目录

[1. 管道(Pipe) 1](#_Toc200967627)

[匿名管道 1](#_Toc200967628)

[命名管道(FIFO) 2](#_Toc200967629)

[2. XSI IPC 2](#_Toc200967630)

[消息队列(Message Queue) 2](#_Toc200967631)

[信号量 2](#_Toc200967632)

[共享内存(Shared Memory) 2](#_Toc200967633)

[3. 本地套接字(Socket) 4](#_Toc200967634)

[创建套接字 4](#_Toc200967635)

[绑定套接字 5](#_Toc200967636)

[监听和接受连接 6](#_Toc200967637)

[客户进程套接字 6](#_Toc200967638)

[4. 内存映射文件(Memory-Mapped File) 6](#_Toc200967639)

## 1. 管道(Pipe)

### 匿名管道

只能用于具有亲缘关系的进程间通信(如父子进程)

#include <unistd.h>

**int pipe(int pipefd[2]);**

pipefd[2]用于返回两个表示管道两端的文件描述符，其中pipefd[0]用作读取数据，pipefd[1]用作写数据。

函数调用成功返回0，失败返回-1

直接使用read()和write()即可分别对管道读写端进行读写

管道中的数据如果没有一次性读完，剩下的数据可以在下一次继续读取

### 命名管道(FIFO)

可以在不相关的进程间进行通信。与匿名管道不同，命名管道是一个实际存在于文件系统中的特殊文件。可以有多个进程同时对其进行读写。

两个进程想要使用FIFO通信，必须都打开FIFO文件后才能进行读写

#include <sys/types.h>

#include <sys/stat.h>

**int mkfifo(const char \*pathname, mode\_t mode);**

pathname: 指定要创建的 FIFO 文件的路径名。

mode: 指定权限位。这些权限位会被进程的文件模式创建掩码（umask）修改。

返回值

成功时，返回 0

失败时，返回 -1 并设置相应的错误号（可通过 errno 获取）。

## 2. XSI IPC

XSI IPC包含的三种方式都需要指定key，不同的进程使用相同的key进行通信

### 消息队列(Message Queue)

消息队列允许进程以消息的形式发送数据。每个消息都有一个类型，接收者可以根据类型选择性地接收消息。相比管道，消息队列提供了更复杂的通信模式，并且消息是有格式的。

### 信号量

### 共享内存(Shared Memory)

共享内存允许多个进程访问同一块内存区域，从而实现高效的数据交换。为了防止多个进程同时修改共享内存内容而导致的数据不一致问题，通常需要配合使用某种形式的同步机制(如信号量)。

#### shmget

* **定义**: int shmget(key\_t key, size\_t size, int shmflg);
* **作用**: 创建或获取一个共享内存段。
  + key: 用于标识共享内存段的唯一标识符。通常使用 ftok 函数来生成一个唯一的键值。
  + size: 共享内存段的大小，通常以字节为单位。
  + shmflg: 操作的标志，常见的标志有：
    - IPC\_CREAT: 如果共享内存段不存在，则创建它。
    - IPC\_EXCL: 与 IPC\_CREAT 一起使用，如果共享内存段已经存在，则返回错误。
    - 权限标志，例如 0666 用于设置共享内存的访问权限。
* **返回值**: 返回共享内存段的标识符，如果失败返回 -1。
* **示例**: int shm\_id = shmget(1234, 1024, IPC\_CREAT | 0666);

#### shmat

* **定义**: void\* shmat(int shm\_id, const void \*shm\_addr, int shmflg);
* **作用**: 将共享内存段映射到当前进程的地址空间。
  + shm\_id: 共享内存段的标识符，通常是通过 shmget 获取的。
  + shm\_addr: 指定映射的起始地址，通常传 NULL 让系统选择一个合适的地址。
  + shmflg: 映射的标志，常用的标志包括：
    - SHM\_RDONLY: 只读方式映射共享内存。
    - SHM\_RND: 映射到一个页边界。
* **返回值**: 如果成功，返回共享内存段的地址（可以通过它访问共享内存）。如果失败，返回 (void\*) -1。
* **示例**: char \*shm\_ptr = (char \*)shmat(shm\_id, NULL, 0);

#### shmdt

* **定义**: int shmdt(const void \*shm\_addr);
* **作用**: 将共享内存从当前进程的地址空间中分离。每个通过 shmat 映射的共享内存段，都需要通过 shmdt 来解除映射。
  + shm\_addr: 映射的共享内存段的起始地址，通常是 shmat 返回的地址。
* **返回值**: 如果成功，返回 0。如果失败，返回 -1。

