ÍNDICE

1. PROGRAMACIÓN DISTRIBUIDA UTILIZANDO COLAS DE MENSAJES

1.1. El concepto de cola de mensajes

Las colas de mensajes componen, junto con la memoria compartida y los semáforos, el tercer mecanismo IPC. Una cola de mensajes es una lista ordenada de mensajes mantenida en el kernel del S.O. Tienen alguna similitud con los segmentos de memoria compartida como por ejemplo el estar identificada por una clave numérica, tener propietario y modo de acceso y además existen independientemente de cualquier proceso de usuario. Cualquier proceso que conozca la clave de la cola de mensajes y tenga los permisos necesarios podrá enviar mensajes a la cola y recuperar mensajes de la misma.

Cada mensaje enviado a una cola tiene un tipo asociado y transporta datos del usuario. El tipo es simplemente un entero positivo, y se utiliza para tener algún tipo de control en el orden en que los mensajes son recuperados de la cola.

En términos de funcionalidad, las colas de mensajes están a medio camino entre los pipes y la memoria compartida.

1.2. Principales características de las colas de mensajes

Entre las principales características de las colas de mensajes podemos destacar las siguientes:

- Ofrecen un acceso mucho más flexible que el esquema FIFO de los pipes, pero no ofrecen completamente el acceso aleatorio proporcionado por la memoria compartida. Los mensajes pueden ser recuperados por su tipo de mensaje, no sólo en orden FIFO.
- No es necesario que haya un proceso que esté leyendo mensajes de una cola antes de que otro pueda escribir un mensaje, lo cual es un contraste con los pipes, donde es necesario que exista un proceso lector antes de que exista un escritor. Un proceso puede escribir algunos mensajes en una cola, terminar y dejar los mensajes para que los lea otro proceso más tarde.
- Al no utilizar descriptores de ficheros, no se puede utilizar entrada/salida multiplexada a través de **select** y **poll**. Si un proceso quiere estar atento a la recepción de datos en dos o más colas de mensajes deberá o bien utilizar algún mecanismo de lectura no bloqueante (espera activa) ☹ o bien utilizar hilos ☺.
- Son un mecanismo IPC orientado a conexión1.
- Son fiables al estar restringidas a la misma máquina.
- Implementan control de flujo2.

1 Por orientado a conexión entendemos el tener que hacer algún tipo de llamada de inicialización antes de poder enviar algún mensaje.
2 Por control de flujo entendemos que el proceso es puesto a dormir si intenta escribir en la cola y hay una bajada de recursos del sistema (buffers). Una vez que la condición desaparece, el proceso es despertado.

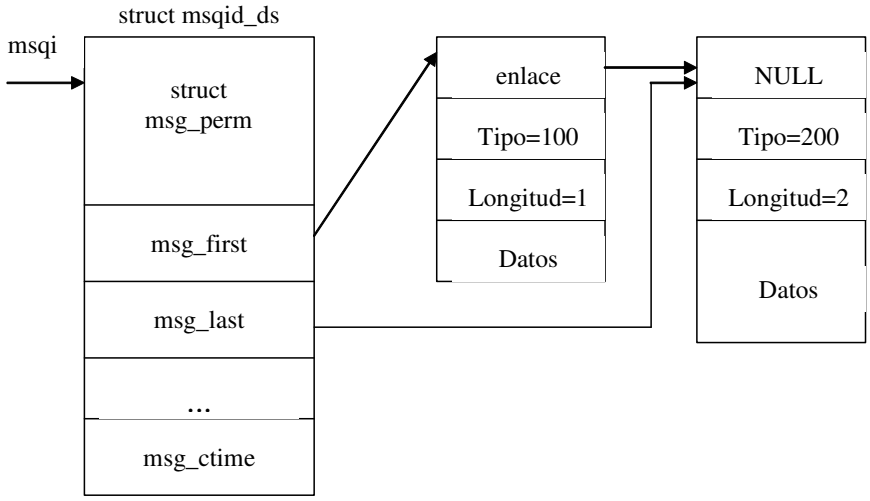
1.3. Representación interna de las colas de mensajes

