TP VII – IPC – Cola de Mensajes – Queue's

Objetivo: Desarrollar una serie de programas que permitan comprobar distintas funcionalidades vinculadas con la gestión de cola de mensajes como una forma de comunicación entre procesos (IPC: Inter Process Communication) provistas por el kernel del SO. Introducir al desarrollo de programas servidores y clientes. Estimular una programación modular, con código reusable.

Introducción

Existen 3 formas de IPC's:

1. Cola de Mensajes (Message Queues)
2. Semáforos (Semaphores)
3. Memoria Compartida (Shared Memory)

Dentro del kernel se identifican a través de un número entero positivo, este número es de tipo `key_t` y esta definido dentro del archivo de cabecera `sys/types.h` . De alguna forma, si dos o más programas pretenden compartir cualquiera de estos IPC's deberán conocer este número para poder accederlo de forma unívoca. Para compartir este número se podría hacer lo siguiente:

1. Un programa servidor podría crear un IPC de tipo IPC_PRIVATE, obtener su identificador y luego guardarlo en forma persistente en el sistema de archivos para que luego el o los programas cliente puedan leerlo.
2. El programa servidor y cliente acuerdan un número fijo de clave, indicado por ejemplo en un archivo de cabecera común a ambos programas e incorporado en la compilación de ambos.
3. El programa servidor y cliente acuerdan un path (una ruta de acceso, la ubicación de una carpeta dentro del sistema de archivos) y un id de proyecto (un simple carácter). Este path y id de proyecto pueden estar en un archivo de cabecera común a ambos. Con estos dos datos se puede invocar a la función `ftok()`, la cual, a partir de estos dos datos genera una clave única de IPC para que ambos programas puedan acceder al mismo recurso.

Cada una de estas alternativas tiene sus ventajas y desventajas, en nuestro caso, se optó por la opción 3.

¿Qué es una cola de mensajes? Es una lista enlazada de mensajes almacenada dentro del kernel del SO e identificada a través de un id (de tipo `key_t`).

¿Qué estructura tiene cada mensaje? Todos los mensajes tienen idéntica estructura y la misma esta compuesta por un número de tipo `long`, que representa el tipo de mensaje y luego continúa con una serie de caracteres acorde con la necesidad de cada aplicación. Esto permite gran flexibilidad de implementación.

Esta simple estructura de cola de mensajes nos permite pensar en una implementación client-server. Se podría implementar un servidor que espera recibir mensajes de tipo 1 de la cola de mensajes, estos mensajes son peticiones de los clientes, una vez recibido el mensaje, lo procesa y … ¿cómo devolver una respuesta a los clientes? Se nos ocurrió la siguiente alternativa: dentro de la petición del cliente, incluir su id de proceso, entonces, el servidor, como respuesta a la petición cliente puede introducir un mensaje en la cola de mensajes de tipo X en donde X es el id de proceso cliente a quien el servidor le esta respondiendo.

El Analista Y pretende diseñar un servidor que informe a los clientes acerca de los totales de ventas de las sucursales de la Empresa X bajo este enfoque, los mensajes que enviarán los clientes tendrán el formato: `<id de proceso cliente/número de sucursal>` y

serán de tipo 1, mientras que el formato del mensaje a devolver por parte del servidor será: <número de sucursal/total de ventas> y será un mensaje de tipo <id de proceso cliente>.
El formato propuesto en los mensajes a intercambiar entre cliente y servidor, permiten los siguientes diseños:

Programa servidor

```
main()
      inicializar totales de ventas de sucursales
      crear clave IPC según path y id proyecto
      crear cola de mensajes usando clave
      si pude crear Cola Ok entonces
            mientras no salir
                  recibir mensaje de tipo 1 de cola
                  parsear mensaje recibido (id proceso cliente/sucursal)
                  si número de sucursal es correcto entonces
                        obtener total de ventas de sucursal
                        enviar mensaje a cola de tipo <id proceso cliente>
                  fin si
            fin mientras
            borrar cola
      sino
            error creando cola
      fin si
fin main()
```

