202X

操作系统寒假帮扶讲座1

2024.2.21

特别提醒：此PPT受众主要为补考同学，对于参加正常考试的同学仅供参考！

第1章 计算机系统概述

取指-指行

- 计算机部件：

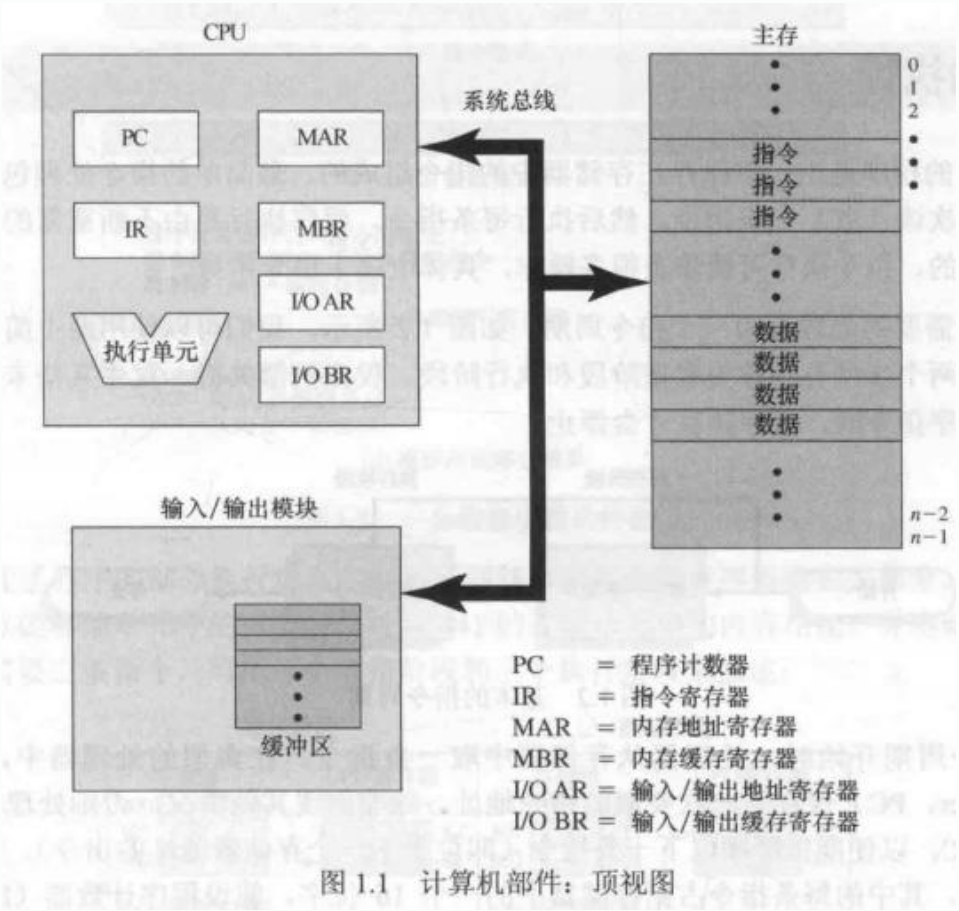


图 1.1 计算机部件：顶视图

第1章 计算机系统概述

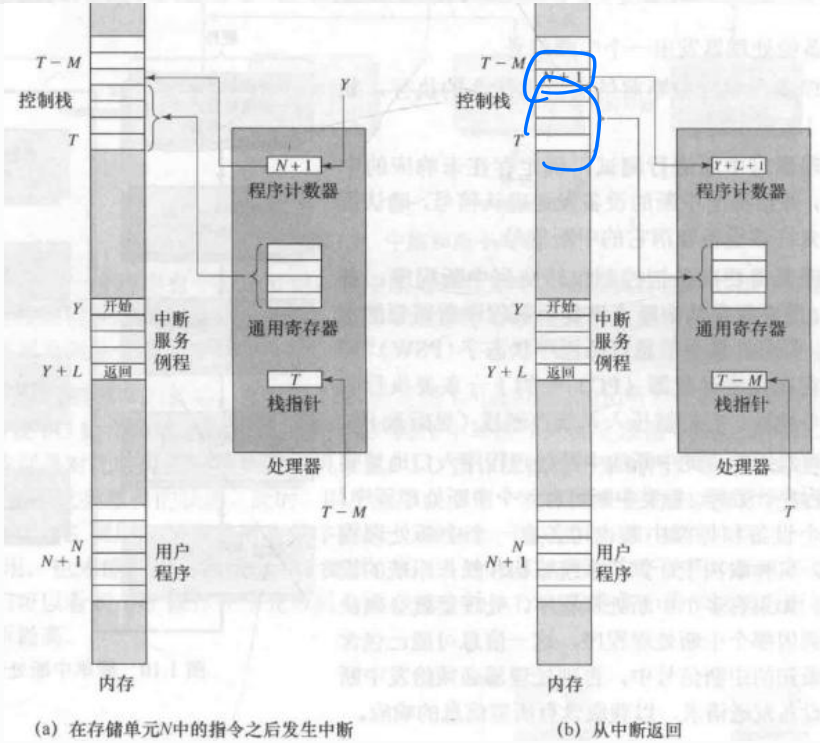
- 中断：

表 1.1 中断的分类

程序中断	在某些条件下由指令执行的结果产生，如算术溢出、除数为 0、试图执行一条非法机器指令及访问用户不允许的存储器位置
时钟中断	由处理器内部的计时器产生，允许操作系统以一定的规律执行函数
I/O 中断	由 I/O 控制器产生，用于发信号通知一个操作的正常完成或各种错误条件
硬件失效中断	由诸如掉电或存储器奇偶校验错之类的故障产生

- 中断处理：

图 1.10 简单中断处理



(a) 在存储单元N中的指令之后发生中断　(b) 从中断返回

第2章 操作系统概述

- 操作系统：**控制**应用程序执行的**程序**，是应用程序和计算机硬件间的**接口**
- 功能：用户/计算机接口、资源管理器