#### shmctl

* **定义**: int shmctl(int shm\_id, int cmd, struct shmid\_ds \*buf);
* **作用**: 控制共享内存段，执行一系列管理操作。
  + shm\_id: 共享内存段的标识符。
  + cmd: 控制命令，常见的命令有：
    - IPC\_STAT: 获取共享内存的信息。
    - IPC\_RMID: 删除共享内存段。
    - SHM\_LOCK: 锁定共享内存段。
    - SHM\_UNLOCK: 解锁共享内存段。
  + buf: 一个指向 struct shmid\_ds 结构体的指针，该结构体用于获取或设置共享内存的详细信息。
* **返回值**: 如果成功，返回 0。如果失败，返回 -1。

#### ftok

* **定义**: key\_t ftok(const char \*pathname, int proj\_id);
* **作用**: 将文件路径和一个项目标识符 proj\_id 转换为一个唯一的共享内存键值 key\_t。这个键值可以作为 shmget 函数的 key 参数。
* **返回值**: 返回生成的唯一键值。
* **示例**: key\_t key = ftok("/tmp", 65);

#### shmget 返回的 struct shmid\_ds 结构体

* struct shmid\_ds 是与共享内存段相关的控制信息结构体，它包含了共享内存段的元数据。它通常用于与 shmctl 一起使用来获取或设置共享内存的属性。

struct shmid\_ds

{

struct ipc\_perm shm\_perm; // 权限信息

size\_t shm\_segsz; // 共享内存段的大小

time\_t shm\_atime; // 最近一次附加时间

time\_t shm\_dtime; // 最近一次分离时间

time\_t shm\_ctime; // 最近一次操作时间

pid\_t shm\_cpid; // 创建共享内存的进程 PID

pid\_t shm\_lpid; // 最后一个操作共享内存的进程 PID

shmatt\_t shm\_nattch; // 附加到共享内存的进程数量

};

## 3. 本地套接字(Socket)

### 创建套接字

#include <sys/types.h>

#include <sys/socket.h>

**int socket(int domain, int type, int protocol);**

domain: 指定使用的协议族（Protocol Family），常见的值包括：

AF\_INET：IPv4 网络协议。

AF\_INET6：IPv6 网络协议。

AF\_UNIX 或 AF\_LOCAL：本地通信。

type: 指定服务类型或语义，常见的值包括：

SOCK\_STREAM：TCP模式，发送字节流

SOCK\_DGRAM：UDP模式，发送数据包

SOCK\_RAW：原始套接字，允许对底层协议直接访问，如 ICMP、IGMP。

protocol: 通常设置为 0，表示使用给定 domain 和 type 的默认协议。

返回值

成功时返回一个新的 socket 描述符，失败时返回 -1，并设置 errno 来指示错误类型。

使用本地套接字时，domain的值为AF\_UNIX 或 AF\_LOCAL，type的值可以是SOCK\_STREAM也可以是SOCK\_DGRAM，protocal的值一般为0。具体如下：

int local\_sfd = socket(**AF\_LOCAL**, **SOCK\_STREAM**, **0**);

### 绑定套接字

使用bind()对套接字进行地址绑定

和网络套接字不同，本地套接字绑定之后会创建一个套接字文件

**int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);**

sockfd：套接字描述符

addr：指向包含地址信息的结构体(本地套接字使用sockaddr\_un类型，下面介绍)

addrlen：地址结构体的大小。

返回值：

成功时返回 0；失败时返回 -1 并设置 errno。

#include <sys/un.h>

**struct sockaddr\_un**

{

sa\_family\_t sun\_family; //使用的协议族，上面用AF\_LOCAL这里也用AF\_LOCAL

char sun\_path[]; //套接字路径

};

具体使用方式如下:

struct sockaddr\_un addr;

addr.sun\_family = AF\_LOCAL;

strcpy(addr.sun\_path, "server.socket"); //字符串只能复制

//由于 sun\_path 的长度不固定，所以传入的长度应该只包括实际使用的部分

int len = **offsetof(struct sockaddr\_un, sun\_path) + strlen(servaddr.sun\_path);**

//防止已存在同名套接字文件

**unlink(SERV\_ADDR);**

bind(local\_sfd, (struct sockaddr\*)&addr, len);

### 监听和接受连接

使用listen()和accept()分别监听和接受连接

此过程和网络套接字相同

### 客户进程套接字

客户进程本地套接字通信过程为**socket() -> bind() -> connect()**

socket()和connect()过程除了使用的套接字地址结构为sockaddr\_un以外，其他和与网络套接字相同。最大的不同是与网络套接字相比多了bind()过程

bind()绑定了客户进程套接字的信息。网络套接字没有bind()是因为客户端可以依赖操作系统自动选择一个临时端口来进行通信。

## 4. 内存映射文件(Memory-Mapped File)

内存映射文件允许一个文件的内容被映射到进程的地址空间中，这样不同的进程就可以通过读写这块内存区域来间接地访问同一个文件，从而实现进程间的通信。

**注意：用于映射的文件必须要有实际大小**

#include <sys/mman.h>

void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);

匿名映射：flags带有MAP\_ANONYMOUS字段。此时fd参数无效，只能在血缘进程间传递信息