華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

操作系统原理

第4章 进程及进程管理

华中科技大学计算机学院 谢美意

目 录

CONTENT

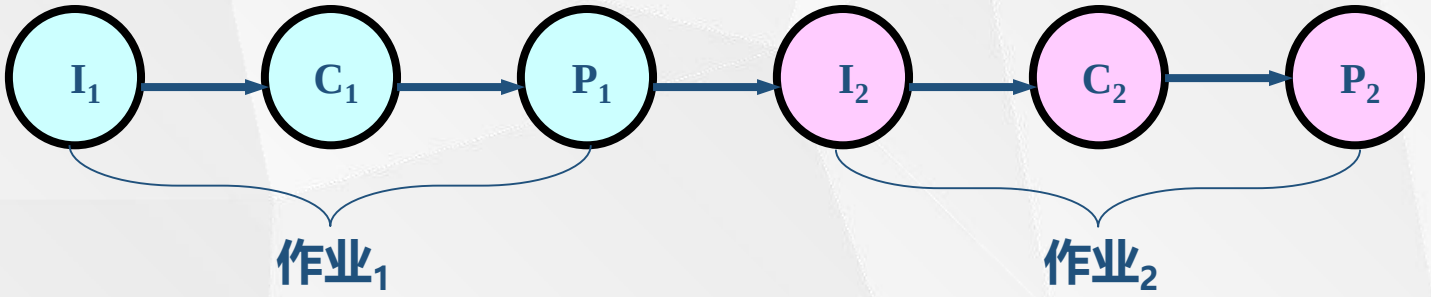
- 进程的引入
- 进程概念
- 进程控制
- 进程的相互制约关系
- 进程互斥与同步的实现
- 进程通信
- 线程

華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

进程的引入

顺序程序

- 程序的一次执行过程称为一个计算，它由许多简单操作所组成。
- 一个计算的若干操作必须按照严格的先后次序顺序地执行，这个计算过程就是程序的顺序执行过程。



I：输入操作；C：计算操作；P：输出操作

单道系统的工作情况

顺序程序的特点

- **顺序性**

  处理机的操作严格按照程序所规定的顺序执行。

- **封闭性**

  程序一旦开始执行，其计算结果不受外界因素的影响。

- **可再现性**

  程序执行的结果与它的执行速度无关 (即与时间无关)，而只与初始条件有关。

并发程序

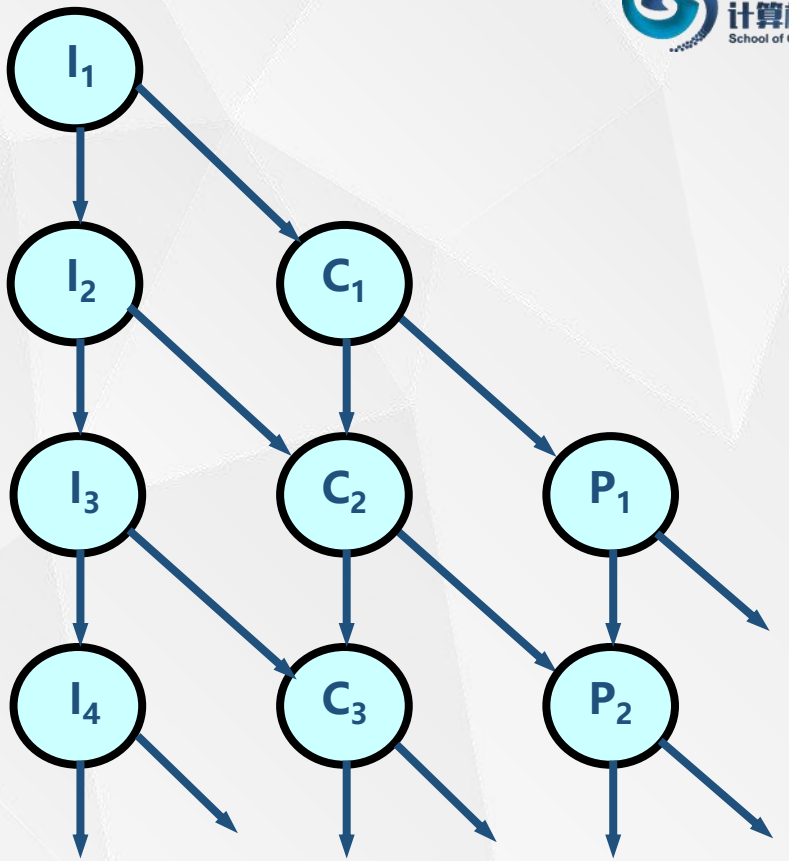
若干个程序段同时在系统中运行，这些程序段的执行在时间上是重叠的，一个程序段的执行尚未结束，另一个程序段的执行已经开始，即使这种重叠是很小的一部分，也称这几个程序段是并发执行的。

并发执行的描述方法

cobegin

S_1；S_2；…；S_n；

coend



多道系统的工作情况

- 哪些程序段的执行必须是顺序的？为什么？
- 哪些程序段的执行可以是并行的？为什么？

并发程序的特点

① 程序与计算不再一一对应

例1：

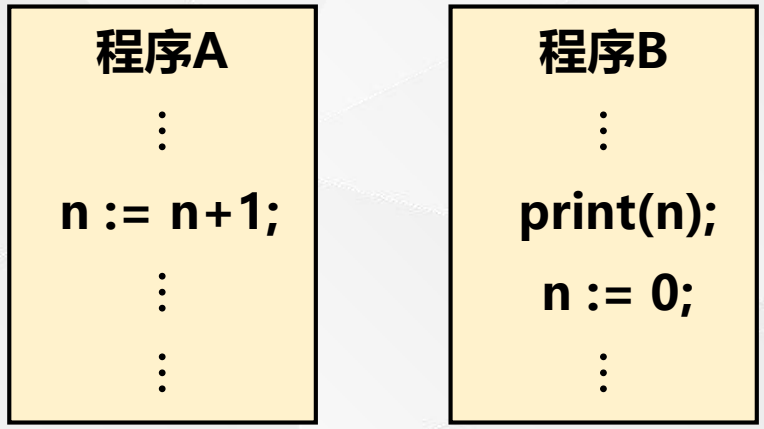
例2：

② 程序并发执行的相互制约

- 间接的相互制约关系 —— 资源共享
- 直接的相互制约关系 —— 公共变量

③ 失去程序的封闭性和可再现性

若一个程序的执行可以改变另一个程序的变量，那么，后者的输出就可能有赖于各程序执行的相对速度，即失去了程序的封闭性特点。



程序A的语句n :=n+1与程序B的两个语句的关系	之前	之后	之间
n的赋值	10	10	10
打印的结果	11	10	10
n的最终赋值	0	1	0

程序并发执行时的结果与各并发程序的相对速度有关，即给定相同的初始条件，若不加以控制，也可能得到不同的结果，此为**与时间有关的错误**。

进程概念

进程（Process）定义

进程是指一个具有一定独立功能的程序关于某个数据集合的一次运行活动。

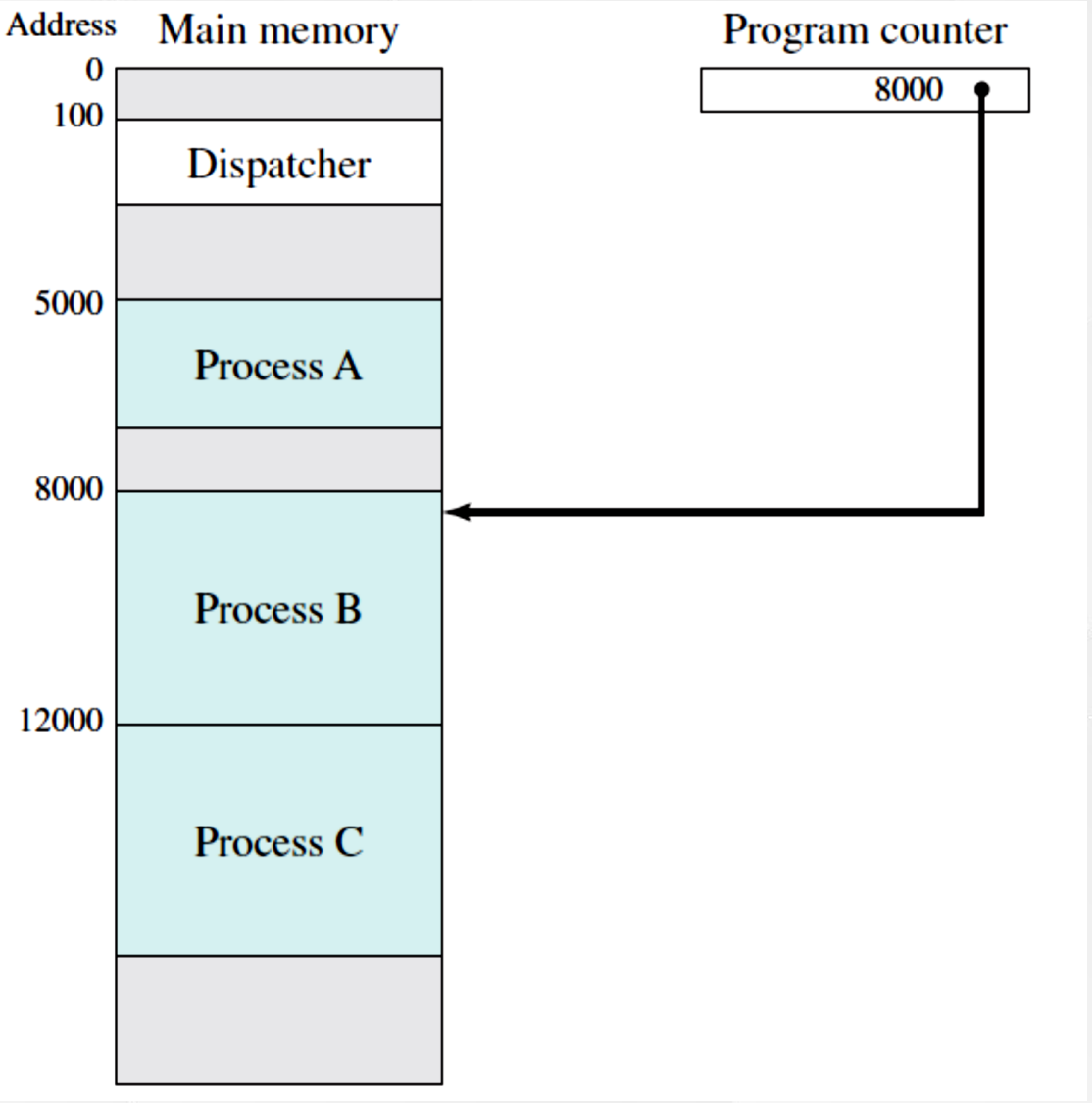
进程与程序的区别

① 程序是静态的概念，进程是动态的概念；

② 进程是一个独立运行的活动单位；

③ 进程是竞争系统资源的基本单位；

④ 一个程序可以对应多个进程，一个进程至少包含一个程序。



1	5000	27	12004
2	5001	28	12005
3	5002	----------Time-out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------Time-out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	]103	36	5007
11	]104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------Time-out	
16	8003	41	100
----------I/O request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------Time-out	

