第二章 进程的描述与控制

上节课学习内容：

1.理解前趋图和程序执行的概念；

2.掌握程序顺序执行和并发执行的特征；

3.理解进程的概念；

4.掌握进程的状态转换；

5.理解PCB的作用。

09:53

37

第二章 进程的描述与控制

今日学习任务整理单：

1.理解OS实现进程控制的方式；
2. 理解原语的概念；
3. 明确进程创建、终止、阻塞和唤醒；

进程控制

4. 理解进程同步、临界资源；
5. 掌握记录型信号量。

进程同步

今日学习资料：

1. 雨课堂中的MOOC资源-第二章进程管理（4）进程控制；
4. 教材2.3和2.4。

今日作业：认真阅读教材，重点理解信号量。

09:54

41

进程控制是用来管理从进程诞生、运行、到结束过程中的一切事情，故而要由操作系统内核中的原语来实现。

2.3.1 操作系统内核

操作系统内核：操作系统中与硬件紧密相关的模块（如中断处理程序、设备驱动程序等）以及运行频率较高（如时钟管理、进程调度等）的模块，它们常驻内存。

设置操作系统内核的作用：

(1) 便于对软件进行保护，防止被其它程序破坏；
(2) 提高操作系统的运行效率。

处理机执行状态：

(1) 系统态（管态、内核态）；
(2) 用户态（目态）。

2.3.1 操作系统内核

操作系统内核有两大功能：

1. 支撑功能

(1) 中断处理　　内核对中断进行“有限处理”后，转入相关进程接续后续处理

(2) 时钟管理　　定时产生时钟信号

(3) 原语操作　　原语是完成一定功能的过程，具有不可分割性。

2.3.1 操作系统内核

1. 支撑功能

2. 资源管理功能

(1) 进程管理　运行频率较高的模块（如进程调度、创建与撤销等），常被调用模块如同步原语等均放入内核。

(2) 存储器管理　运行频率较高也要放入内核。

(3) 设备管理　与硬件相关也要放入内核。

2.3.2 进程的创建

1. 进程的层次结构

进程的层次结构 体现为进程创建过程中生成的进程家族树，如unix下的进程树 图2-13。

而windows中不存在进程层次结构的概念，进程之间（通过句柄）只有控制与被控制的关系。

2.3.2 进程的创建

1. 进程的层次结构

2. 进程图(Process Graph)

