaircut ();  }
        else signal(mutex);
    }
}
```

exercise

- 某理发店有3位理发师，3把理发椅，n把供等候理发的顾客坐的椅子。如果没有顾客到来，理发师们便在理发椅上睡觉。当顾客到来时，必须先唤醒一个理发师，如果理发师们正在理发时又有顾客到来，则如果有空椅子，则顾客坐下休息排队等待，如果没有空椅子了，则顾客离开。
- 请使用信号量机制解决理发师进程和顾客进程之间的同步/互斥问题。
- **Semaphores**
  - **customers=0; // 正在等待的顾客数，包括正在理发的**
  - **barbers=3; // 空闲的理发师数**
  - **mutex=1; // 控制对waiting的互斥访问**
- **Variable**
  - **int waiing=0; // 坐在椅子上等待的顾客数**

Answer

```
void barber() {
  while(true){
      wait(customers);
      wait(mutex);
      waiting--;
      signal(mutex);
      cuthair();
      signal(barbers);
  }
}
```

```
void custumer (){
  while(true){
      wait(mutex);
      if (waiting<n) {
          waiting++;
          signal(mutex);
          signal(customers);
          wait(barbers);
          gethaircut();
      }
      else signal(mutex);
  }
}
```