第3章 进程管理 习题参考答案

1、程序并发执行时与顺序执行有哪些不同？

解答：程序并发执行与顺序执行的不同点有：（1）多个程序顺序执行时，每个程序都可以一口气连续运行完成，中间不需要暂停；但多个程序并发运行时，由于资源的限制和竞争，很多程序都很难做到一口气连续运行完成，中间不可避免的会出现暂停，即是以走走停停的方式进行的；（2）程序顺序执行时具有封闭性特征，当前运行程序独占系统所有资源，因此程序的执行具有可再现性特征。多个程序并发运行时，共享系统中的资源，程序的执行具有不可再现性特征。

2、什么是进程？进程有哪些特征？简述进程与程序的区别和联系。

解答：进程定义：进程是具有一定独立功能的程序关于某个数据集合的一次运行过程，是系统进行资源分配和调度的一个独立单位。

进程相较于程序来说，具有如下一些特征：动态性、并发性、独立性、异步性。

进程与程序的区别：

①程序是静态概念，本身可以作为软件资源长期保存；而进程是程序的一次执行过程，是动态的，有一定的生命期。

②进程是一个能独立运行的单位，是系统进行资源分配和调度的基本单位，能与其它进程并发执行，而程序因其自身不能描述并发运行过程中的动态信息，所以无法参与并发运行。

③各进程在并发执行过程中存在异步性特征，而程序因为只能顺序执行，故没有这个特征。

进程与程序的联系：进程是程序的一次执行过程，程序是进程映像的构成部分；多道环境中，必须为程序创建进程后才能得到运行；进程与程序之间是多对多关系。

3、进程有哪些基本状态？试说明引起进程状态转换的典型原因。

解答：进程的基本状态有：就绪态、执行态、阻塞态（或等待态）。

引起进程状态转换的典型原因有：

（1） 就绪→运行：典型原因是进程调度

（2） 运行→就绪：在分时系统中，当前运行进程如果时间片用完将暂停运行；在抢占调度方式中，如有更高优先级的就绪进程到达，将迫使当前运行进程让出 CPU。于是当前运行进程从运行状态转换成就绪状态，重新进入就绪队列排队，等待下一次CPU 调度。

（3） 运行→阻塞：当前运行进程因需要等待某事件的发生而无法继续运行，如等待 I/O 操作的完成，或未能申请到所需的系统资源，或等待其他进程发来信号等，则进程让出 CPU，转变为阻塞状态。

（4） 阻塞→就绪：处于阻塞状态的进程，所等待的事件已经发生，如 I/O 操作已完成或获得了所需的资源等，则进程转变为就绪状态等待 CPU 调度。

4、进程控制块的作用是什么？在进程控制块中主要包括哪些信息？

解答：PCB 的作用：PCB 用于保存系统管理和控制进程所需要的全部信息，是进程动态特性的集中反映，是进程存在的唯一标志，系统是根据 PCB 来感知进程的存在的。通过 PCB，使得原来不能并发执行的程序，成为能并发执行的进程。

PCB 中包括的主要信息有：（1）进程标识信息：进程标识符、用户标识符、家族关系。

（2） 进程调度信息：进程状态、进程优先级、等待事件、其他余进程调度相关的信息。
（3） 进程现场信息：CPU 中通用寄存器内容、指令计数器的值、PSW、栈指针等。（4）进程控制信息：进程的程序和数据的地址、进程同步和通信信息、进程资源管理信息、相关链接指针等。

5、进程创建、进程撤销、进程阻塞、进程唤醒几个原语主要应完成哪些工作？

解答：进程创建主要完成的工作有：（1）为新进程申请一个尚未被使用的 pid 和一个空白 PCB。（2）为新进程分配必要的资源，比如建立进程地址空间，为其分配内存空间设置堆栈、存放程序、数据，此外还有文件、I/O 设备以及初始时间片长度等。这些资源或者从系统分配，或者从父进程继承或共享。（3）初始化新进程的 PCB，填入相关信息，如 pid，进程名、父进程标识符、处理器初始状态、进程状态、进程优先级、进程对应程序的入口地址、进程同步信息、进程地址空间信息、资源分配情况等。（4）最后，将新进程状态置为就绪状态，并插入就绪队列，等待 CPU 调度。

撤销进程主要完成的工作有：（1）根据被撤销进程的标识符（pid），从系统的 PCB 集合中找到该进程的 PCB，读出其状态，若处于运行状态，则立即终止运行，并将系统调度标志置为“真”，以便撤销该进程后系统立即重新进行 CPU 调度。（2）若被撤销进程还有子进程， 则应该进一步撤销其所有子进程，或者为这些子进程临时指定新的父进程，比如 Linux 会指定其同组进程或 init 进程为其子进程的临时父进程。（3）回收被撤销进程所拥有的全部系统资源：或者归还给系统，如内存、I/O 设备等；或者归还给父进程，如从父进程继承来的未使用完的时间片等。（4）最后，将被撤销进程的 PCB 从所在队列移出，待以后父进程或系统从中收集相关信息后，最终释放它，则该进程就永远从系统中消失了。

进程阻塞主要完成的工作有：首先立即停止该进程的执行，保存其 CPU 现场信息到 PCB 中，并将 PCB 中的进程状态由“运行”改为“阻塞”，然后插入相应的阻塞队列。最后转进程调度程序重新进行调度，将CPU 分配给另一就绪进程运行。

进程唤醒主要完成的工作有：首先确定要唤醒的进程：查找该事件的阻塞队列，或者唤醒队首进程，或者唤醒指定进程，或者唤醒队列中所有进程，不同的操作系统具体实现机制会有区别；然后将被唤醒进程的 PCB 从所在阻塞队列中移出，并将其 PCB 中的进程状态由“阻塞”改为“就绪”，最后插入就绪队列中等待 CPU 调度。

6、简述进程互斥与同步的概念。同步机制应遵循的四个准则是什么？

解答：所谓进程互斥，是指多个进程在共享临界资源时，必须以互斥方式共享，即当一个临界资源正在被某个进程访问时，其他进程不允许同时访问该资源，只能等待前一进程访问完毕释放资源后，后一进程才能接着访问。

所谓进程同步是指相互合作的进程需按一定的先后顺序执行，以顺利完成某共同任务。具体说，这些进程之间需要交换一定的信息，当某进程未获得其合作进程发来的信息之前，该进程等待，直到接收到相关信息时才继续执行，从而保证诸进程正确的协调运行。

同步机制应遵循的四个准则是：空闲让进、忙则等待、有限等待、让权等待。

7、如何利用硬件方法解决进程互斥问题？

解答：解决进程互斥问题的硬件机制主要有禁止中断和使用专用机器指令两种方式。

（1） 禁止中断：让进程进入临界区之前禁止一切中断，从而使进程切换不可能发生；在进程退出临界区时再开放中断。

（2） 使用 TSL 指令：首先为临界资源申明一个全局 boolean 变量 lock，表示资源的两种状态：TRUE 表示资源被占用，FALSE 表示资源空闲，初始化为 FALSE；进程 P_i 进入临界区之

前必须先执行 TSL 指令，只有检测到 lock 值为 FALSE 时（临界资源空闲）才能进入临界区执行；执行完临界区后必须将 lock 置为 FALSE，表示已退出临界区。

（3） 使用 Swap 指令：应为每个临界资源设置一个全局布尔变量 lock，初值为 FALSE，表示资源空闲；在每个进程中再设置一个局部布尔变量 key，用于与 lock 交换信息，进程进入临界区之前先利用 Swap 指令交换 lock 与 key 的内容，然后检查 key 的值，若为 FALSE 表示临界资源空闲，进程可进入临界区执行，否则需等待直到资源空闲。

8、系统中只有一台打印机，有三个进程在运行中都需要使用打印机进行打印输出，问：这三个进程间有什么样的制约关系？试用信号量描述这种关系。

解答：三个进程之间是互斥关系。算法描述如下：

```
semaphore mutex;
void main() {
  mutex=1;
parbegin(Pi());
}
```

```
void  Pi(i=1,2,3 ) {
    ……
    wait(mutex);
       printer();
    signal(mutex);
     ……
 }
```

9、在生产者-消费者问题中，如果缺少了 signal(full)或 signal(empty)，或者将 wait(full)与 wait(mutex)互换位置，或者将 signal(full)与 signal(mutex)互换位置，分别会有什么后果？

解答：如果缺少了 signal(full)，则生产者向缓冲区投放消息后，并不对 full 加 1，full 的值永远为 0，消费者执行 wait(full)时将阻塞，当生产者在缓冲区中放满消息后，后续投放消息的生产者都将阻塞，最后全部进入死锁状态。

如果缺少了 signal(empty)，则消费者取出消息后，并不对 empty 加 1，当生产者连续投放 n 个消息后，empty 变为 0，则后续生产者投放消息将失败，进入阻塞状态，之后消费者在取消息时也将进入阻塞状态，最后全部进入死锁状态。

如果将 wait(full)与 wait(mutex)互换位置，当某个时刻 mutex=1，full=0，empty=n 时，如果系统首先调度消费者执行，则它执行 wait(mutex)时成功，mutex 变为 0，再进一步执行 wait(full)时将阻塞；而生产者执行 wait(mutex)时将失败阻塞，最后进入死锁状态。

如果将 signal(full)与 signal(mutex)互换位置，由于没有阻塞情况，进程将顺利往下执行。只不过释放临界资源的时间稍晚，可能会造成临界资源的利用率下降。