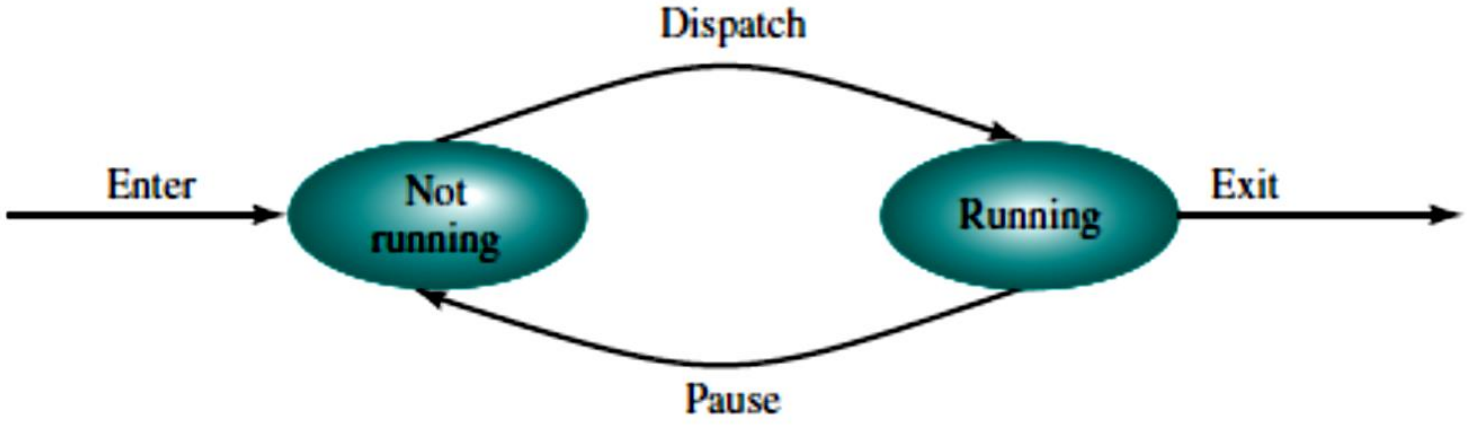
内存中有A、B、C三个进程
和操作系统调度程序

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

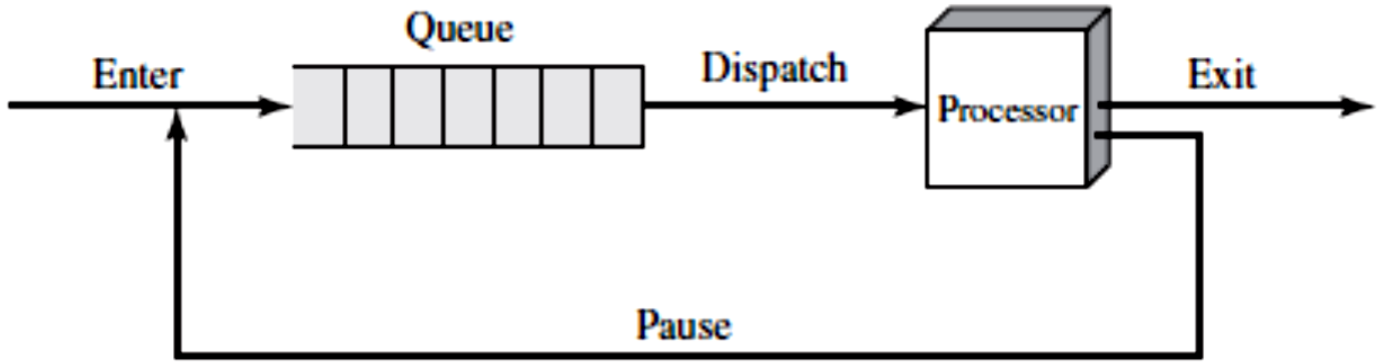
(a) Trace of process A (b) Trace of process B (c) Trace of process C

某一时刻A、B、C三个进程的执行轨迹

进程的状态



(a) State transition diagram



(b) Queueing diagram

进程的两个基本状态

进程的状态

进程的三个基本状态

① 运行状态(running)

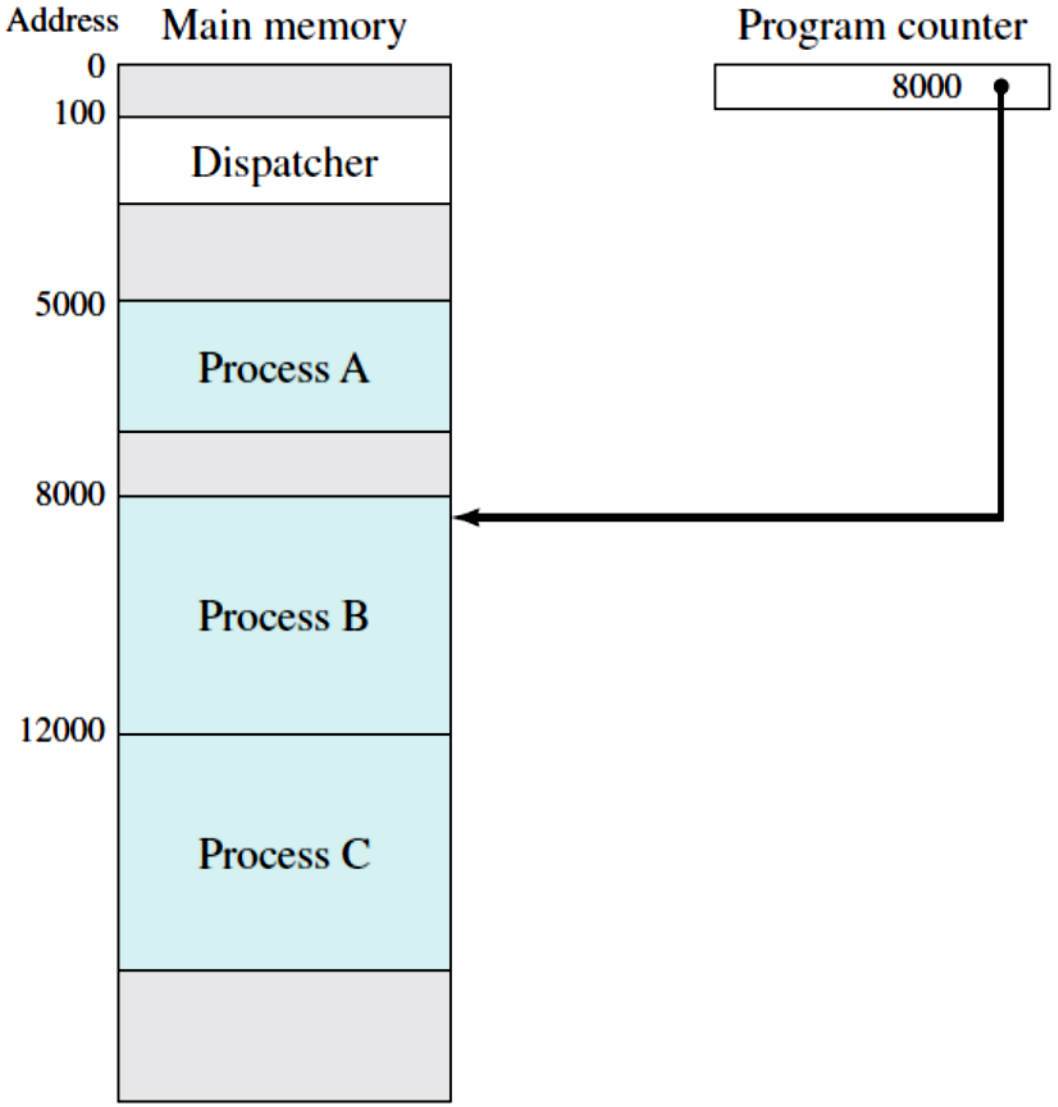
该进程已获得运行所必需的资源，它的程序正在处理机上执行。

② 等待状态(wait)

进程正等待着某一事件的发生而暂时停止执行。这时，即使给它CPU控制权，它也无法执行。

③ 就绪状态(ready)

