几种 linux 下的进程通信手段

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）在进程间通信方面的侧重点有所不同。前者对 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”,通信进程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。Linux 则把两者继承了下来，如图示：

其中，最初 Unix IPC 包括：管道、FIFO、信号；System V IPC 包括：System V 消息队列、System V 信号灯、System V 共享内存区；Posix IPC 包括：Posix 消息队列、Posix 信号灯、Posix 共享内存区。有两点需要简单说明一下：

1）由于 Unix 版本的多样性，电子电气工程协会（IEEE）开发了一个独立的 Unix 标准，这个新的 ANSI Unix 标准被称为计算　肪车目梢浦残圆僮飧低辰缑妫≒SOIX）。现有大部分 Unix 和流行版本都是遵循 POSIX 标准的，而 Linux 从一开始就遵循 POSIX 标准；

2）BSD 并不是没有涉足单机内的进程间通信（socket 本身就可以用于单机内的进程间通信）。事实上，很多 Unix 版本的单机 IPC 留有 BSD 的痕迹，如 4.4BSD 支持的匿名内存映射、4.3+BSD 对可靠信号语义的实现等等。

linux 下进程间通信的几种主要手段简介：

1.管道

管道是进程间通信中最古老的方式，它包括无名管道和有名管道两种，前者可用于具有亲缘关系进程间的通信，即可用于父进程和子进程间的通信，后者额克服了管道没有名字的限制，因此，除具有前者所具有的功能外，它还允许无亲缘关系进程间的通信，即可用于运行于同一台机器上的任意两个进程间的通信。

无名管道由 pipe（）函数创建：

```
#include

int pipe（int filedis）；
```

参数 filedis 返回两个文件描述符：filedes[0]为读而打开，filedes 为写而打开。filedes 的输出是 filedes[0]的输入。

在 Linux 系统下，有名管道可由两种方式创建：命令行方式 mknod 系统调用和函数 mkfifo.下面的两种途径都在当前目录下生成了一个名为 myfifo 的有名管道：

方式一：mkfifo（“myfifo”,“rw”）；

方式二：mknod myfifo p

生成了有名管道后，就可以使用一般的文件 I/O 函数如 open、close、read、write 等来对它进行操作。

2.消息队列

消息队列是消息的链接表，包括 Posix 消息队列 system V 消息队列。消息队列用于运行于同一台机器上的进程间通信，它和管道很相似，有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

我们可以用流管道或者套接口的方式来取代它。

3.共享内存

共享内存是运行在同一台机器上的进程间通信最快的方式，因为数据不需要在不同的进程间复制。通常由一个进程创建一块共享内存区，其余进程对这块内存区进行读写。共享内存往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

首先要用的函数是 shmget，它获得一个共享存储标识符。

```
#include

#include

#include

int shmget（key_t key, int size, int flag）；
```

这个函数有点类似大家熟悉的 malloc 函数，系统按照请求分配 size 大小的内存用作共享内存。Linux 系统内核中每个 IPC 结构都有的一个非负整数的标识符，这样对一个消息队列发送消息时只要引用标识符就可以了。这个标识符是内核由 IPC 结构的关键字得到的，这个关键字，就是上面第一个函数的 key.数据类型 key_t 是在头文件 sys/types.h 中定义的，它是一个长整形的数据。在我们后面的章节中，还会碰到这个关键字。

当共享内存创建后，其余进程可以调用 shmat（）将其连接到自身的地址空间中。

```
void *shmat（int shmid, void *addr, int flag）；
```

shmid 为 shmget 函数返回的共享存储标识符，addr 和 flag 参数决定了以什么方式来确定连接的地址，函数的返回值即是该进程数据段所连接的实际地址，进程可以对此进程进行读写操作。

使用共享存储来实现进程间通信的注意点是对数据存取的同步，必须确保当一个进程去读取数据时，它所想要的数据已经写好了。通常，信号量被要来实现对共享存储数据存取的同步，另外，可以通过使用 shmctl 函数设置共享存储内存的某些标志位如 SHM_LOCK、SHM_UNLOCK 等来实现。

4. 信号量

信号量又称为信号灯，它是用来协调不同进程间的数据对象的，而最主要的应用是前一节的共享内存方式的进程间通信。本质上，信号量是一个计数器，它用来记录对某个资源（如共享内存）的存取状况。一般说来，为了获得共享资源，进程需要执行下列操作：

（1） 测试控制该资源的信号量。

（2） 若此信号量的值为正，则允许进行使用该资源。进程将进号量减 1.

（3） 若此信号量为 0,则该资源目前不可用，进程进入睡眠状态，直至信号量值大于 0,进程被唤醒，转入步骤（1）。

（4） 当进程不再使用一个信号量控制的资源时，信号量值加 1.如果此时有进程正在睡眠等待此信号量，则唤醒此进程。

维护信号量状态的是 Linux 内核操作系统而不是用户进程。我们可以从头文件/usr/src/linux/include/linux/sem.h 中看到内核用来维护信号量状态的各个结构的定义。信号量是一个数据集合,用户可以单独使用这一集合的每个元素。要调用的第一个函数是 semget,用以获得一个信号量 ID.

```
#include

#include

#include

int semget（key_t key, int nsems, int flag）;
```

key 是前面讲过的 IPC 结构的关键字，它将来决定是创建新的信号量集合，还是引用一个现有的信号量集合。nsems 是该集合中的信号量数。如果是创建新集合（一般在服务器中），则必须指定 nsems;如果是引用一个现有的信号量集合（一般在客户机中）则将 nsems 指定为 0.

semctl 函数用来对信号量进行操作。

int semctl（int semid, int semnum, int cmd, union semun arg）；

不同的操作是通过 cmd 参数来实现的，在头文件 sem.h 中定义了 7 种不同的操作，实际编程时可以参照使用。

semop 函数自动执行信号量集合上的操作数组。

int semop（int semid, struct sembuf semoparray[], size_t nops）；

semoparray 是一个指针，它指向一个信号量操作数组。nops 规定该数组中操作的数量。

下面，我们看一个具体的例子，它创建一个特定的 IPC 结构的关键字和一个信号量，建立此信号量的索引，修改索引指向的信号量的值，最后我们清除信号量。

5.套接口

套接口（socket）编程是实现 Linux 系统和其他大多数操作系统中进程间通信的主要方式之一。我们熟知的 WWW 服务、FTP 服务、TELNET 服务等都是基于套接口编程来实现的。除了在异地的计算机进程间以外，套接口同样适用于本地同一台计算机内部的进程间通信。