

第3章 进程管理课后习题解答

1. 操作系统为什么要引入进程的概念？

【解答】程序在并发执行方式下，运行时具有异步性的特征。这样，就需要一个数据结构 PCB 来记录程序的状态，以及控制其状态转换所需的一些信息。因此，将 PCB、程序、数据三者组成一个完整的实体，就是进程实体。进程是程序的一次执行，引入进程的概念，便于操作系统对于程序的运行进行控制。

2. 试比较进程和程序的区别。

【解答】进程和程序之间存在着密切的联系，但它们是两个完全不同的概念。它们之间的主要区别是：

(1) 程序是静态的概念，本身可以作为一种软件资源长期保存着；而进程则是程序的一次执行过程。它是动态的概念。

(2) 进程是一个能独立运行的单位，能与其它进程并发执行。进程是作为资源申请和调度单位存在的；而通常的程序是不能作为一个独立运行的单位而并发执行的。

(3) 程序和进程无一对应的关系。一个程序可由多个进程共用；另一方面，一个进程在其活动中又可顺序地执行若干个程序。一个程序，运行一次，便创建了一个进程；同一个程序，若运行十次，就产生了十个进程。

(4) 各个进程在并发执行过程中会产生相互制约关系，造成各自前进速度的不可预测性。而程序本身是静态的，不存在这种异步特征。

3. 程序并发执行，为什么会失去封闭性和可再现性？

【解答】在程序并发执行的环境下，多个进程共享系统中的资源，这些资源是否被使用，及每一时刻由哪个进程使用，将由多个进程的相互作用而决定。这样，程序的执行就失去了封闭性。某个进程的执行必然受到其它进程的影响。

由于进程在并发执行的过程中失去了封闭性，当下一次再运行该程序时，系统中存在的进程及各进程的状态都可能发生变化，导致进程的执行失去可再现性。

4. 什么叫进程的并发性？试举一个进程并发执行的例子。

【解答】多个进程在同一时间间隔内同时发生，叫做进程的并发性。

例如：我们在 Linux 系统下使用编辑器 vi 进行编辑，而此时系统正在后台运行一个编译程序，此时，两个进程并发执行。

5. 举一个例子，说明一个程序可能同时属于多个进程。

【解答】例如：超市的收银系统。在每个终端上，收银员在运行收款程序，因此每个收银员都在执行一个进程。超市里同时有很多收款终端，同一个程序被并发执行多次，造成了多个进程并发执行。

6. 试说明 PCB 的作用，为什么说 PCB 是进程存在的惟一标志？

【解答】PCB 的作用是：在一个在多道程序环境下，集中反映了一个进程的动态特征。在进程并发执行时，由于资源共享，带来各进程之间的相互制约。显然，为了反映这些制约关系和资源共享关系，必须使用 PCB 中的信息，才能对进程实施有效的管理和控

制。

PCB 是进程实体的一部分，是操作系统中非常重要的数据结构，存放着进程所需的运行信息和控制信息，系统在创建进程时，首先创建 PCB，然后通过 PCB 感知进程的存在，进程在被撤消时，PCB 也随之被撤消。因此，PCB 是进程存在的惟一标志。

7. 说明进程由哪几部分构成？

【解答】进程实体通常就是由程序、数据集合和 PCB 这三部分构成，也称为“进程映像”。

8. 什么叫临界区？为什么进程在进入临界区之前，要先执行申请操作，离开临界区，要做释放操作？

【解答】将每个进程中访问临界资源的那段代码称为临界区，因此也可以说：不能被多个进程同时进入的程序或数据区域叫做临界区。

因为临界区不能由多个进程同时进入，因此，在进入临界区之前，要先执行申请操作，确保在没有其它进程进入的情况下，该进程才能进入该临界区；在离开临界区之前，要做释放操作，以便其它进程可以进入该临界区。

9. 试说明进程的基本状态及转换的原因。

【解答】在操作系统中，进程通常至少有三种基本状态：就绪状态、执行状态、阻塞状态。

进程状态转换的原因是：系统中的进程是并发执行的。在系统中，CPU 在不同的时间，按一定的算法为不同的进程服务；系统中的资源是有限的；各进程之间存在相互作用。因此，进程并非固定处于某个状态，它将随着自身的推进和外界条件的变化而发生变化。对于一个系统，处于就绪状态的进程，在调度程序为之分配了处理机之后，该进程便由就绪态转变为执行状态。当前进程，如果因分配给它的时间片已用完而被暂停执行时，该进程便由执行状态又回到就绪状态；一个处在执行状态的进程，因为等待某外部事件发生，而使该进程状态转变为阻塞状态。一个处于阻塞状态的进程，当它所需的外部事件满足，它应由阻塞状态变为就绪状态。