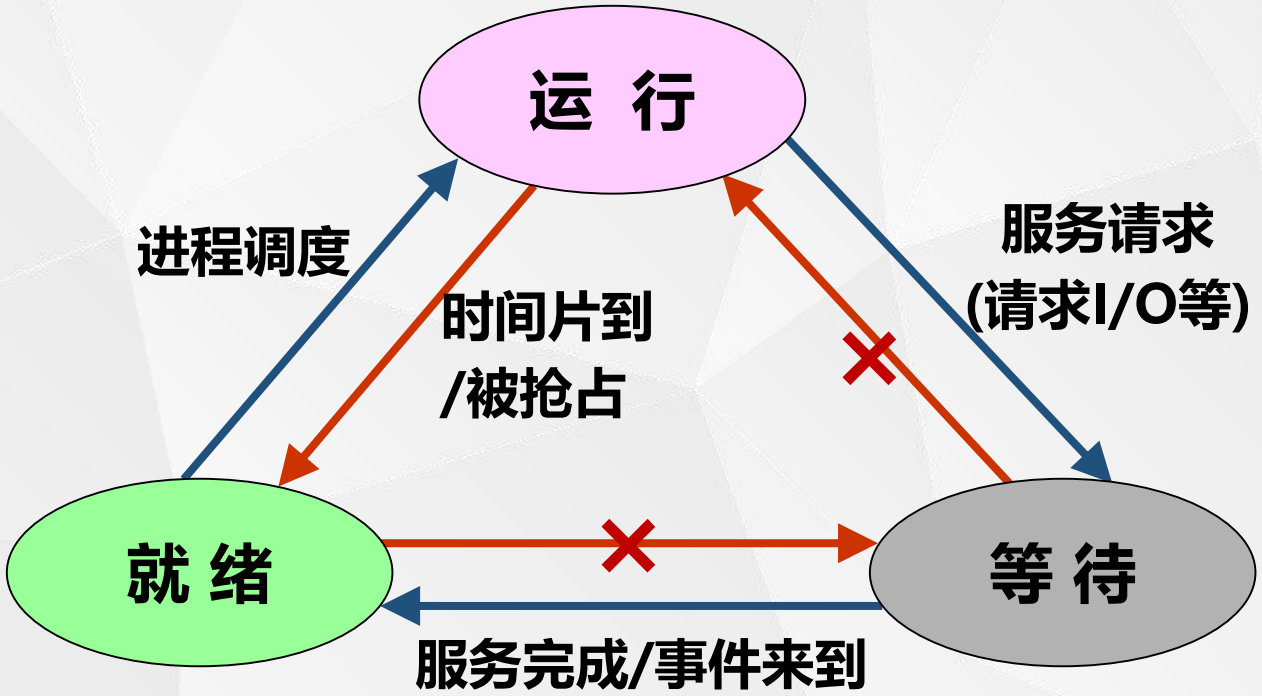
进程已获得除CPU之外的运行所必需的资源，一旦得到CPU控制权，立即可以运行。



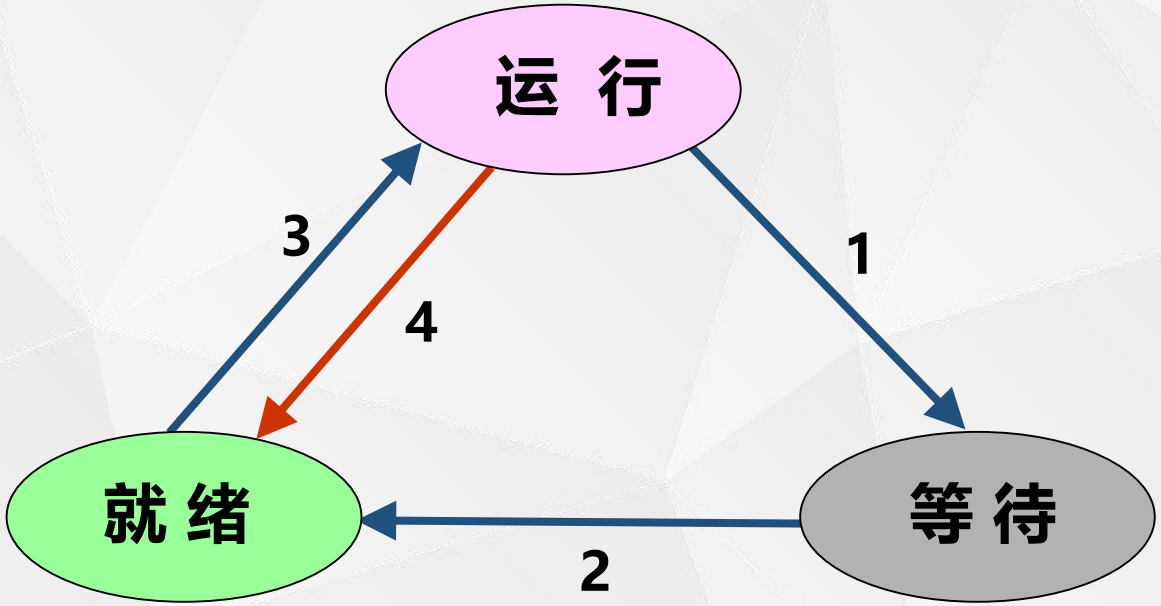
1	5000	27	12004
2	5001	28	12005
3	5002	----------Time-out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------Time-out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	]103	36	5007
11	]104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------Time-out	
16	8003	41	100
----------I/O request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------Time-out	

在红色箭头指向的这个时刻，A、B、C三个进程分别处于什么状态?

