实验三　进程间通信

一．实验目的

1．掌握利用管道机制实现进程间的通信的方法
2．掌握利用消息缓冲队列机制实现进程间的通信的方法
3．掌握利用共享存储区机制实现进程间的通信的方法
4．加深对上述三种通信机制的理解。

二．实验工具与设备

已安装 Linux 操作系统的计算机。

三．实验预备内容

阅读 Linux 系统的 msg.c、sem.c 和 shm.c 等源代码文件，熟悉 Linux 的三种通信机制。

四．实验内容

1．掌握实现进程间通信的系统调用的功能和方法

进程通信，是指进程之间交换信息。从这个意义上讲，进程之间的同步、互斥也是一种信息交换，也是一种通信。但是，这里所说的“通信”是指进程之间交换较多的信息这样一种情况，特别是在由数据相关和有合作关系的进程之间，这种信息交换是十分必要和数量较大的。

进程间通信是协调解决多个进程之间的约束关系，实现进程共同进展的关键技术，是多道系统中控制进程并发执行必不可少的机制。

（1）进程的通信方式：

a. 直接通信是指信息直接传递给接收方，如管道。在发送时，指定接收方的地址或标识，也可以指定多个接收方或广播式地址, send(Receiver, message)。在接收时，允许接收来自任意发送方的消息，并在读出消息的同时获取发送方的地址, receive(Sender,message)。

b. 间接通信：借助于收发双方进程之外的共享数据结构作为通信中转，如消息队列。这种数据结构称为缓冲区或信箱。通常收方和发方的数目可以是任意的。

（2）进程间通信的类型：

a. 共享存储器系统：**基于共享数据结构的通信方式**：只能传递状态和整数值（控制信息），包括进程互斥和同步所采用的信号量机制。速度快，但传送信息量小，编程复杂，属于低级通信；**基于共享存储区的通信方式**：能够传送任意数量的数据，属于高级通信。

b. 消息传递系统：在消息传递系统中，进程间的数据交换以消息为单位，用户直接利用系统提供的一组通信命令（原语）来实现通信。

c. 管道通信：管道是一条在进程间以字节流方式传送的通信通道。它由 OS 核心的缓冲区（通常几十 KB）来实现，是单向的；在实质上，是一个有 OS 维护的特殊共享文件，常用于命令行所指定的输入输出重定向和管道命令。在使用管道前要建立相应的管道，然后才可使用。

（3）Linux 管道

a. 通过 pipe 系统调用创建无名管道，得到两个文件描述符，分别用于写和读。
   – int pipe(int fd[2]);　//头文件 unistd.h，返回值：0（成功），-1（发生错误）
   – 文件描述符 fd[0]为读端，fd[1]为写端
   – 通过系统调用 write 和 read 进行管道的写和读
   – 进程间双向通信，通常需要两个管道

– 只适用于父子进程之间或父进程安排的各个子进程之间

b. Linux 中的有名管道，可通过 mknod 系统调用建立：指定 mode 为 S_IFIFO，或调用 C 库函数 mkfifo 产生

– int mknod(const char *path, mode_t mode, dev_t dev)

– int mkfifo(const chat *path, mode_t mode)

（4）消息缓冲机制

消息(message) 与窗口系统中的“消息”不同。通常是不定长数据块。消息的发送不需要接收方准备好，随时可发送。相应的数据结构：

```
type message buffer = record
                        sender
                        size
                        text
                        next
                      end
```

Linux 消息：

消息队列(message queue)：每个 message 不定长，由类型(type)和正文(text)组成

Linux 消息队列 API：头文件 sys/types.h, sys/ipc.h, sys/msg.h

–**msgget** 依据用户给出的整数值 key，创建新消息队列或打开现有消息队列，返回一个消息队列 ID；

–**msgsnd** 发送消息；

–**msgrcv** 接收消息,可以指定消息类型；没有消息时，返回-1；

–**msgctl** 对消息队列进行控制，如删除消息队列；

通过指定多种消息类型，可以在一个消息队列中建立多个虚拟信道

注意：消息队列不随创建它的进程的终止而自动撤销，必须用 msgctl(msgqid, IPC_RMID, 0)。另外，msgget 获得消息队列 ID 之后，fork 创建子进程，在子进程中能继承该消息队列 ID 而不必再一次 msgget。

int msgget(key_t key, int flag);