10、假设有三个并发进程 A，B，C，其中 A 负责从输入设备上读入信息并传送给B，B 将信息加工后传送给 C，C 负责将信息打印输出。写出下列条件的并发程序：

（1） 进程 A、B 共享一个单缓冲区，进程 B、C 共享另一个单缓冲区。

（2） 进程 A、B 共享一个由 m 个缓冲区组成的缓冲池，进程 B、C 共享另一个由 n 个缓冲区组成的缓冲池。

解答：（1）分析：由于是单缓冲区，所以各进程间可以用两个同步关系来解决。

三个进程代码如下：

```
empty1，empty2，ful1，full2：semaphore
main() {
  empty1，empty2=1，1;
  ful,1，full2=0,0;
  parbegin(A,B,C)
```

```
void A ()
  {do{
    get data
    wait(empty1);
      put data to buffer1;
```

}
 	
	signal(full1);
	}while(1)
	}

void B ()
{do{
wait(full1);
get data from buffer1;
signal(empty1);
wait(empty2);
put data to buffer2;
signal(full2);
}while(1)
}
	void C ()
	{do{
	wait(full2);
	get data from buffer2;
	signal(empty2);
	}while(1)
	}

（2）由于多个进程共享的缓冲池由多个缓冲区，所以进程间必须互斥和同步。

三个进程代码如下：

empty1，empty2，ful1，full2,mutex1,mutex2：semaphore;
mutex1,mutex2：semaphore
main() {
empty1，empty2=m，n;
ful,1，full2=0,0;
mutex1,mutex2=1,1
parbegin(A,B,C)
}
	void A ()
	{do{
	get data
	wait(empty1);
	wait(mutex1);
	put data to buffer1;
	signal(mutex1);
	signal(full1);
	}while(1)
	}

void B ()
{do{
wait(full1);
wait(mutex1);
get data from buffer1;
signal(mutex1);
signal(empty1);
wait(empty2);
wait(mutex2);
put data to buffer2;
signal(mutex2);
signal(full2);
}while(1)
}
	void C ()
	{do{
	wait(full2);
	wait(mutex2);
	get data from buffer2;
	signal(mutex2);
	signal(empty2);
	}while(1)
	}

11、 病人在医院中的就医流程通常是：若没有病人，则叫号系统睡眠，医生休息；当有病人报到就诊时，叫号系统按一定规则叫号，自动分配病人给医生；医生收到叫号信号后给病人看病。病人到医院后首先通过自助挂号终端挂号，自助挂号终端一段时间只能一个人操作；

拿到挂号凭证后到门诊大厅的一台报到机上扫描挂号凭证完成就医报到操作，报到机一段时间也只能一个人操作；然后坐在大厅里等待就医叫号；病人报到成功后会唤醒叫号系统，叫号系统为病人分配医生，同时唤醒医生工作，完成叫号工作后等待下一个病人报到；当病人收到就诊通知时，就到诊疗室看病，医生开出药方并提交后系统自动完成缴费操作；已完成缴费的病人到取药柜台将药方交给发药员，取药柜台一段时间只能被一个病人使用；发药员根据药方把药交给病人，病人离开医院。请用信号量实现医院就诊流程中病人、叫号系统、医生、发药员之间的工作流程管理。

解答：分析：有同步关系的进程：医生、病人、发药员

病人之间需要互斥使用的资源：自助挂号终端，报到机，取药柜台

```
patient，call_patient，finish：semaphore;
mutex1,mutex2,mutex3：semaphore
nurse,medicine:semaphore;
main() {
   patient，call_patient ,finish=0;
   mutex1,mutex2,mutex3=1,1,1
   nurse,medicine=0,0
   parbegin(doctor,patient-i,nurce)
}
```

```
void doctor()
   {do{
      wait(patient);
      signal(call_patient);
      给病人看病;
        开药方；
      signal(finish)
   }while(1)
 }
```

```
void nurce ()
    {do{
      wait(nurse);
        给病人拿药;
      signal(medicine);
   }while(1)
 }
```

```
void patient-i() {
    wait(mutex1)；    //申请挂号终端
    挂号
    signal(mutex1);
    wait(mutex2)；    //申请报到机
    扫描报到
    signal(mutex2);
    signal(patient);
    wait(call_patient);
    看病
    wait(finish);
    wait(mutex3); //申请取药柜台
    将药方交给发药员
    signal(nurse);
    wait(medicine);
    拿药
    signal(mutex3);
    离开医院
 }
```

12、桌上有一个盘子，只能放一只水果。请用信号量分别实现以下几个同步问题：

（1） 爸爸放苹果，妈妈放桔子，儿子只吃桔子，女儿只吃苹果。

（2） 爸爸放苹果，妈妈放桔子，儿子吃桔子、苹果。

（3） 爸爸放苹果或橘子，儿子只吃桔子，女儿只吃苹果。

解答：（1）

```
empty，full：semaphore;
main() {
   empty，full =1,0;
   parbegin(father,mother,son,daughter)
```

```
void son ()
   {do{
      wait(full);
         get orange or apple;
```

<table>
<tr><td>}</td><td>signal(empty);
}while(1)
}</td></tr>
<tr><td>void father ()
{do{
wait(empty);
put orange;
signal(full);
}while(1)
}</td><td>Void mother ()
{do{
wait(empty);
put apple;
signal(full);
}while(1)
}</td></tr>
</table>

（2）

<table>
<tr><td>empty， orange， apple： semaphore;
main() {
empty， orange， apple =1,0,0;
parbegin(father,mother,son,daughter)
}</td><td>void son ()
{do{
wait(orange);
get orange;
signal(empty);
}while(1)
}</td><td>void daughter ()
{do{
wait(apple);
get apple;
signal(empty);
}while(1)
}</td></tr>
<tr><td>void father ()
{do{
wait(empty);
put orange;
signal(orange);
}while(1)
}</td><td colspan="2">Void mother ()
{do{
wait(empty);
put apple;
signal(apple);
}while(1)
}</td></tr>
</table>

（3）

<table>
<tr><td>empty， orange， apple： semaphore;
main() {
empty， orange， apple =1,0,0;
parbegin(father,son,daughter)
}</td><td>void son ()
{do{
wait(orange);
get orange;
signal(empty);
}while(1)
}</td></tr>
<tr><td>void father ()
{do{
wait(empty);
put orange or apple;
if(orange)
signal(orange);
else
signal(apple);
}while(1)</td><td>void daughter ()
{do{
wait(apple);
get apple;
signal(empty);
}while(1)
}</td></tr>
</table>

}	

13、图书馆为师生提供论文查重服务。在查重室有一个技术员专门进行查重工作，有一张椅子给正在查重的师生坐（称为查重椅），有 10 张椅子供师生坐下等待（称为等候椅）。如果没有查重任务，则技术员休息；当有查重任务时马上进行查重工作。当师生来到查重室时，如果有空的等候椅就坐下来，如果查重椅也空着，则离开等候椅坐到查重椅上并请技术员进行查重；如果没有空的等候椅就在查重室外等待，直到有空的等候椅再进来坐下；查重完成后从查重椅上站起并离开查重室。试用信号量实现师生与查重技术员之间的同步关系。

解答：Semaphore　chair1=10；空的等候椅数量

Semaphore　ready=0；表示查重椅上是否有师生在等待查重，并用作阻塞技术人员进程

Semaphore　chair2=1；查重椅是否空闲

Semaphore　finished=0；当前师生的查重工作是否已经完成

i
void main() {
parbegin(Technician,Tea_Stu-i);
}
void　Technician ()
{do {
wait(ready);
Duplicate checking;
signal（finished）
}while(1);
}
	void　Tea_Stu -i ()
	{wait(chair1);
	wait(chair2);
	sit_in_chair2;
	signal(chair1);
	signal(ready);
	get Duplicate checking;
	wait(finished);
	stand_from_chair2;
	signal(chair2);
	}

14、在一个机票订购系统中，共有三个工作进程：P1，P2 和 P3。P1 负责某个航班的机票信息查询处理（只读）；P2 负责某个航班的机票退票处理（只写）；P3 负责某个航班的机票售票处理（先读后写）。当一个进程正在修改某航班的机票信息时，其他进程不能访问（即不能读/写）；但多个进程同时查询某航班机票信息是允许的。请使用信号量实现 P1、P2、P3 三个进程间的同步互斥关系，要求：①正常运行时不产生死锁；②机票信息的访问效率高（即系统并发度高）。

解答：

wmutex，rmutex：semaphore;
readcount，ticketcount ： int;
main() {
wmutex，rmutex =1,1;
readcount，ticketcount =0
parbegin(p1,p2,p3)
}
	void p2()
	{do{
	wait(wmutex);
	退票处理
	signal(wmutex);
	}while(1)
	}

void p1 ()
{do{
wait(readcount);
if(readcount==0) wait(wmutex);
	void p3()
	{do{
	wait(wmutex);
	ticketcount=某航班机票数量

readcount++;
signal(readcount);
查询机票信息;
wait(readcount);
readcount--;
if(readcount==0) signal(wmutex);
signal(readcount);
}while(1)
}
	if(ticketcount>0)
	售票，机票数量减 1；
	else
	显示“机票已售完”信息；
	endif
	signal(wmutex);
	}while(1)
	}

15、系统中有多个生产者进程和多个消费者进程，共享一个能存放 500 件产品的环形缓冲区（开始时为空）。当缓冲区未满时，生产者进程可以放入其生产的一件产品，否则等待；当缓冲区未空时，消费者进程可以从缓冲区取走一件产品，否则等待。当一个消费者进程获得取出产品的机会时，必须连续取出 3 件产品后，其他消费者才可以取产品。请写出该问题的同步算法。（2014 年计算机联考真题）