Cada cola de mensajes tiene asociada con ella una estructura **msqid_ds**. Dicha estructura define el estado actual de la cola de mensajes.

```
struct msqid_ds{
  struct  ipc_perm  msg_perm;   /* información sobre permisos y modo de acceso */
  struct  msg msg_first;        /* puntero al primer mensaje de la cola */
  struct  msg msg_last;         /* puntero al último mensaje de la cola */
  ulong  msg_cbytes;            /* número actual de bytes de la cola */
  ulong  msg_qnum;              /* número de mensajes en la cola */
  ulong  msg_qbytes;            /* máximo número de bytes de la cola */

  pid_t  msg_lspid;             /* pid del último msgsnd() */
  pid_t  msg_lrpid;             /* pid del último msgrcv() */
  time_t msg_stime;             /* tiempo del último msgsnd() */
  time_t msg_rtime;             /* tiempo del último msgrcv() */
  time_t msg_ctime;             /* tiempo del último cambio */
  struct msg_wait msg_wait_list;       /* lista de mensajes que están esperando */
}
```

Los punteros **msg_first** y **msg_last** no tienen ningún valor para el usuario ya que apuntan al área de almacenamiento de mensajes en el kernel, y esta zona de la memoria está protegida del acceso de los procesos de usuario. El aspecto de una cola de mensajes dentro del kernel aparece en la siguiente figura:



En la siguiente tabla se muestran los límites del sistema que afectan a las colas de mensajes. Dichos valores dependen de la implementación del S.O. y los que mostramos aquí son los correspondientes a la máquina centauro (OSF/Digital Unix v4.0E).

Nombre	Descripción	Valor
MSGMAX	El tamaño en bytes del mensaje más largo que podemos enviar.	8192
MSGMNB	El tamaño máximo en bytes de una cola particular (la suma de los tamaños de todos los mensajes en la cola).	16384
MSGMNI	El máximo número de colas de mensajes en el sistema.	50
MSGTQL	El máximo número de mensajes en todo el sistema.	40

1.4. Operaciones sobre colas de mensajes

1.4.1. Creación de una cola de mensajes: msgget()

Esta es la primera función que se invoca normalmente para crear una cola de mensajes o conectarnos a una ya existente. La sintaxis de esta función es la siguiente:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key_t clave, int flag);
```

Esta función nos va a devolver el handler de la nueva cola de mensajes o bien –1 en caso de error. Si no devuelve –1 el handler devuelto será el valor utilizado para referenciar a la cola por las otras tres funciones que veremos a continuación.

El parámetro clave especifica la clave que identifica la cola de mensajes. Este valor debe ser un valor distinto de 0 o la constante simbólica **IPC_PRIVATE** que garantiza la creación de un canal IPC privado, es decir, creado ex profeso para la aplicación.

En el campo flag se pueden suministrar parámetros adicionales para la creación de la cola. Uno de estos parámetros es el flag **IPC_CREAT** que indica que queremos crear una cola nueva para nuestra aplicación. Si la función **msgget()** tiene éxito nos devolverá el handler de la nueva cola de mensajes creada. Si ya existiese una cola de mensajes para el valor de clave suministrado y no suministramos el flag **IPC_EXCL**, entonces la función msgget nos devolvería el handler asociado a la cola identificada con esa clave.

En el caso de haber especificado el flag **IPC_CREAT** y estar otra aplicación utilizando esa clave, la llamada **msgget()** fallaría y nos devolvería un error a nuestra aplicación.

El campo flag se utiliza para especificar los modos de acceso a la cola (en notación octal) para las tres categorías de permisos existentes en el entorno Unix (ugo, usuario-grupo-otros). Para combinar los permisos de creación de la cola con los flags vistos anteriormente se utiliza la operación OR a nivel de bits.

Para más información sobre esta función consultar las páginas del man.

Ej.: Creación de una cola con clave 4144 y permisos de lectura y escritura para el propietario, lectura para el grupo del propietario y sin permisos para el resto.