10. 在创建一个进程时，所要做的工作有哪些？

【解答】操作系统通过进程创建原语来创建一个进程。创建原语通过下述步骤创建一个进程：

- (1) 申请空白 PCB。
- (2) 为新进程分配资源。
- (3) 初始化进程控制块。
- (4) 将新建进程插入就绪态队列。

11. 从概念上说明记录型信号量的构成，描述 wait 原语和 signal 原语所进行的操作。

【解答】记录型信号量的数据结构由两部分构成，即：数值分量和指针分量。数值表示系统中可用的该类临界资源的数量，而指针分量为进程链表指针，指向等待该类资源的 PCB 队列。

申请临界资源的原语 wait 操作可描述为：

```
procedure wait(S)
  var S: semaphore;
```

```
begin
  s:=s-1;
  if s≥0 then 本进程继续;
else
  将本进程放入阻塞态队列;
  转进程调度
end
```

释放临界资源的原语 **signal** 操作可描述为:

```
procedure signal(S)
  var S: semaphore;
begin
  s:=s+1;
  if s≤0 then 唤醒指针 L 所指的阻塞态进程; ;
end
```

12. 在生产者—消费者问题中, 如果缺少了 **signal(full)**或 **signal(empty)**, 对执行结果将会有何影响?

【解答】若缺少释放资源的原语操作, 则会导致生产者或消费者进程不能再继续工作。

如缺少了 **signal(full)**, 则消费者进程可能得不到所需的临界资源如缓冲区, 不能取一件产品;

同样地, 如果缺少 **signal(empty)**, 则生产者进程又可能得不到所需的资源, 不能存放一件产品。

13. 在生产者—消费者问题中, 如果两个 **wait** 操作即 **wait(mutex)**和 **wait(empty)**位置互换, 会产生什么后果?

【解答】如果两个 **wait** 操作即 **wait(mutex)**和 **wait(empty)**位置互换, 则有可能产生死锁。

14. 进程的高级通信方式有哪几种?

【解答】高级通信方式可分为三大类: 共享存储器系统、消息传递系统和管道通信系统。

在共享存储器系统中, 相互通信的进程共享某些数据结构或共享存储区; 消息传递系统中, 进程间的数据交换以消息为单位, 用户直接利用系统提供的一组通信原语来实现通信, 消息传递系统可分为消息缓冲通信和信箱通信; 管道通信通信的方法是, 向管道提供输入的发送进程以字符流形式将大量的数据送入管道, 而接受管道输出的接收进程可从管道中接收数据。

15. 什么是线程? 说明它与进程的主要区别。

【解答】线程是进程中执行运算的最小单位, 亦即执行处理机调度的基本单位。

线程与进程的的主要区别:

(1) 调度性方面: 进程是资源分配的基本单位。而线程是分配处理机的基本单位, 它与资源分配无关。即真正在处理机上运行的是线程。

(2) 拥有资源方面: 进程是拥有资源的独立单位, 而线程基本是不拥有资源的。同一进程的线程共用该进程的资源。

(3) 并发性方面：进程和线程都可以并发执行。但同一个进程的线程在并发执行时，状态转换速度较快。

(4) 系统开销方面：进程在创建、状态转换、撤消时开销较大，而对于同样的工作，线程的开销较小。

16. 什么是多线程机制？引入它有什么好处？

【解答】多线程机制是指操作系统支持在一个进程内执行多个线程的能力。

引入线程的好处有以下几点：

(1) 易于调度。由于线程只作为独立调度的基本单位，同一进程的多个线程共享进程的资源，所以线程易于切换。

(2) 提高了系统的效率。通过线程可方便有效地实现并发性。进程可创建多个线程来执行同一程序的不同部分。

(3) 创建一个线程比创建一个进程花费的开销少，创建速度快。

(4) 在多处处理器的系统中，有利于发挥多处理器的功能，提高进程的并行性。

17. 在读者—写者问题中，如果修改问题中的同步算法，要求对写进程优先，即一旦写进程到达，后续的读者进程必须等待，而无论是否有读者进程在读文件。写出相应进程的程序段。