- 操作系统开发中的主要成就：进程、信息保护和安全、内存管理、调度和资源管理

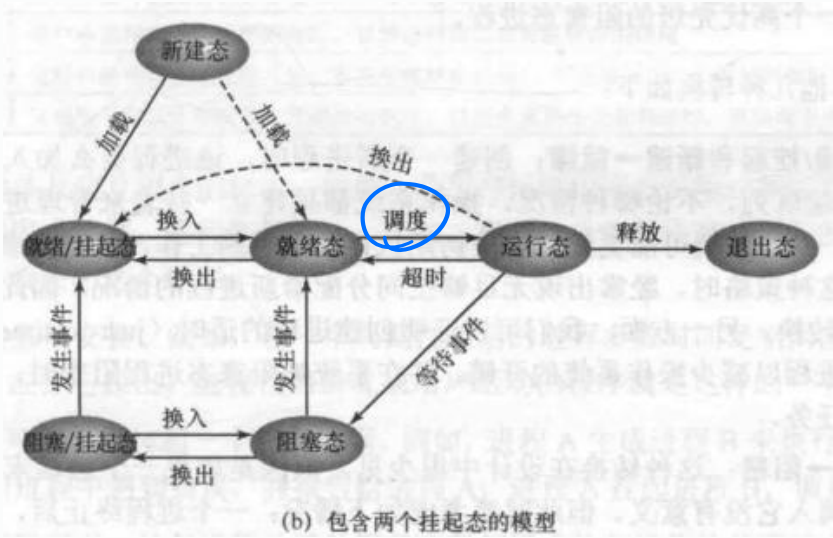
第3章 进程描述和控制

- 进程：一个**正在**执行的**程序**/一个**正在**计算机上执行的**程序实例**/能分配给处理器并由处理器执行的实体/由一组执行的指令、一个当前状态和一组相关的系统资源**表征的活动单元**
- 进程的组成：程序代码+相关数据+进程控制块 PCB
- 进程控制块：由操作系统创建和管理，包含了操作系统**所需进程的所有信息**

①标识符：与进程相关的**唯一标识符**，用来区分其他进程
②状态：就绪态、运行态、阻塞态等
③优先级：相对于其他进程的优先顺序
④程序计数器：程序中即将执行的下一条指令的地址
⑤内存指针：包含程序代码和相关数据的指针，以及与其他进程共享内存块的指针
⑥上下文数据：进程执行时处理器的寄存器中的数据
⑦I/O状态信息：包括显式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等
⑧记账信息：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

