联航精英训练营

UNIGRESS ELITE TRAINING CAMP

Linux高级程编程（五）

张勇涛

IPC（interprocess communication）

- 广义上一切能是进程间互相交流的对象和方法都是IPC,比如文件、管道、socket等。
- 狭义上的IPC特指消息队列、信号量和共享内存三种对象。

- IPC的应用范围：

消息队列应用于不同进程之间少量数据的顺序共享。

信号量应用于进程间的同步与互斥的控制。

共享内存则应用于进程间大批量数据的随机共享访问。

System V IPC

System V IPC 包括三种不同的进程间通信机制。

- **消息队列用来在进程之间传递消息。**消息队列与管道有点像，但存在两个重大差别。第一是消息队列是存在边界的，这样读者和写者之间以消息进行通信，而不是通过无分隔符的字节流进行通信的。第二是每条消息包括一个整型的 type 字段，并且可以通过类型类选择消息而无需以消息被写入的顺序来读取消息。
- **信号量允许多个进程同步它们的动作。**一个信号量是一个由内核维护的整数值，它对所有具备相应权限的进程可见。一个进程通过对信号量的值进行相应的修改来通知其他进程它正在执行某个动作。
- **共享内存使得多个进程能够共享内存**（即同被映射到多个进程的虚拟内存的页帧）的同一块区域（称为一个段）。由于访问用户空间内存的操作是非常快的，因此共享内存是其中一种速度最快的 IPC 方法：一个进程一旦更新了共享内存，那么这个变更会立即对共享同一个内存段的其他进程可见。

这三种 IPC 机制在功能上存在着很大的差异，但把它们放在一起讨论是有原因的。其中一个原因是它们是一同被开发出来的，它们在 20 世纪 70 年代后期首次出现在了 Columbus UNIX 系统中。这是 Bell 内部实现的一种 UNIX，用于运行电话公司记录保存和管理过程中用到的数据库和事物处理系统。在 1983 年左右，这些 IPC 机制出现在了主流的 System V UNIX系统上——System V IPC 的名称由此而来。

key值和ID值

- Linux系统为每个IPC机制都分配了唯一的ID，所有针对该IPC机制的操作都使用对应的ID。因此，通信的双方都需要通过某个办法来获取ID值。显然，创建者根据创建函数的返回值可获取该值，但另一个进程如何实现呢？显然，Linux两个进程不能随意访问对方的空间（一个特殊是，子进程可以继承父亲进程的数据，实现父亲进程向子进程的单向传递），也就不能够直接获取这一ID值。

- 为解决这一问题，IPC在实现时约定使用key值做为参数创建，如果在创建时使用相同的key值将得到同一个IPC对象的ID（即一方创建，另一方获取的是ID），这样就保证了双方可以获取用于传递数据的IPC机制ID值。

4

消息队列

管道的缺陷

- 管道是一种最古老的方式，他不打包就直接发送过去了

消息队列的优点

1. 消息队列先进先出
2. 消息队列将输出的信息进行了打包处理，这样就可以保证以每个消息为单位进行接收。
3. 消息队列可以对货物进行分类服务，标记各类别的货物，这样可以根据货物类别分别出货。

消息队列

消息队列就是消息的一个链表，它允许一个或多个进程向它写消息，一个或多个进程从中读消息。具有一定的FIFO的特性，但是可实现消息的随即查询。这些消息存在于内核中，由“队列ID”来标识。

消息队列的实现包括创建和打开队列、添加消息、读取消息和控制消息队列这四种操作。

msgget:	创建和打开队列，其消息数量受系统限制。
msgsnd:	添加消息，将消息添加到消息队列尾部。
msgrcv:	读取消息，从消息队列中取走消息。
msgctl:	控制消息队列。

消息队列的创建

int msgget (key_t key, int flag)

key：返回新的或已有队列的ID，IPC_PRIVATE

flag：低9位决定了共享内存属主,属组,和其它用户的访问权限，　其它位指定了共享内存的创建方式

消息队列创建方式参数:

