计算机操作系统

第一章

1.5 操作系统概念

第二章 进程与线程

2.1 进程

2.1.1 进程模型

概念：

> 在进程模型中，计算机上所有可运行的软件，通常也包括操作系统，被组织成若干顺序进程，简称进程。一个进程就是一个正在执行程序的实例，包括程序计数器、寄存器和变量的当前值，是操作系统分配内存的最小单位。从概念上说，每个进程拥有它自己的虚拟CPU。

2.1.2 进程的创建

1. 系统初始化
2. 正在运行的查询执行了创建进程的系统调用
3. 用户请求创建一个新进程
4. 一个批处理作业的初始化

在UNIX系统中，只有一个系统调用才能创建新进程：fork。这个系统会调用一个与调用进程相同的副本。在调用了fork后，这两个进程（父进程和子进程）拥有相同的内存映像、同样的环境字符串和同样的打开文件。通常，子进程会接着执行execve或者类似的系统调用，以修改其内存映像并运行一个新的程序。

2.1.3 进程的终止

1. 正常退出
2. 出错退出
3. 严重错误（非自愿）
4. 被其他进程杀死（非自愿）UNIX：kill，Win32：TerminateProcess

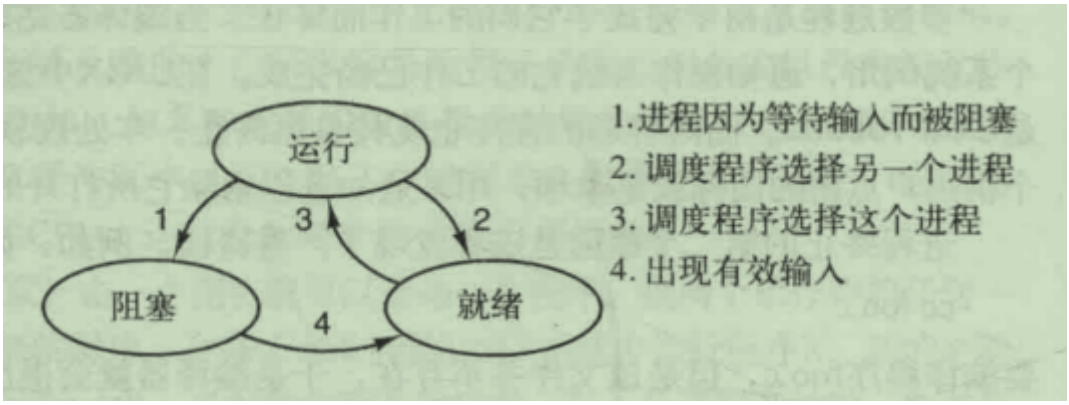
UNIX中是调用exit，Windows则是调用ExitProcess

2.1.4 进程的层次结构

UNIX中父子进程组成进程树，根节点是init

Windows没有进程层次的概念，所有的进程地位都是相同的。创建进程的时候，父进程得到要给句柄，用于控制子进程。但是该句柄可以传送给其他进程，所以就不存在进程层次了。