解答：

mutex1,mutex2,empty,full：semaphore;
int i,j;
main() {
empty，full =500,0;
mutex1,mutex2=1,1
i,j=0
parbegin(producer-m,consumer-n)
}
	void producer-n ()
	{do{
	wait(mutex2);
	for(int
	k=0;k<=3;k++){wait(
	full); wait(mutex1);
	data=buf(j);
	j=(j+1)%500
	signal(mutex1);
	signal(empty);
	signal(mutex2);
	}while(1)
	}

void producer-m ()
{do{
wait(empty);
wait(mutex1);
buf(i)=(i+1)%500;
signal(mutex1);
signal(full);
}while(1)
}	

16、有一个阅览室，共有 100 个座位。读者进入阅览室时必须在入口处进行登记；离开阅览室时必须在出口处进行注销，入口和出口都不允许多人同时使用。请为阅览室设计一个管理读者进入/离开阅览室的操作流程。

解答：

mutex1,mutex2,S：semaphore;
main() {
mutex1，rmutex2 =1,1;
S =100
parbegin(pi)
}
	void pi()
	{do{
	wait(S)
	wait(mutex1);
	入口登记
	signal(mutex1);

	阅览
	wait(mutex2);
	出口注销
	signal(mutex2);
	signal(S);
	}while(1)
	}

17、某工厂有一个可以存放设备的仓库，总共可以存放 10 台设备。生产的每一台设备都必须入库，销售部门可从仓库提取设备供应客户。设备的入库和出库都必须借助运输工具。现只有一台运输工具，每次只能运输一台设备。请设计一个能协调工作的自动调度管理系统。

解答：

生产者-消费者问题：

Mutex：semaphore=1，互斥使用运输工具Empty：

semaphore=10，仓库还能存放的设备数量Full：

semaphore=0，仓库中的设备数量

void Producer-i (i=1,2,…,m)
{do{
produce an item nextp；
wait(empty);
wait(mutex)；
buffer(in) = nextp;
in = (in +1) mod k;
signal(mutex)；
signal(full);
}while (1);
}
	void Consumer-j (j=1,2,…,n)
	{do{
	wait(full);
	wait(mutex)；
	nextc = buffer(out);
	out = (out +1) mod k;
	signal(mutex)；
	signal(empty);
	consume the item in nextc;
	}while (1);
	}

18、设有两个生产者进程 A、B 和一个销售者进程 C，他们共享一个无限大的仓库，生产者每次循环生产一个产品，然后入库供销售者销售；销售者每次循环从仓库中取出一个产品销售。如果不允许同时入库，也不允许边入库边出库。请回答以下问题：

（1）对仓库中 A、B 两产品的库存件数无要求，但要求生产 A 产品和 B 产品的件数满足以下关系：

-n≤生产 A 的件数－生产 B 的件数≤m

其中 n，m 是正整数，请用信号量机制写出 A，B，C 三个进程的工作流程。

（2）对生产 A 产品和 B 产品的件数无要求，但要求仓库中 A、B 两产品的库存件数满足以下关系：

-n≤A 产品件数－B 产品件数≤m

其中 n，m 是正整数，请用信号量机制写出 A，B，C 三个进程的工作流程。

解答：（1）

Mutex,Sa,Sb,S: Semaphore；
main() {
Mutex =1; S=0;
	Void PC() {
	While(1) {
	wait(S);

```
   Sa=M;          Sb=N;
   parbegin(PA,PB,PC)
}
```

```
           wait(Mutex);
            出库一个商品；
           signal(Mutex);
       }
    };
```

```
Void    PA()
      { While(
      1)
           wait(Sa);
           生产 A
           signal(Sb);
           wait(Mutex);
           产品 A 入库；
           signal(Mutex);
           signal(S);
           }
    }
```

```
Void    PB() {
      While(1)
           wait(Sb);
           生产 B
           signal(Sb);
           wait(Mutex);
           产品 B 入库；
           signal(Mutex);
           signal(S);
           }
    }
```

（2）

```
 Mutex,Sa,Sb,S: Semaphore；
 countA,countB : semaphore ;
 // A、B 的数量
 diff : int ; //P C 销售 A、B 的差值：
 // diff=-n 时，只能取 A
 //diff=m 时，只能取 B
 //-n<diff<m 时，既可以取A，也可以取 B
main() {
   Mutex =1;
   countA,countB,S=0;
   Sa=M;   Sb=N;
   parbegin(PA,PB,PC)
}
```

```
void PC() {
    while(1) {
          wait(S);
          if(diff<=-n)
            { wait(count
            A)
            wait(Mutex);
            出库一个A 商品；
            signal(Mutex);
            diff++;
           }
          else   if(diff>=m)
            {wait(countB)
            ;wait(Mutex);
            出库一个B 商品；
            signal(Mutex);
            diff--;   }
          else {   //-n<diff<m，可取 A 或 B
            wait(Mutex);
            出库一个A 商品或 B；
            signal(Mutex);
            if(出库 A) {
                wait(countA);
                diff++;   }
            else {   //出库 B
            wait(countB);
            diff--;}
```

	} } }

void PA() {
while(1) {
生产 A
wait(Sa);
wait(Mutex);
产品 A 入库；
signal(Mutex);
signal(Sb);
signal(countA);
signal(S);
}
}
	void PB() {
	while(1) {
	生产 B
	wait(Sb);
	wait(Mutex);
	产品 B 入库；
	signal(Mutex);
	signal(Sa);
	signal(countB);
	signal(S);
	}
	}

19、一个管程由哪几部分组成？管程中引入条件变量有什么用处？

解答：从管程定义可知，管程是一个软件模块，由 4 部分组成：① 管程的名字；②局部于管程的共享数据结构的说明；③对该数据结构进行操作的一组过程，每个过程完成进程对上述数据结构的某种操作；④对局部于管程的共享数据结构进行初始化的代码。

条件变量用于实现进程间的同步关系，一个条件变量代表了进程继续执行所需要的一个条件，比如生产者若想成功投放产品，必须满足有空缓冲区的条件；一个条件变量通常对应一个等待队列，当条件不满足时，进程将进入这个等待队列阻塞，直到条件满足被唤醒。

20、高级调度与低级调度的主要任务是什么？为什么要引入中级调度？

解答：高级调度又称作业调度或长程调度，其任务是根据系统的资源情况，按照一定的原则从外存上的后备队列中选择若干个作业调入内存，为他们创建进程，分配必要的资源，如内存、外设等，并将新创建的进程插入就绪队列，准备执行；此外，当作业执行完毕时，还要负责回收其所占据的资源。

低级调度通常称为进程调度，有时也称短程调度，其任务是决定就绪队列中的哪个进程获得处理器，然后由分派程序把处理器分配给该进程，并为它恢复运行现场，让其运行。

引入中级调度的主要目的是为了提高内存利用率和系统吞吐量。当内存紧张时，就将那些暂时不能运行的进程（如处于阻塞状态的进程）换出到外存，回收其内存空间给别的进程，通常称此时的进程状态为挂起状态；当内存空间较充裕时，又从外存选择若干具备运行条件的挂起进程换入到内存。中级调度其实就是存储器管理中的对换功能，故又称为对换调度。

21、选择进程调度算法时需要考虑哪些因素？评价一个调度系统性能的指标有哪些？

解答：选择进程调度算法时需要考虑的因素有：系统设计目标、调度的公平性、资源的均衡利用、合理的系统开销。

评价一个调度系统性能的指标有：CPU 的利用率、系统吞吐量、周转时间和带权周转时间、响应时间、对截止时间的保证。

22、请分别为批处理系统、分时系统和实时系统选择合适的进程调度算法，并说明理由。

解答：批处理系统：其性能评价指标是系统吞吐量和系统资源利用率以及平均周转时间，

所以可以选择非抢占式的优先权调度算法、短作业优先调度算法。分时系统的性能评价指标主要是系统对用户的响应时间，选择时间片轮转算法、多级反馈队列调度算法。实时系统的性能评价指标主要是对截止时间的保证，因此可选择抢占式的优先权调度算法。

23、什么是静态优先级和动态优先级？各有何优缺点？如果让你为一个调度系统设计动态优先级机制，你会怎样设计？你的设计对系统的调度性能有哪些改善？

解答：静态优先级。是在进程创建时根据进程的类型、运行时间长短、对资源的需求（如要求内存大小、需要的外部设备种类及数量、需打开的文件数等)以及用户的要求而确定的， 在进程的整个运行期间保持不变。静态优先级实现简单，系统开销小，但可能出现“饥饿”现象，比如当系统负荷较重时，不断有高优先级就绪进程到达，会导致低优先级进程长时间得不得 CPU 调度

动态优先级。通常也是在创建进程时为其赋予一个初始优先级（比如可简单地直接从父进程处继承），以后在进程的运行过程中随着进程特性的变化，按照一定的原则不断修改其优先级，可防止一个进程长期垄断 CPU 而导致低优先级进程产生“饥饿”现象，但实现的复杂度变高。

动态优先级机制的设计：

思路 1：随着进程在就绪队列中等待时间的增长，可提高进程的优先级；随着进程连续占用 CPU 时间的增长，比如，用完一个时间片时，可降低其优先级，可防止一个进程长期垄断 CPU 而导致低优先级进程产生“饥饿”现象。

思路 2：随着进程剩余运行时间的减少，提升其优先级，可以让这些进程加速完成，提高系统的吞吐量，减小平均周转时间。

24、试给某系统设计进程调度的解决方案，期望能满足以下性能要求：①对各进程有合理的响应时间；②有较好的外部设备利用率；③能适当照顾计算量大的进程；④系统调度开销（调度算法运行时间开销）与系统中就绪进程的数量无关；⑤紧迫型任务能得到及时处理。
请详细说明你的设计方案是如何满足上述性能要求的。