父进程
子进程
祖先
子孙

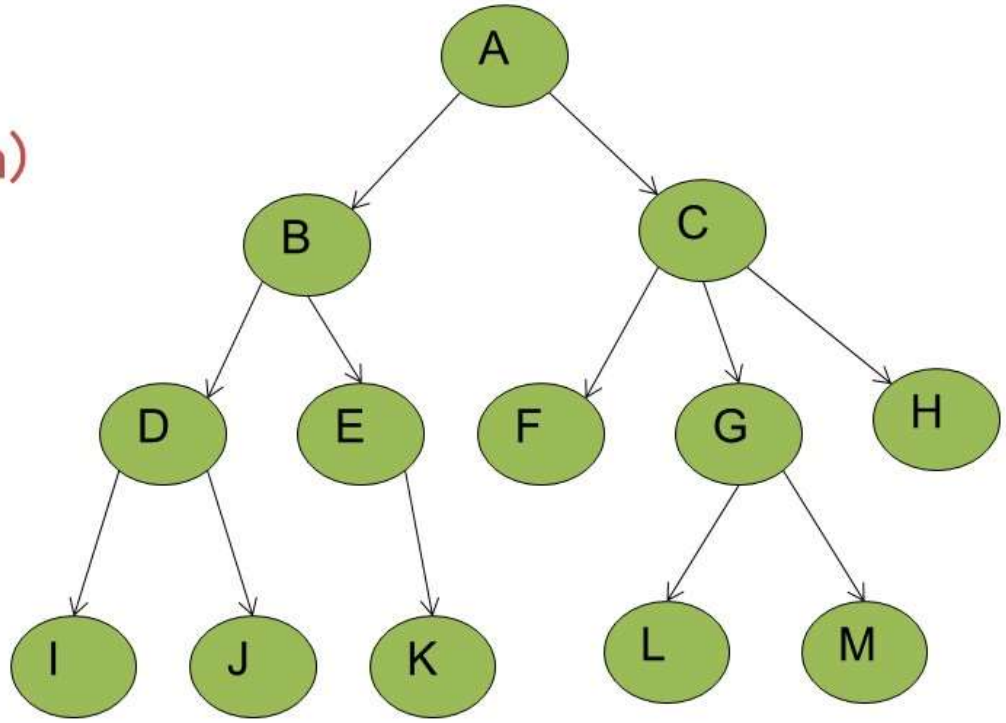


图 2-13 进程树

2.3.2 进程的创建

1. 进程的层次结构
2. 进程图(Process Graph)
3. 引起创建进程的事件

(1) 用户登录。

(2) 作业调度。

(3) 提供服务。

(4) 应用请求。

2.3.2 进程的创建

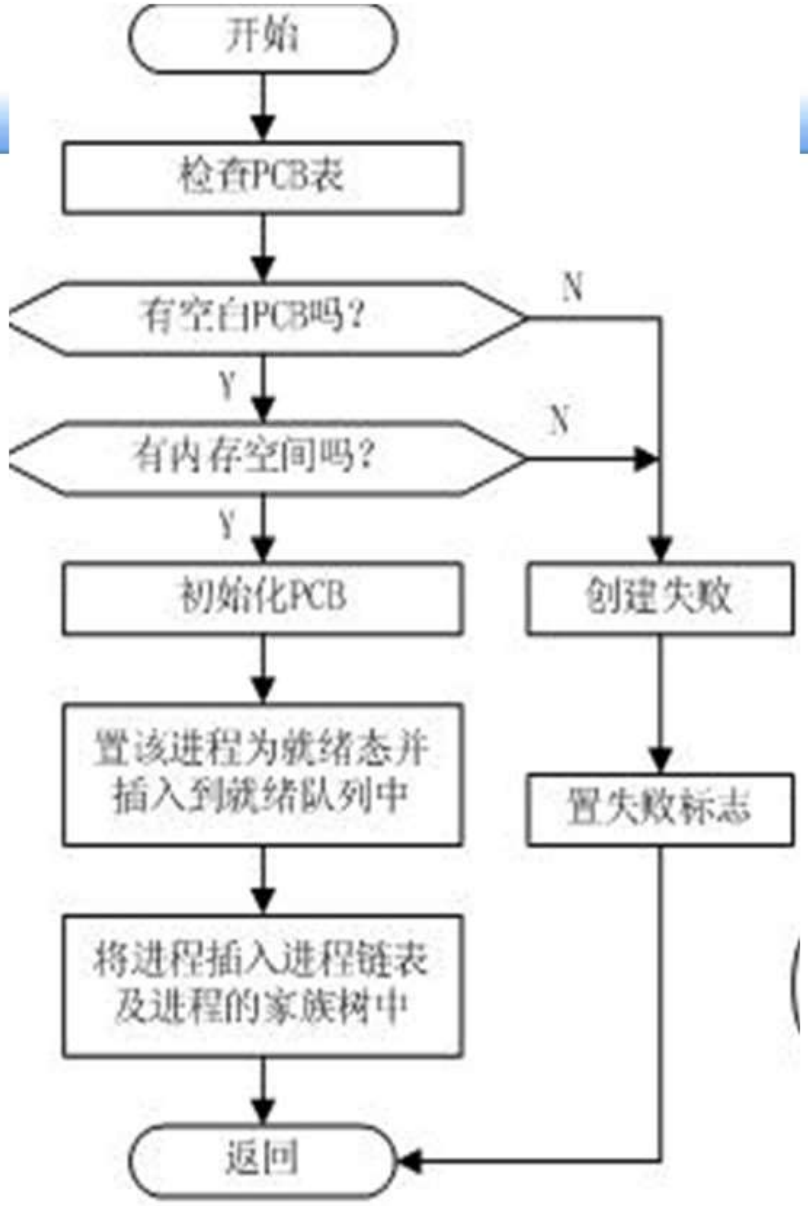
1. 进程的层次结构
2. 进程图(Process Graph)
3. 引起创建进程的事件
4. **进程的创建(Creation of Progress)**