进程状态变迁图



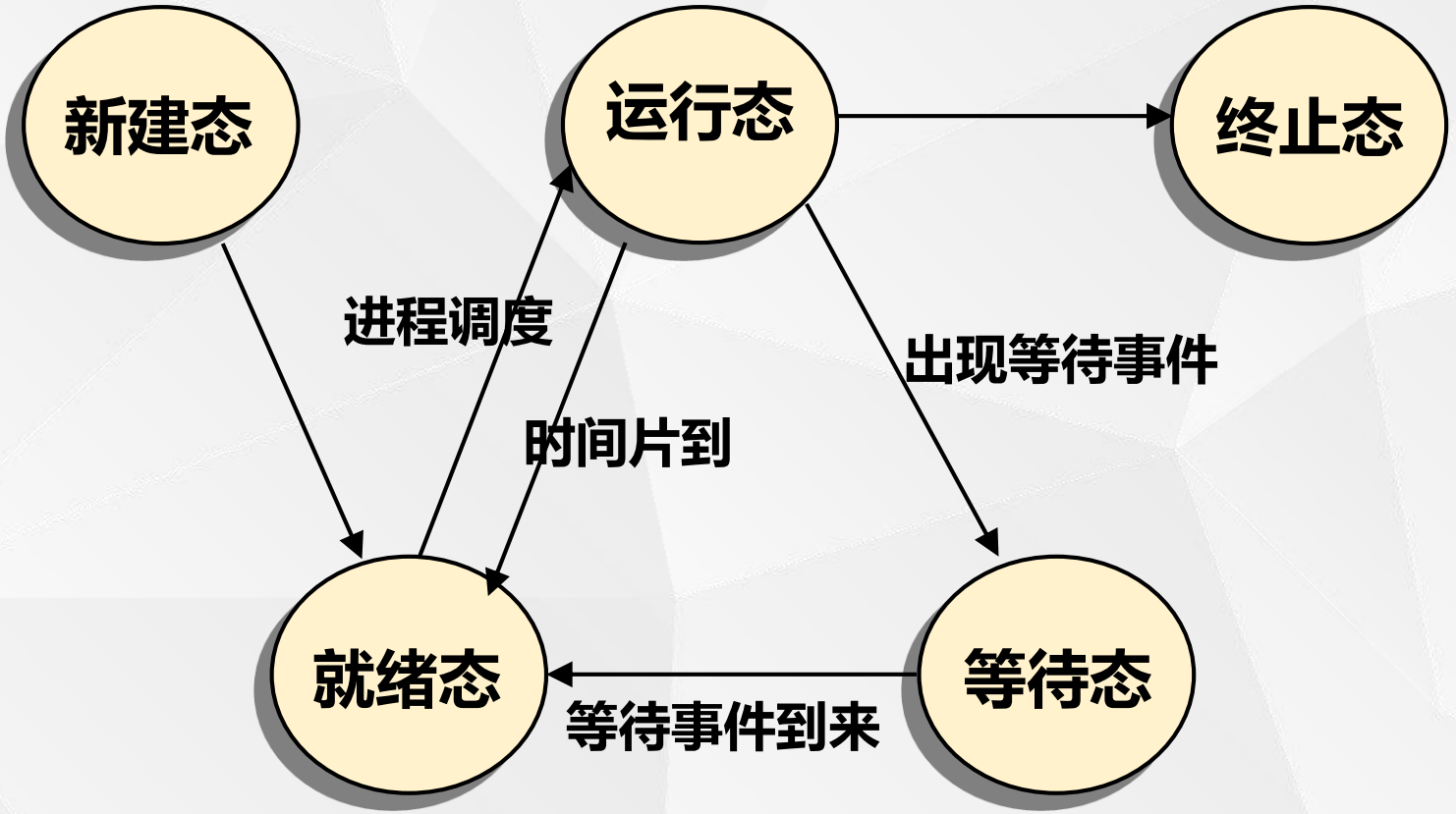
因果变迁



变迁1——> 变迁3，是否会发生？需要什么条件？

变迁4——> 变迁3，是否会发生？需要什么条件？

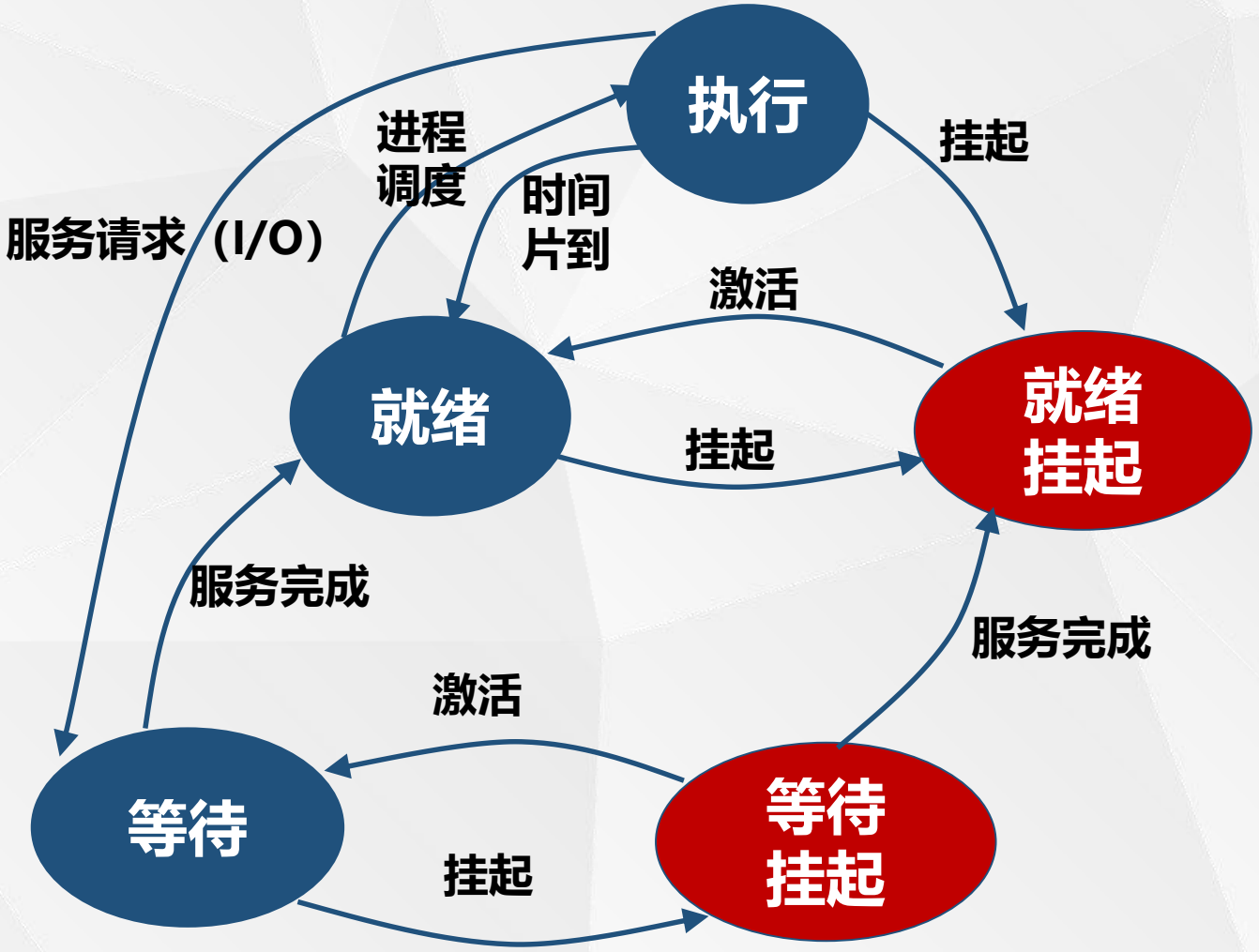
进程的五个状态及其转换




```
main()
{
    sleep(2);
    return();
}
```

讨论：此程序执行过程中的进程状态变迁。

具有挂起状态的进程状态变迁图



挂起：把内存中的进程换出到磁盘中，以便为其他进程让出空间。

- 等待→等待挂起
- 等待挂起→等待
- 等待挂起→就绪挂起
- 就绪挂起→就绪
- 就绪→就绪挂起
- 执行→就绪挂起

进程描述

- **进程控制块**

  描述进程与其他进程、系统资源的关系以及进程在各个不同时期所处的状态的数据结构，称为进程控制块 (process control block，PCB)。

- **进程的组成（进程映像）**

进程控制块

程序与数据

① 程序与数据

描述进程本身所应完成的功能。

② 进程控制块PCB

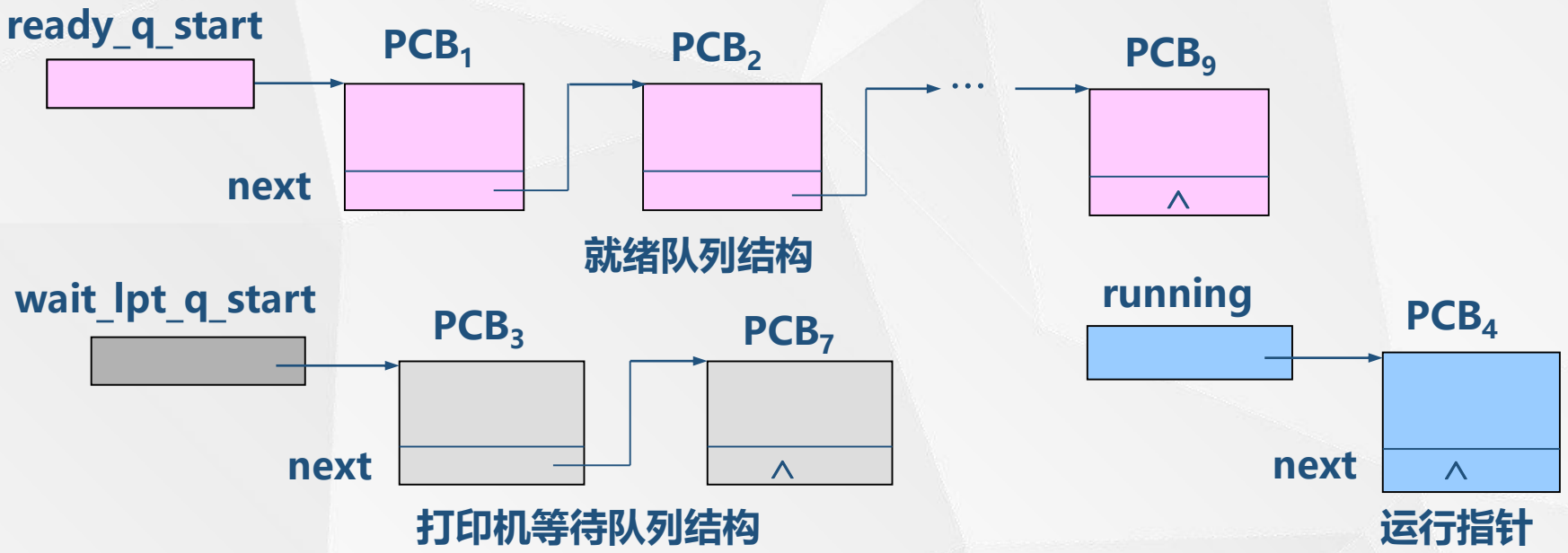
进程的动态特征，该进程与其他进程和系统资源的关系。

进程控制块的主要内容

① **进程标识符**　　进程符号名或内部 id号

② **进程当前状态**　　本进程目前处于何种状态

Q：大量的进程如何组织?



进程控制块的主要内容 （续）

③ 当前队列指针next

该项登记了处于同一状态的下一个进程的 PCB地址。

④ 进程优先级

反映了进程要求CPU的紧迫程度。

⑤ CPU现场保护区

当进程由于某种原因释放处理机时，CPU现场信息被保存在PCB的该区域中。

⑥ 通信信息

进程间进行通信时所记录的有关信息。

⑦ 家族联系

指明本进程与家族的联系

⑧ 占有资源清单

- Linux中的PCB结构叫task_struct，每个进程都把自己的信息放在一个task_struct结构里， task_struct包含了这些内容：
  - **标示符**： 描述本进程的唯一标示符，用来区别其他进程。
  - **状态**： 任务状态，退出代码，退出信号等。
  - **优先级**： 相对于其他进程的优先级。
  - **程序计数器**： 程序中即将被执行的下一条指令的地址。
  - **内存指针**： 包括程序代码和进程相关数据的指针，还有和其他进程共享的内存块的指针
  - **上下文数据**： 进程执行时处理器的寄存器中的数据。
  - **I/O状态信息**：包括显示的I/O请求，分配给进程的I / O设备和被进程使用的文件列表。
  - **记账信息**： 可能包括处理器时间总和，使用的时钟数总和，时间限制，记账号等。
- task_struct的定义可以在include/linux/sched.h里找到。所有运行在系统里的进程都以task_struct链表的形式存在内核里。

进程控制

1. 进程控制的概念

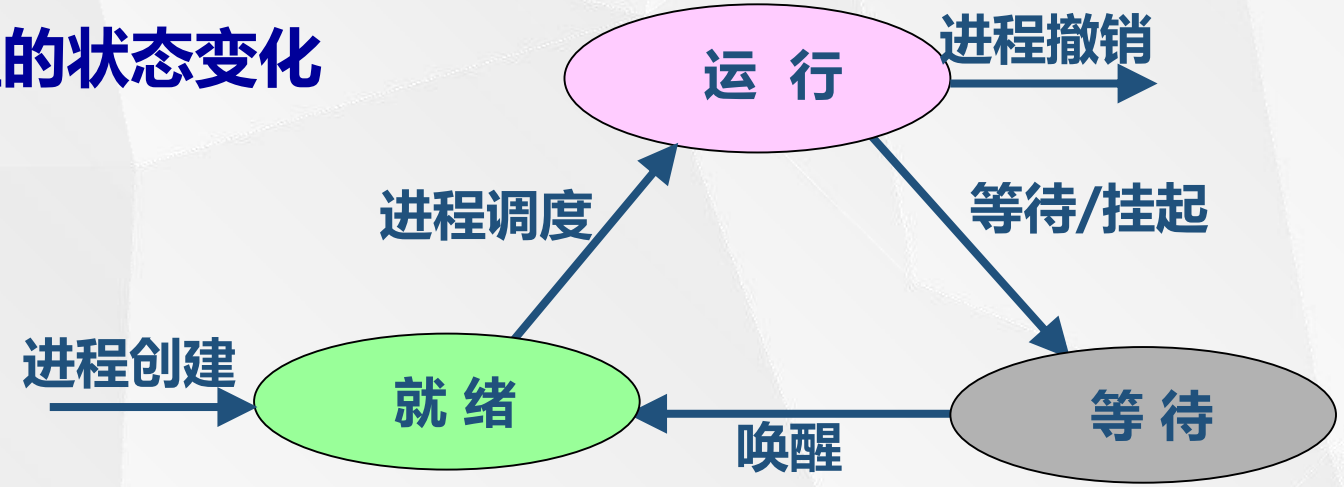
(1) 进程控制的职责

对系统中的进程实施有效的管理，负责进程状态的改变。

① 常用的进程控制原语（什么叫原语？）

创建原语、撤消原语、等待原语、唤醒原语

② 进程的状态变化



2 进程创建

① 进程创建原语的形式

create (name，priority)