第3章 进程描述和控制

- 进程状态：



(b) 包含两个挂起态的模型

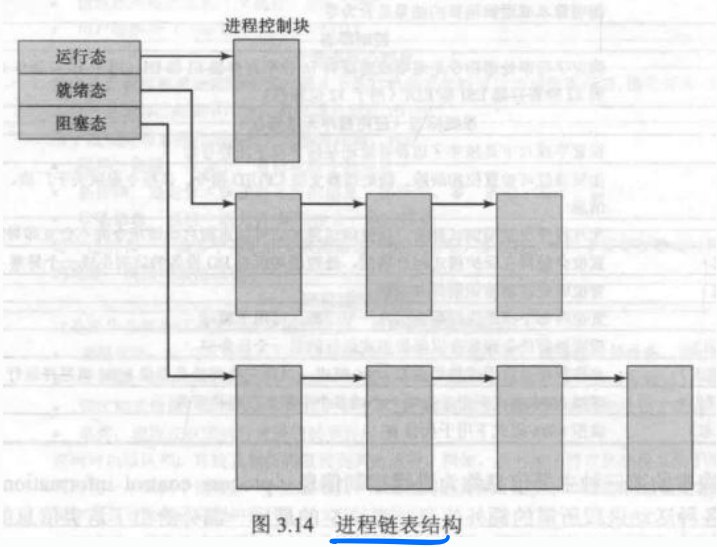


图 3.14 进程链表结构

第3章 进程描述和控制

- 操作系统控制表：操作系统管理进程和资源，掌握每个进程和资源的当前状态
- 进程映像：

表 3.4 进程映像中的典型元素

项　目	说　明
用户数据	用户空间中的可修改部分，包括程序数据、用户栈区域和可修改的程序
用户程序	待执行的程序
栈	每个进程有一个或多个后进先出（LIFO）栈，栈用于保存参数、过程调用地址和系统调用地址
进程控制块	操作系统控制进程所需的数据（见表 3.5）

- 进程映像在虚存中的结构：

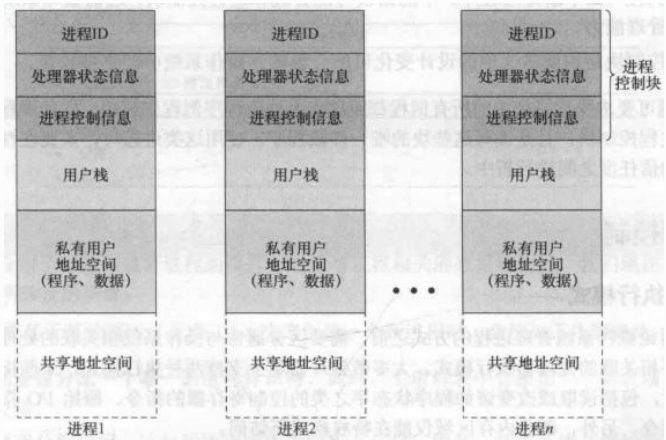


图 3.13 虚存中的用户进程

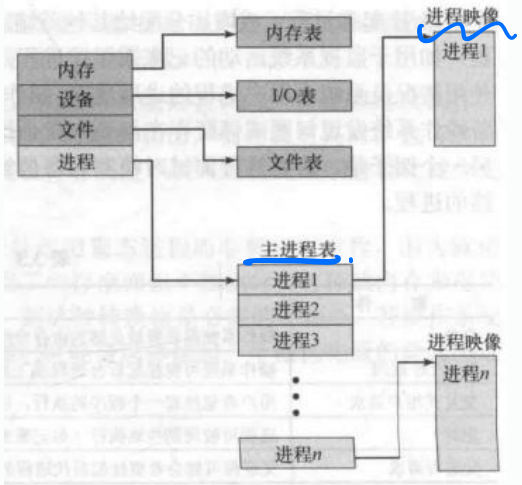


图 3.11 操作系统控制表的通用结构

第3章 进程描述和控制

- 执行模式：

  ①用户模式：**用户程序**通常在该模式下运行

  ②内核/系统/控制/特权模式：可以**读取或改变**如PSW之类的**控制寄存器的指令**、**原始I/O指令**和与**内存管理**相关的指令，部分内存区域仅能在该模式下访问，保护操作系统和重要的操作系统表不受用户程序的干扰