```
#define CLAVE 4144
#define PERMISOS 0640

...
msgqid = msgget(CLAVE,PERMISOS|IPC_CREAT);
...
```

Ej.: Conexión a una cola con clave 4144 desde otro proceso distinto al que la crea.

```
#define CLAVE 4144
...
msgqid = msgget(CLAVE,0);
...
```

1.4.2. Envío de mensajes a una cola de mensajes: msgsnd()

Esta función nos va a permitir enviar mensajes a una cola de mensajes. La sintaxis de esta función es la siguiente:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd (int msgqid,const void *msgp, size_t msgsz,int msqflg);
```

En este caso **msgqid** representa el handler de la cola de mensajes a donde vamos a enviar el mensaje. Este parámetro se obtiene al hacer una invocación previa a **msgget()**.

El segundo parámetro es un puntero a la estructura que contiene el mensaje. Esta estructura debe de tener como primer campo uno de tipo long que almacena el tipo del mensaje (entero positivo). El segundo campo de esta estructura debe de ser un array de caracteres cuya longitud debe definir el usuario y queda limitado por los valores límites de la implementación. El aspecto de esta estructura sería la siguiente:

```
struct msgbuf{
  long int mtype;
  char mtext[];
}
```

El tercer parámetro pasado a la función **msgsnd()** es el tamaño del campo array de datos de la estructura **msgbuf**. Este valor puede ser 0 si no hay campo de datos en la estructura.

El cuarto y último parámetro puede tomar el valor **IPC_NOWAIT** o bien 0. Si el flag **IPC_NOWAIT** está presente y se produce una de las siguientes condiciones:

- El número de mensajes en la cola alcanzó el límite para el sistema.
- El número total de bytes en la cola es igual al límite del sistema.

Entonces, la llamada a la función retorna inmediatamente devolviendo un error y el kernel no envía el mensaje. Si este flag no está presente, y ocurre una de las condiciones anteriores:

a) el kernel suspende al proceso que la invoca hasta que desaparece la condición de bloqueo, momento en el que el kernel envía el mensaje y el proceso continúa,
b) o bien la cola es eliminada del sistema con lo cual la función retornará devolviendo un error,
c) o bien el proceso atrapa una señal, en cuyo caso el proceso continúa su ejecución en la rutina que atrapa la señal.

Para más información sobre el funcionamiento de esta función consultar las páginas del man.