Programa cliente

```
main()
      crear clave IPC según path y id proyecto
      obtener cola de mensajes usando clave
      si pude obtener Cola Ok entonces
            obtener id de proceso
            para cada sucursal hacer
                  armar mensaje <id proceso/sucursal>
                  enviar mensaje de tipo 1 a cola
                  si pude enviar mensaje Ok entonces
                        recibir mensaje de tipo <id proceso> de cola
                        si pude recibir mensaje Ok entonces
                              parsear mensaje recibido(id sucursal/total ventas)
                              mostrar mensaje recibido
                        fin si
                  fin si
            fin para
      sino
            error obteniendo cola
      fin si
fin main()
```

1. El Analista de Sistemas Y esta muy entusiasmado con IPC y está convencido de que este no será el último desarrollo en esta línea, por lo tanto, se propone implementar una pequeña librería para el manejo de cola de mensajes, semáforos y memoria compartida: `myipc.c` cuyo archivo de cabecera será `myipc.h`. Este archivo de cabecera y librería será compartido tanto por cliente como por servidor, por lo tanto, aquí podemos acordar el path y id de proyecto según lo dicho en la introducción, así que dentro de `myipc.h` deberemos incluir:

```
#define IPC_PATH "."
#define IPC_KEY 'm'
```

También se deberán incluir dentro de `myipc.h` una serie de archivos de cabecera a ser utilizados por los programas servidor y cliente, para que ambos accedan a las funciones de ipc y otras librerías:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <limits.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
```

La API de `myipc.c` definida dentro de `myipc.h` y compartida por clientes y servidores será:

```
int borrar_cola(int qid);
int crear_cola(key_t key);
int enviar_msg(int qid, void *qbuf, int msglen, long type);
void init(double *ventas);
key_t obtener_clave(char *path,char car);
int obtener_cola(key_t key);
int recibir_msg(int qid, void *qbuf,int msglen, long type);
```

Tanto cliente como servidor ejecutarán:

```
key_t clave = obtener_clave(IPC_PATH,IPC_KEY);
```

para generar una clave IPC con la cual identificar a la cola de mensajes.
El servidor creará la cola de mensajes: `int idcola = crear_cola(clave);` mientras que el cliente se conectará a una cola previamente creada: `int idcola = obtener_cola(clave);`
Estas funciones devolverán `-1` cuando exista alguna condición de error.
El servidor inicializará el total de ventas de las sucursales (generará totales de ventas):

```
double ventas[10];
…
init(&ventas[0]);
```

Tanto para el envío como para la recepción de mensajes: `qid` se refiere al id de la cola de mensajes (`idcola`), `qbuf` se refiere al mensaje a enviar/recibir, `msglen` se refiere al largo del mensaje[1] y

[1] Cabe aclarar que este número no incluye lo que ocupa el tipo de mensaje. Únicamente se contabiliza el largo de los

`type` al tipo de mensaje a enviar (`1` o bien `<id de proceso clientes>`).

`Myipc.c` permitirá un uso más abstracto de IPC y permitirá aislar el código del Analista Y de los detalles de implementación. ¿Para qué podría servirle esto? Hay varias razones para hacerlo, pero mencionemos al menos una: portabilidad. El Analista Y desarrollará sus servidores y clientes en términos de la API definida por `myipc.h` , no es el caso de la cola de mensajes, pero nosotros utilizaremos lo que se denomina una implementación System V R4 (SVR4) de semáforos, pues resulta que también existen los semáforos POSIX, para ambos tipos de semáforos se utilizan API's muy distintas para comunicarse con el kernel. El Analista Y podría tener distintas versiones de `myipc.c` (la implementación de la API `myipc.h`), pero todas con la misma API (`myipc.h`), según la plataforma donde corra su código o acorde con las características de la aplicación a desarrollar podrá usar una u otra, sin que ni siquiera lo noten los programadores que trabajarán con El, puesto que ellos hacen todas las operaciones en términos de `myipc.h`.

Comencemos con la implementación de esta librería (`myipc.c`), ¿Qúe implica obtener una clave? simplemente llamar a la función `ftok()` para que ésta haga su trabajo:

```
key_t obtener_clave(char *path,char car) {
      return ftok(path, car);
}
```

Para crear/obtener acceso a una cola de mensajes podemos usar la función `msgget()` indicándole la clave y el modo (formado por el tipo de acción a realizar (`IPC_CREAT` para crear una cola, sino se trata solo de obtener el id de una cola previamente creada) y el permiso que tendrá este proceso sobre dicho recurso). Aquí queda de manifiesto que hay una cuestión de permisos vinculada con IPC, no cualquier proceso puede crear recursos a nivel del kernel[2]:

```
/* Crea cola de mensajes */
int crear_cola(key_t key) {
   return msgget(key, IPC_CREAT|0660);
}

/* Obtiene cola de mensajes previamente creada */
int obtener_cola(key_t key) {
   return msgget(key, 0660);
}
```

Una vez que hemos creado una cola/obtenido su id, podemos cambiar algunas características de la misma utilizando la función `msgctl()`, ésta también nos servirá para destruir la cola:

```
/* Destruye cola de mensajes */
int borrar_cola(int qid)  {
   return msgctl(qid, IPC_RMID, 0);
}
```

Sabemos que todo mensaje comienza con un `long` y continúa con una serie de caracteres

caracteres del mensaje.

[2] En este sentido, tenga presente con qué usuario esta ejecutando su aplicación, podría tener problemas en la ejecución de programas que utilizan IPC si el mismo no posee los privilegios suficientes. En tal caso, pruebe de ejecutar el código utilizando el usuario root (administrador).

(totalmente variable, según nuestra necesidad), para las funciones de envío y recepción de mensajes (`msgsnd()` y `msgrcv()`) obviamente, el mensaje deberá simbolizarse como algo de tipo `void *` (un puntero o referencia a un tipo de dato `void`, un tipo genérico, al cual pueda adaptarse cualquier mensaje). Nuestra API continuará con esta idea, pero obviamente habrá que indicar el largo del mensaje (que no incluye al `long` inicial). También se agregó que se pueda indicar el tipo, de esta forma, el programador no tiene necesidad de cargar el tipo de mensaje en este `long` incial, ya que lo haremos nosotros haciendo uso de `memcpy()` y del hecho que sabemos que todo mensaje comienza con este `long`:

```
/* Envia un mensaje de determinado tipo a la cola */
int enviar_msg(int qid,void *qbuf, int msgsize, long type)  {
   memcpy(qbuf,&type,sizeof(long)); //el efecto generico de: qbuf->mtype = type;
   return msgsnd(qid,qbuf,msgsize, 0);
}
/* Obtiene un mensaje de la cola de un determinado tipo */
int recibir_msg(int qid, void *qbuf, int msgsize,long type) {
   memcpy(qbuf,&type,sizeof(long));
   return msgrcv(qid, qbuf, msgsize, type, 0);
}
```

Esto otorga gran flexibilidad, por ejemplo, nuestro servidor podría declarar el formato de los mensajes de la cola como:

```
#define SUC_MSG_LEN 20
...
struct sucmsg {
   long mtype;
   char mtext[SUC_MSG_LEN];
};
...
```

y luego hacer uso de la librería para recibir mensajes de tipo `1` (`1L`) con largo `SUC_MSG_LEN`:

```
...
   struct sucmsg qbuf;
   ...
        if ( recibir_msg(idcola,&qbuf,SUC_MSG_LEN,1L) != -1) {
            printf("main():recibi [%s] como peticion\n",qbuf.mtext);
         }
   ...
...
```

Puede completar el resto de la API, hacer un pequeño programa de prueba de la librería y compilar el código.

2. Utilice el programa de prueba de la librería del punto anterior y cópielo como `tp71.c`. Implemente en este programa el servidor de la cola de mensajes, tal como se indicó en la introducción.
Esta primera versión simplemente creará la cola, esperará por mensajes, mostrará el contenido de los mensajes recibidos.
Compile y ejecute `tp71.c`. El programa quedará bloqueado esperando mensajes de una cola a quien nadie envía nada, efectivamente, la función `msgrcv()` es bloqueante, deje por ahora el

programa bloqueado y abra otra consola, ejecute el comando: `ipcs` (consulte su sintaxis: `man ipcs`), este comando Unix es muy importante porque permite la administración (por fuera de las aplicaciones) de los recursos IPC actualmente existentes en el kernel del SO, de este comando podrá obtener una salida como:

```
...
grchere@debian:~/gccwork/src/msg$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere    600        196608     2          dest
0x00000000 65537      grchere    600        196608     2          dest
0x00000000 98306      grchere    600        393216     2          dest
0x00000000 131075     grchere    600        393216     2          dest
0x00000000 163844     grchere    600        12288      2          dest
0x00000000 196613     grchere    600        196608     2          dest
0x00000000 229382     grchere    600        393216     2          dest
0x00000000 262151     grchere    600        393216     2          dest
0x00000000 360456     grchere    600        393216     2          dest
0x00000000 393225     grchere    600        196608     2          dest
0x00000000 720906     grchere    600        393216     2          dest
0x00000000 753675     grchere    600        196608     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

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

grchere@debian:~/gccwork/src/msg$
```

Tendrá que presionar `Ctrl+C` para interrumpir el programa servidor, en tal caso, podrá quedar la cola de mensajes creada, use el comando `ipcs` para destruirla en caso de que sea necesario.

3. Modifique el programa `tp71.c` para permitir una salida más elegante del mismo y que pueda borrar la cola de mensajes sin problemas, para ello implemente la señal `SIGUSR2` en combinación con una variable global (`volatile`) para indicar que el programa debe salir. Al enviar una señal `SIGUSR2` el programa servidor deberá detenerse:

```
grchere@debian:~/gccwork/src/msg$ ./tp71 &
main():inicio servidor Cola de Mensajes!
main():para salir envie se?al SIGUSR2 a proceso 3265
main():inicializo total de ventas de sucursales
main():solicito clave ipc
main():creo cola de mensajes con clave ipc
main():quedo a la espera de mensajes en cola [98304]...
[1] 3265
grchere@debian:~/gccwork/src/msg$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere    600        196608     2          dest
0x00000000 65537      grchere    600        196608     2          dest
0x00000000 98306      grchere    600        393216     2          dest
0x00000000 131075     grchere    600        393216     2          dest
0x00000000 163844     grchere    600        12288      2          dest
```

```
0x00000000 196613     grchere    600        196608     2          dest
0x00000000 229382     grchere    600        393216     2          dest
0x00000000 262151     grchere    600        393216     2          dest
0x00000000 360456     grchere    600        393216     2          dest
0x00000000 393225     grchere    600        196608     2          dest
0x00000000 720906     grchere    600        393216     2          dest
0x00000000 753675     grchere    600        196608     2          dest
0x00000000 851980     grchere    600        393216     2          dest
0x00000000 884749     grchere    600        196608     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x6d0229fa 98304      grchere    660        0            0

grchere@debian:~/gccwork/src/msg$

grchere@debian:~/gccwork/src/msg$ kill -SIGUSR2 3265
grchere@debian:~/gccwork/src/msg$ main():recibi se?al de salida
main():fin recepcion de mensajes, borra cola [98304]
main():fin servidor  Cola de Mensajes! retorno=0

[1]+  Done                    ./tp71
grchere@debian:~/gccwork/src/msg$
```

3. Crear el programa `tp72.c`. Este programa será el cliente de `tp71.c`. Hágalo según lo indicado en la introducción. Este programa solicitará los totales de ventas de `10` sucursales, indicando su id de proceso y número de sucursal, enviando mensajes de tipo `1` y por cada envío esperará la respuesta, mostrando lo recibido del servidor en la consola.
Fácilmente puede armar el mensaje a enviar al servidor utilizando la función `snprintf()` de la siguiente forma:

```
...
snprintf(qbuf.mtext,SUC_MSG_LEN,"<%d/%d>",pid,suc);
...
```

Como siempre, trabaje con -al menos- dos consolas, una para el servidor y otra para el cliente, podrá obtener salidas como esta:

```
consola server:
grchere@debian:~/gccwork/src/msg$ ./tp71 &
main():inicio servidor Cola de Mensajes!
main():para salir envie se?al SIGUSR2 a proceso 3316
main():inicializo total de ventas de sucursales
main():solicito clave ipc
main():creo cola de mensajes con clave ipc
main():quedo a la espera de mensajes en cola [131072]...
[1] 3316
grchere@debian:~/gccwork/src/msg$
grchere@debian:~/gccwork/src/msg$ main():recibi [<3332/0>] como peticion
main():recibi [<3332/1>] como peticion
main():recibi [<3332/2>] como peticion
main():recibi [<3332/3>] como peticion
```

```
main():recibi [<3332/4>] como peticion
main():recibi [<3332/5>] como peticion
main():recibi [<3332/6>] como peticion
main():recibi [<3332/7>] como peticion
main():recibi [<3332/8>] como peticion
main():recibi [<3332/9>] como peticion

grchere@debian:~/gccwork/src/msg$
```

consola cliente:

```
grchere@debian:~/gccwork/src/msg$ ./tp72
main():inicio cliente Cola de Mensajes!
main():solicito clave ipc
main():obtengo cola de mensajes con clave ipc
main():envie [<3332/0>] Ok, ahora espero respuesta
main():envie [<3332/1>] Ok, ahora espero respuesta
main():envie [<3332/2>] Ok, ahora espero respuesta
main():envie [<3332/3>] Ok, ahora espero respuesta
main():envie [<3332/4>] Ok, ahora espero respuesta
main():envie [<3332/5>] Ok, ahora espero respuesta
main():envie [<3332/6>] Ok, ahora espero respuesta
main():envie [<3332/7>] Ok, ahora espero respuesta
main():envie [<3332/8>] Ok, ahora espero respuesta
main():envie [<3332/9>] Ok, ahora espero respuesta
main():fin envio de mensajes a cola [131072]
main():fin cliente Cola de Mensajes! retorno=0
grchere@debian:~/gccwork/src/msg$
```

4. El programa servidor requiere hacer un *parsing* de lo recibido. Copie `tp71.c` en `tp73.c`. Vamos a trabajar sobre `tp73.c` para hacer una nueva versión del servidor. Este código ya lo hemos hecho, debemos reutilizarlo. Si Ud. tiene conocimientos en cuanto a la compilación y creación de librerías estáticas[3], podría hacer una partiendo de lo hecho en el TP VI - FIFO's, para reutilizar el código de *parsing*. Caso contrario, para no complejizar aún más esta práctica, puede copiar dicho código (`myfifo.h`, `myfifo.c`) a su directorio de trabajo y compilar este programa con esta libreria (`gcc -Wall -o tp73 tp73.c myipc.c myfifo.c`). Cabe dejar bien en claro que esto no es reutilizar código, lo cual supone la existencia de código en un único lugar a ser compartido en `n` proyectos.
Modificar `tp73.c`, usar `strcpyentre()` para hacer *parsing* de petición de clientes, obtener id proceso y sucursal, tomar el total de ventas de dicha sucursal y enviar mensaje de respuesta a cliente[4]. Compilar con la librería `myfifo.c`.
El programa cliente también requiere algunos cambios. Copiar `tp72.c` en `tp74.c`. Vamos a trabajar sobre `tp74.c` para hacer una nueva versión del cliente. El cliente deberá mostrar el mensaje recibido del servidor, luego del envío de la petición.
Podrá obtener una salida como esta:

[3] Se recomienda la lectura de todo el apunte /Apuntes/ProgramacionI-2008-gcc.doc que se encuentra en el cd-rom de programación de esta asignatura, titulado "Trabajando con el Compilador GNU C/C++ GCC", allí encontrará la forma de crear un ambiente de trabajo y como crear/catalogar librerias a ser reutilizadas en n proyectos.
[4] Obsérvese que tal vez Ud. no encontró sentido en los cambios propuestos en el punto 10 del TP VI - FIFO's, en cuanto a la eliminación de código repetitivo y poco reusable. Si no se hubieran implementado esos cambios, aquí no podríamos hacer esta reutilización!. De eso se trata la programación.

consola server:

```
grchere@debian:~/gccwork/src/msg$ ./tp73 &
main():inicio servidor Cola de Mensajes!
main():para salir envie se?al SIGUSR2 a proceso 3461
main():inicializo total de ventas de sucursales
main():solicito clave ipc
main():creo cola de mensajes con clave ipc
main():quedo a la espera de mensajes en cola [196608]...
[1] 3461
grchere@debian:~/gccwork/src/msg$ main():recibi [<3463/0>] como peticion
main():envie [<0/176.74>] como respuesta a peticion
main():recibi [<3463/1>] como peticion
main():envie [<1/139.95>] como respuesta a peticion
main():recibi [<3463/2>] como peticion
main():envie [<2/53.98>] como respuesta a peticion
main():recibi [<3463/3>] como peticion
main():envie [<3/153.95>] como respuesta a peticion
main():recibi [<3463/4>] como peticion
main():envie [<4/184.47>] como respuesta a peticion
main():recibi [<3463/5>] como peticion
main():envie [<5/53.13>] como respuesta a peticion
main():recibi [<3463/6>] como peticion
main():envie [<6/76.13>] como respuesta a peticion
main():recibi [<3463/7>] como peticion
main():envie [<7/129.18>] como respuesta a peticion
main():recibi [<3463/8>] como peticion
main():envie [<8/139.09>] como respuesta a peticion
main():recibi [<3463/9>] como peticion
main():envie [<9/38.64>] como respuesta a peticion

grchere@debian:~/gccwork/src/msg$
```

consola cliente:

```
grchere@debian:~/gccwork/src/msg$ ./tp74
main():inicio cliente Cola de Mensajes!
main():solicito clave ipc
main():obtengo cola de mensajes con clave ipc
main():envie [<3463/0>] Ok, ahora espero respuesta
main():recibi [<0/176.74>] del servidor de cola de mensajes
main():envie [<3463/1>] Ok, ahora espero respuesta
main():recibi [<1/139.95>] del servidor de cola de mensajes
main():envie [<3463/2>] Ok, ahora espero respuesta
main():recibi [<2/53.98>] del servidor de cola de mensajes
main():envie [<3463/3>] Ok, ahora espero respuesta
main():recibi [<3/153.95>] del servidor de cola de mensajes
main():envie [<3463/4>] Ok, ahora espero respuesta
main():recibi [<4/184.47>] del servidor de cola de mensajes
main():envie [<3463/5>] Ok, ahora espero respuesta
main():recibi [<5/53.13>] del servidor de cola de mensajes
main():envie [<3463/6>] Ok, ahora espero respuesta
main():recibi [<6/76.13>] del servidor de cola de mensajes
main():envie [<3463/7>] Ok, ahora espero respuesta
main():recibi [<7/129.18>] del servidor de cola de mensajes
main():envie [<3463/8>] Ok, ahora espero respuesta
main():recibi [<8/139.09>] del servidor de cola de mensajes
main():envie [<3463/9>] Ok, ahora espero respuesta
main():recibi [<9/38.64>] del servidor de cola de mensajes
```

```
main():fin envio de mensajes a cola [196608]
main():fin cliente Cola de Mensajes! retorno=0
grchere@debian:~/gccwork/src/msg$
```

5. Teniendo en cuenta lo aprendido acerca de Señales (Signals), se recomienda poner alarma de `n` segundos en el programa cliente (usar captura de señal `SIGALRM`, combinado con la función `alarm()`), ya que, si el servidor nunca contesta la petición del cliente, éste quedará bloqueado (en función `msgrcv()`) esperando una respuesta que nunca llegará.

Bibliografía

- Stevens, Richard, "Advanced Programming in the UNIX Environment", Addison-Wesley Professional Computing Series, 1993, ISBN 0-201-56317-7
- LooseSandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram, Ulrich Drepper, "The GNU C Library Reference Manual", Free Software Foundation, 2007, Boston, USA.

Espero que esta práctica haya sido de vuestro agrado y le permita comprender mejor los mecanismos básicos de manejo de cola de mensajes, uno de los tres mecanismos de IPC que nos provee el kernel del SO.