语法格式：msgqid=msgget(key, flag)

功能：获得一个消息的描述符，该描述符指定一个消息对流以便于其他系统调用。

其中，key 由系统规定类型，sys/type.h。flag 本身由操作允许权和控制命令值相 “或”得到，如：IPC_CREAT|0400 表示是否该队列应被创建，IPC_EXCL|0400 表示该队列的创建应是互斥的。msgqid 是该系统调用返回的描述符，失败则返回－1。

int msgsnd(int id, struct msgbuf *msgp, int size, int flag);

功能：发送一消息。其中，id 为返回消息队列的描述符。msgp 指向用户存储区的一个构造体指针。size 指示由 msgp 指向的数据结构中字符数组的长度，即消息长度，该数组的最大值由 MSGMAX 字体可调用参数来确定。flag 规定当核心用尽内部缓冲空间时应执行的动作：若在 flag 中未设置 IPC_NOWAIT，则当该消息队列中的字节数超过一最大值时，或系统范围的消息数超过某一最大值时，调用 msgsnd 进程阻塞；若设置 IPC_NOWAIT，则在此情况下，msgsnd 直接返回。

int msgrcv(int id, struct msgbuf *msgp, int size, int type, int flag);

语法格式：count=msgrcv(id, msgp, size, type, flag)

功能：接收一消息，count 返回消息正文的字节数。其中，struct msgbuf{long mtyoe; char mtext[ ];}; id 消息队列的描述符。msgp 用来存放欲接收消息的用户数据结构的地址。size 指示 msgp 中数据数组的大小。

type 为 0 接收该队列的第一个消息；为正，接收类型为 type 的第一个消息；为负，接收小于或

等于 type 绝对值的最低类型的第一个消息。

flag 规定倘若该队列无消息，核心应当做什么事。若设置 IPC_NOWAIT，则立即返回；若在 flag 中设置 MSG_NOERROR，且所接收的消息大学大于 size，核心截断所接收的消息。

int msgctl(int id, int cmd, struct msgid_ds *buf);

功能：查询一个消息描述符的状态，设置它的状态及删除一个消息描述符。其中，id 用来识别该消息的描述符。

cmd 规定命令的类型：IPC_STAT 将与 id 相关联的消息队列首标读入 buf; IPC_SET 为这个消息队列设置有效的用户和小组表示及操作允许权和字节的数量；IPC_RMID 删除 id 的消息队列。

buf 是含有控制参数或查询结果的用户数据结构的地址。

```
struct msgid_ds{
    struct ipc_perm msgperm;        //许可权结构
    short pad1[7];                  //由系统使用
    ushort onsg_qnum;               //队列上消息
    ushort msg_qbytes;              //队列上最大字节数
    ushort msg_lspid;               //最后发送消息的PID
    ushort msg_lrpid;               //最后接收消息的PID
    time_t msg_stime;               //最后发送消息的时间
    time_t msg_rtime;               //最后接收消息的时间
    time_t msg_ctime;               //最后更改消息的时间
}
```

```
struct ipc_perm{
    ushort uid;     //当前用户 id
    ushort uid;     //当前进程组 id
    ushort cuid;    //创建用户 id
    ushort cgid;    //创建进程组 id
    ushort mode; //存取许可权
    {short pad1; long pad2;}
                    //由系统使用
}
```

（5）共享存储区(shared memory)

相当于内存，可以任意读写和使用任意数据结构（当然，对指针要注意），需要进程互斥和同步的辅助来确保数据一致性。

Linux 的共享存储区：

- 创建或打开共享存储区(shmget)：依据用户给出的整数值 key，创建新区或打开现有区，返回一个共享存储区 ID。
- 连接共享存储区(shmat)：连接共享存储区到本进程的地址空间，可以指定虚拟地址或由系统分配，返回共享存储区首地址。父进程已连接的共享存储区可被 fork 创建的子进程继承。
- 拆除共享存储区连接(shmdt)：拆除共享存储区与本进程地址空间的连接。
- 共享存储区控制(shmctl)：对共享存储区进行控制。如：共享存储区的删除需要显式调用 shmctl(shmid, IPC_RMID, 0)；
- 头文件：sys/types.h, /sys/ipc.h, sys/shm.h