取值	描述
IPC_CREAT	创建消息队列,如果消息队列已经存在,就获取消息队列的标示符
IPC_EXCL	与宏IPC_CREAT一起使用,单独使用无意义,此时只能创建一个不存在的共享内存,如果内存已经存在则调用失败

消息队列发送函数

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <msg.h>

int msgsnd (int msqid, struct msgbuf *msgp,
                    size_t msgsz, int msgflag)
```

参数：

msqid 是消息队列的队列ID；

msgp 是消息内容所在的缓冲区；

msgsz 是消息的大小,不包括消息类型部分而且取值必须大于0；

msgflg 是控制消息发送的方式，当前有堵塞和非堵塞两种。如果设置了IPC_NOWAIT,消息采用非阻塞方式发送,否则采取阻塞方式发送。

msgsnd堵塞的原因是：1. 消息队列满 2.消息总数满

msgsnd返回情况表

情况	Msgflg设置	返回值	errno
消息发送成功		0	无
消息队列被删除		-1	EIDRM
信号中断		-1	EINTR
消息发送阻塞	设置IPC_NOWAIT	-1	EAGAIN
	未设置	等待直到堵塞条件消失或者异常出现	

消息队列发送模型

1. 定义消息结构
2. 打开/创建消息队列
3. 组装消息
4. 发送消息
5. 发送判断

1. 消息结构

```
struct msgbuf
{
    long mtype; /* type of message */
    char mtext[100]; /* message text */
};
```

2. 打开/创建消息队列

```
int msid;
/* 打开（或创建）消息队列*/
if ((msid = msgget(0x1234, 0666|IPC_CREAT)) < 0)
{
    fprintf(stderr, "open msg %X failed.\n", 0x1234);
    return ;
}
```

3. 组装消息

```
struct mymsgbuf buf;      /* 申请消息缓冲*/
buf.mtype = 100;
strcpy(buf.ctext,  “hello world\n”  );
```

4.发送消息/5.发送判断

```
while((msgsnd(msid, &buf, strlen(buf.ctext), 0)) < 0)
{
    if (errno == EINTR)
        continue;
    return;
}
```

消息队列接收函数

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <msg.h>

int msgrcv (int msqid,    struct    msgbuf *msgp,
                  size_t msgsz,        long    msgtyp,  int msgflg)
```

msqid 是消息队列的引用标识符；

msgp 是接收到的消息将要存放的缓冲区；

msgsz 是消息的大小；

msgtyp 是期望接收的消息类型；

msgflg是标志。

msgrcv返回情况表

情况	msgflg设置	返回值	errno	队列情况
消息接收成功		实际接收消息的长度	无	删除消息
消息队列被删除		-1	EIDRM	已删除
信号中断		-1	EINTR	
无满足要求的信息	IPC_NOWAIT	-1	EAGAIN	
	未设置	等待直到堵塞条件消失或异常出现		
消息数据的长度大于msgsz	MSG_NOERROR	msgsz	无	截断消息
	未设置	-1	E2BIG	删除消息

