



## 2.6 进程通信

---



# 进程通信

---

## ■ 进程间信息交换

### ■ 控制信息的传送

- 低级通信，状态或值
- 程序员实现
- 目的是控制执行速度

### ■ 大批量数据传送

- 高级通信
- 目的是交换信息
- 隐藏进程通信细节，降低编程复杂度



## 2.6.1 进程通信的类型

### ■ 1 共享存储器系统

- 基于共享数据结构的通信方式
  - 程序员负责同步，效率低
- 基于共享存储区的通信方式
  - 高级通信
  - 向系统申请共享存储区的一个分区，对此分区读写

### ■ 2 消息传递系统

- 是以格式化的消息(**message**)为单位的进行信息交换
- 程序员直接利用系统提供的一组通信命令(**原语**)进行通信。
- 操作系统隐藏了通信的实现细节，大大减化了通信程序编制的复杂性，属于高级通信方式。

### ■ 3 管道通信

- 读写进程通过称为管道的共享文件进行通信
- 管道机制协调：互斥；同步；确定对方存在



## 2.6.2 消息传递通信的实现，

### ■ 1 直接通信方式

- 发送进程利用**OS**所提供的发送命令，直接把消息发送给目标进程。
- 要求发送进程和接收进程都以显式方式提供对方的标识符。
  - Send(Receiver, message); 发送一个消息给接收进程;
  - Receive(Sender, message); 接收Sender发来的消息
  - 若接收进程可与多个发送进程通信，无法事先指定发送进程。可设成接受任何进程的消息，发送进程的标识为是完成通信后的返回值

## 2.6.2 消息传递通信的实现。

### ■ 2 间接通信方式

- 通信双方需要通过共享数据结构暂存消息—邮箱
- 发送进程发消息到邮箱，接受进程从信箱取消息
- 系统提供原语
  - 信箱的创建和撤消。
    - 创建者进程应给出信箱名字、信箱属性(公用、私用或共享)；对于共享信箱，还应给出共享者的名字。当进程不再需要读信箱时，可用信箱撤消原语将之撤消。晶
  - 消息的发送和接收
- 信箱类型
  - 用户和系统均可创建并拥有信箱
  - 私用信箱：进程的一部分
  - 公用信箱：操作系统创建，供核准的进程使用
  - 共享信箱：创建者指名明共享的进程名字
- 发送进程和接受进程的关系
  - 一对一；一对多；多对一；多对多

## 2.6.3 消息传递系统实现中的若干问题

- **1. 建立通信链路(communication link)**
  - 由发送进程在通信之前，用显式的“建立连接”命令(原语)请求系统为之建立一条通信链路；在链路使用完后，也用显式方式拆除链路。这种方式主要用于计算机网络中。
  - 发送进程无须明确提出建立链路的请求，只须利用系统提供的发送命令(原语)，系统会自动地为之建立一条链路。这种方式主要用于单机系统中。
- **2 消息的格式**
  - 变长，定长
- **3 进程同步方式**
  - 发送进程阻塞，接受进程阻塞：
    - 用于无缓冲时，平时阻塞，直到有消息传递。称为汇合
  - 发送不进程不阻塞，接受进程阻塞
    - 广泛使用，如平时阻塞的服务进程
  - 均不阻塞