【解答】增加一个信号量 S，用于在写进程到达后封锁后续的读进程。

```
semaphore S=1;
```

在读进程开头，增加两句：

```
P (S) ;
```

```
V (S) ;
```

即所有访问该文件的进程，在申请读以前，先申请信号量 S。如果没有写进程，则可以直接通过；如果存在写进程，则该信号量 S 申请不到，该读者进程等待。

对于写进程，首先该申请信号量 S，由于读者进程申请到以后接着释放该信号量，所以写者进程更容易申请到 S。写者进程申请到 S 以后，直到写操作完成以后才释放该信号量。因此，读者进程和写进程改为：

```
Var:rmutex, wmutex:semaphore:=1,1;
    Readcount:integer:=0;
begin
  parbegin
    读者进程:
      Reader:
        begin
          repeat
            wait(s);
            signal (s);
            wait(rmutex);
            if readcount=0 then wait(wmutex);
            readcount:=Readcount+1;
            signal(rmutex);
            ...
```

```

        进行读操作;
        ...
        wait(rmutex);
        readcount:=readcount+1;
        if readcount=0 then signal(wmutex);
        signal(rmutex);
    until false;
end
写者进程:
writer:
begin
    repeat
        wait(s);
        wait(wmutex);
        执行写操作;
        signal(wmutex);
        signal(s);
    until false;
end
parend
end

```

18. 试利用记录型信号量写出一个不会出现死锁的哲学家进餐问题的算法。

【解答】有多种算法可解决该问题。

方法（1）至多只允许有四位哲学家同时去拿左边的筷子。

实现方法：在每个进程的程序段前定义一个信号量 S，初值为 4，每个哲学家在使用筷子之前，先申请信号量 S，吃完放下筷子后，再释放 S。各进程执行的程序段为：

```

Pi ( )
Begin
S: semaphore=4
Var chopstick: array[0, ..., 4] of semaphore=[1,1,1,1,1];
repeat
    wait(s);
    wait(chopstick[i]);
    wait(chopstick[(i+1) mod 5]);
    eat;
    ...
    signal(chopstick[i]);
    signal(chopstick[(i+1) mod 5]);
    signal(s);
    think;
until false;
end

```

方法（2）仅当哲学家的左、右两只筷子均可用时，才允许他拿起筷子进餐。解决的方法：可用 AND 型信号量。各进程执行的程序段为：

```

Pi ( )

```

```

begin
Var chopstick: array[0, ..., 4] of semaphore=[1,1,1,1,1];
repeat
    wait(chopstick[i], chopstick[(i+1) mod 5]);
    eat;
    ...
    signal(chopstick[i], chopstick[(i+1) mod 5]);
    think;
until false;
end

```

方法（3）规定奇数号哲学家先拿他左边的筷子，然后再去拿右边的筷子；而偶数号哲学家则相反。

实现的方法：在第 i 个哲学家所执行的程序段中判定 i 的值是偶数还是奇数，然后再做资源申请。各进程执行的程序段为：

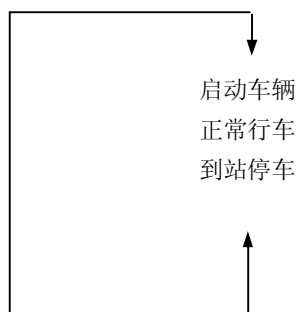
```

Pi ( )
begin
Var chopstick: array[0, ..., 4] of semaphore=[1,1,1,1,1];
repeat
    if i mod 2==1 then
        {wait(chopstick[i]);
        wait(chopstick[(i+1) mod 5]); }
    else
        { wait(chopstick[(i+1) mod 5]);
        wait(chopstick[i]);}
    eat;
    ...
    signal(chopstick[i]);
    signal(chopstick[(i+1) mod 5]);
    think;
until false;
end

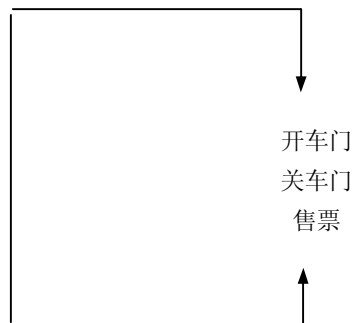
```

19. 设公共汽车上有一位司机和一位售票员，它们的活动如下：

司机进程：



售票员进程：



请分析司机与售票员之间的同步关系，如何用 P、V 操作实现。

【解答】为了安全起见，显然要求：关车门后才能启动车辆；到站后才能开车门。所以司机和售票员在到站、开门、关门、启动车辆这几个活动之间存在着同步关系。两个信号量 S1、S2 分别表示可以开车和可以开门，S1 的初值为 1，S2 的初值为 0。用 PV 操作实现司机进程和售票员进程同步的算法描述如下：

```
semaphore s1,s2:=1,0;
main()
{cobegin
  driver( );
  busman( );
coend
}
driver()
{
  while(1)
  p(s1);
  启动车辆;
  正常行车;
  到站停车;
  V (s2) ;
}
busman( )
{while(1)
  p(s2);
  开车门;
  等待上下乘客;
  关车门;
  V(s1);
  售票;
}
```

[注：为提高系统的效率，提高进程的并发性，将售票放在汽车行驶的过程中。]