訊息佇列函數由 msgget、msgctl、msgsnd、msgrcv 四個函數組成。下面的表格列出了這四個函數的函數原型及其具體說明。

1. msgget 函數原型

msgget(得到訊息佇列識別字或創建一個訊息佇列物件)		
所需標頭檔	#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h>	
函數說明	得到訊息佇列識別字或創建一個訊息佇列物件並返回訊息佇列識別字	
函數原型	int msgget(key_t key, int msgflg)	
函數傳入值	key	0(IPC_PRIVATE)：會建立新的訊息佇列
		大於 0 的 32 位元整數：視參數 msgflg 來確定操作。通常要求此值來源於 ftok 返回的 IPC 鍵值
	msgflg	0：取訊息佇列識別字，若不存在則函數會報錯
		IPC_CREAT：當 msgflg&IPC_CREAT 為真時，如果內核中不存在鍵值與 key 相等的訊息佇列，則新建一個訊息佇列；如果存在這樣的訊息佇列，返回此訊息佇列的識別字
		IPC_CREAT|IPC_EXCL：如果內核中不存在鍵值與 key 相等的訊息佇列，則新建一個訊息佇列；如果存在這樣的訊息佇列則報錯
函數返回值	成功：返回訊息佇列的識別字	
	出錯：-1，錯誤原因存於 error 中	
附加說明	上述 msgflg 參數為模式標誌參數，使用時需要與 IPC 物件存取許可權（如 0600）進行|運算來確定訊息佇列的存取許可權	

錯誤代碼	EACCES：指定的訊息佇列已存在，但調用進程沒有許可權訪問它 EEXIST：key 指定的訊息佇列已存在，而 msgflg 中同時指定 IPC_CREAT 和 IPC_EXCL 標誌 ENOENT：key 指定的訊息佇列不存在同時 msgflg 中沒有指定 IPC_CREAT 標誌 ENOMEM：需要建立訊息佇列，但記憶體不足 ENOSPC：需要建立訊息佇列，但已達到系統的限制

如果用 msgget 創建了一個新的訊息佇列物件時，則 msqid_ds 結構成員變數的值設置如下：

msg_qnum、msg_lspid、msg_lrpid、 msg_stime、msg_rtime 設置為 0。

msg_ctime 設置為當前時間。

msg_qbytes 設成系統的限制值。

msgflg 的讀寫許可權寫入 msg_perm.mode 中。

msg_perm 結構的 uid 和 cuid 成員被設置成當前進程的有效使用者 ID，gid 和 cuid 成員被設置成當前進程的有效組 ID。

2. msgctl 函數原型

msgctl (獲取和設置訊息佇列的屬性)	
所需標頭檔	#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h>
函數說明	獲取和設置訊息佇列的屬性
函數原型	int msgctl(int msqid, int cmd, struct msqid_ds *buf)
	msqid訊息佇列識別字

函數傳入值	cmd	IPC_STAT:獲得 msgid 的訊息佇列頭資料到 buf 中
		IPC_SET：設置訊息佇列的屬性，要設置的屬性需先存儲在 buf 中，可設置的屬性包括：msg_perm.uid、msg_perm.gid、msg_perm.mode 以及 msg_qbytes
		buf：訊息佇列管理結構體，請參見訊息佇列內核結構說明部分
函數返回值	成功：0	
	出錯：-1，錯誤原因存於 error 中	
錯誤代碼	EACCESS：參數 cmd 為 IPC_STAT，確無許可權讀取該訊息佇列 EFAULT：參數 buf 指向無效的記憶體位址 EIDRM：識別字為 msqid 的訊息佇列已被刪除 EINVAL：無效的參數 cmd 或 msqid EPERM：參數 cmd 為 IPC_SET 或 IPC_RMID，卻無足夠的許可權執行	

