Linux中的进程间通信主要有：半双工管道PIPE、popen和pclose、命名管道FIFO、消息队列、信号量、共享内存、socket套接字。

消息队列、信号量、共享内存统称为XSI IPC，它们都分别有SystemV和POSIX两个版本。

共享内存是最快的进程通信方式，而socket是最具有通用性的进程通信方式。如果对于性能不是有太高的要求，使用socket进行进程间通信应该是最佳选择。

1. 半双工管道PIPE

这种形式的IPC被各个版本的UNIX版本与衍生版本（包括linux）所支持，尤其是半双工的PIPE在现实的应用中使用的最多。半双工的PIPE只能使用在具有亲缘关系的两个进程中，最常见的就是父子进程。通常的使用方法是父进程调用pipe函数，然后再调用fork函数产生一个子进程，在父子进程中关闭自己不需要的管道，然后通过复制文件描述符来确定最终的输入和输出。需要注意的是，在pipe中传输的是无符号的字节流形式，所以pipe不适用于传输一个struct，如果一定要这样做，就需要注意传输时不能超过PIPE\_BUFFER，并且在有多个进程写同一个管道时，可能会产生交错的数据。最后还需要注意的是pipe的读写之间的关系和写对读的依赖性，以及读操作可能产生的各种结果。

pipe的操作需要经过内核的中转才能进行，读写过程共需要两次内核空间的拷贝和输入/输出操作。

1. FILE \*popen(const char \*cmd, const char \*type)

Int pclose(FILE \*fp)

cmd为要执行的可执行程序，type为”r”或者”w”。

popen的作用是执行一个可执行程序产生一个子进程，然后返回一个连接子进程的标准输入或者准备输出的文件描述符。通过文件描述符读取子进程的标准输出或者对子进程的标准输入进行写操作。

1. 命名管道FIFO

管道只能由相关进程使用，这些相关进程的共同祖先创建了管道。但是通过FIFO，不相关的进程也能交换数据。

FIFO是通过 int mkfifo(const char \*pathname, mode\_t mode) 函数来创建一个命名管道，创建命名管道后就可以使用open函数来打开命名管道，然后就可以使用read、write、close等函数来操作命管道。

类似于管道，若用write写一个尚无进程为读而打开的FIFO，则产生SIGPIPE信号。基某个FIFO的最后一个写进程关闭了FIFO，是将为该FIFO的读进程产生一个文件结束标志。

1. 消息队列

消息队列有一些缺点，它没有访问计数。如果一个进程创建了一个消息队列，在该进程中放入了几则消息，然后终止，但是该消息队列及其内容并不会被删除。它们残留在系统中直到以下的情况发生：由某个进程调用msgrcv或者msgctl读取消息或删除消息队列；或者某个进程执行ipcrm命令删除消息队列；或由正在启动的系统删除消息队列。而对于管道和FIFO来说，最后一个访问它们的进程结束后，它们的数据都会被消除掉。由于以上的原因，新的应用程序中最好不要再使用消息队列。

1. 信号量

信号量不是为了在进程间传输数据，它是一个计数器，用于多进程间对共享对像的访问。为了获得共享资源，进程需要执行以下操作：

1. 测试信号量的值
2. 若此信号量的值大于0，表示进程可以使用该资源。然后进程将该信号量减1，表示它使用了一个资源单位。
3. 若此信号量的值为0，则进程进入休眠状态，直到信号量大于0时被唤醒。进程被唤醒后回到第(1)步。

由于SystemV的信号量太难用，所以只要系统支持POSIX版本的信号量，就应该尽量使用POSIX的信号量。

POSIX命名信号量有以下几个系统调用：

1. sem\_t \*sem\_open(

const char \*name, /\*信号量的命名，直接写一个文件名，不要带路径\*/

int flag, /\*创建还是打开信号量，若是创建则要指定后面两个参数\*/

mode\_t perms, /\*权限\*/

unsigned value /\*信号量的初始值\*/)

1. int sem\_close(sem\_t \*sem) //关闭命名信号量
2. int sem\_unlink(const char \*name) //删除命名信号量
3. int sem\_wait(sem\_t \*sem) //减少信号量
4. int sem\_post(sem\_t \*sem) //增加信号量
5. int sem\_getvalue(sem\_t \*sem) //如果信号量的值大于0则返回该值；如果信号量的值为0且没有任何进行在等待信号量则返回0；如果信号量的值为0且有进程在等待信号量则返回等待的进程的数量。
6. int sem\_trywait(sem\_t \*sem) //如果可能就减小信号量，如果信号量已经是0则进程不会阻塞而是返回-1。
7. int sem\_timewait(sem\_t \*sem) //一直等待信号量直到时间超时。这个时间不是一个相对时间，而是一个绝对时间（比如1:23:17PM）。

POSIX的未命名信号量：

1. int sem\_init(sem\_t \*sem, //指向信号量的指针，信号量可以是全局变量也可以动态分配，还可以将它放在共享内存中，这样就可以在进行中使用未命名信号量。

int pshared, //信号量是否在进程中共享

unsigned value) //信号量的初始值

1. sem\_destroy(sem\_t \*sem) //删除未命名信号量，但并不释放信号量的内存，因为这个函数并不知道信号量的内存是如何分配的，有可能是全局变量，还有可能是动态分配的。
2. 共享内存

POSIX共享内存特有的两个系统调用为：

1. int shm\_open(const char \*name, int flags, mode\_t perms) //创建共享内存对像。

name: POSIX IPC name

flags: 可以是O\_CREATE、O\_EXCL、O\_TRUNC、O\_RDONLY和O\_RDWR，但是不能是O\_WRONLY。

Perms: 如果对像是新建的，那么第3个参数指定它的权限。

1. int shm\_unlink(const char \*name) //删除共享内存对像。

创建共享内存后可以使用ftruncate(int fd, off\_t length)来设置共享内存的大小，因为其原始大小是0字节。 然后再使用mmap(void \*addr, size\_t len, int port, int flags, off\_t off)来将共享内存对像映射到进程的地址空间。ftruncate和mmap都是通用系统调用，可以在普通文件中使用。

1. socket套接字

套接字是最具通用性的进程间通信的方法。它不仅可以在同一台主机间进行进程通信，还可以在不同主机间进行进程通信。Socket套接字的系统调用分主要有：socket、bind、listen、accept、connect、send、sendto、recv、recvfrom、read、write、close等。