分析：

① 设置 3 个就绪队列：高中低
② 对各进程有合理的响应时间：引入时间片
③ 有较好的外部设备利用率：对完成 I/O 操作的被唤醒的进程放入中等优先级的就绪队列，设置较短的时间片，如 20-50ms
④ 能适当照顾计算量大的进程：设置低优先级就绪队列，计算量大的进程放入这个这个队列，设置较长的时间片，比如 100-300ms
⑤ 系统调度开销（调度算法运行时间开销）与系统中就绪进程的数量无关：设置一个 1B 的位示图
⑥ 紧迫型任务能得到及时处理：进行优先级抢占，紧迫性任务位于高优先级就绪队列，且抢占低优先级就绪进程

25、设有五个进程，它们到达就绪队列的时刻和运行时间如表 3-23 所示。若分别采用先来先服务调度算法、短进程优先调度算法和高响应比优先调度算法，试给出各进程的调度顺序并计算平均周转时间。

表 3-23 进程调度信息表

进程	到达时刻	运行时间
P1	10.1	0.3

P2	10.3	0.9
P3	10.4	0.5
P4	10.5	0.1
P5	10.8	0.4

解答：

（1） 先来先服务算法：

调度顺序	进程	到达时刻	运行时间	开始时间	完成时间	周转时间
1	P1	10.1	0.3	10.1	10.4	0.3
2	P2	10.3	0.9	10.4	11.3	1.0
3	P3	10.4	0.5	11.3	11.8	1.4
4	P4	10.5	0.1	11.8	11.9	1.4
5	P5	10.8	0.4	11.9	12.3	1.5
平均周转时间：T＝（0.3 + 1.0 + 1.4 + 1.4 + 1.5）/ 5 = 1.12						

（2）短进程优先：（非抢占式）

调度顺序	进程	到达时刻	运行时间	开始时间	完成时间	周转时间
1	P1	10.1	0.3	10.1	10.4	0.3
2	P3	10.4	0.5	10.4	10.9	0.5
3	P4	10.5	0.1	10.9	11.0	0.5
4	P5	10.8	0.4	11.0	11.4	0.6
5	P2	10.3	0.9	11.4	12.3	2.0
平均周转时间：T＝（0.3 + 0.5 + 0.5 + 0.6 + 2.0）/ 5 = 0.78						

短进程优先（SPF）（剩余时间抢占）

调度顺序	进程	到达时刻	要求运行时间	开始时间	暂停时间	剩余时间	完成时间	周转时间
1	P1	10.1	0.3	10.1	10.4	0	10.4	0.3
2	P3	10.4	0.5	10.4	10.5	0.4	未	
3	P4	10.5	0.1	10.5	10.6	0	10.6	0.1
4	P3		0.4	10.6	11.0	0	11.0	0.6
5	P5	10.8	0.4	11.0	11.4	0	11.4	0.6
6	P2	10.3	0.9	11.4	12.3	0	12.3	2.0
平均周转时间：T＝（0.3 + 0.1 + 0.6 + 0.6 + 2.0）/ 5 = 0.72								

（3）高响应比优先调度算法

1）P1 首先运行，10.1 开始，运行 0.3，10.4 结束，周转时间：0.3

2）P2、P3 在就绪队列中：P2 响应比=1+等待时间/要求服务时间=1+0.1/0.9=1.1

P3 响应比=1+0/0.5=1

所以先运行 P2，从 10.4 开始，运行 0.9，11.3 结束，周转时间=11.3-10.3=13）

11.3 时刻：P3、P4、P5 都在就绪队列中：

P3 响应比=1+0.9/0.5=2.8

P4 响应比=1+0.8/0.1=9

P5 响应比=1+0.5/0.4=2.25

P4 运行，从 11.3 开始，11.4 运行结束，周转时间=0.9 4）

11.4 时刻：P3、P5 在就绪队列中：

P3 响应比=1+1/0.5=3

P5 响应比=1+0.6/0.4=2.25

P3 运行，11.4 开始，11.9 结束，周转时间=1.5 5）

最后运行 P5：11.9 开始，12.3 结束，周转时间=1.5

6） 平均周转时间=（0.3+1+0.9+1.5+1.5）/5=1.04

26、设有四个进程，它们到达就绪队列的时刻、要求运行时间及优先级（此处优先级 4 为最低优先级，优先级 1 为最高优先级）如表 3-24 所示。若分别采用非抢占式优先级调度算法和抢占式优先级调度算法，试给出各进程的调度顺序并计算平均周转时间。

表 3-24 进程调度信息表

进程	到达时刻	运行时间	优先级
P1	0	8	1
P2	1	3	3
P3	2	7	2
P4	3	12	4

解答：（1）非抢占式优先级调度算法：

调度顺序	进程	到达时刻	运行时间	开始时间	完成时间	周转时间
1	P1	0	8	0	8	8
2	P3	2	7	8	15	13
3	P2	1	3	15	18	17
4	P4	3	12	18	30	27
平均周转时间：T＝（8 + 13 + 17 + 27）/ 4 = 16.25						

（2）抢占式优先级调度算法：结果与非抢占式一样。

27、某分时系统的进程出现如下图 3-60 所示的状态变化：

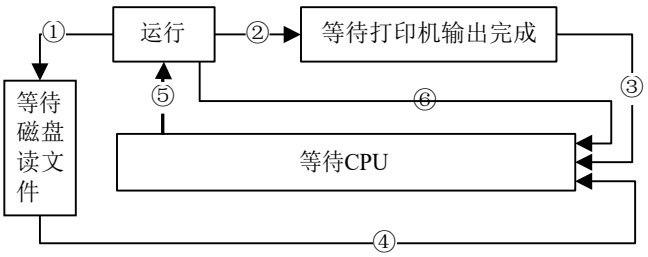


图 3-60 调度示意图

（1） 分析该系统采用的是何种进程调度算法？系统需要设置哪些进程队列？

（2） 写出图中所标示的 6 种状态变化及原因。

（3） 为了照顾等待 I/O 操作完成的进程能优先得到调度：①等待磁盘 I/O 操作完成的进程最优先得到照顾；②等待打印机输出完成的进程第二优先。应如何修改上述调度算法？请详细描述你的算法修改思路。

解答：（1）从图中可以看出在⑤、⑥和就绪队列之间存在一个环路，有的进程在执行中被剥夺处理机，并被排入就绪队列的末尾，等待下一次调度，同时进程调度程序又调度

就绪队列的队首进程到处理机上运行，因此是典型的时间片轮转调度算法。

（2） 进程状态变化及原因：

① 运行→阻塞：等待磁盘读操作的完成；

② 运行→阻塞：等待打印操作的完成；

③ 阻塞→就绪：所等待的打印操作已经完成；

④ 阻塞→就绪：所等待的磁盘读操作已经完成；

⑤ 就绪→运行：处理机调度；

⑥ 运行→就绪：当前进程时间片到。

（3） 可设置 3 个就绪队列：普通就绪队列（优先级最低）、磁盘 I/O 就绪队列（优先级最高），打印输出就绪队列（优先级第 2）。刚开始所有就绪进程都进入普通就绪队列，按时间片轮转进行调度；如果一个进程因为所等待的磁盘 I/O 操作完成被唤醒，则进入磁盘 I/O 就绪队列，并最优先得到调度，如果时间片到还没有运行完成则再次进入普通就绪队列；如果一个进程因为所等待的打印输出操作完成被唤醒，则进入打印输出就绪队列，并较普通就绪队列优先得到调度，如果时间片到还没有运行完成则再次进入普通就绪队列；当前进程运行过程中，如果有更高优先级进程到达，则抢占当前进程的 CPU。

28、进程通信主要有哪几种类型？各有何特点？

解答：进程通信可分为四大类：共享存储器系统、消息传递系统、管道通信系统以及客户-服务器系统通信。

共享存储器系统通信机制最大的特点是没有中间环节，通信双方直接读写共享存储分区而实现信息交换，因此通信速度非常快。此外，每次通信能够传输的数据量由共享存储区大小决定，能方便地实现大量数据的交换。

消息传递系统通信机制可实现不同主机间多个进程的通信。在消息传递系统中，进程间的数据交换以格式化的消息为单位，程序员直接利用系统提供的一组通信命令（原语）进行通信。这种方式既提高了通信效率，又简化了程序编制的复杂性，因此成为最常用的进程通信机制。

管道通信机制是通信双方利用一个共享文件来实现信息交换，分为无名管道和有名管道两种。无名管道是一个存在于高速缓存中的临时文件，没有对应的磁盘映像，常用于有亲缘关系的父子进程或兄弟进程之间的通信；有名管道可用于系统中任意进程间的通信，它是可以在文件系统中长期存在的具有路径名的文件。通信双方对管道文件的读写必须严格遵守 FIFO 的原则。

用户要访问的资源可能存放在网络上的某服务器中，于是客户端进程向服务器进程发请求，服务器进程可不断地接收来自多个客户端进程的服务请求，并为之提供相应服务，最后向客户端进程返回处理结果，这种通信机制称为客户/服务器系统通信（Client/Server 方式，简称 C/S 模式），目前已成为网络环境中主流的通信机制

29、有名管道和无名管道两种通信机制有什么不同？

解答：无名管道是一个存在于高速缓存中的临时文件，没有对应的磁盘映像，没有文件名，常用于有亲缘关系的父子进程或兄弟进程之间的通信，由系统调用 pipe()建立， 当通信双方终止时，管道文件也随之从缓存中删除。

