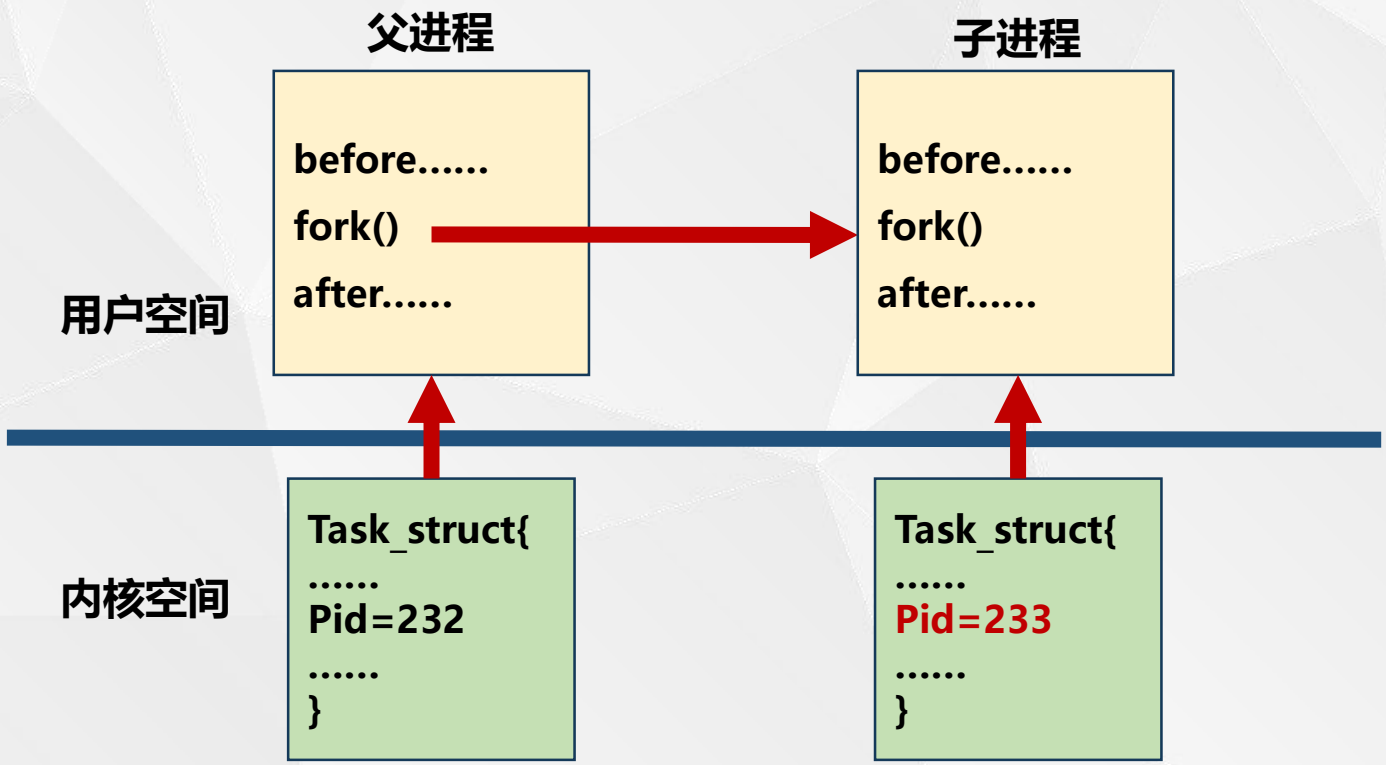
Linux——fork()