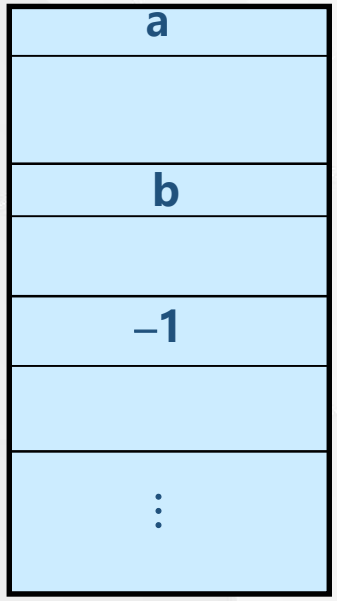
- name为被创建进程的标识符
- priority为进程优先级

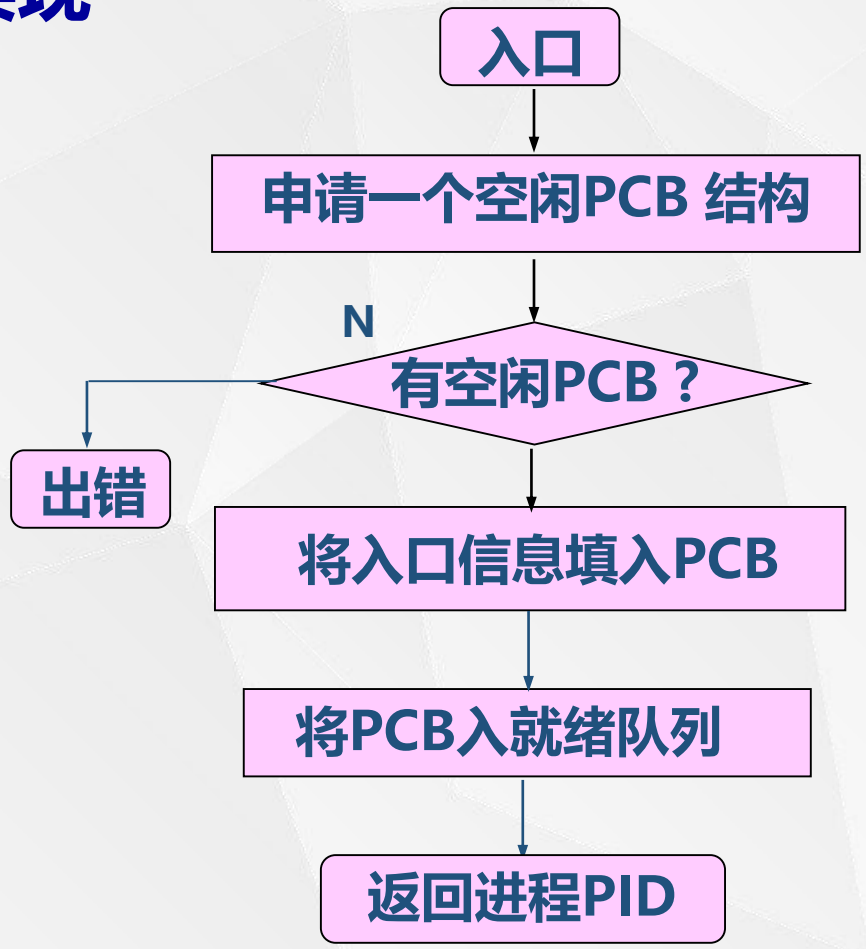
② 进程创建原语的功能

创建一个具有指定标识符的进程，建立进程的PCB结构。

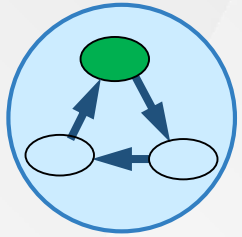
③ 进程创建原语的实现

- PCB池

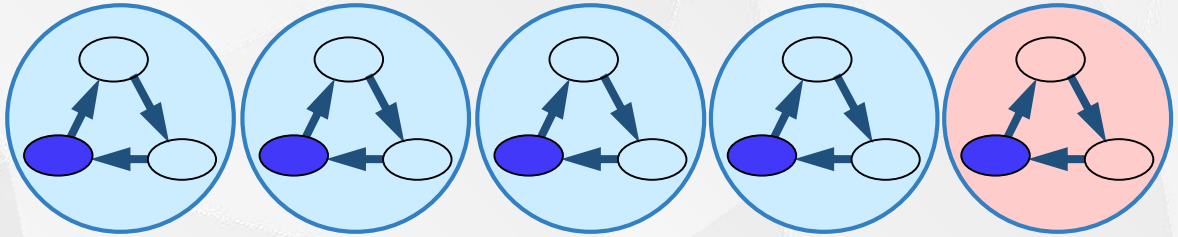




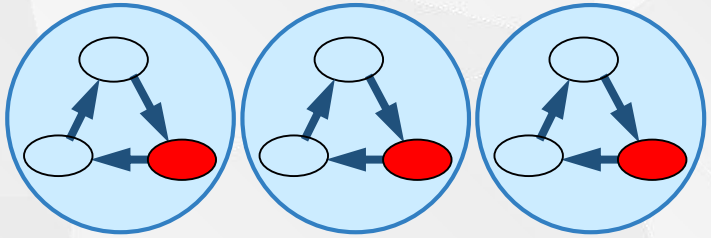
运行指针



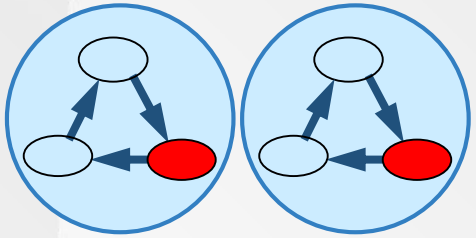
就绪队列



等待队列1



等待队列2



创建进程的典型原因

新建一个批处理任务	在一个批处理作业流中，当操作系统完成了上一个作业，从作业流中读出下一个作业时。
交互式登录	用户通过一个交互式终端登录到某个系统中时。
操作系统创建，以提供某种服务	现代操作系统往往需要创建某种服务（如打印服务）以辅助用户进程完成其工作。
被已经存在的进程所创建	例如为了提高多核系统的资源利用率，某用户进程创建一个子进程。

Linux——fork()

Linux用fork()创建一个子进程，它从父进程继承整个进程的地址空间，包括：**进程上下文、进程堆栈、内存信息、打开的文件描述符、信号控制设置、进程优先级、进程组号、当前工作目录、根目录、资源限制、控制终端**等。

① 为新进程分配一个新的PCB结构；

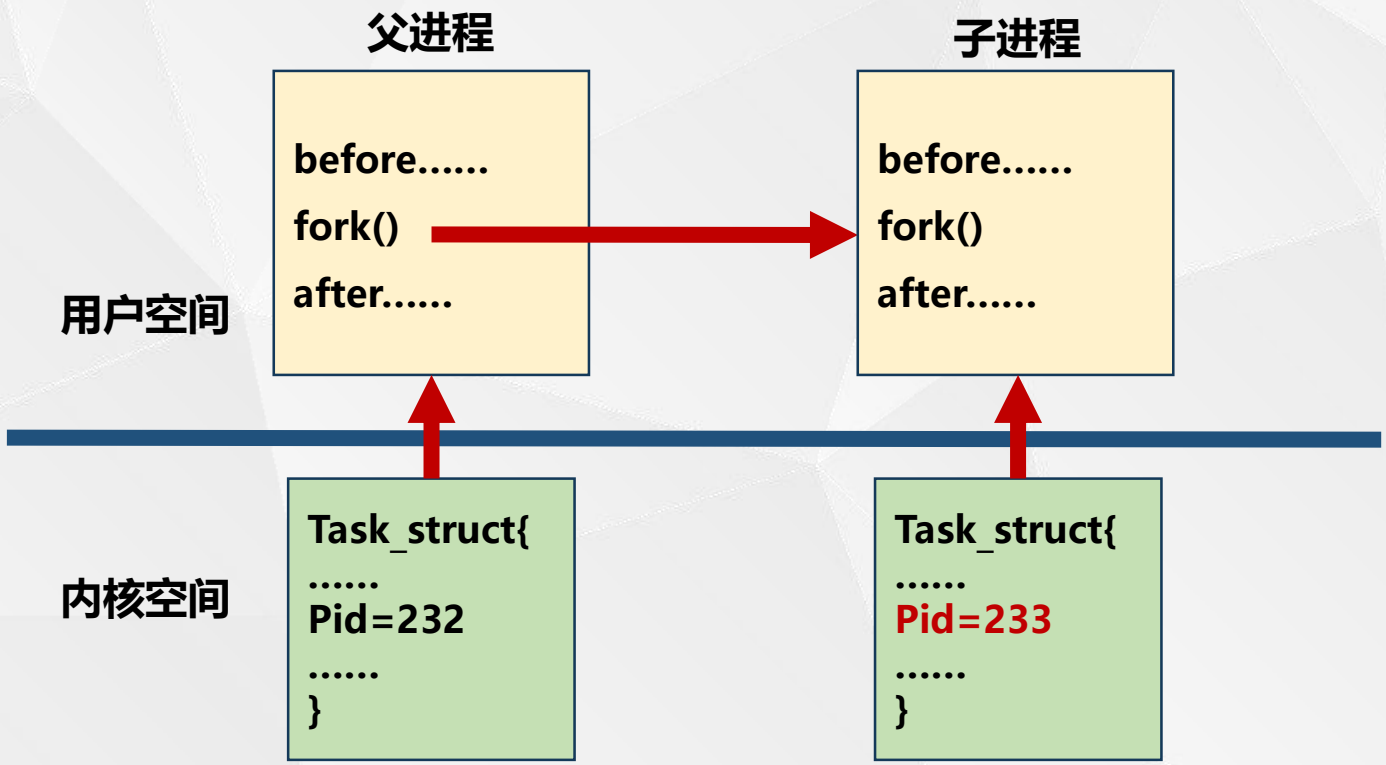
② 为子进程赋一个唯一的进程标识号 (PID)；

③ 为子进程做一个父进程上下文的逻辑副本。**这意味着父子进程将执行完全相同的代码。**

④ 增加与该进程相关联的文件表和索引节点表的引用数。这就意味着父进程打开的文件子进程可以继续使用。

⑤ **对父进程返回子进程的进程号，对子进程返回零。**

Linux——fork()


父进程
子进程
before……
fork()
after……
before……
fork()
after……
用户空间
内核空间
Task_struct{
……
Pid=232
……
}
Task_struct{
……
Pid=233
……
}

fork举例（1）

```
main(){
    int pid;
    pid=fork( );
    if  (pid==0)
        printf("I'm the child process!\n");
    else if (pid>0)
        printf("I'm the parent process!\n");
    printf("Fork TEST!\n");
}
```

Q：这个程序的执行结果是什么？

① 子进程打印

I'm the child process!