3. msgsnd 函數原型

msgsnd (將消息寫入到訊息佇列)		
所需標頭檔	#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h>	
函數說明	將 msgp 消息寫入到識別字為 msqid 的訊息佇列	
函數原型	int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg)	
函數傳入值	msqid	訊息佇列識別字
	msgp	發送給佇列的消息。msgp 可以是任何類型的結構體，但第一個欄位必須為 long 類型，即表明此發送消息的類型，msgrcv 根據此接收消息。msgp 定義的參照格式如下： struct s_msg{ /*msgp 定義的參照格式*/ long type; /* 必須大於 0,消息類型 */ char mtext[256]; /*訊息文字，可以是其他任何類型*/ } msgp;

	msgsz	要發送消息的大小，不含消息類型佔用的 4 個位元組,即 mtext 的長度
	msgflg	0：當訊息佇列滿時，msgsnd 將會阻塞，直到消息能寫進訊息佇列
		IPC_NOWAIT：當訊息佇列已滿的時候，msgsnd 函數不等待立即返回
		IPC_NOERROR：若發送的消息大於 size 位元組，則把該消息截斷，截斷部分將被丟棄，且不通知發送進程。
函數返回值	成功：0	
	出錯：-1，錯誤原因存於 error 中	
錯誤代碼	EAGAIN：參數 msgflg 設為 IPC_NOWAIT，而訊息佇列已滿 EIDRM：識別字為 msqid 的訊息佇列已被刪除 EACCESS：無許可權寫入訊息佇列 EFAULT：參數 msgp 指向無效的記憶體位址 EINTR：佇列已滿而處於等待情況下被信號中斷 EINVAL：無效的參數 msqid、msgsz 或參數消息類型 type 小於 0	

msgsnd()為阻塞函數，當訊息佇列容量滿或消息個數滿會阻塞。訊息佇列已被刪除，則返回 EIDRM 錯誤；被信號中斷返回 E_INTR 錯誤。

如果設置 IPC_NOWAIT 訊息佇列滿或個數滿時會返回-1，並且置 EAGAIN 錯誤。

msgsnd()解除阻塞的條件有以下三個條件：

① 不滿足訊息佇列滿或個數滿兩個條件，即訊息佇列中有容納該消息的空間。

② msqid 代表的訊息佇列被刪除。

③　調用 msgsnd 函數的進程被信號中斷。

4. msgrcv 函數原型

msgrcv (從訊息佇列讀取消息)		
所需標頭檔	#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h>	
函數說明	從識別字為 msqid 的訊息佇列讀取消息並存於 msgp 中，讀取後把此消息從訊息佇列中刪除	
函數原型	ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);	
函數傳入值	msqid	訊息佇列識別字
	msgp	存放消息的結構體，結構體類型要與 msgsnd 函數發送的類型相同
	msgsz	要接收消息的大小，不含消息類型佔用的 4 個位元組
	msgtyp	0：接收第一個消息
		>0：接收類型等於 msgtyp 的第一個消息
		<0：接收類型等於或者小於 msgtyp 絕對值的第一個消息
	msgflg	0: 阻塞式接收消息，沒有該類型的消息 msgrcv 函數一直阻塞等待
		IPC_NOWAIT：如果沒有返回條件的消息調用立即返回，此時錯誤碼為 ENOMSG
		IPC_EXCEPT：與 msgtype 配合使用返回佇列中第一個類型不為 msgtype 的消息
		IPC_NOERROR：如果佇列中滿足條件的消息內容大於所請求的 size 位元組，則把該消息截斷，截斷部分將被丟棄
函數返回值	成功：實際讀取到的消息資料長度	
	出錯：-1，錯誤原因存於 error 中	
錯誤代碼	E2BIG：消息資料長度大於 msgsz 而 msgflag 沒有設置 IPC_NOERROR	