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

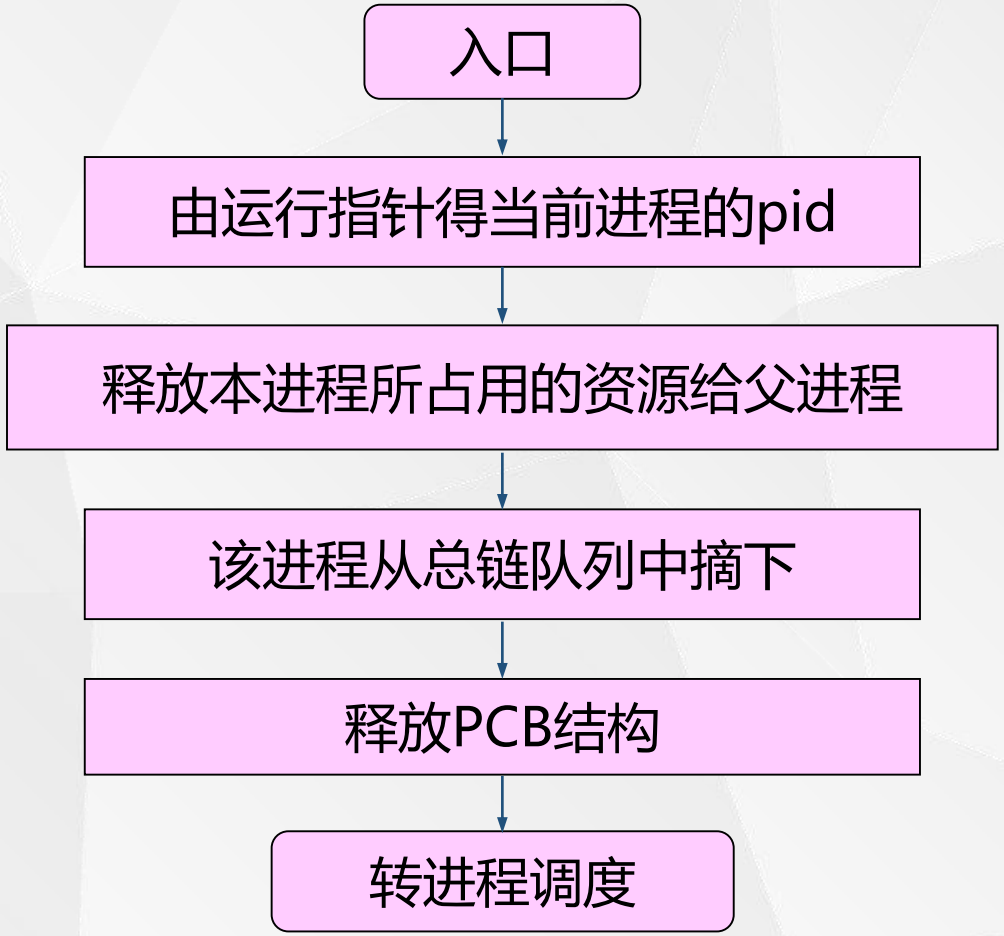
进程完成任务或希望终止时使用进程撤消原语。

Kill ( ) 或 exit( )