1.4.3. Lectura de mensajes de una cola de mensajes: msgrcv()

Esta función nos va a permitir recuperar los mensajes existentes en una cola de mensajes. Su sintaxis es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv (int msgqid,void *msgp, size_t msgsz,long msgtype, int msqflg);
```

En este caso **msgqid** representa el handler de la cola de mensajes de donde vamos a recibir el mensaje. Este parámetro se obtiene al hacer una invocación previa a **msgget()**.

El segundo parámetro es un puntero a una variable de tipo estructura **msgbuf** donde se va a almacenar el mensaje leído de la cola.

El tercer parámetro especifica el tamaño del campo de la estructura **msgbuf** donde se van a recibir los datos y no incluye el tamaño del campo donde se almacena el tipo del mensaje.

El cuarto parámetro nos permite especificar cómo queremos leer los mensajes existentes en la cola o más bien el orden en que los leeremos. Si este parámetro es 0, la función nos devuelve el primer mensaje de la cola en orden FIFO. Si el parámetro **msgtype** es mayor que 0, entonces la función nos devolverá el primer mensaje de la cola cuyo tipo sea igual al especificado en **msgtype**. Por último, si **msgtype** fuese menor que 0, la función nos devolvería el primer mensaje de la cola cuyo tipo de mensaje sea el menor valor menor o igual que el valor absoluto del parámetro **msgtype**.

El parámetro **msqflag** puede tomar el valor **IPC_NOWAIT**, en cuyo caso la operación de lectura de mensajes es no bloqueante, es decir, si no hubiese mensajes del tipo especificado en la cola, **msgrcv()** retornaría un error. Si no se pone el flag **IPC_NOWAIT**, la función bloquea al proceso que la invoca hasta:

a) que haya un mensaje en la cola del tipo especificado,
b) la cola sea eliminada por otro proceso en cuyo caso se retornaría un error o
c) se atrapa una señal en cuyo caso también se retornaría un error.

Para más información consultar las páginas del man.

1.4.4. Operaciones de control de la cola de mensajes: msgctl()

Esta función nos va a permitir realizar operaciones de control de la cola de mensajes. Su sintaxis es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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

Aunque se pueden realizar varias operaciones de control sobre una cola, el uso principal de este comando es el de permitirnos eliminar una cola de mensajes. Para ello **cmd** debe de ser **IPC_RMID**. Al realizar esta operación se elimina la cola de mensajes y se libera el espacio de memoria que el kernel utiliza para ella. Esta operación esta restringida, es decir, el usuario efectivo del proceso que la invoca debe de ser el root o bien ser el propietario o el creador de la cola. Para más información consultar las páginas del man.

1.5. Usos de las colas de mensajes

Las colas de mensajes pueden utilizarse para multiplexar flujos de datos de múltiples productores a un solo consumidor. El productor del mensaje podría identificarse dentro del tipo del mensaje, por ejemplo, fijando como tipo del mensaje el PID del productor. También es posible demultiplexar datos de un simple productor sobre múltiples consumidores. Los consumidores podrían ser generalistas, es decir, cada uno tomaría el siguiente mensaje de la cola sin importar su tipo o bien podrían ser especialistas, es decir, tomando los mensajes de la cola por su tipo.

Los tipos de los mensajes pueden utilizarse para implementar un esquema de prioridades. Utilizando tipos de mensajes entre, por ejemplo, 1 y 100 nos proporcionaría un esquema simple de ordenación por prioridad donde los mensajes de tipo 1 serían los de más alta prioridad y los de 100 los de más baja prioridad. Por ejemplo, si invocásemos **msgrcv()** solicitando un tipo de –100 nos devolvería el mensaje de prioridad más alta de la cola. Se podría utilizar un esquema ligeramente más complejo para separar los mensajes de prioridad "normal" de los mensajes "urgentes", con tipos en el rango, por ejemplo, de 1 a 10 como urgentes y el resto considerados como de prioridad normal. Un consumidor podría comprobar si existen mensajes urgentes en la cola invocando **msgrcv()** con un tipo de –10 y el valor **IPC_NOWAIT** en el campo **flag**. Si no hubiese mensajes urgentes, esta llamada retornaría –1, con lo cual el consumidor podría seguir adelante y procesar el primer mensaje no urgente de la cola.

1.6. Gestión de las colas de mensajes desde la línea de comando

En Unix el comando **ipcs** puede utilizarse para obtener estadísticas de funcionamiento de varios mecanismos IPC entre los que se encuentran las colas de mensajes. Este comando tiene varias opciones que permiten obtener información tan diversa como quién es el propietario de la cola, los permisos, el número de bytes en la cola, etc. Así mismo, las máquinas Unix disponen de otro comando (**ipcrm**) que puede utilizarse fácilmente para eliminar mecanismos IPC como las colas de mensajes.

Para más información, consultar las páginas del man.

ANEXO

```
/* Fichero ej1.h*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define CLAVE_COLA      777
#define PERMISOS_COLA   0600
#define TAM_BUFFER      1024

typedef struct {
           long tipo;
           char datos[TAM_BUFFER];
        } mensaje;
```