Fork TEST!

② 父进程打印

I'm the parent process!

Fork TEST!

fork举例(2)


```
main(){
    int pid1, pid2;
    pid1=fork();
    pid2=fork();
    printf("pid1=%d\n", pid1);
    printf ("pid2=%d\n", pid2);
}
```

Q：这个程序执行后的输出结果有几行?

Linux——exec()

功能：更换进程执行代码，更换正文段，数据段。

格式：exec (文件名，参数表，环境变量表)

例： execlp（“max”,15,18,10,0); execvp（“max”,argp);

```
main()
{
    if(fork()==0)
        {    printf（“a”）;
             execlp（“file1”,0);
             printf（“b”）;
        }
    printf（“c”）;
}
```

```
file1:
main()
{
     printf（“d”）;
}
```

以下哪个输出结果是正确的?

cad

abdc

adbcc

adbc

3 进程撤销

① 进程撤销原语的形式

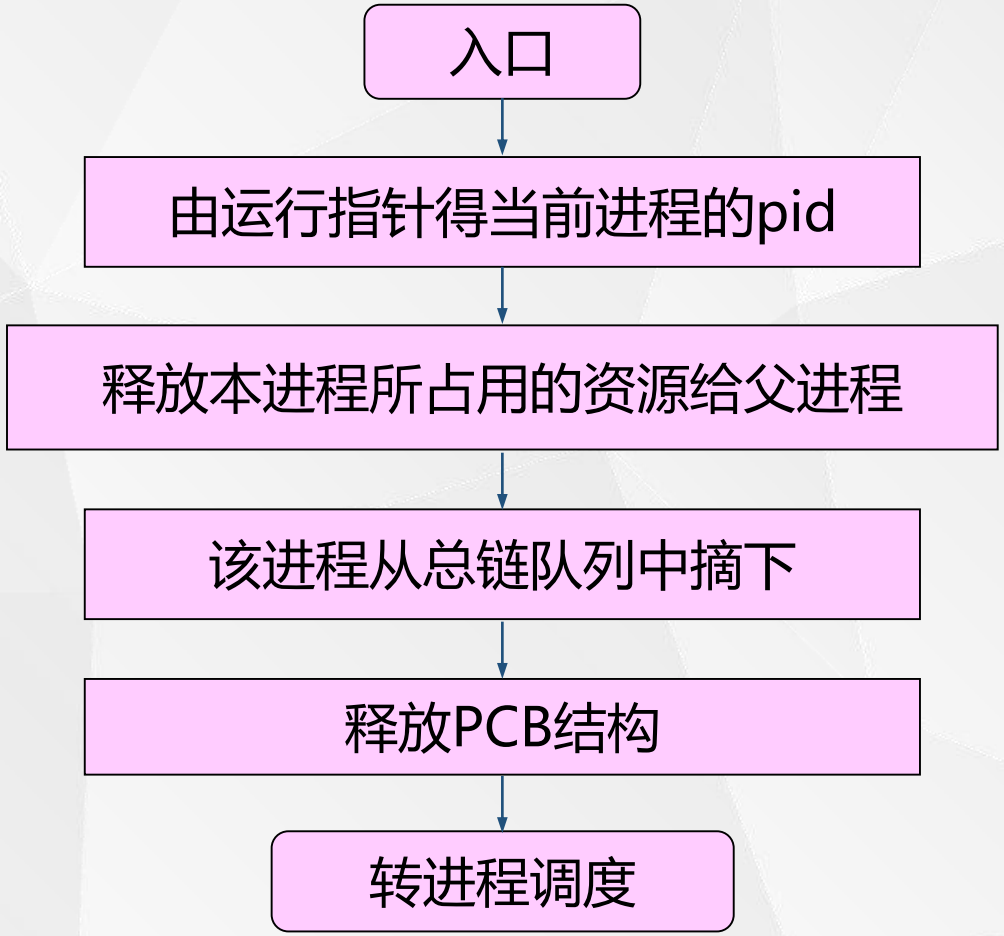
进程完成任务或希望终止时使用进程撤消原语。

Kill ( ) 或 exit( )

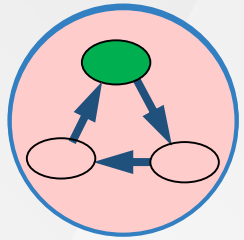
② 进程撤销原语的功能

撤消当前运行的进程。将该进程的PCB结构归还到PCB资源池，所占用的资源归还给父进程，从总链队列中摘除它，然后转进程调度程序。

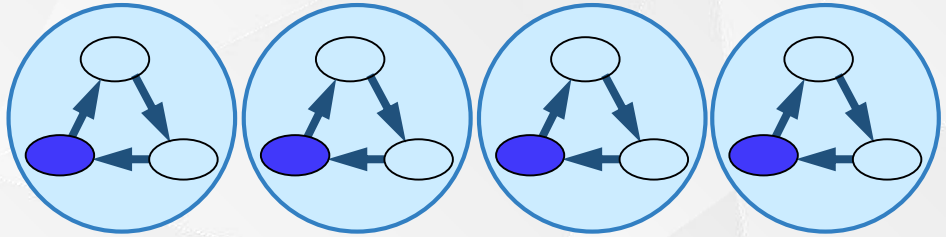
③ 进程撤销原语的实现


入口
由运行指针得当前进程的pid
释放本进程所占用的资源给父进程
该进程从总链队列中摘下
释放PCB结构
转进程调度

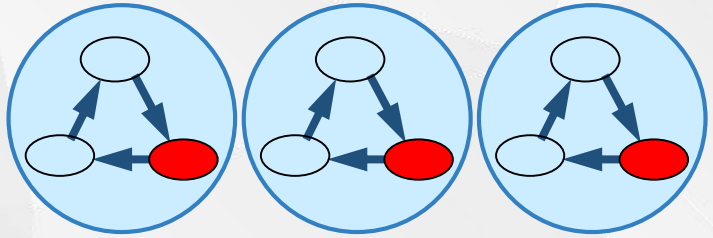
运行指针



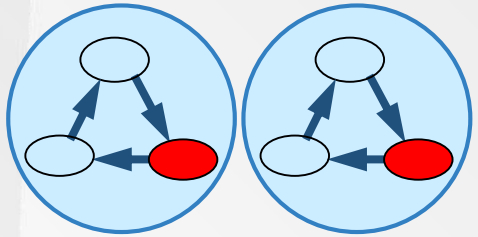
就绪队列



等待队列1



等待队列2



④ 进程撤销的主要原因

- 进程正常运行结束
- 进程运行中出错，如执行了非法指令、在常态下执行了特权指令、运行时间超越了分给的最大时间段、申请的内存超过了系统能提供最大量、越界错误等
- 操作员或操作系统干预
- 父进程撤销其子进程
- 操作系统终止

Linux——exit()

- Linux用exit(status)撤销一个进程，它停止当前进程的运行，清除其使用的内存空间，销毁其在内核中的各种数据结构，但仍保留其PCB结构，等待父进程回收。
- 进程的状态变为zombie（僵尸状态）。
- 若其父进程正在等待它的终止，则父进程可立即得到其返回的整数status。
- **僵尸进程**
  - 若子进程调用exit()，而父进程并没有调用wait()或waitpid()获取子进程的状态信息，那么子进程的PCB仍然保存在系统中。这种进程称之为僵尸进程。
  - **僵尸进程的坏处?**
- **孤儿进程**

  当一个父进程由于正常完成工作而退出或由于其他情况被终止，它的一个或多个子进程却还在运行，那么那些子进程将成为孤儿进程。
  - 孤儿进程将被1号进程接管
  - 1号进程定期清除僵尸进程

4 进程等待

① 进程等待原语的形式

当进程需要等待某一事件完成时，它可以调用等待原语挂起自己。

susp(chan)

