## 第10课 讲稿（进程间通信-6）

1. **消息队列**
2. 消息队列概述

顾名思义，消息队列就是一些消息的列表。用户可以从消息队列中添加消息和读取消息等。从这点上看，消息队列具有一定的FIFO 特性，但是它可以实现消息的随机查询，比FIFO 具有更大的优势。同时，这些消息又是存在于内核中的，由“队列ID”来标识，可以在不同的进程间进行通信。

1. 消息队列的应用
2. 函数说明

消息队列的应用包括创建或打开消息队列、添加消息、读取消息和控制消息队列这4 种操作。其中创建或打开消息队列使用的函数是msgget()，这里创建的消息队列的数量会受到系统消息队列数量的限制；添加消息使用的函数是msgsnd()函数，它把消息添加到已打开的消息队列末尾；读取消息使用的函数是msgrcv()，它把消息从消息队列中取走，与FIFO 不同的是，这里可以指定取走某一条消息；最后控制消息队列使用的函数是msgctl()，它可以完成多项功能。

1. 函数格式

表23列举了msgget()函数的语法要点。

|  |  |
| --- | --- |
| 表23 msgget()函数语法要点 | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> |
| 函数原型 | int msgget(key\_t key, int msgflg) |
| 函数传入值 | key：消息队列的键值，可由ftok()生成，多个进程可以通过它访问同一个消息队列。 |
| msgflg：同open()函数的权限位，也可以用八进制表示法。此外，IPC\_CREAT用于消息队列不存在时创建消息队列。IPC\_PRIVATE用于创建当前进程的私有消息队列，IPC\_EXCL用于判断消息队列是否存在，若已经存在，则在创建时出错返回； |
| 返回值 | 成功：消息队列ID |
| 出错：-1（错误原因在errno中） |

表24 列举了msgsnd()函数的语法要点。

|  |  |  |
| --- | --- | --- |
| 表24 msgsnd()函数语法要点 | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> | |
| 函数原型 | int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg) | |
| 函数传入值 | msqid：消息队列的队列ID | |
| msgp：指向消息缓冲区的指针, 此位置用来暂时存储发送和接收的消息，**是一个用户可定义的通用结构**。该消息结构msgbuf通常为：  struct msgbuf  {  long mtype; /\* 消息类型，必须大于0 \*/  char mtext[1]; /\* 消息正文 \*/  } | |
| msgsz：消息正文的字节数（不包括消息类型指针变量） | |
| msgflg： | IPC\_NOWAIT：若消息无法立即发送（比如：当前消息队列已满），函数会立即返回 |
| 0：阻塞直到发送成功为止 |
| 返回值 | 成功：0 | |
| 出错：-1（错误原因在errno中） | |

表25列举了msgrcv()函数的语法要点。

|  |  |  |
| --- | --- | --- |
| 表25 msgrcv()函数语法要点 | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> | |
| 函数原型 | int msgrcv(int msgid, void \*msgp, size\_t msgsz, long int msgtyp, int msgflg) | |
| 函数传入值 | msqid：消息队列的队列ID | |
| msgp：消息缓冲区, 同于msgsnd()函数的msgp | |
| msgsz：消息正文的字节数（不包括消息类型指针变量） | |
| msgtyp： | 0：接收消息队列中第一个消息 |
| 大于0：接收消息队列中第一个类型为msgtyp的消息 |
| 小于0：接收消息队列中第一个类型值不小于msgtyp绝对值且类型值又最小的消息 |
| msgflg： | MSG\_NOERROR：若返回的消息比msgsz字节多，则消息就会截短到msgsz字节，且不通知消息发送进程 |
| IPC\_NOWAIT若在消息队列中并没有相应类型的消息可以接收，则函数立即返回 |
| 0：阻塞直到接收一条相应类型的消息为止 |
| 返回值 | 成功：0 | |
| 出错：-1（错误原因在errno中） | |

