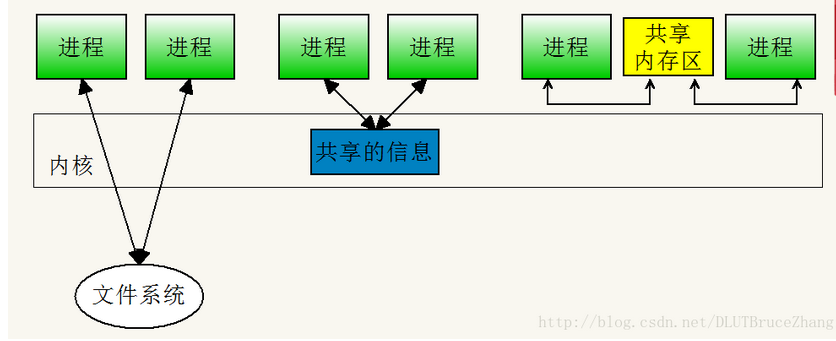
## 进程间通信介绍（一）

1. 互斥与同步
   1. 互斥：不同间进程竞争资源。该资源叫做临界资源；操作该资源的代码代码叫做临界区
   2. 同步：不同进程相互合作完成一项任务
2. 进程间通讯目的
   1. 传输数据
   2. 共享资源（互斥同步）
   3. 事件通知
   4. 控制进程（debug）
3. 发展
   1. 管道
   2. system V
   3. POSIX
4. 分类
   1. 文件（数据传输）
   2. 文件锁（资源共享）
   3. 管道（数据传输）
   4. 信号（事件通知、进程控制）
   5. 消息队列（数据传输）
   6. 共享内存（资源共享）
   7. 信号量（资源共享）
   8. 互斥量（资源共享）
   9. 条件变量（资源共享）
   10. 读写锁（资源共享）
   11. 套接字（tcp、udp、unix域）
5. 共享信息三种方式
6. IPC对象的持续性
   1. 随进程持续
   2. 随内核持续
   3. 随文件系统持续

## 进程间通信介绍（二）

1. 死锁

|  |  |
| --- | --- |
| 构成条件 | * 互斥 * 请求与等待 * 不可剥夺 * 环路等待 |
| 解决方案 | * 资源统一分配：只有请求资源全部可满足时才会分配 * 可抢占 * 资源有序分配：破坏环路等待 |
| 典型算法：  银行家算法  哲学家就餐问题 |  |

1. 信号量和PV原语

|  |
| --- |
| struct semaphore**{**  int value**;**  pointer\_PBC queue**;**  **}**  P**(**s**){**  **--**s**.**value**;**  **if(**s**.**value**<**0**){**  进程入等待队列  **}**  **}**  V**(**s**){**  **++**s**.**value**;**  **if(**s**.**value**<=**0**){**  唤醒进程  **}**  **}** |
| 总结：  P、V操作用于不同进程时，同步；  P、V用于一个进程，互斥 |

1. 实际应用

|  |  |
| --- | --- |
| 公交司机和售票员 | 解法1：  司机进程：  S1**(**0**)**  **while(**1**){**  P**(**S1**);**  开车**;**  停车**;**  V**(**S2**);**  **}**  售票员进程：  S2**(**0**)**  **while(**1**){**  关门**;**  V**(**S1**);**  售票**;**  P**(**S2**);**  开门**;**  **}**  解法2：  司机进程：  S1**(**0**);**  **while(**1**){**  停车  V**(**S2**);**  P**(**S1**);**  开车  **}**  售票员进程：  S2**(**0**);**  **while(**1**){**  P**(**S2**)**  开门**;**  关门；  售票；  V**(**S1**);**  **}** |
| 民航售票 | 民航售票：  余票 **=** x  S**(**1**);**  P**(**S**);**  **if(**x**>**0**)**  **--**x**;**  V**(**S**);**  其中x称为时临界资源；这块代码称为临界区 |
| 汽车租赁：  总共有2辆车，四位顾客 | S**(**2**);**  P**(**S**);**  租车**;**  还车**;**  V**(**S**);** |

## System V IPC概述

1. 消息队列、共享内存、信号量

|  |  |  |
| --- | --- | --- |
| 消息队列 | 共享内存 | 信号量 |
| msgget | shmget | semget |
| msgctl | shmctl | semctl |
| msgsnd  msgrcv |  |  |

1. ipc\_perm结构

|  |
| --- |
| struct ipc\_perm**{**  key\_t key**;**  uid\_t uid**;**  gid\_t gid**;**  cid\_t cuid**;**  gid\_t cgid**;**  mode\_t mode**;**  unsigned short seq**;**  **};** |