消息队列接收模型

1. 定义消息结构
2. 打开/创建消息队列
3. 准备接受消息缓冲区
4. 接收消息
5. 接收判断

消息队列的例子

- msg1.c
- msg2.c

消息队列的控制

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <msg.h>

int msgctl (int msqid, int cmd, struct msqid_ds *buf)
```

- 功能:

  对消息队列执行各种控制,包括查询消息队列数据结构,改变消息队列的访问权限、改变消息队列属主信息和删除消息队列等

消息队列与管道

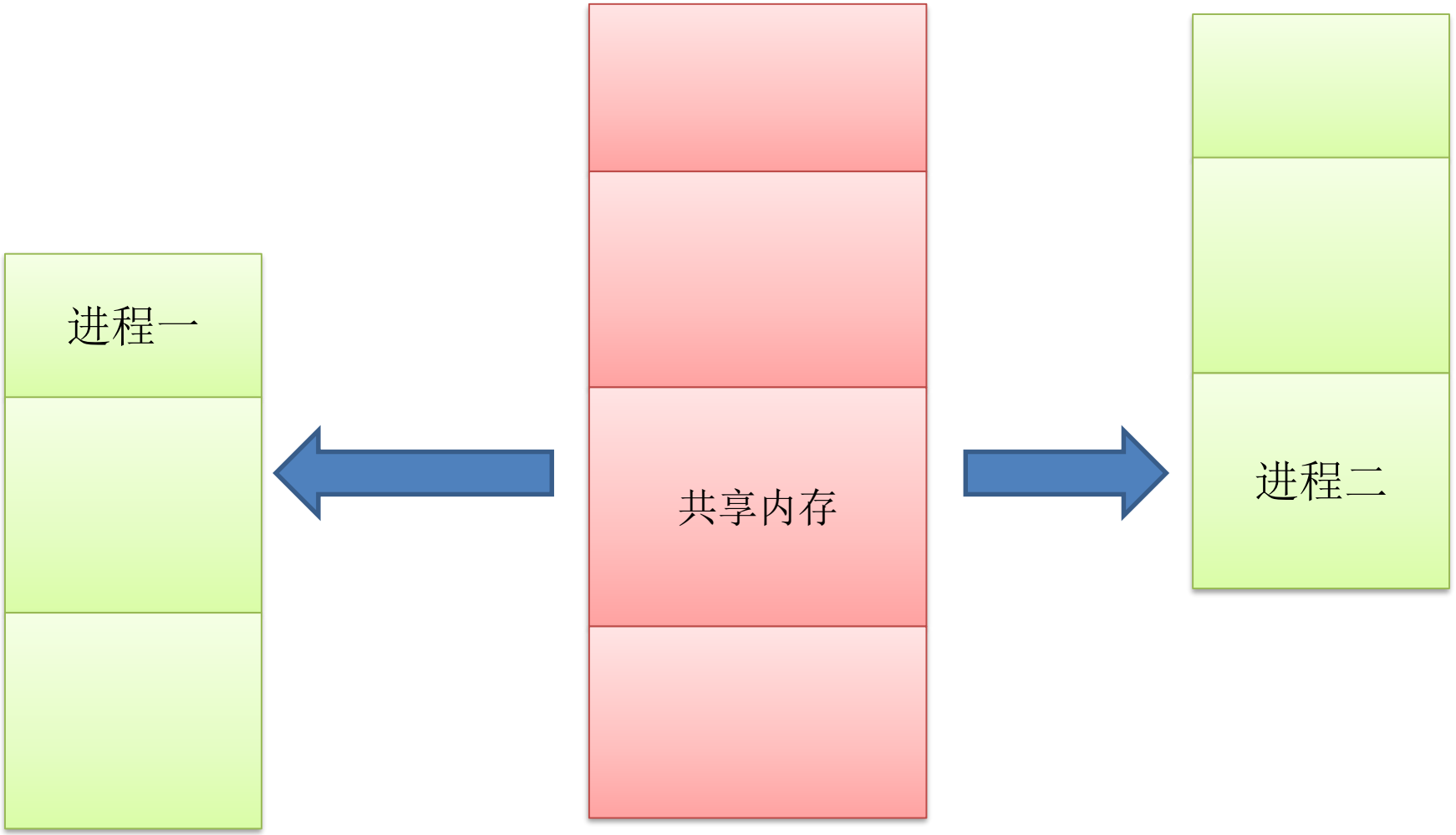
- 消息队列和管道都是一头写，一头读！

消息队列和管道的不同点：

1. 消息队列无需固定的读写进程，任何进程都可以读写
2. 消息队列可以支持多个进程，多个进程可以读写消息队列，即消息队列可以实现多对多，而管道只能是点对点
3. 消息队列只在内存中实现。
4. 消息队列不是只在UNIX和类UNIX操作系统中实现。几乎所有主流操作系统都支持消息队列。

SYSTEM V共享内存

共享内存



什么是共享内存

- 共享内存区域是被多个进程共享的一部分物理内存。
- 如果多个进程都把该内存区域映射到自己的虚拟地址空间，则这些进程就都可以直接访问该共享内存区域，从而可以通过该区域进行通信。
- 共享内存是进程间共享数据的一种最快的方法，一个进程向共享内存区域写入了数据，共享这个内存区域的所有进程就可以立刻看到其中的内容。
- 共享内存由进程创建,但是进程结束时共享内存仍然保留,除非该共享内存被显示的删除或重启系统。
- ipcs –m 查看系统中共享内存的基本信息

实现共享内存的步骤

一、创建共享内存，使用shmget函数。

二、映射共享内存，将这段创建的共享内存映射到具体的进程空间去，使用shmat函数。

- 注意:共享内存由进程创建,但是进程结束时,共享内存仍然保留,除非是共享内存被显示地删除或重启操作系统.

共享内存的创建