## 2.7 线程 (Thread)

---



## 2.7.1 线程的基本概念,

### ■ 1 引入

#### ■ 进程的两个基本属性

- 资源的拥有者;资源分配的基本单位
- 执行过程中的调度单位

#### ■ 缺点

- 创建进程 —— 撤销进程 —— 进程切换 过程中涉及到资源的管理和分配
- 处理机调度时系统时间空间开销大，并发度有限

#### ■ 目的

- 减少处理机空转时间和调度切换时间
- 提高系统的执行效率
- 适应对称多处理机





## 2.7.1 线程的基本概念,

### ■ 2 线程与进程的比较

- 轻量型进程, 进程元, 重型进程
- 进程拥有若干个线程, 至少一个线程

#### ■ 调度

- 线程作为调度单位; 进程资源拥有单位
- 同一进程内线程切换不会引起进程切换
- 减少了切换开销, 提高了切换速度

#### ■ 并发性

- 线程可以并发

#### ■ 拥有资源

- 线程不拥有资源, 访问所属进程的资源, 进程的资源对其所属线程是共享的

#### ■ 系统开销

- 线程切换仅涉及一些寄存器, 不涉及存储器
- 线程有相同地址空间, 同步通信较容易



## 2.7.1 线程的基本概念,

### ■ 3 线程的属性

#### ■ 独立轻型实体

- 不拥有系统资源
- 只有很少的必要的资源

- **TCB**, 程序计数器, 用于保留局部变量、状态参数和返回地址的寄存器和堆栈

#### ■ 调度和分派的基本单位

- 切换开销小

#### ■ 可并发执行

#### ■ 共享进程资源

- 同一进程的线程共享该进程的资源
- 地址空间, 文件, 定时器, 信号量机构等

## 2.7.1 线程的基本概念，

### ■ 4 线程的状态

#### ■ 状态参数。

- ① 寄存器状态， 它包括程序计数器**PC**和堆栈指针中的内容；
- ② 堆栈， 在堆栈中通常保存有局部变量和返回地址；
- ③ 线程运行状态， 用于描述线程正处于何种运行状态；
- ④ 优先级， 描述线程执行的优先程度；
- ⑤ 线程专有存储器， 用于保存线程自己的局部变量拷贝；
- ⑥ 信号屏蔽， 即对某些信号加以屏蔽。

#### ■ 线程运行状态。

- 各线程之间也存在着共享资源和相互合作的制约关系，致使线程在运行时也具有间断性。
- 相应地，线程在运行时，也具有下述三种基本状态：① 执行状态，表示线程正获得处理机而运行；② 就绪状态，指线程已具备了各种执行条件，一旦获得**CPU**便可执行的状态；③ 阻塞状态，指线程在执行中因某事件而受阻，处于暂停执行时的状态。



## 2.7.1 线程的基本概念,

### ■ 5 进程的创建和终止

#### ■ 创建

- 应用程序启动时，通常仅有一个初始化线程在执行
- 可根据需要再去创建若干个线程。
- 在创建新线程时，需要利用一个线程创建函数(或系统调用)，并提供相应的参数，如指向线程主程序的入口指针、堆栈的大小，以及用于调度的优先级等。在线程创建函数执行完后，将返回一个线程标识符供以后使用。 葛

#### ■ 终止线程的方式

- 在线程完成了自己的工作后自愿退出；
- 线程在运行中出现错误或由于某种原因而被其它线程强行终止。



## 2.7.1 线程的基本概念,

### ■ 5. 多线程OS中的进程

- 作为系统资源分配的单位
  - 用户地址空间、同步通信机制、文件、设备、地址映射表
- 可包括多个线程
- 进程不是一个可执行的实体
  - 线程为独立运行单位
  - 进程执行实质上是线程执行

## 2.7.2 线程的同步和通信，

### ■ 1 互斥锁(mutex) 葛

- 开锁，上锁，实现对资源互斥访问的机制。
- 时间和空间开销低， 适合高频度使用的共享资源
- 故障
  - 线程1：—— **mutex1** → 临界区C → 申请**mutex2**(失败 → 阻塞
  - 线程2：占有 **mutex2** → 申请**mutex1** → 阻塞

### ■ 2 条件变量

- 每一个条件变量通常都与一个互斥锁一起使用，亦即，在创建一个互斥锁时便联系着一个条件变量
- 单纯的互斥锁用于短期锁定，主要是用来保证对临界区的互斥进入；而条件变量则用于线程的长期等待，直至所等待的资源成为可用的。



## 2.7.2 线程的同步和通信,

### ■ 2 条件变量

申请资源:

Lock mutex

check data structures;

while(resource busy)

wait(condition variable);

mark resource as busy;⌘

unlock mutex;