- 如何确定处理器正在什么模式下运行：2位RPL字段，**级别0**为最高特权级别，**级别3**为最低特权级别，内核模式使用级别0，用户模式使用其他级别

- 进程创建：

  ①为新进程分配一个唯一的进程标识符

  ②为进程分配空间：包括进程映像中的所有元素

  ③初始化进程控制块

  ④设置正确的链接

  ⑤创建或扩充其他数据结构

第3章 进程描述和控制

- 何时切换进程：可在操作系统从当前正运行进程中获得控制权的**任何时刻**发生

  ①中断：控制权首先交给中断处理器，完成一些基本工作后，再将控制权转给相关的操作系统例程。如时钟中断、I/O中断、内存失效

  ②陷阱

  ③系统调用

- 进程切换步骤：

  ①保存处理器的上下文，包括程序计数器和其他寄存器

  ②更新当前处于运行态进程的进程控制块

  ③把该进程的进程控制块移到相应的队列

  ④选择另一个进程执行（调度算法）

  ⑤更新所选进程的进程控制块，包括把进程的状态改为运行态

  ⑥更新内存管理数据结构

  ⑦载入程序计数器和其他寄存器先前的值，恢复处理器的上下文

第4章 线程

- 多线程：操作系统在**单个进程**内支持**多个并发**执行路径的能力
- 线程与进程的区别：

①单线程进程模型：进程的表示包括：PCB、用户地址空间、用户栈、内核栈
②多线程进程模型：进程的表示包括：PCB、用户地址空间
线程的表示包括：单独的线程控制块、单独的栈

进程中所有线程**共享**该进程的状态和资源，所有线程都驻留在同一块地址空间中，可以访问相同的数据

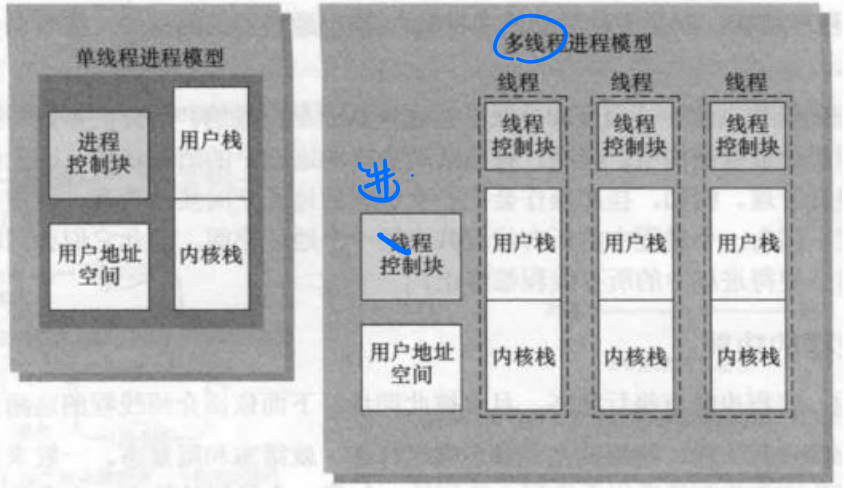


图 4.2 单线程和多线程进程模型

第5章 并发：互斥和同步

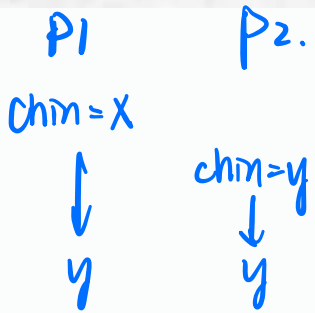
- 关键术语：

表 5.1 与并发相关的关键术语

原子操作	一个函数或动作由一个或多个指令的序列实现，对外是不可见的；也就是说，没有其他进程可以看到其中间状态或能中断此操作。要保证指令序列要么作为一个组来执行，要么都不执行，对系统状态没有可见的影响。原子性保证了并发进程的隔离
临界区	一段代码，在这段代码中进程将访问共享资源，当另外一个进程已在这段代码中运行时，这个进程就不能在这段代码中执行
死锁	两个或两个以上的进程因每个进程都在等待其他进程做完某些事情而不能继续执行的情形
活锁	两个或两个以上的进程为响应其他进程中的变化而持续改变自己的状态但不做有用的工作的情形
互斥	当一个进程在临界区访问共享资源时，其他进程不能进入该临界区访问任何共享资源的情形
竞争条件	多个线程或进程在读写一个共享数据时，结果依赖于它们执行的相对时间的情形
饥饿	一个可运行进程尽管能继续执行，但被调度程序无限期地忽视，而不能被调度执行的情形

- 举例：

```
void echo()
{
 chin = getchar();
 chout = chin;
 putchar(chout);
}
```