入口参数chan：进程等待的原因

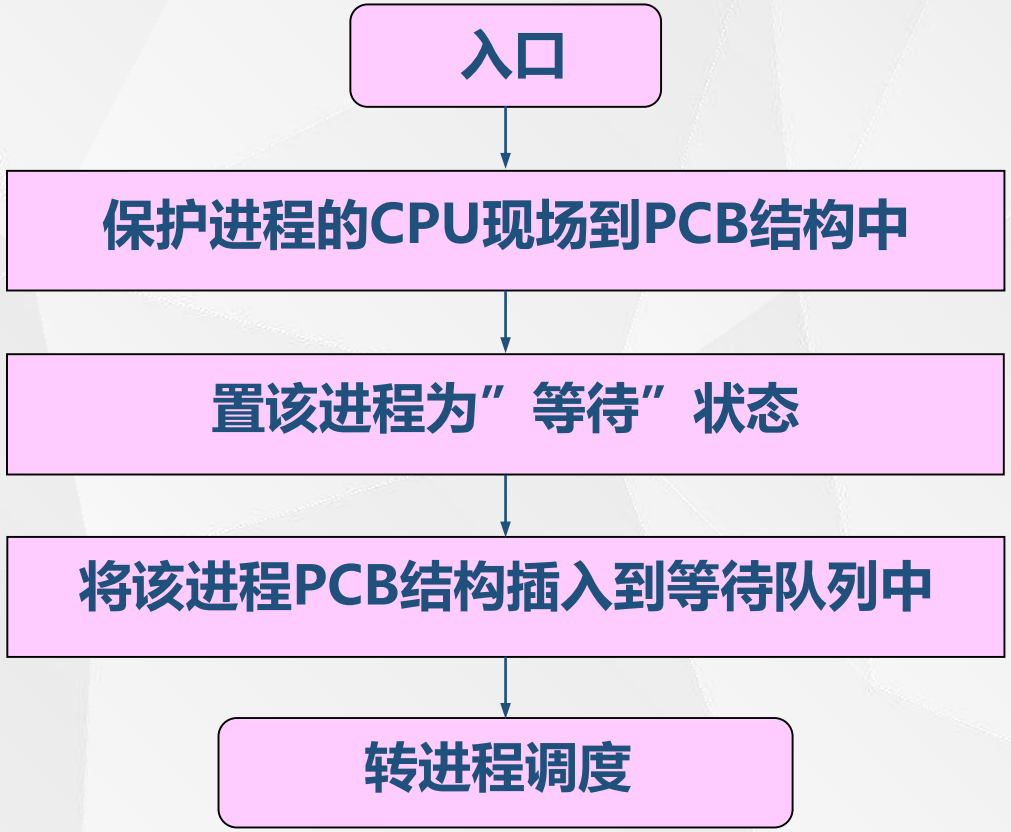
② 进程等待原语的功能

中止调用进程的执行，并加入到等待chan的等待队列中；

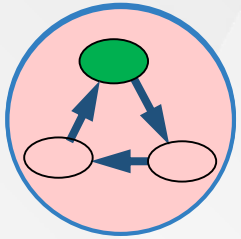
最后使控制转向进程调度。

注意：并没有一个唯一特定的系统调用来将进程挂起，导致进程挂起的原因很多，所以很多系统调用例程都可能使用此原语将当前进程挂起。

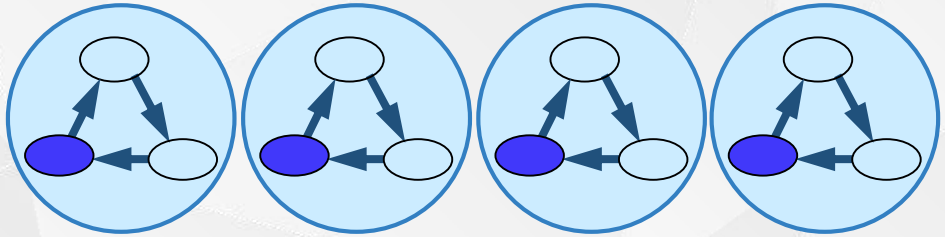
③ 进程等待原语的实现


入口
保护进程的CPU现场到PCB结构中
置该进程为”等待”状态
将该进程PCB结构插入到等待队列中
转进程调度

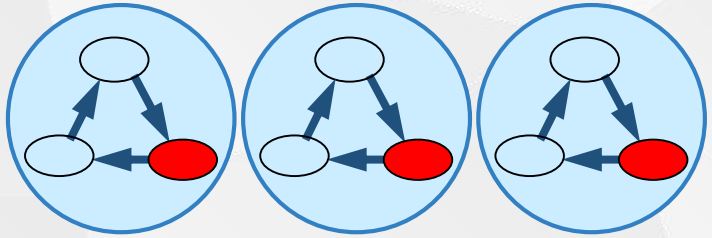
运行指针



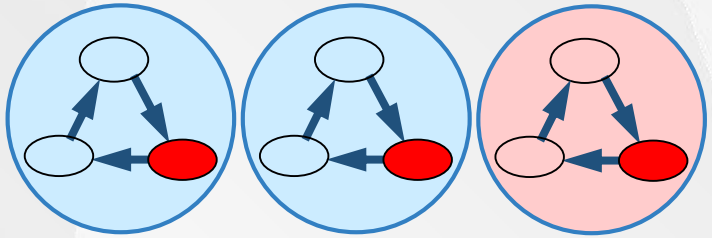
就绪队列



等待队列1



等待队列2



④ 进程等待的原因

- 进程在执行过程中通过系统调用请求一些暂时得不到而必须等待的服务。例如：
  - 读写文件、访问共享内存等，而这些服务操作系统可能无法立即提供；
  - 进程在执行过程中发出了I/O请求，从而必须等待该I/O请求的完成才能继续执行；
  - 因为进程间通讯的原因，一个进程在执行过程中必须等待另一个进程的消息才能继续往下执行。

Linux——wait()

① wait()：等待子进程结束。

pid=wait(int * status);

wait()函数使父进程暂停执行，直到它的一个子进程结束为止，该函数的返回值是终止运行的子进程的PID。参数status所指向的变量存放子进程的退出码，即从子进程的main函数返回的值或子进程中exit()函数的参数。

② waitpid()：等待某个特定子进程结束。

waitpid(pid_t pid, int * status, int options)

- **pid**为要等待的子进程的PID；
- **status**的含义与wait()函数中的status相同。

wait 与 exit举例


```
main( )
{
  int  n;
  if (fork()==0)
  {
        printf( “a” );
        exit(0);
  }
  wait(&n);
  printf( “b” );
}
```

输出结果是什么?

5 进程唤醒

① 进程唤醒原语的形式

当处于等待状态的进程所期待的事件来到时，由发现者进程使用唤醒原语唤醒它。

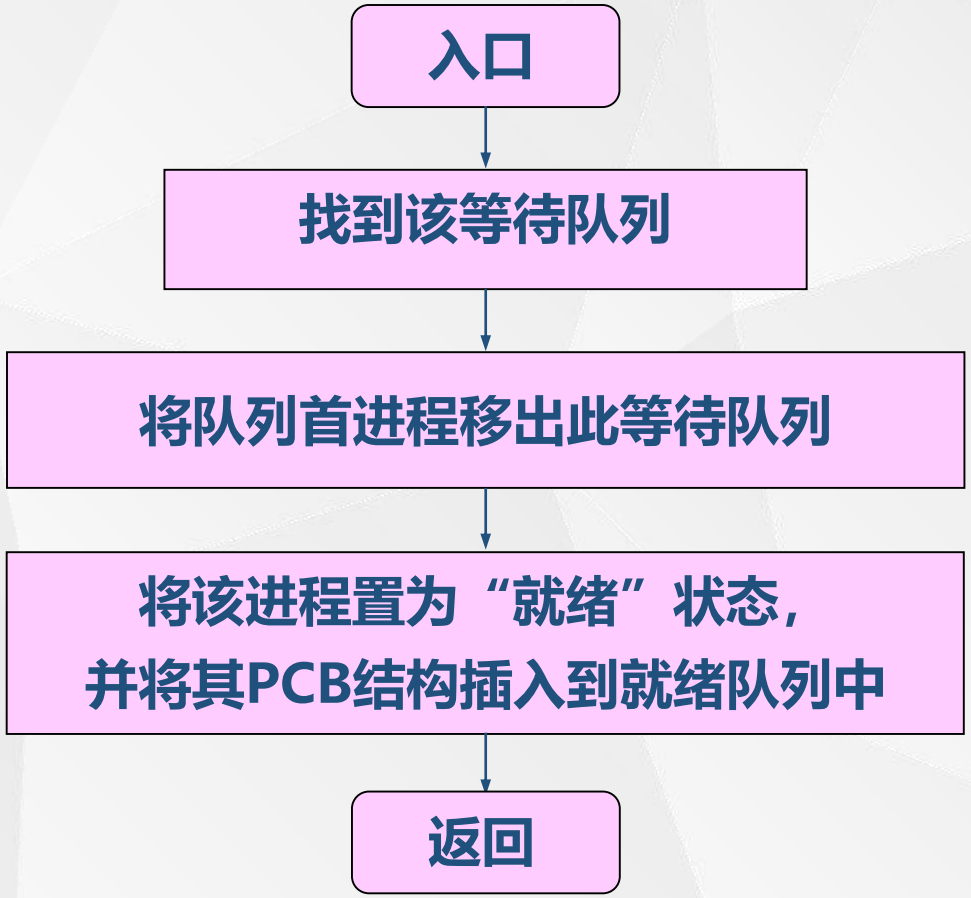
wakeup(chan)

入口参数chan：进程等待的原因。

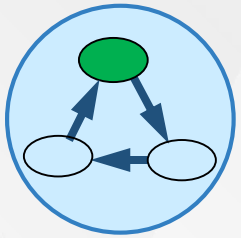
② 进程唤醒原语的功能

当进程等待的事件发生时，唤醒等待该事件的进程。

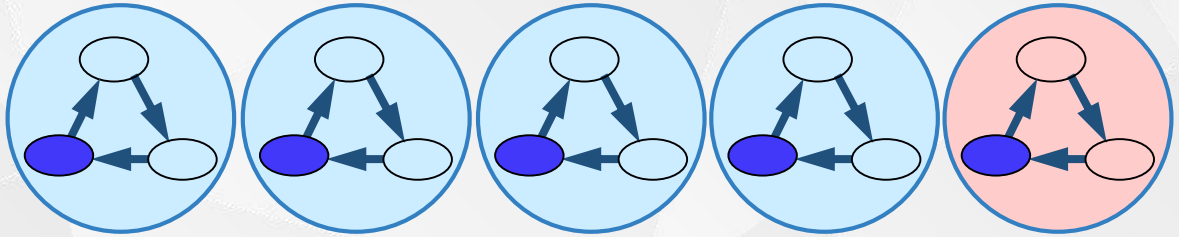
③ 进程唤醒原语的实现


入口
找到该等待队列
将队列首进程移出此等待队列
将该进程置为“就绪”状态，
并将其PCB结构插入到就绪队列中
返回

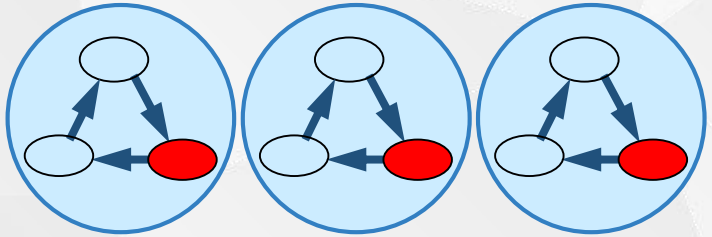
运行指针



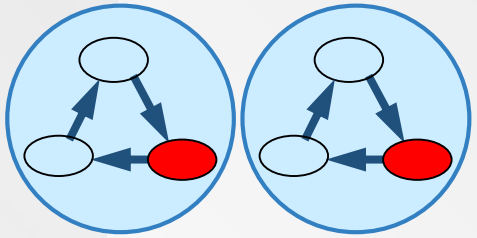
就绪队列



等待队列1



等待队列2



④ 进程唤醒的时机

- 例如，I/O完成时会产生一个中断，在该中断的中断处理函数中会唤醒等待该I/O的进程；
- 或者进程在等待某个被上锁的资源，当另一个进程通过系统调用来释放其施加在该资源上的锁时，系统调用服务例程会唤醒该进程；
- 或者进程在等待另一个进程的消息，当另一个进程通过系统调用向其发送消息时，系统调用服务例程会唤醒该进程。

进程切换

① 什么时候进行进程切换?