1. 各个字段解释

使用msgget函数将会填充以下字段

|  |  |
| --- | --- |
| key | 1. 使用ftok   #include<unistd.h>  #include<sys/ipc.h>  #include<sys/stat.h>  #include<stdio.h>  #include<assert.h>  int main**(**int argc**,**char**\*\*** argv**){**  struct stat st**;**  assert**(**argc **==** 2**);**  stat**(**argv**[**1**],&**st**);**  printf**(**"st\_dev: %lx, st\_ino: %lx, key: %lx\n"**,**  st**.**st\_dev**,**st**.**st\_ino**,**ftok**(**argv**[**1**],**0x57**));**  **return** 0**;**  **}**  输出：  st\_dev**:** 801**,** st\_ino**:** 400a3**,** key**:** 570100a3  可以看出：key = fotk参数2（L8）+st\_dev（L8）+st\_ino(L16)组成   1. PRIVATE |
| uid  gid  cuid  cgid | uid和gid可以使用xxxctl改变  cuid和cgid不可以改变 |
| mode | 用户|组|其他 权限控制 |
| seq | xxxget返回值，用于标识消息队列。重用每一个表项时增加表项个数（MAXMNI）。 |

1. 限制

|  |  |
| --- | --- |
| 最大消息队列数 | MSGMNI |
| 一个队列最大消息数 | MSGMNB |
| 一个消息最大字节数 | MSGMAX |

## 互斥锁与条件变量

### 问题及解决

|  |  |
| --- | --- |
| man pthread\_mutex\_lock显式没有相关的函数 | 安装manpages-posix-dev  apt-get install manpages-posix-dev |
|  |  |

### 互斥锁

|  |
| --- |
| 1 #include**<**unistd**.**h**>**  2 #include**<**pthread**.**h**>**  3 #include**<**stdio**.**h**>**  4  5 #include**<**assert**.**h**>**  6 #include**<**string**.**h**>**  7  8 #define MAXITEM 100000  9 #define MAXNTHREADS 10  10 struct Shared**{**  11 pthread\_mutex\_t mutex**;**  12 int buf**[**MAXITEM**];**  13 int index**;**  14 int nval **;**  15 **}**Shared**;**  16 struct Shared shared **=** **{**PTHREAD\_MUTEX\_INITIALIZER**,{**0**},**0**,**0**}** **;**  17 void**\*** product**(**void**\*** arg**){**  18 **while(**1**){**  19 pthread\_mutex\_lock**(&**shared**.**mutex**);**  20 **if(**shared**.**nval **>=** MAXITEM**){**  21 pthread\_mutex\_unlock**(&**shared**.**mutex**);**  22 **return** **NULL;**  23 **}**  24 shared**.**buf**[**shared**.**index**++]** **=** shared**.**nval**++;**  25 //shared.buf[shared.index] = shared.nval;  26 //shared.index++;  27 //shared.nval++;  28 pthread\_mutex\_unlock**(&**shared**.**mutex**);**  29 **++\*((**int**\*)**arg**);**  23 **}**  24 shared**.**buf**[**shared**.**index**++]** **=** shared**.**nval**++;**  25 pthread\_mutex\_unlock**(&**shared**.**mutex**);**  26 **++\*((**int**\*)**arg**);**  27 **}**  28 **}**  29 void consume\_wait**(**int i**){**  30 **while(**1**){**  31 pthread\_mutex\_lock**(&**shared**.**mutex**);**  32 **if(**i **<** shared**.**index**){**  33 pthread\_mutex\_unlock**(&**shared**.**mutex**);**  34 **break;**  35 **}**  36 pthread\_mutex\_unlock**(&**shared**.**mutex**);**  37 **}**  38 **}**  39 void**\*** consume**(**void **\*){**  40 **for(**int i **=** 0 **;**i**<**MAXNTHREADS**;++**i**){**  41 consume\_wait**(**i**);**  42 **if(**shared**.**buf**[**i**]** **!=** i**){**  43 printf**(**"consume error\n"**);**  44 **}**  45 **}**  46 **}**  47 int main**(){**  48 pthread\_t mts**[**MAXNTHREADS**+**1**];**  49 int count**[**MAXNTHREADS**]** **=** **{**0**};**  50 **for(**int i **=** 0 **;** i **<** MAXNTHREADS**;++**i**){**  51 pthread\_create**(&**mts**[**i**],NULL,**product**,&**count**[**i**]);**  52 **}**  53 pthread\_create**(&**mts**[**MAXNTHREADS**],NULL,**consume**,NULL);**  54 **for(**int i **=** 0 **;** i**<** MAXNTHREADS**;++**i**){**  55 pthread\_join**(**mts**[**i**],NULL);**  56 printf**(**"%d : %d\n"**,**i**,**count**[**i**]);**  57 **}**  58 pthread\_join**(**mts**[**MAXNTHREADS**],NULL);**  59 **return** 0**;**  60 **}**  输出：（每次运行结果不一样）  0 : 0  1 : 0  2 : 0  3 : 0  4 : 0  5 : 0  6 : 60714  7 : 39286  8 : 0  9 : 0 |