	EIDRM：識別字為 msqid 的訊息佇列已被刪除 EACCESS：無許可權讀取該訊息佇列 EFAULT：參數 msgp 指向無效的記憶體位址 ENOMSG：參數 msgflg 設為 IPC_NOWAIT，而訊息佇列中無消息可讀 EINTR：等待讀取佇列內的消息情況下被信號中斷

msgrcv()解除阻塞的條件有以下三個：

① 訊息佇列中有了滿足條件的消息。

② msqid 代表的訊息佇列被刪除。

③ 調用 msgrcv()的進程被信號中斷。

訊息佇列使用程式範例

5. 訊息佇列控制範例

msgctl.c 原始程式碼如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <error.h>
```

```
#define TEXT_SIZE   512
struct msgbuf
{
    long mtype ;
    char mtext[TEXT_SIZE] ;
}   ;
int main(int argc, char **argv)
{
    int msqid ;
    struct msqid_ds info ;
    struct msgbuf buf ;
    struct msgbuf buf1 ;
    int flag ;
    int sendlength, recvlength ;

    msqid = msgget( IPC_PRIVATE, 0666 ) ;
    if ( msqid < 0 )
    {
        perror("get ipc_id error") ;
        return -1 ;
    }
```

```
buf.mtype = 1 ;
strcpy(buf.mtext, "happy new year!") ;
sendlength = sizeof(struct msgbuf) - sizeof(long) ;
flag = msgsnd( msqid, &buf, sendlength , 0 ) ;
if ( flag < 0 )
{
    perror("send message error") ;
    return -1 ;
}
buf.mtype = 3 ;
strcpy(buf.mtext, "good bye!") ;
sendlength = sizeof(struct msgbuf) - sizeof(long) ;
flag = msgsnd( msqid, &buf, sendlength , 0 ) ;
if ( flag < 0 )
{
    perror("send message error") ;
    return -1 ;
}

flag = msgctl( msqid, IPC_STAT, &info ) ;
if ( flag < 0 )
{
```

```
        perror("get message status error") ;

        return -1 ;

    }

    printf("uid:%d, gid = %d, cuid = %d, cgid= %d\n" ,

        info.msg_perm.uid,   info.msg_perm.gid,   info.msg_perm.cuid,
info.msg_perm.cgid   ) ;

    printf("read-write:%03o, cbytes = %lu, qnum = %lu, qbytes=
%lu\n" ,

        info.msg_perm.mode&0777, info.msg_cbytes, info.msg_qnum,
info.msg_qbytes ) ;

    system("ipcs -q") ;

    recvlength = sizeof(struct msgbuf) - sizeof(long) ;

    memset(&buf1, 0x00, sizeof(struct msgbuf)) ;


    flag = msgrcv( msqid, &buf1, recvlength ,3,0 ) ;

    if ( flag < 0 )

    {

        perror("recv message error") ;

        return -1 ;

    }

    printf("type=%d, message=%s\n", buf1.mtype, buf1.mtext) ;


    flag = msgctl( msqid, IPC_RMID,NULL) ;
```

```
    if ( flag < 0 )
    {
        perror("rm message queue error") ;
        return -1 ;
    }
    system("ipcs -q") ;

   return 0 ;
}
```

編譯 gcc msgctl.c –o msgctl。

執行 ./msg，執行結果如下:

```
uid:1008, gid = 1003, cuid = 1008, cgid= 1003
read-write:666, cbytes = 1024, qnum = 2, qbytes= 163840

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000000 65536      zjkf       666        1024         2

type=3, message=good bye!