有名管道可用于系统中任意进程间的通信，它是可以在文件系统中长期存在的具有路径名的文件，有文件名。进程对有名管道的访问方式与访问其它普通文件一样，都需要先用 open()系统调用打开它，通信结束时使用 close()关闭它。当不需要某个管道文件时，必须使用文件删除命令显式地删除它。

30、什么是死锁？产生死锁的原因和必要条件是什么？处理死锁的基本方法有哪些？

解答：若系统中存在一组进程（两个或两个以上），它们中的每一个都无限等待被该组进程中另一进程所占用的且永远无法释放的资源，这种现象称为进程“死锁”，或说这一组进程处于“死锁”状态。

产生死锁的原因有两个：竞争资源、进程推进顺序非法。

产生死锁的必要条件有 4 个：互斥条件、占有且等待条件、不可剥夺条件、循环等待条件。

处理死锁的基本方法有：预防死锁、避免死锁、检测死锁、解除死锁。

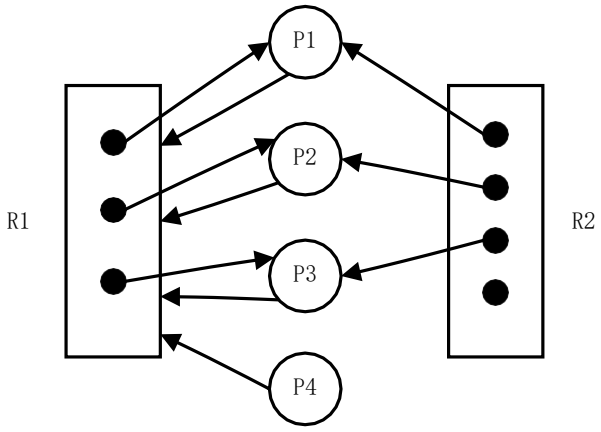
31、（南京大学，1995 年）假定某计算机系统中有 R1 设备 3 台，R2 设备 4 台，它们被 P1，P2，P3 和 P4 这 4 个进程共享，且已知这 4 个进程均以下面所示的顺序使用设备：→申请 R1→申请 R2→申请 R1→释放 R1→释放 R2→释放 R1→

（1） 系统运行过程中是否有产生死锁的可能性？为什么？

（2） 如果有可能产生死锁，请列举一种情况，并画出此时系统的资源分配图。

解答：（1）系统运行过程中有可能产生死锁：系统中只有 3 台 R1 设备，它们要被 4 个进程共享，且每个进程对 R1 的最大需求都为 2.由于 R1 数量不足，并且是需要互斥、不可剥夺的资源，而系统又没有采取任何措施破坏产生死锁的四个必要条件，因此，可能出现死锁。

（2）当 P1、P2、P3 个得到一个 R1 时，它们继续运行，都顺利申请到一个 R2；当三个进程第二次申请 R1 时，系统已无空闲的 R1，所以全部阻塞，并进入循环等待的死锁状态。这时候的资源分配图如下：



32、（西安电子科技大学，2002 年）有三个进程 P1，P2，P3 并发工作。进程 P1 需要资源 S3 和 S1，进程 P2 需要资源 S2 和 S1，进程 P3 需要资源 S3 和 S2。问：

（1） 若对资源分配不进行限制，会发生什么情况？为什么？

（2） 为保证进程正确运行，应采用怎样的资源分配策略？列出你能想到的方法。

解答：（1）可能会发生死锁。比如 P1 先申请并获得 S3，P2 进程申请并获得 S1，P3 进程申请并获得 S2，之后 P1 申请 S1 阻塞，P2 申请 S2 阻塞，P3 申请 S3 阻塞，并且资源都不能被释放，三个进程进入死锁状态。

（2）不会产生死锁的资源分配方法：

思路 1：三个进程都必须把手中已占据的资源归还给系统后才能申请下一个资源。

思路 2：采取静态分配资源的方法，每个进程必须同时获得所需要的两个资源，否则一

个资源都不分配。

思路 3：采取按资源序号递增的顺序申请资源，即 P1 顺序申请 S1 和 S3，P2 顺序申请 S1 和 S2，P3 顺序申请S2 和 S3。

33、在银行家算法中，某时刻 T 出现如表 3-25 所示资源分配情况。

Process	Max A B C	Allocation A B C	Available A B C
P0	7，5，3	0，1，0	3，2，2
P1	3，2，2	2，1，0	
P2	9，0，2	3，0，2	
P3	2，2，2	2，1，1	
P4	4，3，3	0，0，2	

试问：

（1） 此时系统状态是否安全？请给出详细的检查过程。

（2） 若进程依次有如下资源请求，则系统该如何进行资源分配，才能避免死锁？

P1：资源请求 Request（1,0,2）；

P2：资源请求 Request（3,3,0）；

P3：资源请求 Request（0,1,0）；

（3） 使用银行家算法解决死锁问题时，请分析该算法在实现中的局限性。

解答：（1）系统安全，因为存在安全序列（P1,P3,P0,P2,P4），判断过程略。

（2）系统要想避免死锁，就必须保证每次分配后都能得到安全序列，否则就拒绝分配。根据这一原则，对于进程的请求应考虑分配以后是否安全，若不安全，则不能进行此次分配。题目中有 3 个请求，按照顺序依次考虑，先考虑能否满足 P1，分配后系统仍处于安全状态， 因此存在安全序列（P1,P3,P2,P0,P4）。满足 P1 的请求之后，剩余资源为（2,2,0），对于 P2 的请求，由于系统没有那么多剩余资源，因此无法满足，系统拒绝 P4 的请求。最后考虑 P3 的请求，如果满足 P3，分配后剩余资源为（2,1,0），可以找到安全序列（P1,P3,P0,P2,P4），因此可以满足 P3 的请求。

（3）银行家算法的局限性：1）系统中进程数量及资源数量众多，而且每当进程申请资源时都要运行该算法，因此算法本身的运行开销大；2）银行家算法是按照最严苛的条件判断安全性的，过于保守，会降低资源利用率及延迟进程推进速度；3）银行家算法在寻找安全序列时完全没有考虑进程之间的同步关系：相互协作进程间的前驱后继关系，这在实际系统中是很不现实的。

34、某时刻系统的 A、B、C、D 四种资源状态如表 3-26 所示：

（1） 系统中四类资源各自的总数是多少？请写出Need 矩阵。

（2） 当前系统状态是否安全？请写出一个安全序列。

（3） 如果 P1 发出请求（0,4,2,0），是否可以满足该请求？如果可以，请给出安全序列。

表 3-26 系统资源分配状态表

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	0，0，1，2	0，1，1，2	1，5，4，0
P1	1，0，0，0	1，7，5，0	
P2	1，3，5，4	2，3，5，6	
P3	0，0，1，4	0，6，5，6	

（1） Need 矩阵=Max-Allocation，结果见表

Process	Allocation A B C D	Max A B C D	Need A B C D
P0	0，0，1，2	0，1，1，2	0，1，0，0
P1	1，0，0，0	1，7，5，0	0，7，5, 0
P2	1，3，5，4	2，3，5，6	1, 0, 0, 2
P3	0，0，1，4	0，6，5，6	0, 6, 4, 2

（2） P0、P2、P1、P3 或者 P0、P2、P3、P1，所以安全。（过程略）

（3） 合理请求，系统空闲资源够，尝试分配，之后进行安全性计算，可以得到一个安全系列：P0、P2、P1、P3，所以可以分配。（过程略）

35、某系统 T0 时刻的资源分配图如图 3-61 所示：

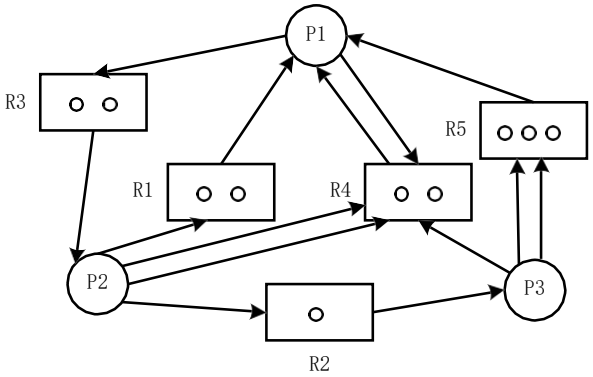


图 3-61 T0 时刻系统资源分配图

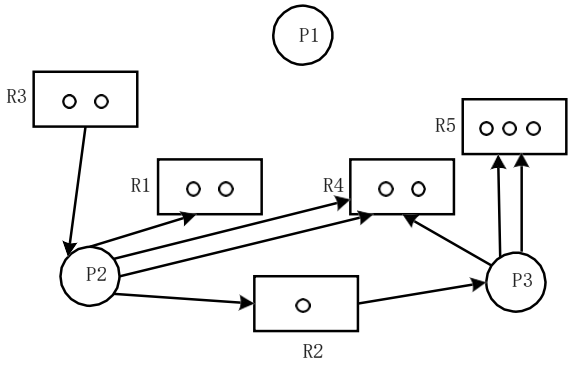
（1） 请问 T0 时刻该系统是否已经发生死锁？给出判断过程。

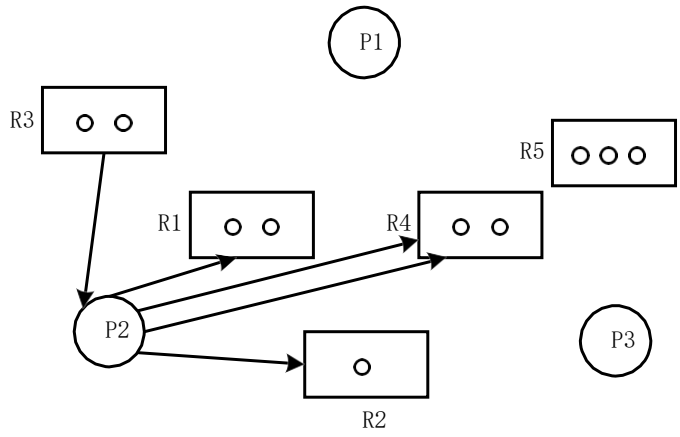
（2） 若此时进程 P2 申请一个R4 资源，按照银行家算法，系统是否可以进行分配？为什么？