```
#include <sys/types.h>        /* For portability */
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
        Returns shared memory segment identifier on success, or –1 on error
```

系统调用：shmget( ) ;

原型：int shmget (key_t key, int size, int shmflg);

参数： key是共享内存的关键字,size指定共享内存的大小

shmflg的低9位决定了共享内存属主,属组,和其它用户的访问权限,其它位指定了共享内存的创建方式

返回值：如果成功，返回共享内存段标识符。

如果失败，则返回- 1：

共享内存创建方式参数:

参数	描述
IPC_CREAT	创建共享内存,如果共享内存已经存在,就获取共享内存的标示符
IPC_EXCL	与宏IPC_CREAT一起使用,单独使用无意义,此时只能创建一个不存在的共享内描述存,如果内存已经存在则调用失败

系统调用：shmget()

- 注意：

  当参数key的取值为IPC_PRIVATE时,将创建关键字为0的共享内存, UNIX内核中可以同时存在多个关键字为0的共享内存

系统调用：shmget()

errno =	EINVAL	(无效的内存段大小)
	EEXIST	(内存段已经存在，无法创建)
	EIDRM	(内存段已经被删除)
	ENOENT	(内存段不存在)
	EACCES	(权限不够)
	ENOMEM	(没有足够的内存来创建内存段)

共享内存的映射

系统调用：shmat();

原型：void *shmat (int shmid, char *shmaddr, int shmflg);

返回值：如果成功，则返回共享内存段连接到进程中的地址。

如果失败，则返回 NULL：

errno = EINVAL (无效的IPC ID 值或者无效的地址)

ENOMEM (没有足够的内存)

EACCES (存取权限不够)

- 在绝大部分情况, 我们指定参数shmaddr值为NULL标志位,系统将自动确认共享内存链接到进程空间的首地址。

共享内存的释放

当一个进程不再需要共享的内存段时，它将会把内存段从其地址空间中脱离。

系统调用：shmdt();

调用原型：int shmdt (char *shmaddr);

返回值：如果失败，则返回- 1;

errno = EINVAL (无效的连接地址)

共享内存的例子

- shm1.c
- shm2.c