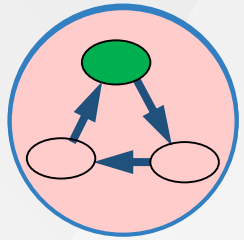
② 进程撤销原语的功能

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。

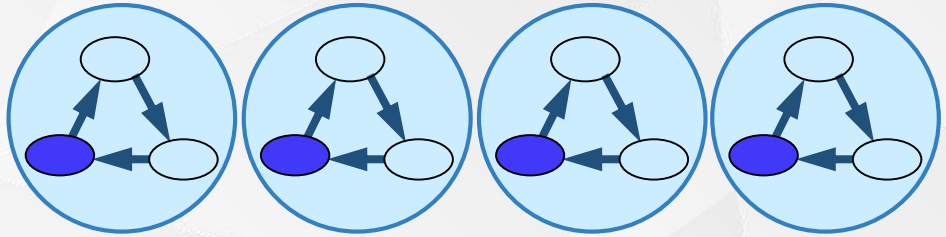
③ 进程撤销原语的实现


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

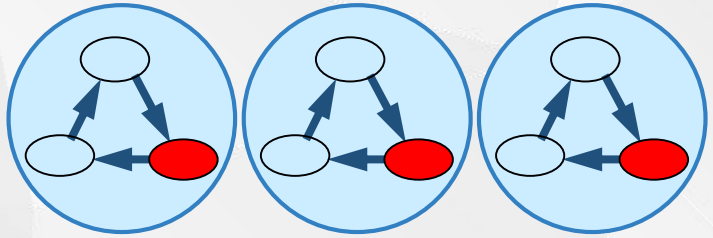
运行指针



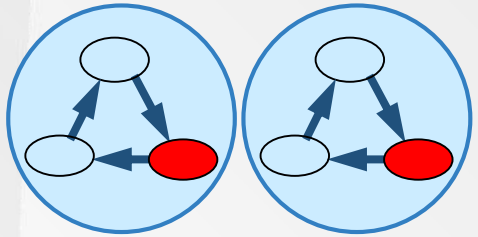
就绪队列



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)