2.1.5 进程的状态



转换2和3是因为进程的调度引起的。

2.1.6 进程的实现

2.2 线程

2.3 进程间通信

Inter Process Communication IPC

2.3.1 竞争条件

> 两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序。

2.3.2 临界区

互斥：

> 以某种手段确保当前一个进程在使用一个共享变量或者文件的时候，其它进程不能做同样的操作。

临界区：

> 对共享内存进行访问的程序片段称作临界区域或者临界区

如何避免多个进程同时进入临界区？

1. 任何两个进程不能同时处于临界区
2. 不应对CPU的速度和数量做任何假设
3. 临界区外运行的进程不得阻塞其他进程
4. 不得使进程无限期等待进入临界区

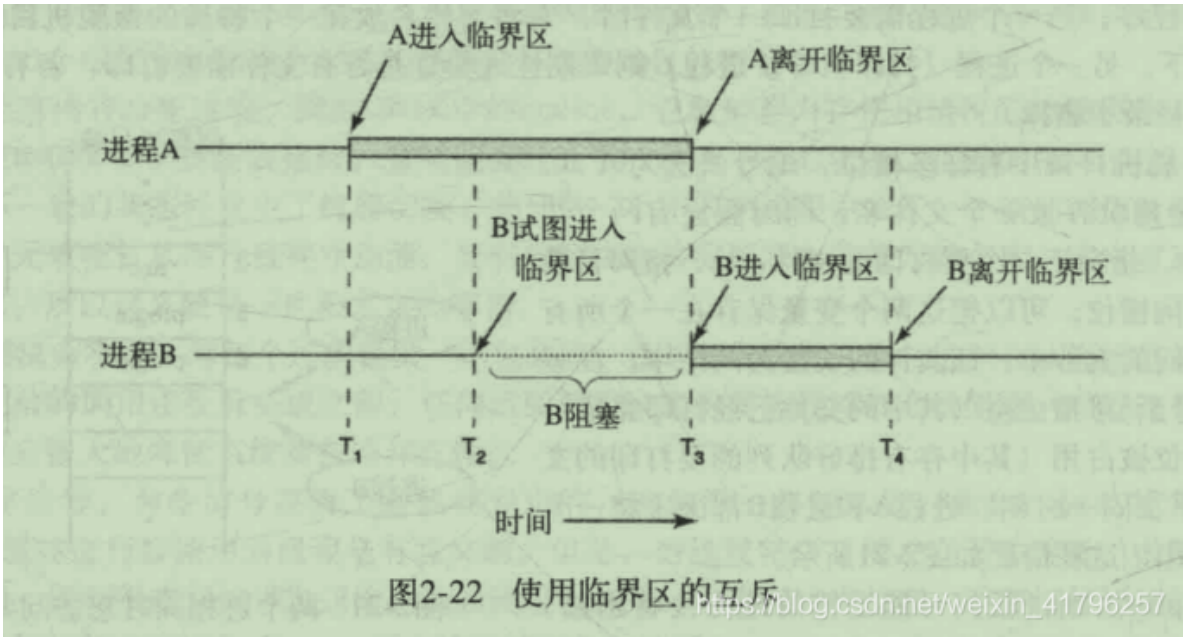


图2-22 使用临界区的互斥

2.3.3 忙等待的互斥

1. 屏蔽中断

在单处理器系统中，可以在进程进入到临界区后立即屏蔽所有中断，这样就不能进行进程切换了。然后在该进程就要离开临界区之前打开中断。

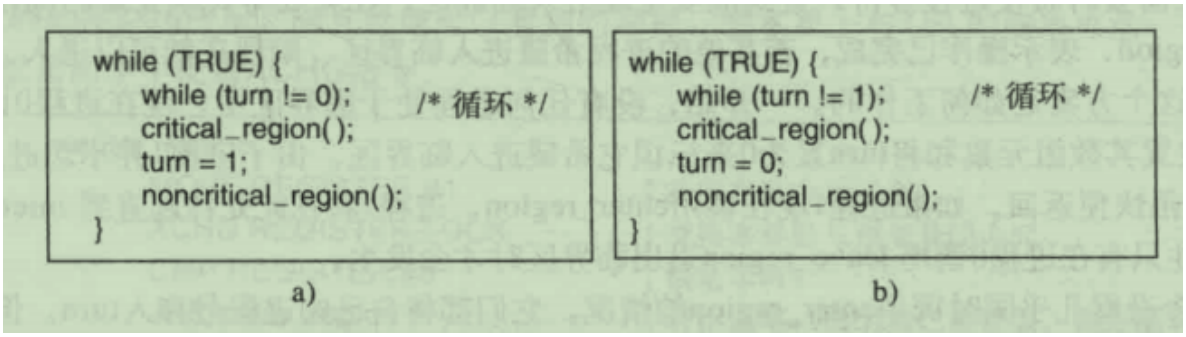
这个方案很多缺点，首先，将屏蔽中断的权利下放到了用户，这是很不明智的。另外，如果系统的多处理器的，对该进程所处处理器的屏蔽中断并不能对其他处理器有效。

2. 锁变量

使用一个变量，初始值为0，当进程进入临界区后，修改为1代表临界区被占用。

但是一样会存在问题，比如一个线程刚刚读取到该变量为0，然后切换线程，gg。

3. 严格轮换法



容易出现忙等的情况，很浪费资源。

4. Peterson解法

5. TSL指令

```
TSL RX, LOCK    测试并加锁
```

它将一个内存字lock读到寄存器RX中，然后在该内存地址上存一个非零值。读字和写字操作保证是不可分割的，即该指令是一个原子指令。执行TSL的CPU将锁住内存总线，以禁止其他CPU在本指令结束之前访问内存。

2.3.4 睡眠与唤醒

sleep和wakeup

生产者-消费者问题

2.3.5 信号量

```
#define N 100                                   /* 缓冲区中的槽数目 */
typedef int semaphore;                          /* 信号量是一种特殊的整型数据 */
semaphore mutex = 1;                            /* 控制对临界区的访问 */
semaphore empty = N;                            /* 计数缓冲区的空槽数目 */
semaphore full = 0;                             /* 计数缓冲区的满槽数目 */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE是常量1 */
          item = produce_item();                /* 产生放在缓冲区中的一些数据 */
          down(&empty);                         /* 将空槽数目减1 */
          down(&mutex);                         /* 进入临界区 */
          insert_item(item);                    /* 将新数据项放到缓冲区中 */
          up(&mutex);                           /* 离开临界区 */
          up(&full);                            /* 将满槽的数目加1 */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* 无限循环 */
          down(&full);                          /* 将满槽数目减1 */
          down(&mutex);                         /* 进入临界区 */
          item = remove_item();                 /* 从缓冲区中取出数据项 */
          up(&mutex);                           /* 离开临界区 */
          up(&empty);                           /* 将空槽数目加1 */
          consume_item(item);                   /* 处理数据项 */
     }
}
```

图2-28 使用信号量的生产者-消费者问题

二元信号量

> 供两个或多个进程使用的信号量，其初值为1，保证同时只有一个进程可以进入临界区。

该信号量的修改是原子的。

2.3.6 互斥量

> 互斥量是一个可以处于两态之一的变量：加锁与解锁

互斥量使用两个过程。当一个线程或者进程需要访问临界区时，它调用mutex_lock。如果该互斥量当前是解锁的，则调用成功。如果互斥量被加锁，调用线程被阻塞，知道临界区里面的线程调用mutex_unlock。如果多个线程被阻塞，则随机选择一个线程并允许它获得锁（非公平锁）。

```
mutex_lock:
    TSL REGISTER,MUTEX      | 将互斥信号量复制到寄存器，并且将互斥信号量置为1
    CMP REGISTER,#0         | 互斥信号量是0吗？
    JZE ok                  | 如果互斥信号量为0，它被解锁，所以返回
    CALL thread_yield       | 互斥信号量忙；调度另一个线程
    JMP mutex_lock          | 稍后再试
ok: RET                     | 返回调用者；进入临界区

mutex_unlock:
    MOVE MUTEX,#0           | 将mutex置为0
    RET                     | 返回调用者
```

图2-29　mutex_lock和mutex_unlock的实现