linux进程间通信介绍

written by 王保明

# 进程同步与进程互斥

|  |
| --- |
| **顺序程序与并发程序特征**   * 顺序程序特征 * 顺序性 * 封闭性：（运行环境的封闭性） * 确定性 * 可再现性 * 并发程序特征 * 共享性 * 并发性 * 随机性 |
| **进程互斥**   * 由于各进程要求共享资源，而且有些资源需要互斥使用，因此各进程间竞争使用这些资源，进程的这种关系为进程的互斥 * 系统中某些资源一次只允许一个进程使用，称这样的资源为临界资源或互斥资源。 * 在进程中涉及到互斥资源的程序段叫临界区 |
| **互斥示例** |
| **进程同步**   * 进程同步指的是多个进程需要相互配合共同完成一项任务。 |
| **同步示例** |
| **进程间通信目的**   * 数据传输：一个进程需要将它的数据发送给另一个进程 * 资源共享：多个进程之间共享同样的资源。 * 通知事件：一个进程需要向另一个或一组进程发送消息，通知它（它们）发生了某种事件（如进程终止时要通知父进程）。 * 进程控制：有些进程希望完全控制另一个进程的执行（如Debug进程），此时控制进程希望能够拦截另一个进程的所有陷入和异常，并能够及时知道它的状态改变。 |

# 进程间通信发展

|  |
| --- |
| * 管道 * System V进程间通信 * POSIX进程间通信 |
| * Linux下的进程通信手段基本上是从UNIX平台上的进程通信手段继承而来的。而对UNIX发展做出重大贡献的两大主力AT&T的贝尔实验室及BSD（加州大学伯克利分校的伯克利软件发布中心）在进程间的通信方面的侧重点有所不同。前者是对UNIX早期的进程间通信手段进行了系统的改进和扩充，形成了“system V IPC”，其通信进程主要局限在单个计算机内；后者则跳过了该限制，形成了基于套接口（socket）的进程间通信机制。而Linux则把两者的优势都继承了下来 |
| • UNIX进程间通信（IPC）方式包括管道、FIFO以及信号。  • System V进程间通信（IPC）包括System V消息队列、System V信号量以及System V共享内存区。  • Posix 进程间通信（IPC）包括Posix消息队列、Posix信号量以及Posix共享内存区。 |
| **有关POSIX**  POSIX 表示可移植操作系统接口（Portable Operating System Interface ，缩写为 POSIX ），POSIX标准定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称，其正式称呼为IEEE 1003，而国际标准名称为ISO/IEC 9945。  POSIX标准意在期望获得源代码级别的软件可移植性。换句话说，为一个POSIX兼容的操作系统编写的程序，应该可以在任何其它的POSIX操作系统（即使是来自另一个厂商）上编译执行。  POSIX 并不局限于 UNIX。许多其它的操作系统，例如 DEC OpenVMS 支持 POSIX 标准，尤其是 IEEE Std. 1003.1-1990（1995 年修订）或 POSIX.1，POSIX.1 提供了源代码级别的 C 语言应用编程接口（API）给操作系统的服务程序，例如读写文件。POSIX.1 已经被国际标准化组织（International Standards Organization，ISO）所接受，被命名为 ISO/IEC 9945-1:1990 标准。  **有关POSIX2**  POSIX的诞生和Unix的发展是密不可分的，电气和电子工程师协会（Institute of Electrical and Electronics Engineers，IEEE）最初开发 POSIX 标准，是为了提高 UNIX 环境下应用程序的可移植性。Unix于70年代诞生于贝尔实验室，并于80年代向美各大高校分发V7版的源码以做研究。加利福尼亚大学伯克利分校在V7的基础上开发了BSD Unix。后来很多商业厂家意识到Unix的价值也纷纷以贝尔实验室的System V或BSD为基础来开发自己的Unix，较著名的有Sun OS，AIX，VMS。  然而，POSIX 并不局限于 UNIX。许多其它的操作系统，例如 DEC OpenVMS 支持 POSIX 标准，尤其是 IEEE Std. 1003.1-1990（1995 年修订）或 POSIX.1，POSIX.1 提供了源代码级别的 C 语言应用编程接口（API）给操作系统的服务程序，例如读写文件。POSIX.1 已经被国际标准化组织（International Standards Organization，ISO）所接受，被命名为 ISO/IEC 9945-1:1990 标准。  POSIX 已发展成为一个非常庞大的标准族，某些部分正处在开发过程中。POSIX 与 IEEE 1003 和 2003 家族的标准是可互换的。  Windows NT-based 系统不能直接支持新版POSIX接口，仅支持第一版POSIX v1:ISO/IEC9945-1:1990[1] 版标准。由于仅支持第一版POSIX的Windows NT-based系统不能创建符合POSIX接口标准的线程和窗体、套接字，所以微软公司提供POSIX兼容层Microsoft POSIX subsystem软件包（Windows Services for UNIX）以支持新版POSIX接口，Windows 系统还可以运行其他POSIX兼容层例如Cygwin. |
| **进程间通信分类**   * 文件 * 文件锁 * 管道（pipe）和有名管理（FIFO） * 信号（signal） * 消息队列 * 共享内存 * 信号量 * 互斥量 * 条件变量 * 读写锁 * 套接字（socket）   **System V IPC & POSIX IPC**   * System V IPC   + System V 消息队列   + System V 共享内存   + System V 信号量 * POSIX IPC   + 消息队列   + 共享内存   + 信号量   + 互斥量   + 条件变量   + 读写锁 |
|  |
| **进程间共享信息的三种方式** |
| **IPC对象的持续性**   * 随进程持续：一直存在直到打开的最后一个进程结束。（如pipe和FIFO） * 随内核持续：一直存在直到内核自举（重启）或显式删除（如System V消息队列、共享内存、信号量） * 随文件系统持续：一直存在直到显式删除，即使内核自举还存在。（POSIX消息队列、共享内存、信号量如果是使用映射文件来实现） |
| **IPC对象是被linux内核管理** |
|  |