(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) 将新进程插入就绪队列，如果进程就绪队列能够接纳新进程，便将新进程插入就绪队列。


开始
检查PCB表
有空白PCB吗?
N
Y
有内存空间吗?
N
Y
初始化PCB
创建失败
置该进程为就绪态并
插入到就绪队列中
置失败标志
将进程插入进程链表
及进程的家族树中
返回

创建原语流程图

2.3.3 进程的终止

1. 引起进程终止(Termination of Process)的事件

1) 正常结束

在任何计算机系统中，都应有一个用于表示进程已经运行完成的指示。例如，在批处理系统中，通常在程序的最后安排一条Holt指令或终止的系统调用。当程序运行到Holt指令时，将产生一个中断，去通知OS本进程已经完成。 在分时系统中，用户可利用Logs off去表示进程运行完毕， 此时同样可产生一个中断，去通知OS进程已运行完毕。

2.3.3 进程的终止

1. 引起进程终止(Termination of Process)的事件

1) 正常结束

2) 异常结束

在进程运行期间，由于出现某些错误和故障而迫使进程终止。

① 越界错误。

② 保护错。访问不当；

③ 非法指令。把数据当成了指令；

④ 特权指令错。

⑤ 运行超时。

⑥ 等待超时。

⑦ 算术运算错。例如被0除；

⑧ I/O故障。

2.3.3 进程的终止

引起进程终止(Termination of Process)的事件

1) 正常结束

2) 异常结束

3) 外界干预

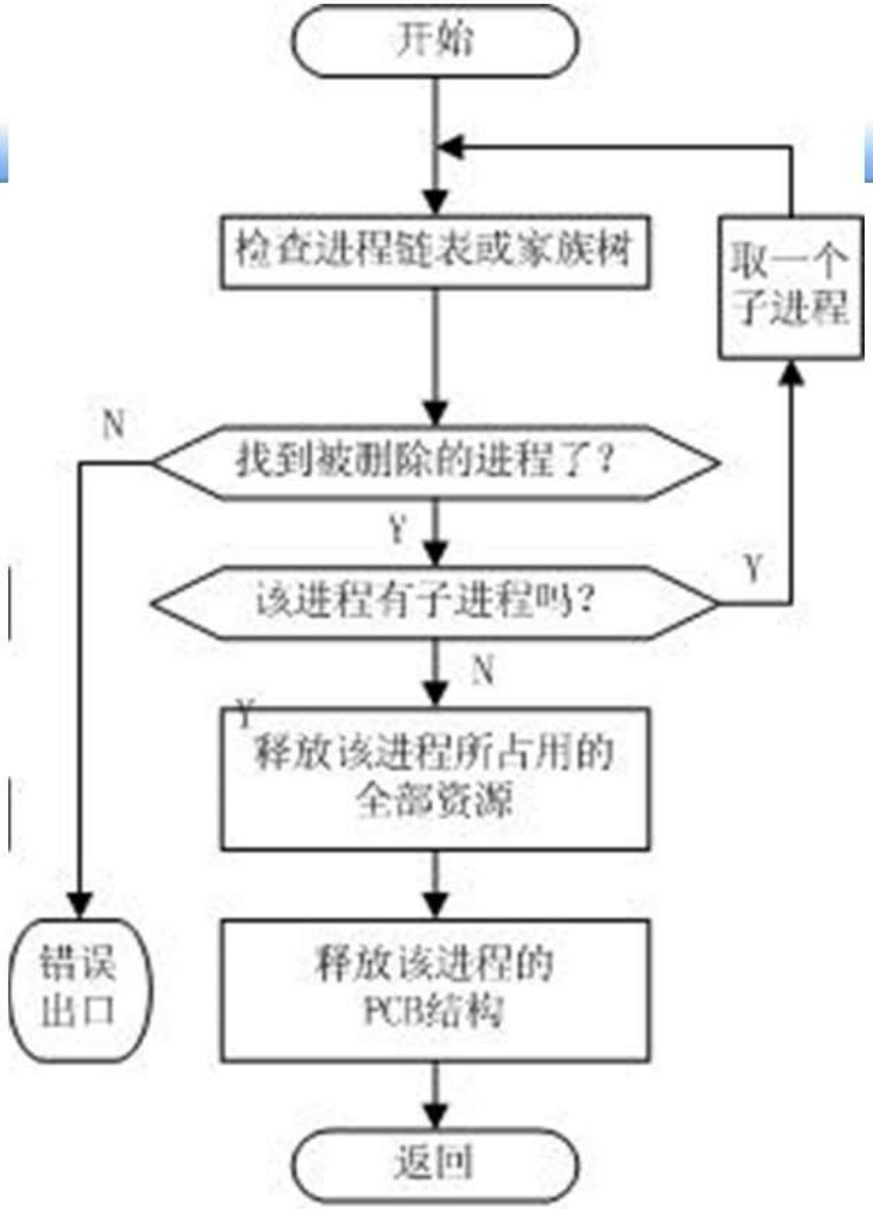
外界干预并非指在本进程运行中出现了异常事件，而是指进程应外界的请求而终止运行。这些干预有：