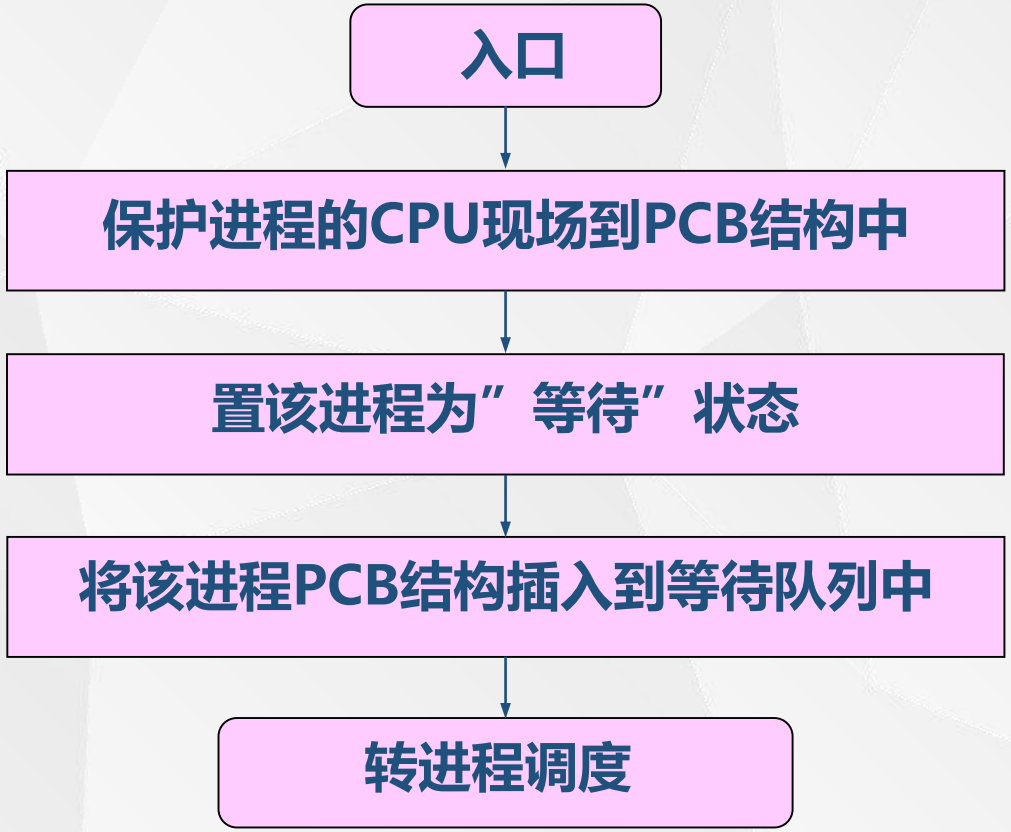
入口参数chan：进程等待的原因

② 进程等待原语的功能

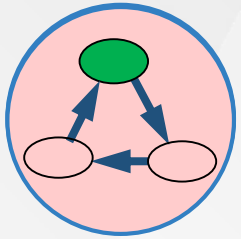
中止调用进程的执行，并加入到等待chan的等待队列中；

最后使控制转向进程调度。

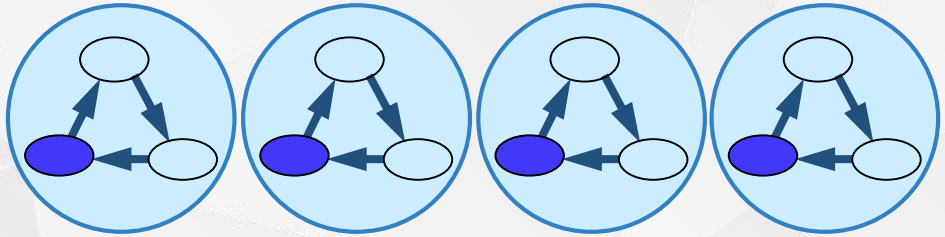
③ 进程等待原语的实现


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