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

6. 兩進程通過訊息佇列收發消息

（1）發送訊息佇列程式

msgsnd.c 原始程式碼如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include   <time.h>
#define TEXT_SIZE   512
struct msgbuf
{
    long mtype ;
    int   status ;
    char time[20] ;
    char mtext[TEXT_SIZE] ;
}  ;
char   *getxtsj()
{
    time_t   tv ;
    struct   tm    *tmp ;
    static   char   buf[20] ;
```

```
    tv = time( 0 ) ;
    tmp = localtime(&tv) ;
    sprintf(buf,"%02d:%02d:%02d",tmp->tm_hour , tmp->tm_min,tmp->tm_sec);
    return    buf ;
}
int main(int argc, char **argv)
{
    int msqid ;
    struct msqid_ds info ;
    struct msgbuf buf ;
    struct msgbuf buf1 ;
    int flag ;
    int sendlength, recvlength ;
    int key ;

    key = ftok("msg.tmp", 0x01 ) ;
    if ( key < 0 )
    {
        perror("ftok key error") ;
        return -1 ;
    }
```

```
msqid = msgget( key, 0600|IPC_CREAT ) ;
if ( msqid < 0 )
{
    perror("create message queue error") ;
    return -1 ;
}

buf.mtype = 1 ;
buf.status = 9 ;
strcpy(buf.time, getxtsj()) ;
strcpy(buf.mtext, "happy new year!") ;
sendlength = sizeof(struct msgbuf) - sizeof(long) ;
flag = msgsnd( msqid, &buf, sendlength , 0 ) ;
if ( flag < 0 )
{
    perror("send message error") ;
    return -1 ;
}
buf.mtype = 3 ;
buf.status = 9 ;
strcpy(buf.time, getxtsj()) ;
```

```
    strcpy(buf.mtext, "good bye!") ;
    sendlength = sizeof(struct msgbuf) - sizeof(long) ;
    flag = msgsnd( msqid, &buf, sendlength , 0 ) ;
    if ( flag < 0 )
    {
        perror("send message error") ;
        return -1 ;
    }
    system("ipcs -q") ;
   return 0 ;
}
```

（2）接收訊息佇列程式

msgrcv.c 原始程式碼如下：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define TEXT_SIZE   512
struct msgbuf
{
```

```
    long mtype ;
    int   status ;
    char time[20] ;
    char mtext[TEXT_SIZE] ;
}   ;
int main(int argc, char **argv)
{
    int msqid ;
    struct msqid_ds info ;
    struct msgbuf buf1 ;
    int flag ;
    int   recvlength ;
    int key ;
    int mtype ;

    key = ftok("msg.tmp", 0x01 ) ;
    if ( key < 0 )
    {
        perror("ftok key error") ;
        return -1 ;
    }
```

```
    msqid = msgget( key, 0 ) ;
    if ( msqid < 0 )
    {
        perror("get ipc_id error") ;
        return -1 ;
    }

    recvlength = sizeof(struct msgbuf) - sizeof(long) ;
    memset(&buf1, 0x00, sizeof(struct msgbuf)) ;
    mtype = 1 ;
    flag = msgrcv( msqid, &buf1, recvlength ,mtype,0 ) ;
    if ( flag < 0 )
    {
        perror("recv message error\n") ;
        return -1 ;
    }
    printf("type=%d,time=%s, message=%s\n", buf1.mtype,
buf1.time,   buf1.mtext) ;
    system("ipcs -q") ;
   return 0 ;
}
```

（3）編譯與執行程式

① 在目前的目錄下利用>msg.tmp 建立空文件 msg.tmp。

② 編譯發送訊息佇列程式 gcc msgsnd.c -o msgsnd。

③ 執行./msgsnd，執行結果如下:

----- Message Queues --------

key msqid owner perms used-bytes messages

0x0101436d 294912 zjkf 600 1072 2

④ 編譯接收消息程式 gcc msgrcv.c -o msgrcv

⑤ 執行./msgrcv，執行結果如下:

type=1,time=03:23:16, message=happy new year!

------ Message Queues --------

key msqid owner perms used-bytes messages

0x0101436d 294912 zjkf 600 536 1

⑥ 利用 ipcrm -q 294912 刪除該訊息佇列。因為訊息佇列是隨內核持續存在的，在程式中若不利用 msgctl 函數或在命令列用 ipcrm 命令顯式地刪除，該訊息佇列就一直存在於系統中。另外信號量和共用記憶體也是隨內核持續存在的。