① 操作员或操作系统干预。例如，发生了死锁；

② 父进程请求。

③ 父进程终止。 当父进程终止时，OS也将他的所有子孙进程终止。

终止原语流程图


开始
检查进程链表或家族树
取一个子进程
N
找到被删除的进程了?
Y
Y
该进程有子进程吗?
N
Y
释放该进程所占用的全部资源
错误出口
释放该进程的PCB结构
返回

2.3.3 进程的终止

1. 引起进程终止(Termination of Process)的事件
2. 进程的终止过程

(1) 根据被终止进程的标识符，从PCB检索出该进程的PCB；

(2) 若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真，用于指示该进程被终止后应重新进行调度。

(3) 终止该进程的所有子孙进程，以防他们成为不可控的进程。

(4) 将被终止进程所拥有的全部资源，或者归还给其父进程，或者归还给系统。

(5) 将被终止进程(它的PCB)从所在队列(或链表)中移出， 等待其他程序来搜集信息。

2.3.2 进程的阻塞与唤醒

1. 引起进程阻塞和唤醒的事件

1）向系统请求共享资源失败

2）等待某种操作的完成

3）新数据尚未到达

4）等待新任务的到达

2.3.2 进程的阻塞与唤醒

1. 引起进程阻塞和唤醒的事件

2. 进程阻塞过程

正在执行的进程，当发现阻塞事件时，由于无法继续执行，于是进程便通过调用阻塞原语block把自己阻塞。可见，进程的阻塞是进程自身的一种主动行为。

进入block过程后，由于此时该进程还处于执行状态，所以应先立即停止执行，把进程控制块中的现行状态由“执行”改为阻塞，并将PCB插入阻塞队列。 最后，转调度程序进行重新调度，将处理机分配给另一就绪进程，并进行切换。