操作系统必须先获取处理器的使用权。

机制	原因	用途
中断	外部事件的发生	响应异步的外部事件
异常	当前指令无法继续执行	处理器错误（如除零错） 或者异常事件（如缺页）
系统调用	由当前运行的进程主动发出请求	调用操作系统的服务例程

② 操作系统获得了处理器使用权后，一定会做进程切换吗?

- **处理机状态切换**（操作系统获得使用权的过程）
  - 设置程序计数器为中断处理例程的入口。
  - 从用户态切换到核态，开始中断处理例程（包含特权指令）的执行。
- 处理机状态切换的开销很小，大多数情况下不会导致进程切换；但是如果当前进程的状态需要变为等待态或就绪态，就必须做进程切换。

③ 进程切换要做哪些事情?

- **保存当前进程的上下文。**
- 改变当前进程的进程控制块内容，包括其状态的改变、离开运行态的原因等。
- 将当前进程的控制块移到相应队列中，如就绪队列、事件等待队列等。
- 选择一个合适的进程开始执行。
- 更新被选择进程的进程控制块，将其状态置为运行态。
- **恢复被选择进程的上下文。**

进程之间的相互制约关系

进程间可能存在的交互关系

无交互	竞争系统资源	• 计算结果与其他进程无关 • 可能影响其他进程的时间特征
间接交互	竞争临界资源 (互斥)	• 可能影响其他进程的计算结果 • 可能影响其他进程的时间特征
直接交互	通过通讯协作 (同步)	• 可能影响其他进程的计算结果 • 可能影响其他进程的时间特征

1. 进程互斥的概念

(1) 临界资源

① 例1：两个进程共享一个变量x

设：x代表某航班机座号，p_1和p_2两个售票进程，售票工作是对变量x加1。这两个进程在一个处理机C上并发执行，分别具有内部变量r_1和r_2。

两个进程共享一个变量x时，两种可能的执行次序

A:

p_1： $r_1 := x$； $r_1 := r_1+1$； $x := r_1$ ；

p_2： $r_2 := x$； $r_2 := r_2+1$； $x := r_2$ ；

B:

p_1： $r_1 := x$；

p_2： $r_2 := x$； $r_2 := r_2+1$； $x := r_2$ ；

$r_1 := r_1+1$； $x := r_1$ ；

设x的初值为10，两种情况下的执行结果：

情况A： x = 10+2　　**情况B：** x = 10+1

特点：当两个进程公用一个变量时，它们必须顺序地使用，一个进程对公用变量操作完毕后，另一个进程才能去访问和修改这一变量。

② 临界资源的定义

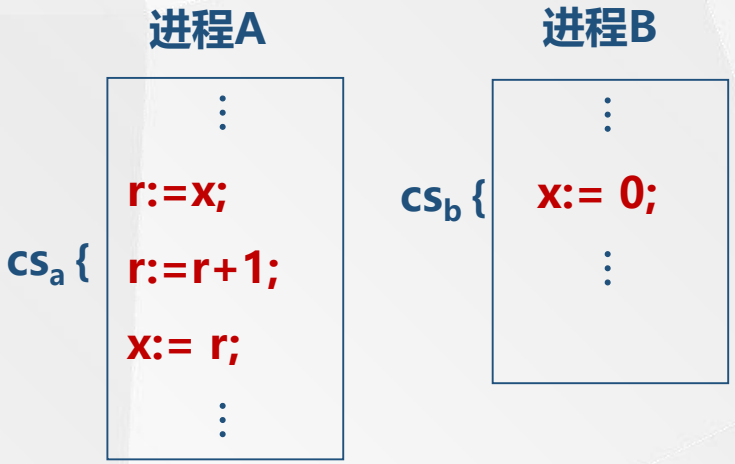
一次仅允许一个进程使用的资源称为临界资源。

硬件：如输入机、打印机、磁带机等

软件：如公用变量、数据、表格、队列等

(2) 临界区

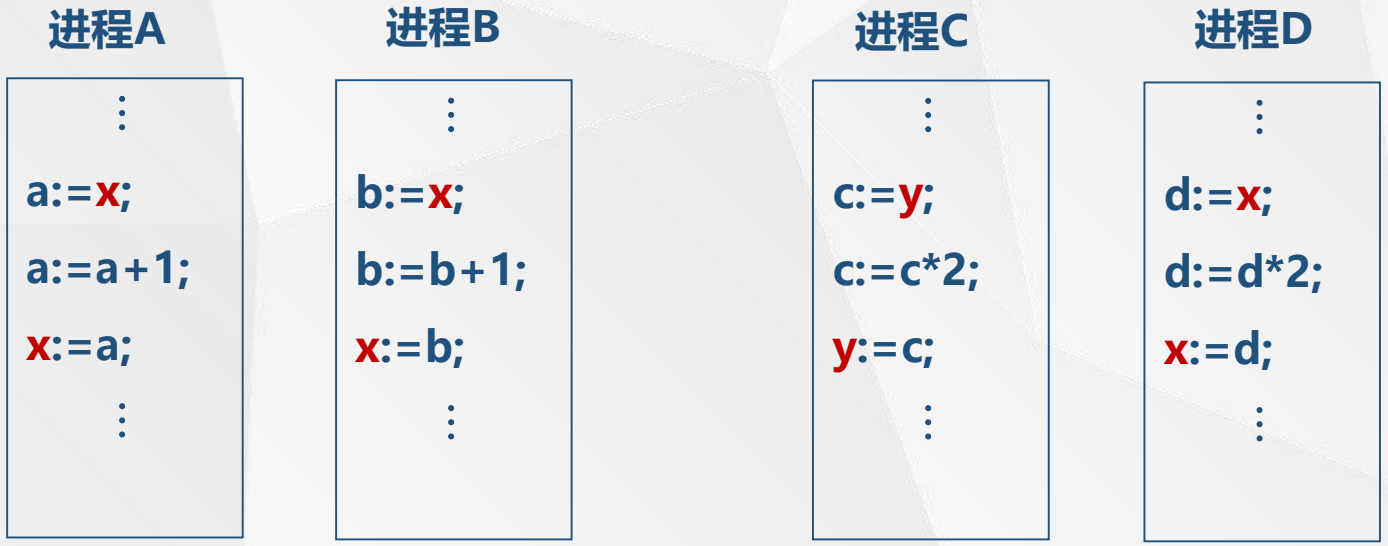
临界区是进程中对公共变量 (或存储区)进行审查与修改的程序段，称为相对于该公共变量的临界区。



(3) 互斥

操作系统中，当某进程正在访问某一存储区域时，就不允许其他进程来读出或者修改该存储区的内容，否则，就会发生后果无法估计的错误。进程间的这种相互制约关系称为互斥。

注意：同一临界资源的临界区才需要互斥进入。



2. 进程同步的概念

(1) 什么是进程同步

并发进程在一些关键点上可能需要互相等待与互通消息，这种相互制约的等待与互通消息称为进程同步。

(2) 进程同步的例子

① 病人就诊

看病活动：

⋮

要病人去化验；

⋮

等化验结果；

⋮

继续诊病；

化验活动：

⋮

需要进行化验？

⋮

进行化验；

开出化验结；

⋮

②誊抄

用卡片输入机将一个文本复写到行式打印机。（输入机：1000卡/分，打印机：600行/分）

- **顺序程序实现方案**

```
while (不空) {
    INPUT;
    OUTPUT;
}
```

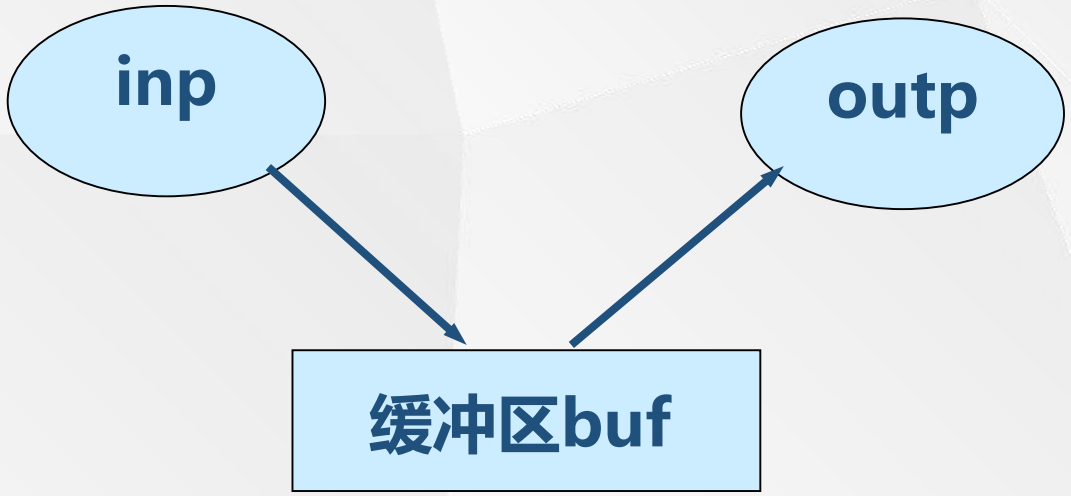
缺点：未能利用卡片输入机与行式打印机的并行操作能力，造成系统效率低。

速度：1/(1/1000+1/600)=375

并发程序实现方案

输入程序inp：
```
while (不空) {
        INPUT;
        写入缓冲区;
}
```

输出程序outp：
```
while (未结束) {
        读缓冲区;
        OUTPUT;
}
```



Q：速度？问题？

同步与互斥的关系

- 进程的互斥从本质上来看也是一种特殊的进程同步。
- 但是为便于区别，一般只把有严格的执行顺序要求的进程同步称为同步问题。
- 同步反映的是合作关系；互斥反映的是竞争关系。