```
/* Fichero cliente.c */
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "ej1.h"

void main(int argc, char *argv[])
{
  FILE *fp;
  int qid;
  mensaje m;
  char buffer[TAM_BUFFER];
  int leidos;

  if (argc!=2)
  {
    printf("Forma de uso: cliente nom_fich\n");
    exit(1);
  }
  if ((fp=fopen(argv[1],"r"))==NULL)
  {
    printf("No se pudo abrir el fichero: %s\n",argv[1]);
    exit(2);
  }
  if ((qid=msgget(CLAVE_COLA,0))<0){
    printf("Error: msgget no pudo abrir la cola de clave %d\n",CLAVE_COLA);
    exit(3);
  }
  m.tipo=1;
  while (! feof(fp))
  {
     leidos=fread(buffer,sizeof(char),TAM_BUFFER,fp);
     memcpy(m.datos,buffer,leidos);
     if (msgsnd(qid,(char *) &m, leidos,0)!=0)
     {
        printf("Ocurrio el error %d en msgsnd\n",errno);
        fclose(fp);
        exit(4);
     }
  }
  m.tipo=2;
  if (msgsnd(qid,(char *) &m, 0,0)!=0)
  {
        printf("Ocurrio el error %d en msgsnd\n",errno);
        fclose(fp);
        exit(4);
  }
  fclose(fp);
  exit(0);
}
```

```
/*
Fichero servidor.c
```

```
*/
#include <stdio.h>
#include "ej1.h"

void main(int argc,char *argv[])
{
  FILE *fp;
  int qid;
  mensaje m;
  int leidos;

  if (argc!=2)
  {
    printf("Forma de uso: servidor nom_fich\n");
    exit(1);
  }
  if ((fp=fopen(argv[1],"w"))==NULL)
  {
    printf("No se pudo abrir el fichero: %s\n",argv[1]);
    exit(2);
  }
  if ((qid=msgget(CLAVE_COLA,PERMISOS_COLA|IPC_CREAT))<0)
  {
    printf("El servidor no pudo crear la cola de mensajes con clave %d\n",CLAVE_COLA);
    exit(3);
  }
  while (1)
  {
    leidos=msgrcv(qid,&m,TAM_BUFFER,0,0);
    switch (m.tipo){
      case 1:
        fwrite(m.datos,sizeof(char),leidos,fp);
        break;
      case 2:
        fclose(fp);
        if (msgctl(qid,IPC_RMID,(struct msqid_ds *) NULL)<0)
        {
          printf("Error al borrar la cola de mensajes de clave %d\n",CLAVE_COLA);
          exit(4);
        }
        exit(0);
    }
  }
}
```

```
# Escuela Técnica Superior de Ingeniería Informática (Gijón)
# Sistemas de Computación (Curso 5)
# Aplicaciones Distribuidas en Unix con Sun RPC
#************************************************
# Variables
#************************************************
CC=gcc
CFLAGS= -Wall
CLIENTE = cliente
SERVIDOR= servidor

todo : $(CLIENTE) $(SERVIDOR)

#************************************************
# Generación del cliente y del servidor
#************************************************
$(CLIENTE) : cliente.c ej1.h
        $(CC) $(CFLAGS) -o $(CLIENTE) cliente.c
$(SERVIDOR) : servidor.c ej1.h
        $(CC) $(CFLAGS) -o $(SERVIDOR) servidor.c

#************************************************
# Utilidades
#************************************************
limpiaobjs :
        rm -f *.o
```

BIBLIOGRAFÍA

Unix. Distributed Programming

Autor: Chris Brown
Editorial: Prentice-Hall
ISBN: 0-13-075896-5

Advanced Programming in the Unix Environment

Autor: W. Richard Stevens
Editorial: Addison-Wesley
ISBN: 0-201-56317-1

Unix Network Programming

Autor: W. Richard Stevens
Editorial: Prentice-Hall
ISBN: 0-13-949876-1

Unix. Manual Pages.