表26列举了msgctl()函数的语法要点。

|  |  |  |
| --- | --- | --- |
| 表26 msgctl()函数语法要点 | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/shm.h> | |
| 函数原型 | int msgctl (int msgqid, int cmd, struct msqid\_ds \*buf ) | |
| 函数传入值 | msqid：消息队列的队列ID | |
| cmd：  命令参数 | IPC\_STAT：读取消息队列的数据结构msqid\_ds，并将其存储在buf 指定的地址中 |
| IPC\_SET：设置消息队列的数据结构msqid\_ds中的ipc\_perm域（IPC操作权限描述结构）值。这个值取自buf参数 |
| IPC\_RMID：从系统内核中删除消息队列 |
| buf：描述消息队列的msgqid\_ds结构类型变量 | |
| 返回值 | 成功：0 | |
| 出错：-1（错误原因在errno中） | |

1. 使用实例

这个实例体现了如何使用消息队列进行两个进程（发送端和接收端）之间的通信，包括消息队列的创建、消息发送与读取、消息队列的撤消和删除等多种操作。

**消息发送端进程和消息接收端进程之间不需要额外实现进程之间的同步**。在该实例中，发送端发送的消息类型设置为该进程的进程号（可以取其他值），因此接收端根据消息类型确定消息发送者的进程号。实例中使用函数ftok()获取键值，它可以根据不同的路径和关键字产生标准的key。

以下是消息队列发送端的代码：

|  |
| --- |
| /\* msgsnd.c \*/  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #define BUFFER\_SIZE 512  /\*自定义消息缓冲区\*/  struct message  {  long msg\_type;  char msg\_text[BUFFER\_SIZE];  };  int main()  {  int qid;  key\_t key;  struct message msg;  /\*根据不同的路径和关键字产生标准的key\*/  if ((key = ftok(".", 'a')) == -1)  {  perror("ftok");  exit(1);  }  /\*创建消息队列\*/  if ((qid = msgget(key, IPC\_CREAT|0666)) == -1)  {  perror("msgget");  exit(1);  }  printf("Open queue %d\n",qid);  while(1)  {  printf("Enter some message to the queue:");  if ((fgets(msg.msg\_text, BUFFER\_SIZE, stdin)) == NULL)  {  printf("no message");  exit(1);  }  msg.msg\_type = getpid();  /\*添加消息到消息队列\*/  if ((msgsnd(qid, &msg, strlen(msg.msg\_text), 0)) < 0)  {  perror("message posted");  exit(1);  }  if (strncmp(msg.msg\_text, "quit", 4) == 0)  {  break;  }  }  exit(0);  } |

以下是消息队列接收端的代码：

|  |
| --- |
| /\* msgrcv.c \*/  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #define BUFFER\_SIZE 512  struct message  {  long msg\_type;  char msg\_text[BUFFER\_SIZE];  };  int main()  {  int qid;  key\_t key;  struct message msg;  /\*根据不同的路径和关键字产生标准的key\*/  if ((key = ftok(".", 'a')) == -1)  {  perror("ftok");  exit(1);  }  /\*获取消息队列\*/  if ((qid = msgget(key, 0666)) == -1)  {  perror("msgget");  exit(1);  }  printf("Open queue %d\n", qid);  do  {  /\*读取消息队列\*/  memset(msg.msg\_text, 0, BUFFER\_SIZE);  if (msgrcv(qid, &msg, BUFFER\_SIZE, 0, 0) < 0)  {  perror("msgrcv");  exit(1);  }  printf("The message from process %d : %s", msg.msg\_type, msg.msg\_text);  } while(strncmp(msg.msg\_text, "quit", 4));  /\*从系统内核中移走消息队列 \*/  if ((msgctl(qid, IPC\_RMID, NULL)) < 0)  {  perror("msgctl");  exit(1);  }  exit(0);  } |

【小结】程序思路

1. 消息发送端程序思路：
2. 定义结构体和结构体变量；
3. 创建键值；
4. 创建（打开）消息队列；
5. 循环获取输入信息；
6. 将信息送入到消息队列；
7. 判断是否输入“quit”，若是，则退出。
8. 消息接收端程序思路：
9. 定义结构体和结构体变量；
10. 创建键值；
11. 创建（打开）消息队列；
12. 循环获取消息队列的信息并输出打印；
13. 判断是否输入“quit”，若是，则退出。

以下是程序的运行结果。输入“quit”则两个进程都将结束。