释放资源:

Lock mutex⌘

mark resource as free;⌘

unlock mutex;⌘

wakeup(condition variable);⌘



## 2.7.2 线程的同步和通信。

### ■ 3 信号量机制

#### ■ 私有信号量(**private semaphore**)。

- 实现同一进程中各线程之间的同步
- 其数据结构是存放在应用程序的地址空间中
- 私有信号量属于特定的进程所有，**OS**并不知道私有信号量的存在

#### ■ 公用信号量(**public semaphore**)。

- 实现不同进程间或不同进程中各线程之间的同步
- 有公开的名字供所有的进程使用，故而把它称为公用信号量
- 其数据结构是存放在受保护的系统存储区中，由**OS**为它分配空间并进行管理，故也称为系统信号量。
- 如果信号量的占有者在结束时未释放该公用信号量，则**OS**会自动将该信号量空间回收，并通知下一进程。可见，公用信号量是一种比较安全的同步机制。





## 2.7.3 线程的实现方式,

### ■ 1 内核支持线程

- 线程的创建、撤销、切换依靠内核，在内核空间完成
- 内核是根据该控制块而感知某线程的存在的，并对其加以控制。

### ■ 2 用户级线程

- 线程的创建、撤消、线程之间的同步与通信等功能，都无须利用系统调用来实现。
- 对于用户级线程的切换，通常是发生在一个应用进程的诸多线程之间，这时，也同样无须内核的支持。
- 由于切换的规则远比进程调度和切换的规则简单，因而使线程的切换速度快。
- 可见，这种线程是与内核无关的。



## 2.7.3 线程的实现方式,

### ■ 2 用户级线程

#### ■ 优点

- 线程切换不需要转换到内核空间，减少了管理开销
- 各进程可以选择适合自己的线程调度方法，与系统无关
- 用户级线程实现与操作系统平台无关，属用户程序

#### ■ 缺点

- 线程进行系统调用时如阻塞，则其所属进程阻塞，该进程内所有线程均阻塞
- 内核每次分配给进程一个**CPU**，进程中只有一个线程执行，对于对称多处理机情形效率低

### ■ 3 组合方式



## 2.7.4 线程的实现，

- **1. 内核支持线程的实现**
  - 创建进程时，为其分配一个任务数据区，其中包含若干的线程控制块**TCB**
  - 进程创建线程时，为新线程分配**TCB**
  - 撤销进程时，回收**TCB**
  - 调度和切换与进程调度切换相似
- **2. 用户级线程的实现**



## 2.7.4 线程的实现,

### ■ 2. 用户级线程的实现

#### ■ 1) 运行时系统(Runtime System)葛

- 用于管理和控制线程的函数(过程)的集合
- 包括用于创建和撤消线程的函数、线程同步和通信的函数以及实现线程调度的函数等。
- 运行时系统中的所有函数都驻留在用户空间, 并作为用户级线程与内核之间的接口。
- 线程切换无需内核, 由运行时系统完成

#### ■ 2) 内核控制线程葛

- 轻型进程LWP(Light Weight Process)。
- 每一个进程都可拥有多个LWP, 每个LWP都有自己的数据结构(如TCB), 其中包括线程标识符、优先级、状态, 另外还有栈和局部存储区等。它们也可以共享进程所拥有的资源。
- LWP可通过系统调用来获得内核提供的服务, 这样, 当一个用户级线程运行时, 只要将它连接到一个LWP上, 此时它便具有了内核支持线程的所有属性。



## 2.7.4 线程的实现。

### ■ 3 用户级线程与内核控制线程的关系

#### ■ 一对一模型

- 每个用户线程都设置一个内核控制线程与之连接
- 并行能力强，系统开销大

#### ■ 多对一模型

- （进程内的）多个用户线程映射到一个内核控制线程
- 管理开销小，线程阻塞会导致进程阻塞

#### ■ 多对多模型

- 内核控制线程的数目可以根据情况变化