2.3.2 进程的阻塞与唤醒

1. 引起进程阻塞和唤醒的事件

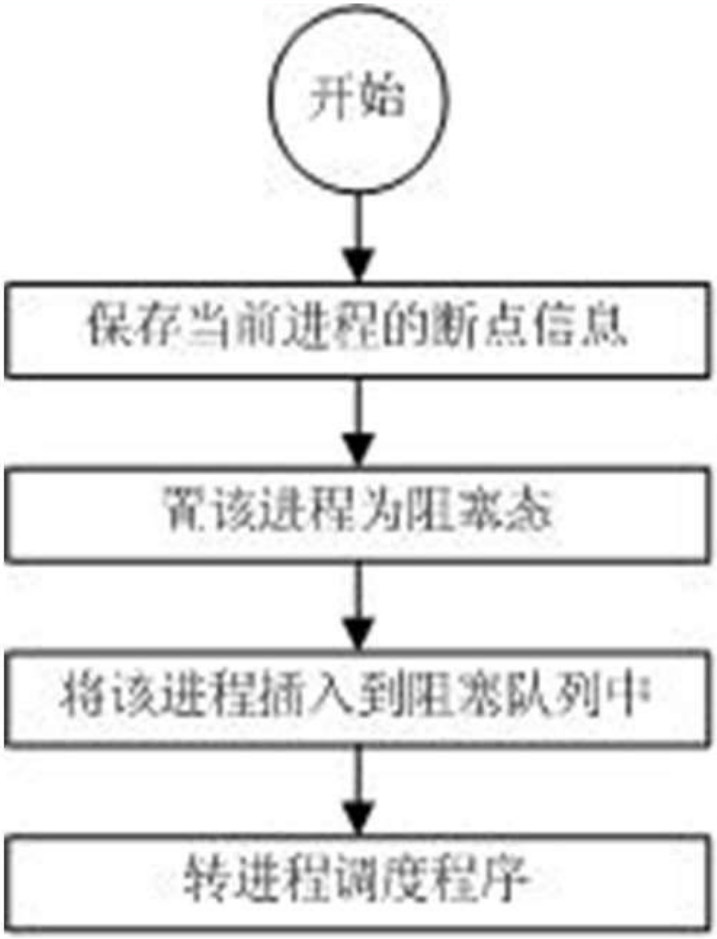
2. 进程阻塞过程

3. 进程唤醒过程

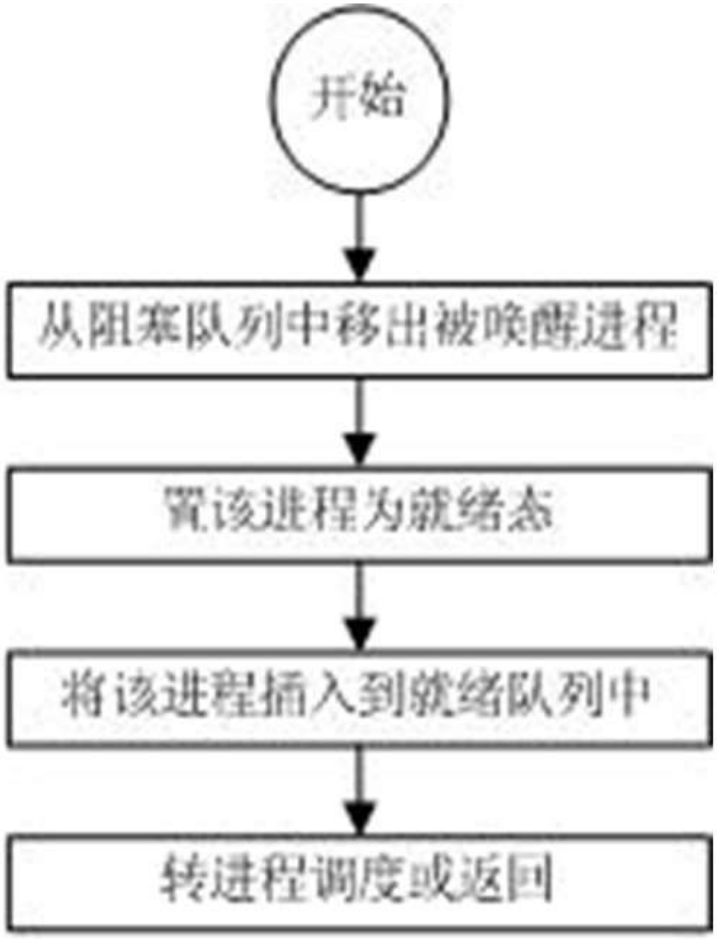
当被阻塞进程所期待的事件出现时，如I/O完成或其所期待的数据已经到达，则由有关进程(比如，用完并释放了该I/O设备的进程)调用唤醒原语wakeup( )，将等待该事件的进程唤醒。