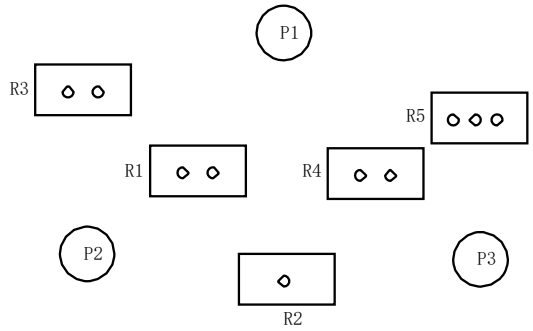
（3） 若要该系统预防死锁，可以采用什么样的资源分配算法？

解答：

（1） 简化资源分配图，简化顺序：







能完全简化，所以没有死锁发生。

（2） T0 时刻系统资源分配情况如下表所示：

进程	Allocation	Need	Available
	R1，R2，R3，R4，R5	R1，R2，R3，R4，R5	R1，R2，R3，R4，R5
P1	1， 0， 0， 1， 1	0， 0， 1， 1， 0	1， 0， 1， 1， 2
P2	0， 0， 1， 0， 0	1， 1， 0， 2， 0	
P3	0， 1， 0， 0， 0	0， 0， 0， 1， 2	

进程 P2 申请一个 R4 资源，是合法请求，且系统资源足够，尝试分配，并修改相关数据结构：进程 P2 的 allocation、need，available：

Allocation=0,0,1,1,0，need=1,1,0,1,0，available=1,0,1,0,2，再进行安全性判定，找不到安全序列，所以不能进行分配，系统资源分配恢复到原来的状态

（3） 若要该系统预防死锁，可以采取的资源分配算法有：静态分配资源、按序分配资源。

36、什么是线程？线程有哪几种实现方式？简述线程与进程的区别和联系。

解答：线程是隶属于进程的一个实体，是比进程更小的一个运行单位。

线程实现机制有三种： 用户级线程、内核级线程、组合模式。

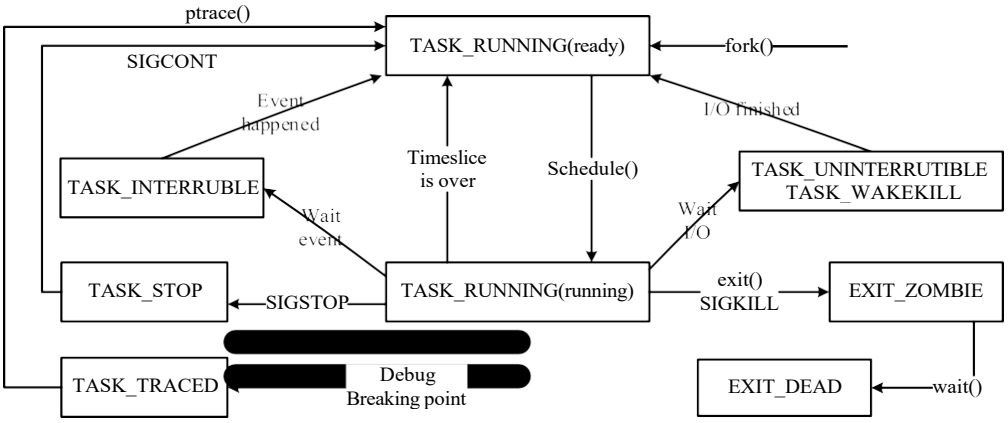
线程与进程的区别有：（1）从调度上来看，引入线程后，CPU 的调度单位是线程，进程是系统分配资源的单位。（2）从拥有资源上看，进程是系统分配资源的单位，线程基本上不拥有资源（只有一点运行时必不可少的资源）。（3）从管理开销上看，进程创建、撤销、调度的开销都远远大于线程开销。

线程与进程的联系：一个线程是隶属于某个进程的，并且共享所属进程的全部资源。引入线程后，多个线程间可以并发运行，从而提高整个进程的并发性。

37、Linux 系统设置了哪几种进程状态？简要说明各状态之间转换的典型原因。

解答：Linux 系统中设置的进程状态主要有：TASK_RUNNING，TASK_INTERRUPTIBLE，TASK_UNINTERRUPTIBLE，TASK_STOPPED，TASK_TRACED，TASK_WAKEKILL，TASK_DEAD，TASK_WAKING。此外还为处于终止过程的进程设置了两个状态：EXIT_DEAD，EXIT_ZOMBIE。

进程在各状态间转换的典型原因如图所示：



38、fork()、vfork()、clone()三个系统调用的区别是什么？

解答：fork()创建一个普通进程，父进程与子进程间采用“写时复制”技术共享代码和数据。vfork()创建的子进程能共享父进程的地址空间，且父进程会一直阻塞，直到子进程调用 exit()终止运行或调用 exec()加载另一个可执行文件，即子进程优先运行。clone()不同于前面两个，它接受一个指向某函数的指针和该函数的参数，刚创建的子进程将马上执行该函数。clone()允许子进程有选择性的继承父进程的资源，因此通常用它来创建线程。

39、do_fork()在创建一个进程时，大致要做哪些事情？

解答：do_fork()在创建一个进程时，大致要做的工作包括：

①设置新进程的跟踪状态：如果父进程被跟踪，且调试器要求跟踪每一个子进程时，则新进程也处于跟踪状态。

②调用 copy_process()完成新进程的进程空间创建和描述符域定义的大部分工作，如为新进程创建内核栈、建立 PCB、复制父进程相关信息等，具体见教材 3.3.4 内容。

③如果clone_flags 包含CLONE_STOPPED 标志，则置新进程状态为TASK_STOPPEND，否则将新进程插入可运行队列；

④如果 clone_flags 包含CLONE_VFORK 标志，则阻塞父进程直到新进程运行结束或执行另外一个程序；

⑤返回子进程的 pid 值，该值就是 fork()系统调用后父进程的返回值。

40、Linux 中当首次将 CPU 调度给子进程时，它将从哪里开始执行指令？

解答：从 fork（）调用的下一条指令开始执行。

41、虽然父子进程可以完全并发执行，但在 Linux 中，成功创建子进程之后，通常让子进程优先获得 CPU，这种做法有什么好处？

解答：父进程调用 fork()创建子进程后，父子进程以“写时复制”方式共享父进程的代码和数据，即执行完全相同的功能，但事实上父进程通常是让新的子进程完成其他与父进程不一样的工作，比如调用 cxec（）系列系统调用执行另外一个可执行程序。如果让父进程先

执行，假设父进程要修改相关数据，则系统马上会为父进程分配新的内存块并复制数据所在页面，而子进程执行时因调用 cxec()执行其他程序，必须重新分配内存空间容纳新程序代码和数据，则前面父进程的复制工作就白做了，浪费系统内存资源和复制时间。所以倾向于让子进程先执行。

42、在 Linux 系统中运行下面程序：

```
main(){
    int num=0；
    fork();
    printf(“hello1\n”);
    fork();
    printf(“hello2\n”);
    fork();
    num++；
    printf(“hello3\n”);
}
```

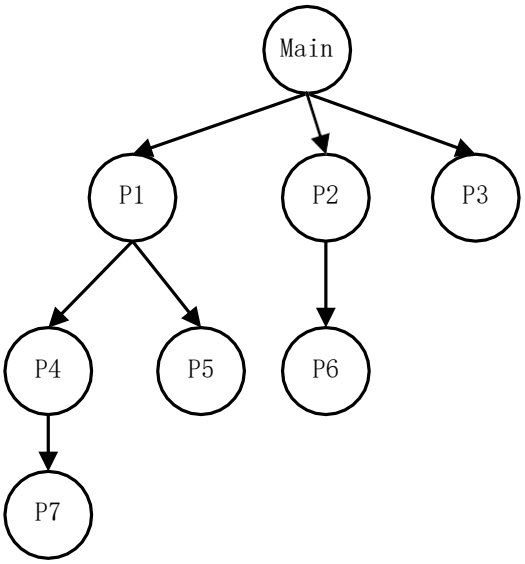
问：（1）最多可产生多少个进程？画出进程家族树。（包含 main 进程在内）

（2） 其中 hello1、hello2、hello3 各被输出多少次？

（3） num 最后的计算结果的最大值是多少？如果将程序中的 fork()换成 vfork()，则 num 最后的计算结果的最大值又是多少？

解答：

（1） 产生 8 个进程，进程图：



（2） hello1、hello2、hello3 各被输出 2 次、4 次、8 次

（3） fork：num 最大值：1

vfork：num 最大值：8

43、试分析 Linux 提供的 clone()操作如何支持线程和进程？

解答：clone()操作也是创建一个子进程，不过可以在系统调用中指定相关参数以说明从父进程那里继承哪些资源，从而达到创建线程的目的。

44、Linux 分别为内核程序和用户程序各提供了哪些同步机制？线程间同步通常采用哪几种同步机制？linux 中 posix 信号量与 System V 信号量有什么区别？