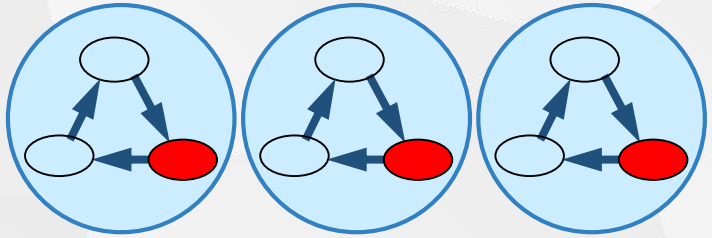
运行指针



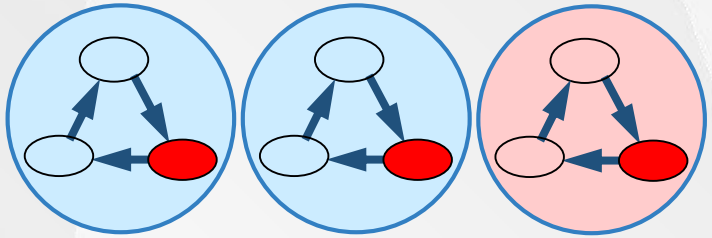
就绪队列



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叫唤醒它。

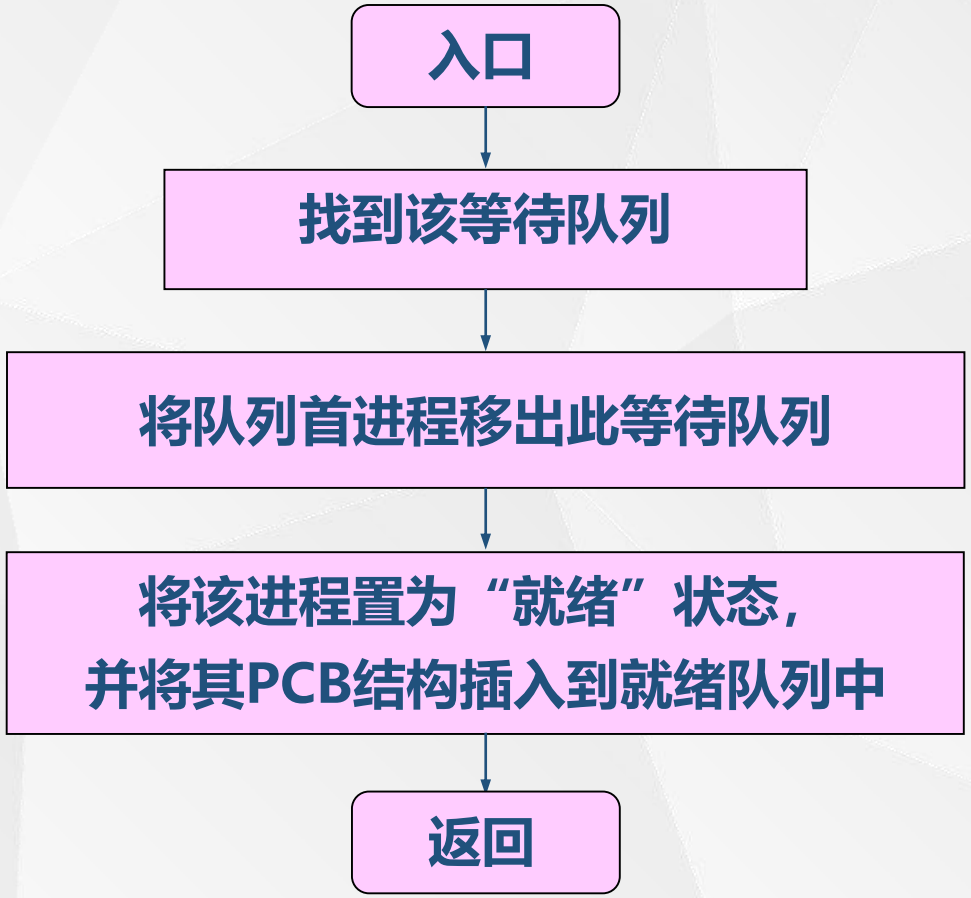
wakeup(chan)