共享内存与管道

1. 使用共享内存机制通讯的两个进程必须在同一台物理机器上
2. 共享内存的访问方式是随机的，而不是只能从一端写，从另一端读，因此，其灵活性比管道和套接字大的多，能够传递的信息也复杂的多。

共享内存的缺点

1. 管理复杂，两个进程必须在同一台物理机器上才能使用这种方式。
2. 安全性脆弱。

SYSTEM V信号量

System V 信号量

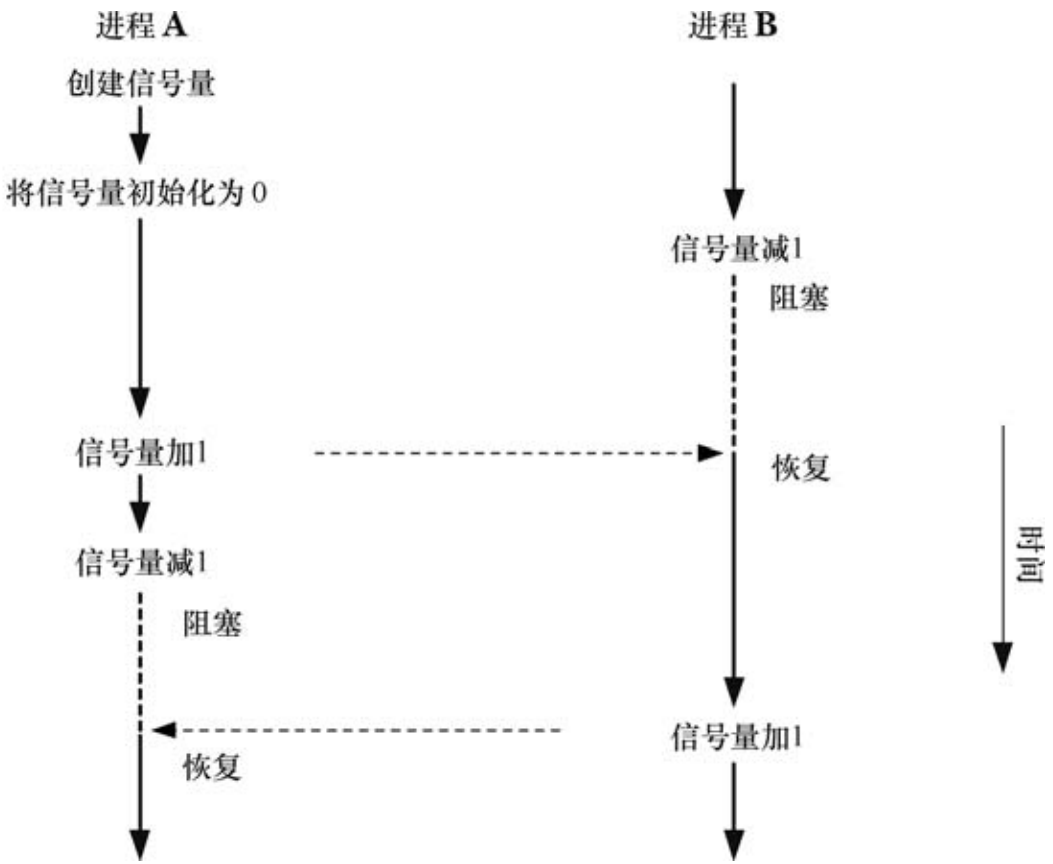
system V 信号量不是用来在进程间传输数据的。相反，它们用来同步进程的动作。 信号量的一个常见用途是同步对一块共享内存的访问以防止出现一个进程在访问共享内存的同时另一个进程更新这块内存的情况。

一个信号量是一个由内核维护的整数，其值被限制为大于或等于 0。
在一个信号量上可以执行各种操作（即系统调用），包括：

- 将信号量设置成一个绝对值；
- 在信号量当前值的基础上加上一个数量；
- 在信号量当前值的基础上减去一个数量；
- 等待信号量的值等于 0。

上面操作中的后两个可能会导致调用进程阻塞。当减小一个信号量的值时，内核会将所有试图将信号量值降低到 0 之下的操作阻塞。类似的，如果信号量的当前值不为 0，那么等待信号量的值等于 0 的调用进程将会发生阻塞。不管是何种情况，调用进程会一直保持阻塞直到其他一些进程将信号量的值修改为一个允许这些操作继续向前的值，在那个时刻内核会唤醒被阻塞的进程。

使用信号量同步两个进程



System V信号量 概述

使用 System V 信号量的常规步骤如下。

- 使用 semget()创建或打开一个信号量集。
- 使用 semctl() SETVAL 或 SETALL 操作初始化集合中的信号量。 （只有一个进程需要完成这个任务。 ）
- 使用 semop()操作信号量值。使用信号量的进程通常会使用这些操作来表示一种共享资源的获取和释放。
- 当所有进程都不再需要使用信号量集之后使用 semctl() IPC_RMID 操作删除这个集合。 （只有一个进程需要完成这个任务。 ）

大多数操作系统都为应用程序提供了一些信号量原语。但 System V 信号量表现出了不同寻常的复杂性，因为它们的分配是以备称为信号量集的组为单位进行的。在使用 semget()系统调用创建集合的时候需要指定集合中的信号量数量。虽然在同一时刻通常只操作一个信号量，但通过 semop()系统调用可以原子地在同一个集合中的多个信号量之上执行一组操作。

专业铸就品质 梦想成就未来

The Specialty Casts Quality,the Dream Gains Future.