解答：Linux 为内核程序提供的同步机制包括原子操作、自旋锁、读写自旋锁、信号量、读写信号量、大内核锁、互斥体、RCU 机制、禁止本地中断等；为用户程序提供的同步机制包括 IPC 信号量、Futex 机制、主要用于线程同步的 POSIX 信号量及条件变量等。

linux 中 posix 信号量包括有名信号量和无名信号量两种，有名信号量可以用于任意进程间的同步，无名信号量主要用于线程间的同步；而 System V 信号量是 Linux 进程通信机制的一种，又称为 IPC 信号量，它是一种信号量集机制，运行进程一次申请多种资源，灵活性比 Posix 信号量更好，但复杂度也更高，可用于任意进程间的同步。

45、Linux 系统为了实现 O(1)级算法复杂度，在调度策略中采用了什么措施？

解答：Linux 系统为了实现 O(1)级算法复杂度，在系统中设置了一个位示图，每个优先级就绪队列都对应一个位图中的二进制位，如果某个就绪队列中有进程存在，则相应的二进制位的值为 1，否则值为 0。当系统进行调度时，首先遍历位示图，找到优先级最高的值为 1 的二进制位，然后取该队列的队首进程参与运行即可，算法的时间开销与就绪队列中进程的数量没有关系。

46、简要说明 Linux 系统的 CFS 调度策略的基本思想。

解答：CFS 是“Completely Fair Scheduler”的缩写，即“完全公平调度器”，其设计思想主要有两点：一是根据系统中各进程的权重分别为各进程分配下一次 CPU 运行时间长度，进程优先级越高，权重越大，分配到的CPU 时间越多，以保证高优先级进程获得更多的 CPU 运行时间；二是选择下一个运行进程时，总是选择“运行速度”最慢的进程参与运行。为实现这个调度思想，CFS 引入了“虚拟运行时间”的概念：首先将进程的 nice 值转化权重，然后依据系统中就绪进程数量确定下一次CPU 运行时间长度，再根据各就绪进程的权重在所有就绪进程的总权重中的比例计算每个就绪进程下一次的CPU 运行时间，采用下面公式计算各进程的虚拟运行时间，T_i=period * $\frac{W_i}{\sum_{i=1}^{n} W_i}$，调度时总是选择虚拟运行时间最短的进程参与运行。

47、Linux 系统中实时进程及普通进程的优先级范围各是多少？

解答：linux 系统中实时进程的优先级是 0-99，普通进程的优先级范围是 100-139.

48、Linux 系统提供了哪几种进程通信机制？Linux 消息队列通信机制与教材中的消息缓冲队列通信机制存在哪些异同？

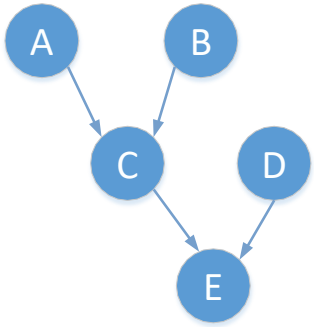
解答：Linux 提高的进程通信机制有管道通信、共享内存通信、消息传递系统通信。Linux 消息队列通信机制与教材中的消息缓冲队列通信机制相同的地方是：都是以消息为单位进行信息传递，发送进程把消息挂入消息队列，接收进程从消息队列取出消息。不同的地方是：教材中的消息缓冲队列是每个进程设置一个，所有权其他进程发生给该进程的消息都在这个队列中，也只允许该进程从消息队列中接收消息。Linux 中的消息队列是给通信各方进程共享的，只要对这个队列具有相应的权限，每个进程都可以从这个队列中取出指定的消息，也可以把消息发送到这个队列中。

49、请分析 Linux 系统的内核线程创建函数 kthread_create()大致是如何实现的？

解答：kthread_create()主要完成的工作有：首先初始化新创建进程的 create 结构，如要执行的函数、函数所需要的参数，再初始化完成量，然后把 cteate 结构挂到 kthread_create_list 链表末尾，然后唤醒 kthreadd 进程（2 号进程，是系统中所有内核线程的父进程），然后等待 kthreadd 完成线程的创建工作，最后检查并返回创建结果。

```
struct task_struct *kthread_create(int (*threadfn)(void
*data),
                                   void *data,
                                   const char namefmt[],
                                   ...)
{
        struct kthread_create_info create;

        create.threadfn = threadfn;
        create.data = data;
        init_completion(&create.started);
        init_completion(&create.done);

        spin_lock(&kthread_create_lock);
        list_add_tail(&create.list, &kthread_create_list);
        spin_unlock(&kthread_create_lock);

        wake_up_process(kthreadd_task);
        wait_for_completion(&create.done);

        if (!IS_ERR(create.result))
                { va_list args;
                va_start(args, namefmt);
                vsnprintf(create.result->comm,
sizeof(create.result->comm),
                          namefmt, args);
                va_end(args);
        }
        return create.result;
}
EXPORT_SYMBOL(kthread_create);
```

50、（2020 年统考题）现有 5 个操作 A、B、C、D 和 E，操作 C 必须在 A 和 B 完成后执行，操作E 必须在 C 和D 完成后执行，请使用信号量的 wait()、signal()操作（P、V 操作）描述上述操作之间的同步关系，并说明所用信号量及其初值。



解答：本题是实现几个操作间的执行顺序，少一个简单的同步问题。可先画出如右图所示前驱图帮助分析。

依据前期后继关系，设置 4 个同步信号量：

Semaphore SAC=0，//控制 A、C 顺序；

Semaphore SBC=0，//控制 B、C 顺序；

Semaphore SCE=0，//控制 C、E 顺序；

Semaphore SDE=0，//控制 D、E 顺序；

Main()

{ parbegin(A,B,C,D,C);}

A(){
完成动作A；
Signal(SAC);
}
	B(){
	完成动作B；
	Signal(SBC);
	}
		C(){ Wait(SAC);
		Wait(SBC);
		完成动作C；
		Signal(SCE);
		}
			D(){
			完成动作D；
			Signal(SDE);
			}
				E(){ Wait(SCE);
				Wait(SDE);
				完成动作E；
				}

51.（2021 年统考题）（1）如下两个操作对 S 为什么要互斥？

```
wait(S):{while(S≤0);        signal(S):{
        S--;                        S++;
        }                           }
```

（2） 算法 1 和算法 2 哪个可以实现临界区的互斥？

算法 1：

```
wait(S){            signal(S){
关中断;             关中断;
while(S≤0);         S++;
S--;                开中断;
开中断;             }
}
```

算法 2：

```
wait(S){            signal(S){
关中断;             关中断;
while(S≤0){         S++;
开中断;             开中断;
关中断;}            }
S--;
开中断;}
```

（3） 用户程序能不能用开关中断实现互斥访问临界资源？

解答：（1）因为 wait（S）和 signal（S）两个操作都要对变量 S 执行修改操作，使得 S 成为两个操作要共享的临界资源，因此必须实现互斥，否则可能会出现与时间有关的错误。

（2） 算法 2 能实现互斥。在算法 1 中，当 S≤0 时，wait（S）将不断空循环，且不能打开中断，导致其他进程不能获得CPU 运行机会，也就不能执行 signal（S）修改 S 的值，从而进入死锁状态。

（3） 用户程序不能用开关中断实现互斥访问临界资源，一是因为开关中断是特权指令，用户程序在用户态下不能执行；二是让用户程序控制中断开关，将给系统带来极大的隐患，使用不当，将会导致中断永远打不开，而让系统进入死锁状态。

52.（2019 年统考题）有 n（n≥3）位哲学家围坐在一张圆桌边，每位哲学家交替地就餐和思考。在圆桌中心有 m（m≥1）个碗，每两位哲学家之间有 1 根筷子。每位哲学家必须取到一个碗和两侧的筷子之后才能就餐，进程完毕，将碗和筷子放回原位，并继续思考。为使尽

可能多的哲学家同时就餐，且防止出现死锁现象，请使用信号量的 P、V 操作（wait()、signal()操作）描述上述过程中的互斥与同步，并说明所用信号量及初值的含义。

解答：semaphore bowl=m；//哲学家互斥使用碗；
semaphore chopsticks[n]；//哲学家互斥使用筷子
semaphore Sm=n-1； //确保最多只允许 n-1 个哲学家去拿筷子，避免死锁
for (int i=0; i＜n; i++)
{ chopsticks[i]=1;}
CoBegin
P -i() { //哲学家 i 执行程序
While(true){
思考问题；
Wait(Sm);
Wait(bowl);
Wait(chopsticks[i]);
Wait(chopsticks[(i+1)MOD n]);
就餐；
Signal(chopsticks[i])
Signal(chopsticks[(i+1)MOD n])
Signal(bowl)
Signal(Sm)
}
}
CoEnd

53.（2017 年统考题）某进程中有 3 个并发执行的线程 thread1,thread2 和 thread3，其伪代码如下所示：

```
//复数的结构类型定义：
typedef struct{
  float a;
  float b;
}cnum;
cnum x,y,z;//全局变量
//计算两个复数之和
cnum add(cnum p,cnum q)
{ cnum s;
  s.a=p.a+q.a;
  s.b=p.b+q.b;
  return s; }
```

```
thread1
{ cnum w;
  w=add(x,y);
  …
}
Thread2
{ cnum w;
  w=add(y,z);
  …
}
```

```
thread3
{ cnum w;
  w.a=1;
  w.b=1;
  z=add(z,w);
  y=add(y,w);
  …
}
```

请添加必要的信号量和P、V(或wait()、signal())操作，要求确保线程互斥访问临界资源，并且最大程度地并发执行。

解答：semaphore mutex_y1=1 //mutex_y1用于thread1与thread3对变量y的互斥访问
semaphore mutex_y2=1 //mutex_y2用于thread2与thread3对变量y的互斥访问
semaphore mutex_z=1 //mutex_z用于变量z的互斥访问

thread1	thread2	thread3

{
cnum w;
wait(mutex_y1);
w=add(x，y);
signal(mutex_y1);
……
}
	{
	cnum w;
	wait(mutex_y2);
	wait(mutex_z);
	w=add(y，z);
	signal(mutex_z);
	signal(mutex_y2);
	……
	}
		{
		cnum　　w;
		w.a=1;
		w.b=1;
		wait(mutex_z);
		z=add(z，w);
		signal(mutex_z);
		wait(mutex_y1);
		wait(mutex_y2);
		y=add(y，w)
		signal(mutex_y1);
		signal(mutex_y2);
		……
		}

54.（2016年统考题）某进程调度程序采用基于优先数(priority)的调度策略，即选择优先数最小的进程运行，进程创建时由用户指定一个nice作为静态优先数。为了动态调整优先数，引入运行时间cpuTime和等待时间waitTime，初值均为0。进程处于执行态时，cpuTime定时加1，且waitTime置0；进程处于就绪态时，cpuTime置0，waitTime定时加1。请回答下列问题。

（1）若调度程序只将nice的值作为进程的优先数，即priority=nice，则可能会出现饥饿现象，为什么?