- 问题本质：共享全局变量chin
- 说明：若需要保护**共享的全局变量**（以及其他共享的全局资源），唯一的办法是控制访问该变量的代码，即**一次只允许一个进程进入echo**，并且只有在echo运行结束后，才允许另一个进程进入

第5章 并发：互斥和同步

- Dijkstra设计的基本原理：两个或多个进程可以通过简单的信号进行合作，可以强迫一个进程在某个位置停止，直到它接收到一个特定的信号。任何复杂的合作需求都可通过适当的信号结构得到满足。为了发信号，使用一个成为**信号量**的特殊变量。
- 发信号：semSignal(s)　　接收信号：semWait(s)
- semWait 使信号量-1，若值变为负数，则阻塞执行semWait的进程，否则进程继续执行
  semSignal 使信号量+1，若值小于等于0，则被semWait操作阻塞的进程解除阻塞
- 信号量初始化成非负数，代表着发出semWait操作后可以立即继续执行的进程的数量
- 解决互斥：

```
/* program mutualexclusion */
const int n = /* 进程数 */;
semaphore s = 1;
void P(int i)
{
     while (true) {
          semWait(s);
          /* 临界区 */;
          semSignal(s);
          /* 其余部分 */;
     }
}
void main()
{
     parbegin (P(1), P(2), … , P(n));
}
```

第5章 并发：互斥和同步

- 生产者/消费者问题：有一个或多个生产者生产某种类型的数据，并放置在缓冲区中；有一个消费者从缓冲区中取数据，每次取一项；系统保证避免对缓冲区的重复操作，即在任何时候**只有一个主体（生产者或消费者）可访问缓冲区**。确保**当缓存已满时**，生产者不会继续向其中添加数据；**当缓存为空时**，消费者不会从中移走数据
- 补充：进程同步：进程的直接制约关系；为了完成某种任务而建立的2个或多个进程，这些进程因为需要在某些位置上协调它们的工作次序而等待。

```
semaphore mutex=1, full=0, empty=n;
void producer(){
    while(1){
        生产;
        semWait(empty);
        semWait(mutex);
        放入缓冲区;
        semSignal(mutex);
        semSignal(full);
    }
}
```

```
void consumer(){
    while(1){
        semWait(full);
        semWait(mutex);
        取出一个产品;
        semSignal(mutex);
        semSignal(empty);
        消费;
    }
}
```

第5章 并发：互斥和同步

- 例：有三个进程A、B和C合作解决文件打印问题：进程A将文件记录从磁盘读入缓冲区1，每执行一次读一个记录；进程B将缓冲区1的内容复制到缓冲区2，每执行一次复制一个记录；进程C将缓冲区2的内容打印出来，每执行一次打印一个记录；缓冲区1和2的大小都等于一个记录大小。请用信号量操作来保证文件的正确打印。

同步：1不满→A放 e1=1　1不空→B取 f1=0　2不满→B放 e2=1　2不空→C取 f2=0

```
A(){
  while(1){
    semWait(e1);
    读入;
    semSignal(f1);
  }
}
```

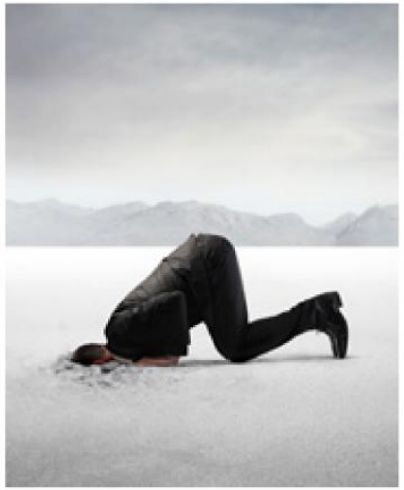
```
B(){
  while(1){
    semWait(f1);
    semWait(e2);
    复制;
    semSignal(e1);
    semSignal(f2);
  }
}
```

```
C(){
  while(1){
    semWait(f2);
    打印;
    semSignal(e2);
  }
}
```

第6章 并发：死锁和饥饿

- 死锁：一组相互**竞争**系统资源或进行通信的进程间的**“永久”阻塞**
- 当一组进程中的**每个进程**都在等待某个事件（如等待释放所请求的资源），而仅有这组进程中**被阻塞**的其他进程才可触发该事件
- 解决方法：**鸵鸟算法**

(a) 遇到危险的鸵鸟



(b) 假装什么都没看见