选择进程间通信

文件实现进程间通信

文件实现进程间通信的基本思想是:服务器进程将随机数写入文件;客户端进程从文件中读出随机数.

命名管道实现进程间通信

命名管道实现进程间通信的做法是:服务器进程将数据写入管道,客户端进程从管道中读出数据并显示在标准输出设备上

共享内存

共享内存是System V的一种进程间通信机制.进程可以使用共享内存的ID连接到某共享内存段,或者指向此段的指针来使用共享内存.共享内存允许两个及以上进程共享同一段物理内存,具有权限的进程可以将共享内存映射为自己空间的一部分.每个共享内存有一个`shmid_ds类型的结构与之对应,该结构记录共享内存的一些属性.

共享内存相关的系统调用

1. 创建共享内存
2. 连接共享内存
3. 使用共享内存
4. 解脱共享内存
5. 删除共享内存

```
int shmget(key_t key,size_t size,int shmflg);
//参数1:用户创建共享内存的键值
//参数2:共享内存的大小(字节)
//阐述3:共享内存的权限
```

shmget函数用来创建共享内存,某个用户的进程创建共享内存后,该用户创建的其他进程都对共享内存有写入权限,其他用户的进程对共享内存只有读权限.进程访问已有共享内存时,也要先调用shmget函数,表示应用此共享内存.创建好共享内存后需连接到进程的地址空间后使用

```
int shmat(int shmid,void * shmaddr, int flag)
//参数1:共享内存ID号
//参数2:共享内存在当前进程中的起始地址
//参数3:访问共享内存的方式
```

连接共享内存使用`shmat`函数,连接后,共享内存加入进程的地址空间,成为地址空间的一部分. shmaddr的取值说明

- shmaddr == 0:将此共享内存连接到内核选择的第一个可用地址上
- shmaddr != 0 && flag中未指定 `SHM_RND`标识:将该共享内存段连接到shmaddr所指定的地址上

- shmaddr != 0 && flag中指定了SHM_RND标识:将该共享内存段连接到shmaddr向下取到的最近的一个`SHMLBA的地址上.

调用成功后,除了返回共享内存的实际起始地址,还会将共享内存中响应的shmid_ds结构中的shm_nattach计数器加一

```
int shmodt(void * shmaddr)
参数1:共享内存的起始位置
```

使用shmdt函数可以将共享内存从进程的地址空间解除出来.调用成功后,将shimid_ds结构中的shm_nattach计数器减一,当共享内存段对应的shm_nattach计数器值为0时,可将共享内存删除.

```
int shmctl(int shmid,int cmd,struct shmid_ds *buf)
//参数1:共享内存的ID号
//参数2:要执行的操作
//参数3:根据cmd的不同而变化
```

该函数常用来参数指定额共享内存,除此之外,还可以对共享内存执行多种操作.

信号量

使用信号量可以协调使用共享内存进程间通信时由于读写造成的冲突.

1. semget获取信号量集合
2. semctl为信号量赋初值
3. semop函数对信号量执行PV操作
4. semctl函数删除信号量集合

```
int semget(key_t key, int nsems, int flag)
//参数1:获取信号量的键值
//参数2:集合中信号量的个数
//参数3:使用信号量集合的权限
```

semget函数创建信号量集合,创建后其他函数引用也需使用semget函数.

```
int semctl(int semid,int semnum,int cmd[,union semun arg]);
// 参数1:信号量集的ID号
// 参数2:信号量编号
// 参数3:要执行的操作
```

该函数用于对信号量或信号量集合进行操作

```
int semop(int semid,struct sembuf semarray[],size_t nops)
// 参数1:信号量集的ID号
// 参数2:存放操作的数组
// 参数3:操作的步数
```

该函数以原子操作方式执行一系列操作,调用该函数给竞争资源加锁或解锁,亦或者是实现PV操作.

System V IPC

消息队列,共享内存和信号量称为System V IPC通信机制.

- 消息队列是一个消息的链式队列,通信双方通过读写消息来实现通信
- 共享内存就是多个进程可以访问同一块内存空间
- 信号量主要作用于进程及以及同一进程的不同线程之间通信的同步手段.

ipcs命令可以显示当前系统中这三种ICP资源的情况.在shell中ipcmk可以创建IPC资源,ipcrm可以删除IPC资源.

消息队列(FIFO)

默认情况,一个系统最多有16个消息队列同时存在.msqid_ds结构用于保存消息队列的内核数据结构.

消息队列相关系统调用

发送方创建消息队列,随后发送消息;接收方引用发送方创建的消息队列,接收消息;消息队列使用完毕后,删除消息队列.

```
int msgget(key_t key,int flag);
// 参数1:生成消息队列ID号所使用的键
// 参数2:消息队列的使用权限
```

mesget函数创建消息队列

```
int msgsnd(int msqid, void *ptr, size_t nbytes, int flag)
// 参数1:消息队列的ID号
// 参数2:指定待发送消息的指针
// 参数3:消息的大小
// 参数4:当消息队列满,如何处理
```

调用msgsnd函数,表示将指针ptr所指的,长度为nbytes的消息发送到ID号为msqid的消息队列中去,如果该消息队列已满,按照flag所设置的方式处理.

```
int msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag)
// 参数1:消息队列的ID号
```

```
// 参数2:指向待发送消息的指针
// 参数3:消息的大小
// 参数4:消息类型
// 参数5:当消息队列对空的时候,如何处理
```

消息发出后,接收方使用`msgrcv`函数接收消息.

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf)
// 参数1:消息队列的ID号
// 参数2:待执行的操作
// 参数3:存放消息队列属性的内存地址
```

当队列使用完毕,使用函数`msgctl`来删除消息队列.

消息队列不同于共享内存之处在于:内核要保证消息队列FIFO的性质,有多个接受方程序时,不会产生冲突.发送方发送的消息添加在队尾,接收方从队首接收消息,读写操作也不存在冲突.