（2）使用nice、cpuTime和waitTime设计一种动态优先数计算方法，以避免产生饥饿现象，并说明waitTime的作用。

解答：(1)由于采用了静态优先数，当就绪队列中总有优先数较小的进程时，优先数较大的进程一直没有机会运行，因而会出现饥饿现象。

(2)优先数priority的计算公式为：

priority=nice+k1×cpuTime-k2×waitTime，其中k1>0，k2>0，用来分别调整cpuTime和waitTime在priority中所占的比例。waitTime可使长时间等待的进程优先数减小，从而避免出现饥饿现象。

55.（2015年统考题）有A、B两人通过信箱进行辩论，每个人都从自己的信箱中取得对方的问题，将答案和向对方提出的新问题组成一个邮件放人对方的信箱中。假设A的信箱最多放M个邮件，B的信箱最多放N个邮件。初始时A的信箱中有x个邮件(0<x<M)，B的信箱中有y个邮件(0<y<N)。辩论者每取出一个邮件，邮件数减1。A和B两人的操作过程描述如下：

A {	B {
while(TRUE){	while(TRUE){
从A的信箱中取出一个邮件；	从B的信箱中取出一个邮件；
回答问题并提出一个新问题；	回答问题并提出一个新问题；
将新邮件放入B的信箱； }	将新邮件放入A的信箱；}
}	}

当信箱不为空时，辩论者才能从信箱中取邮件，否则等待。当信箱不满时，辩论者才能将新邮件放入信箱，否则等待。请添加必要的信号量和P、V(或wait()、signal())操作，以实现上述过程的同步。要求写出完整的过程，并说明信号量的含义和初值。

解答：

```
semaphore Full_A = x ;          //Fuu_A表示A的信箱中的邮件数量
semaphore Empty_A = M-x;        //Empty_A表示A的信箱中还可存放的邮件数量
semaphore Full_B = y ;          //Full_B表示B的信箱中的邮件数量
semaphore Empty_B = N-y;        //Empty_B表示B的信箱中还可存放的邮件数量
semaphore mutex_A = 1 ;         //mutex_A用于A的信箱互斥
semaphore mutex_B = 1 ;         //mutex_B用于B的信箱互斥
main（）{parbegin（A，B）}
```

```
A{
while(TRUE){ P(Full_A)
;  P(mutex_A);
从A的信箱中取出一个邮件;
V(mutex_A);
V(Empty_A);
回答问题并提出一个新问题;
P(Empty_B);
P(mutex_B);
将新邮件放入B的信箱;
V(mutex_B);
V(Full_B);
}
}
```

```
B{
while(TRUE){ P(Full_B
);  P(mutex_B);
从B的信箱中取出一
个邮件;
V(mutex_B);
V(Empty_B);
回答问题并提出一个
新问题;
P(Empty_A);
P(mutex_A);
将新邮件放入A的
信箱;
V(mutex_A);
V(Full_A);
}
}
```

56.（2014年统考题）系统中有多个生产者进程和多个消费者进程，共享一个能存放1000件产品的环形缓冲区（初始为空）。当缓冲区未满时，生产者进程可以放入其生产的一件产品，否则等待；当缓冲区未空时，消费者进程可以从缓冲区取走一件产品，否则等待。要求一个消费者进程从缓冲区连续取出10件产品后，其他消费者进程才可以取产品。请使用信号量P，V(wait()，signal())操作实现进程间的互斥与同步，要求写出完整的过程，并说明所用信号量的含义和初值。

解答：这是典型的生产者和消费者问题，只对典型问题加了一个条件，只需在标准模型上新加一个信号量，即可完成指定要求。设置四个变量mutex1、mutex2、empty和full：

mutex1，用于控制一个消费者进程一个周期(10次)内对于缓冲区的控制，初值为1 mutex2，用于进程单次互斥的访问缓冲区，初值为1

empty，代表缓冲区的空位数，初值为0，

full，代表缓冲区的产品数，初值为1000，具体进程的描述如下：

```
semaphore mutex1=1;
semaphore mutex2=1;
semaphore empty=1000;
semaphore full=0;
 producer()
```

```
{ while(1)
  { 生产一个产品;
    P(empty); //判断缓冲区是否有空位
    P(mutex2); //互斥访问缓冲区
    把产品放入缓冲区; V(mutex2);
    //互斥访问缓冲区V(full); //
    产品的数量加1
    }
  }
consumer()
{ while(1)
  {  P(mutex1) //连续取10次for(int
      i = 0; i <= 10; ++i)
          { P(full); //判断缓冲区是否有产品
            P(mutex2); //互斥访问缓冲区
            从缓冲区取出一件产品;
            V(mutex2); //互斥访问缓冲区
            V(empty); //腾出一个空位消费这件产品;
           }
    V(mutex1)
   }
}
```

57.（2013年统考题）某博物馆最多可容纳500人同时参观，有一个出入口，该出入口一次仅允许一个人通过。参观者的活动描述如右图3-71所示，请添加必要的信号量和P、V（或wait()、signal()）操作，以实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

```
cobegin
参观者进程i：
{   …
    进门;
    …
    参观;
    …
    出门;
    …  }
coend
```

图3-71 参观活动

解答：定义两个信号量：

```
Semaphore empty= 500; // 博物馆可以容纳的最多人数
Semaphore mutex= 1; // 用于出入口资源的控制
Cobegin
参观者进程i;
{   …
    P  (empty);
    P ( mutex);
    进门;
    V( mutex);
    参观;
    P ( mutex);
    出门;
    V( mutex);
    V( empty);
    … }
coend
```

58.（2011年统考题）某银行提供1个服务窗口和10个供顾客等待的座位。顾客到达银行时，若有空座位，则到取号机上领取一个号，等待叫号。取号机每次仅允许一位顾客使用。当营业员空闲时，通过叫号选取一位顾客，并为其服务。顾客和营业员的活动过程描述如下：

process顾客i {
从取号机获取一个号码；
等待叫号；
获取服务；
}
	process 营业员
	{ while
	（TRUE）{
	叫号；
	为客户服务；}
	}

请添加必要的信号量和P、V（或wait()、signal()）操作，实现上述过程中的互斥与同步。

解答：semaphore seats=10
semaphore mutex=1 //互斥使用取号机
semaphore haveCustom=0 //顾客与营业员同步，无顾客时营业员休息
semaphore call=0 //等待营业员叫号

```
Cobegin：
    Process 顾客
        {
            P(seats); //等空位 P(mutex);
            //申请使用取号机从取号机上
            取号;   V(mutex);  //释放取
            号机
            V（haveCudtom）;  //通知营业员有新顾客到来P
            （call）;
            等待叫号;
            V（seats）;   //离开座位
            获取服务服务;
            }
    Process 营业员
    {
        While（true）
        {
            P(haveCustom);  //没有顾客就休息
            叫号;  V
            （call）;  为
            顾客服务;
        }
    }
```

59.（2009年统考题）三个进程P1、P2、P3 互斥使用一个包含N(N>0)个单元的缓冲区。P1每次用produce()生成一个正整数并用put()送入缓冲区某一空单元中；P2 每次用getodd()从该缓冲区中取出一个奇数并用countodd()统计奇数个数；P3 每次用geteven()从该缓冲区中取出一个偶数并用counteven()统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义信号量的含义。要求用伪代码描述。

解答：定义信号量odd控制P1与P2之间的同步；even控制P1与P3之间的同步；empty控制生产者与消费者之间的同步；mutex控制进程间互斥使用缓冲区。程序如下：

```
semaphore odd = 0, even = 0, empty = N, mutex = 1;
P1( )
{
    x = produce(); //生成一个数
    P(empty); //判断缓冲区是否有空单元
    P(mutex); //缓冲区是否被占用Put();
    V(mutex); //释放缓冲区
    if(x%2 == 0)
    V(even); //如果是偶数，向P3 发出信号
    else
    V(odd); //如果是奇数，向P2 发出信号
}
P2( )
{
    P(odd); //收到P1 发来的信号，已产生一个奇数
    P(mutex); //缓冲区是否被占用
    getodd();
    V(mutex); //释放缓冲区
    V(empty); //向P1 发信号，多出一个空单元
    countodd();
}
P3( )
{
    P(even); //收到P1 发来的信号，已产生一个偶数
    P(mutext); //缓冲区是否被占用
    geteven();  V(mutex);
    //释放缓冲区
    V(empty); //向P1 发信号，多出一个空单元
    counteven();
}
```