SYSC 4001 Fall 2014

Tutorial on System V Message Queues

Norman Lim

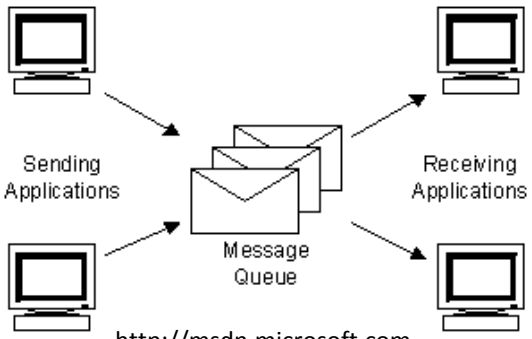
November 18, 2014

Outline

- Introduction
- Creating a Message Queue – `msgget()`
- Message Queue Operations (send and receive)
  - `msgsnd()`
  - `msgrcv()`
- Message Queue Control – `msgctl()`
- IPC Status Commands
- References

Introduction

- Allows two (or more) processes to *exchange information* via access to a common system message queue.
  - One process establishes the message queue that others may access.
- Processes must share a *common key* in order to gain access to the message queue.
- The ***sender*** places a message into the queue using an (OS) message-passing module (i.e. system call).
- The ***receiver*** retrieves the message from the queue also using a system call.



http://msdn.microsoft.com

Creating a Message Queue (1)

- The **`msgget()`** system call creates a new message queue:

```
int msgget(key_t key, int msgflg)
```

- It can also return the message queue ID (`msqid`) of the queue corresponding to the `key` argument.
- **Parameters:**
  - **`key`:** the key for the message queue. It can be specified directly by the user or generated using `ftok()`
    - `key_t key = (key_t) 1234`
    - `key_t key= = ftok("/home/nlim/somefile", 1);`
  - **`msgflg`:** an octal integer with settings for the queue's permissions and control flags.
    - `IPC_CREAT | 0600`

Creating a Message Queue (2)

- **Returns:**
  - Success: *message queue ID*
  - Failure: -1
- **Example:**

```
int msqid = msgget((key_t)1234, IPC_CREAT| 0600);
if (msqid == -1){
   perror("msgget: msgget failed");
   exit(1);
} else {
   //success
}
```

- Need to include the following libraries:
  - <sys/types.h>
  - <sys/ipc.h>
  - <sys/msg.h>

Sending a Message (1)

- The **`msgsnd()`** system call places a message in the message queue:

```
int  msgsnd (int msqid, const void *msgp,
                    size_t msgsz, int msgflg)
```

- **Parameters:**
  - **`msqid:`** the id of the message queue to send to.
  - **`msgp:`** the address of (pointer to) the message to be sent.
  - **`msgsz:`** the size of the message to be sent.
  - **`msgflg:`**
    - 0: *Block* caller if the message queue is full.
    - `IPC_NOWAIT`: Caller *does not wait* if message queue is full.
- **Returns:**
  - Success: 0
  - Failure: -1

Sending a Message (2)

- The message to be sent should be a `struct` that looks like this:

```
struct my_message {
    /*REQUIRED: Message type, which is a positive number*/
    long message_type;

    /*The data to transfer */
    int data;
};
```

- **Example:**

```
struct my_message msg;
msg.message_type = 1; //Note: can send a message to a specific process by specifying its pid
msg.data = 100;

//Create message queue with id = msqid

int msgLength = sizeof(struct my_message) - sizeof(long);
/* The message length only includes the data to transfer (i.e. the size of the message structure minus message type)*/
```

Sending a Message (3)

```
if(msgsnd(msqid, &msg, msgLength, 0) == -1){
   perror("msgsnd: msgsnd failed");
   exit(1);
} else {
   //successfully placed a message with type = 1 into the
   message queue
}
```

Receiving a Message (1)

- The **`msgrcv()`** system call is used to retrieve a message from the message queue:

```
int msgrcv (int msqid, void *msgp, size_t msgsz,
            long msgtyp, int msgflg);
```

- **Parameters:**
  - **`msqid:`** the id of the message queue to receive from.
  - **`msgp:`** a pointer (address) to the variable (`struct`) used to store the retrieved message.
  - **`msgsz:`** the size of the message buffer structure in bytes (excludes message_type field)
  - **`msgtype:`** the type of the message to be retrieved.
    - **> 0** : return first message with type equal to `msgtyp`
    - **0 :** return first message in the queue regardless of *type*
    - **< 0 :** return the first message with lowest type less than or equal to `msgtyp`

Receiving a Message (2)

- **`msgflg:`** indicates what action should be taken.
  - `0:` the calling process blocks until a message arrives in the queue that satisfies the `msgrcv()` parameters
  - `IPC_NOWAIT:` if no messages are available, the caller does not wait for a message (returns -1 and sets errno to ENOMSG)
  - `MSG_NOERROR:` if size of message exceeds msgsz, accept msgsz bytes
- **Returns:**
  - Success: number of bytes received.
  - Failure: -1

Receiving a Message (3)

- **Example:**

```
struct my_message msg;

//Create message queue with id= msqid

int msgLength = sizeof(struct my_message) -
sizeof(long);

if(msgrcv(msqid, &msg, msgLength, 1, 0) == -1){
   perror("msgrcv: msgrcv failed");
   exit(1);
} else {
   //successfully received a message with type = 1
}
```

Message Queue Control (1)

- The **`msgctl()`** system call is used to examine or modify ownership and access permissions of the message queue.

```
int msgctl (int msqid, int cmd, struct msqid_ds *buf);
```

- **Parameters:**
  - **`msqid:`** the id of the message queue to perform operation on
  - **`cmd:`** the action to be performed
    - **`IPC_RMID:`** removes the message queue.
    - **`IPC_STAT:`** return the current value for each member of the `msqid_ds` data structure (contains the permission structure).
    - **`IPC_SET:`** modify a member of the `msqid_ds` structure
  - **`buf:`** pointer to a `msqid_ds struct`
    - `msqid_ds` is an internal struct used by the OS (see References and *man* pages for more detail)

Message Queue Control (2)

- **Returns:**
  - Success: 0
  - Failure: -1

- **Example:** remove a message queue with id = msqid

```
if (msgctl(msqid, IPC_RMID, 0) == -1) {
  perror("msgctl: msgctl failed");
  exit(1);
}
```

IPC Status Commands

- Useful commands that can be invoked from the terminal to display allocated *IPC resources*: semaphores, shared memory, and message queues.
- `ipcs`: displays all IPC resources active in the system
  - Display only message queues: `ipcs -q`
- `ipcrm`: remove a specific IPC resource
  - E.g. remove a semaphore with id 239: `ipcrm -s 239`
- Summary of flags:
  - **-s:** semaphores
  - **-m:** shared memory
  - **-q:** message queues

References

- *Beginning Linux Programming* 4th Edition by Neil Matthew and Richard Stones
  - Available in e-book format at http://www.library.carleton.ca/.
- http://www.tldp.org/LDP/lpg/node27.html
- http://www.cs.cf.ac.uk/Dave/C/node25.html
- http://faculty.kutztown.edu/spiegel/CSc552/PowerPoint/