入口参数chan：进程等待的原因。

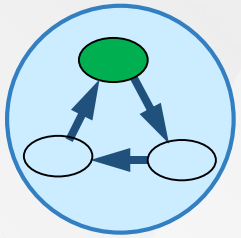
② 进程唤醒原语的功能

当进程等待的事件发生时，唤醒等待该事件的进程。

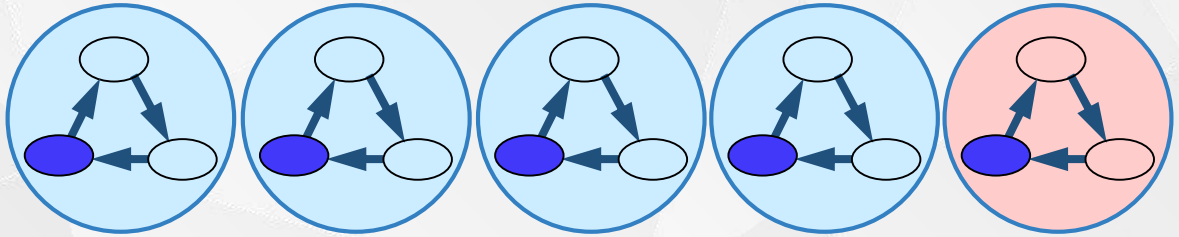
③ 进程唤醒原语的实现


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

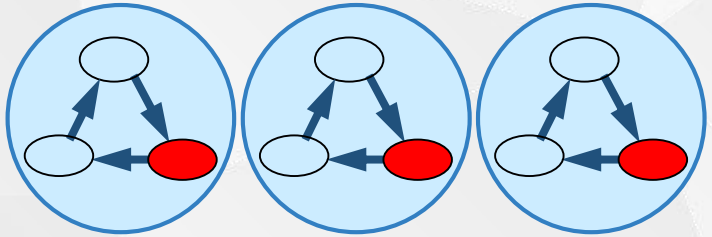
运行指针



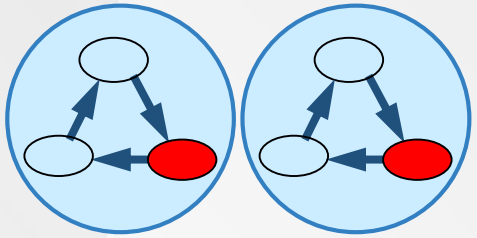
就绪队列



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

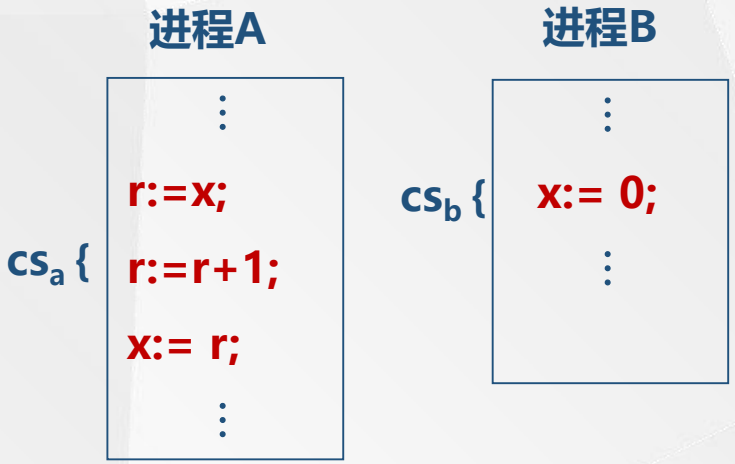
一次仅允许一个进程使用的资源称为临界资源。

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

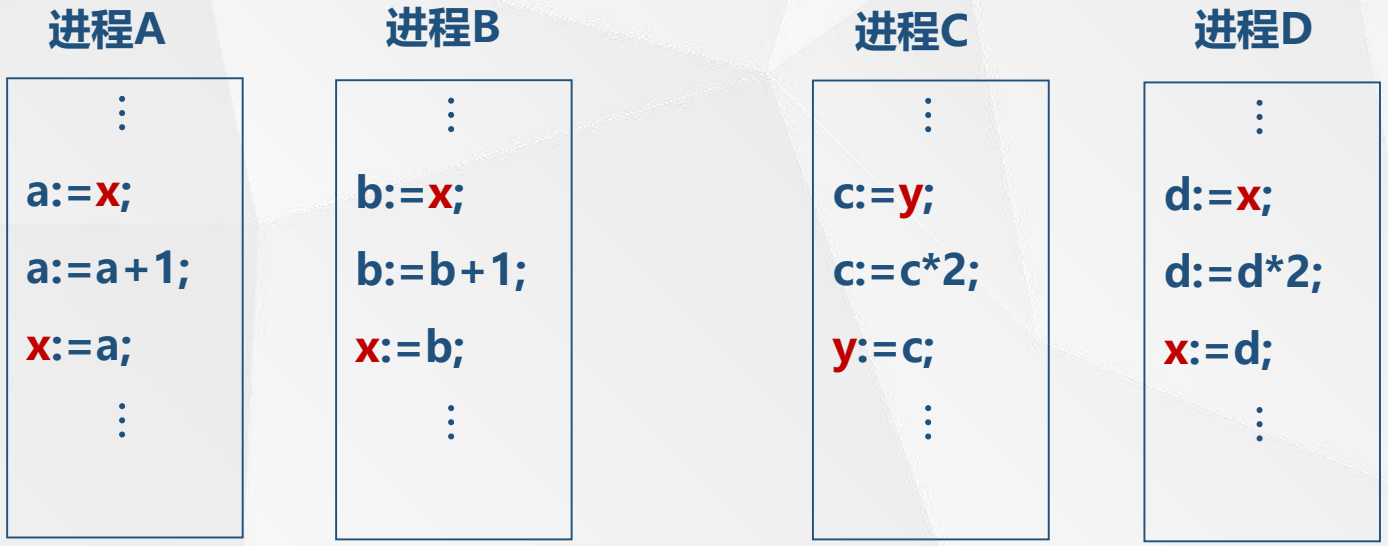
临界区是进程中对公共变量 (或存储区)进行审查与修改的程序段，称为相对于该公共变量的临界区。



(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；

⋮