# 2进程死锁及解决方案

|  |
| --- |
| **死锁**   * 死锁是指多个进程之间相互等待对方的资源，而在得到对方资源之前又不释放自己的资源，这样，造成循环等待的一种现象。如果所有进程都在等待一个不可能发生的事，则进程就死锁了。 |
| **死锁产生的必要条件**   * 互斥条件   + 进程对资源进行排它性使用，即在一段时间内某资源仅为一个进程所占用。 * 请求和保持条件   + 当进程因请求资源而阻塞时，对已获得的资源保持不放。 * 不可剥夺条件   + 进程已获得的资源在未使用完之前，不能被剥夺，只能在使用完时由自己释放。 * 环路等待条件   + 各个进程组成封闭的环形链，每个进程都等待下一个进程所占用的资源 |
| **防止死锁办法**   * 资源一次性分配：（破坏请求和保持条件） * 可剥夺资源：破坏不可剥夺条件） * 资源有序分配法：（破坏循环等待条件） |
| **死锁避免**   * 预防死锁的几种策略，会严重地损害系统性能。因此在避免死锁时，要施加较弱的限制，从而获得较满意的系统性能。 * 由于在避免死锁的策略中，允许进程动态地申请资源。因而，系统在进行资源分配之前预先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，则将资源分配给进程；否则，进程等待。其中最具有代表性的避免死锁算法是银行家算法。 |
| **银行家算法**   * 为保证资金的安全，银行家规定： * (1) 当一个顾客对资金的最大需求量不超过银行家现有的资金时就可接纳该顾客； * (2) 顾客可以分期贷款，但贷款的总数不能超过最大需求量 * (3) 当银行家现有的资金不能满足顾客尚需的贷款数额时，对顾客的贷款可推迟支付，但总能使顾客在有限的时间里得到贷款 * (4) 当顾客得到所需的全部资金后，一定能在有限的时间里归还所有的资金. |
| **哲学家就餐问题**   * 五个哲学家围在一个圆桌就餐，每个人都必须拿起两把叉子才能用餐 * 哲学家就餐问题解法   + 服务生解法   + 最多4个哲学家   + 仅当一个哲学家两边筷子都可用时才允许他拿筷子   + 给所有哲学家编号，奇数号的哲学家必须首先拿左边的筷子，偶数号的哲学家则反之 |
| **信号量**   * 信号量和P、V原语由Dijkstra（迪杰斯特拉）提出 * 信号量   + 互斥：P、V在同一个进程中   + 同步：P、V在不同进程中 * 信号量值含义   + S>0：S表示可用资源的个数   + S=0：表示无可用资源，无等待进程   + S<0：|S|表示等待队列中进程个数 |
| struct semaphore  {  int value;  pointer\_PCB queue;  }  **P原语**  P(s)  {  s.value = s.value--;  if (s.value < 0)  {  该进程状态置为等待状状态  将该进程的PCB插入相应的等待队列s.queue末尾  }  }  **V原语**  V(s)  {  s.value = s.value++;  if (s.value < =0)  {  唤醒相应等待队列s.queue中等待的一个进程  改变其状态为就绪态  并将其插入就绪队列  }  } |
| **用PV原语解决司机与售票员问题** |
| **用PV原语解决民航售票问题** |
| **用PV原语解决汽车租赁问题**   * 有一汽车租赁公司有两部敞篷车可以出租，假定同时来了四个顾客都要租敞篷车，那么肯定会有两个人租不到。   S(2)  P(S)  租车  还车  V(S) |
|  |

eg:

sem\_id = sem\_create(key, num);

sem\_p(sem\_id);其他进程需要临界区资源，内核管理其进入等待队列

xxx

sem\_v(sem\_id);告诉linux内核，因为执行临界区代码而等待的进程队列，其中一个唤醒

sem\_d(sem\_id);

pv操作 cpu从硬件级别是如何实现的 -> 关中断