唤醒原语执行的过程是：首先把被阻塞的进程从等待该事件的阻塞队列中移出，将其PCB中的现行状态由阻塞改为就绪，然后再将该PCB插入到就绪队列中。

2.3 进程控制



阻塞原语流程图



唤醒原语流程图

2.3.5 进程的挂起与激活

1. 进程的挂起

当出现了引起进程挂起的事件时，比如，用户进程请求将自己挂起，或父进程请求将自己的某个子进程挂起， 系统将利用挂起原语suspend( )将指定进程或处于阻塞状态的进程挂起。

挂起原语的执行过程是：首先检查被挂起进程的状态，若处于活动就绪状态，便将其改为静止就绪；对于活动阻塞状态的进程，则将之改为静止阻塞。 为了方便用户或父进程考查该进程的运行情况而把该进程的PCB复制到某指定的内存区域。最后，若被挂起的进程正在执行，则转向调度程序重新调度。

2.3.5 进程的挂起与激活

1. 进程的挂起

2. 进程的激活过程

当发生激活进程的事件时，例如，父进程或用户进程请求激活指定进程，若该进程驻留在外存而内存中已有足够的空间时，则可将在外存上处于静止就绪状态的进程换入内存。这时，系统将利用激活原语active( )将指定进程激活。

> 激活原语先将进程从外存调入内存，检查该进程的现行状态，若是静止就绪，便将之改为活动就绪；若为静止阻塞便将之改为活动阻塞。假如采用的是抢占调度策略，则每当有新进程进入就绪队列时，应检查是否要进行重新调度，即由调度程序将被激活进程与当前进程进行优先级的比较，如果被激活进程的优先级更低，就不必重新调度；否则，立即剥夺当前进程的运行，把处理机分配给刚被激活的进程。

2.4 进程同步

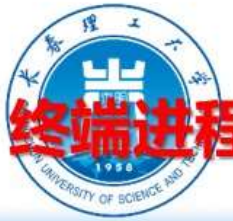
- 本节先要讨论的问题是：

(1) 进程并发执行一定会导致结果不唯一吗？

(2) 在什么情况下会顺利执行

(3) 在什么情况下会出现错误

例1：一个自动图书借阅系统，有两台终端可同时借阅，终端进程描述如下：



终端T1进程：

```
void TT(BOOK *bk)
{  /*BOOK为图书类型*/
  int n;
  n=bk->count;  /*该类图书的数量*/
  if( n>=1)
   {
     n=n-1;
     借出一本书;
     bk->count=n;
   }
}
```

终端T2进程：

```
void TT(BOOK *bk)
{  /*BOOK为图书类型*/
  int n;
  n=bk->count;  /*该类图书的数量*/
  if( n>=1)
   {
     n=n-1;
     借出一本书;
     bk->count=n;
   }
}
```

本节要解决的问题是：

(1) 并发进程之间有什么关系

(2) 进程并发执行相互间会有什么影响

(3) 结果的不可再现性（与时间有关的错误）

2.4.1 进程同步的基本概念

1. 两种形式的制约关系

(1) 间接相互制约关系。

(2) 直接相互制约关系。

2.4.1 进程同步的基本概念

1. 两种形式的制约关系

2. 临界资源(Critical Resouce)

生产者-消费者(producer-consumer)问题是一个著名的进程同步问题。它描述的是：有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费。为使生产者进程与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池，生产者进程将它所生产的产品放入一个缓冲区中； 消费者进程可从一个缓冲区中取走产品去消费。尽管所有的生产者进程和消费者进程都是以异步方式运行的，但它们之间必须保持同步，即不允许消费者进程到一个空缓冲区去取产品；也不允许生产者进程向一个已装满产品且尚未被取走的缓冲区中投放产品。