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

语法格式：shmid=shmget(key, size, flag)

功能：创建一个关键字为 key，长度为 size 的共享存储区。其中，size 为存储区的字节数。key、flag 与系统调用 msgget 相同。

int shmat(int id, char *addr, int flag);

语法格式：virtaddr=shmat(id, addr, flag)

功能：从逻辑上将一个共享存储区附接到进程的虚拟地址空间上。

id 为共享存储区的标识符。addr 用户要使用共享存储区附接的虚地址，若为 0，系统选择一个适当的地址来附接该共享区。

flag 规定对此区的读写权限，以及系统是否应对用户规定的地址做舍入操作：如果 flag 中设置了 shm_rnd 即表示操作系统在必要时舍去这个地址；如果设置了 shm_rdonly，表示只允许读操作。

viraddr 是附接的虚地址。

int shmdt(char *addr);

功能：把一个共享存储区从指定进程的虚地址空间断开，当调用成功，返回 0 值；不成功，返回-1。addr 为系统调用 shmat 所返回的地址。

int shmctl(int id, int cmd, struct shmid_ds *buf);

功能：对与共享存储区关联的各种参数进行操作，从而对共享存储区进行控制。调用成功返回 0，否则-1。其中，id 为被共享存储区的描述符。

cmd 规定命令的类型：IPC_STAT 返回包含在指定的 shmid 相关数据结构中的状态信息，并且把它放置在用户存储区中的*buf 指针所指的数据结构中。执行此命令的进程必须有读取允许权；IPC_SET 对于指定的 shmid，为它设置有效用户和小组标识和操作存取权；IPC_RMID 删除指定的 shmid 以及与它相关的共享存储区的数据结构；SHM_LOCK 在内存中锁定指定的共享存储区，必须是超级用户才可以进行此项操作。

buf 是一个用户级数据结构地址。

2. 编写一段程序，实现进程间的管道通信。 其中，父进程通过管道向子进程发送一个字符串（子进程的进程号），子进程将它显示出来。

3. 编写一段程序，使用消息缓冲队列来实现 client 进程和 server 进程之间的通信。

消息的创建、发送和接收。

先使用 fork( )建立两个子进程 server 和 client。

server 进程先建立一个关键字为 MSGKEY(如 75)的消息队列，等待其他进程发来的消息。server 进程每接收到一个消息，便显示字符串“Server has received message from Client!”。当遇到类型为 1 的消息，则作为结束信号，取消该队列，并退出 server。

client 进程则使用关键字为 MSGKEY 的消息队列，先后发送类型从 10 到 1 的消息，然后退出。client 进程每发送一条消息，便显示字符串“Client has sent message to Server!”。最后一条消息，是 server 进程需要结束的信号。

父进程在 server 和 client 均退出后结束。

4. 编写一个与上述功能相同的程序，使其用共享存储区来实现两个进程之间的通信。

共享存储区的创建、附接和断接。

先使用 fork( )建立两个子进程 server 和 client。

server 进程先建立一个关键字为 SHMKEY(如 75)的共享区，并将第一个字节置为-1，作为数据空的标志。等待其他进程发来的消息，当该字节的值发生变换时，表示收到了信息，进行处理，然后再次把它的值置为-1。server 进程每接收到一个消息，便显示字符串“Server has received message from Client!”。当遇到的值为 0，则视为结束信号，取消该存储区，并退出 server。

client 进程则建立一个关键字为 SHMKEY(如 75)的共享区，当共享取得第一个字节为-1 时，server 端空闲，可发送请求。client 随机填入 9 到 0。期间等待 server 端的再次空闲。进行完这些操作后，client 退出。client 进程每发送一条消息，便显示字符串“Client has sent message to Server!”。

父进程在 server 和 client 均退出后结束。

五．思考题

1．上述哪些通信机制提供了发送进程和接收进程之间的同步功能？这些同步是如何进行的？

2．上述通信机制各有什么特点，它们分别适合于何种场合？