TP VI – Tuberías - FIFO's

Objetivo: Desarrollar una serie de programas que permitan comprobar distintas funcionalidades vinculadas con la gestión de tuberías (pipes) de tipo FIFO y su aplicación. Introducir al desarrollo de programas servidores y clientes. Estimular una programación modular, con código reusable. Generalizar funciones. Usar técnicas avanzadas de I/O y manejo de punteros.

Introducción

En el TP de Pipes se habló de dos limitaciones en cuanto a los pipes, para salvar el problema de unidireccionalidad de los pipes se pueden utilizar los *stream pipes*, mientras que para salvar el problema de que los pipes requieren de un ancestro en común, se pueden utilizar pipes de tipo FIFO (first-input-first-output) también llamados *named pipes*.

Un programa servidor puede crear un FIFO que sea conocido por todos sus clientes utilizando la función `mkfifo()`, este FIFO queda asociado a un archivo dentro del sistema de archivos del SO, por ejemplo: `/tmp/myfifo` se pueden realizar operaciones de I/O sobre dicho FIFO para comunicar procesos como si se tratase de un archivo más del sistema de archivos y sin necesidad de que los procesos tengan una relación padre-hijo.

Las operaciones de I/O se considerarán atómicas siempre y cuando la operación involucre un número de bytes menor que PIPE_BUF (valor predefinido para el SO).

Esto permite el desarrollo de aplicaciones client-server a través del uso de *named pipes* o FIFO's; un servidor puede crear un FIFO, abrirlo de modo lectura, mientras que los clientes abrirán ese mismo FIFO de modo grabación. Los clientes "grabarán" sus peticiones y el servidor "leerá" las peticiones para su proceso. Sin embargo, queda una cuestión por resolver: ¿Cómo puede el servidor saber cuales son sus clientes? ¿Cómo podemos implementar un "ida y vuelta" entre cliente y servidor? El cliente envía la petición, pero necesita enterarse si la misma fue procesada o no y en tal caso, el resultado del proceso.

Una posibilidad es plantear la siguiente solución: los clientes crearán un FIFO en una ubicación conocida tanto por cliente como por servidor, el nombre del FIFO incluirá el número de proceso del cliente (de modo tal, que sea un FIFO distinto para cada cliente); los clientes, junto con su petición, enviarán su id de proceso al servidor; el servidor tomará su id de proceso y grabará la respuesta de su petición en el FIFO previamente abierto por cliente. De esta forma, tendremos la posibilidad de implementar una "vuelta" por parte del servidor; en el párrafo anterior explicamos una forma de "ida".

Una vez que el cliente ha terminado de enviar y recibir peticiones, cerrará el pipe con el servidor y borrará el pipe creado para leer las respuestas del servidor.

Una vez que el servidor ha terminado de trabajar, cerrará el pipe creado para atender a sus clientes y lo borrará del sistema de archivos.

De esta forma, tanto clientes como servidores dejarán "limpio" al sistema de archivos.

1. El Analista de Sistemas Y de la Empresa X ha podido comprobar las ventajas de una arquitectura client-server. Entonces ahora pretende implementar un servidor que pueda atender a clientes. Este servidor se encargará de enviar a los clientes los totales de ventas de las sucursales que los clientes requieran. El diseño de este cliente y servidor, responde a lo expuesto en la sección anterior.
Crear la carpeta `fifo` para almacenar este proyecto.
Dentro de la carpeta `fifo`, crear la carpeta `v1` para almacenar la versión 1 de este proyecto.
Dentro de `v1` vamos a crear el programa `tp61.c`.
El servidor tendrá el siguiente diseño:

```
main()
      crear FIFO en sistema de archivos
      si pude crear FIFO Ok entonces
            otorgar permisos a FIFO en sistema de archivos
            abrir FIFO de R/W
            crear buffer de MAX_BUFFER caracteres
            mientras VERDAD (loop infinito)
                  leer buffer de FIFO
                  si lectura FIFO Ok entonces
                        imprimir buffer
                  fin si
            fin mientras
            cerrar FIFO
            borrar FIFO del sistema de archivos
      sino
            error en creacion de FIFO!
      fin si
fin main()
```

Este servidor permitirá comprobar si recibimos los mensajes que enviarán los clientes, pero requerirá de que terminemos anormalmente el proceso, puesto que quedará atrapado en un loop infinito. Comencemos a desmenuzar este código.
Ejemplo de crear FIFO:

```
int mf = mkfifo(MY_FIFO,O_CREAT);
```

donde MY_FIFO esta definido dentro de `myfifo.h` el cual es un archivo de cabecera compartido entre cliente y servidor para indicar allí el nombre del FIFO, en este caso:

```
#define MAX_BUFFER 255          // deberia ser menor que PIPE_BUF
#define MY_FIFO "/tmp/myfifo"
```

¿Cómo saber si el FIFO se pudo crear Ok?:

```
if ( mf == 0 || errno == EEXIST ) {
      <<FIFO creado Ok>>
} else {
      <<error en creacion de FIFO >>
}
```

¿Cómo otorgar permisos a un archivo?, usando la función `chmod()` (análoga al comando `chmod`):

```
chmod(MY_FIFO,0777);
```

en este caso, otorgamos todo tipo de permisos (rwx) para el dueño del archivo, el grupo al que pertenece y el resto de usuarios.

Abrir un FIFO o cualquier archivo:

```
int fdi = open(MY_FIFO,O_RDWR,0777);
```

`open()` devuelve un descriptor de archivo (file descriptor) a través del cual podremos hacer todas las operaciones de I/O.
Crear un buffer no es mas que reservar memoria automática (en este caso) para un conjunto de bytes en donde almacenaremos la petición de los clientes:

```
char buffer[MAX_BUFFER+1];
```

donde MAX_BUFFER esta definido dentro de `myfifo.h`.
Leer de un archivo:

```
int nread = read(fdi,buffer,MAX_BUFFER);
```

donde `nread` indica la cantidad efectiva de bytes leidos, cuidando de no sobrepasar la capacidad de buffer. Si `nread` es igual a -1, indica que hubo un error en la lectura.

Para cerrar un FIFO o cualquier archivo:

```
close(fdi);
```

Para borrar el FIFO del sistema de archivos, usar la función `remove()`:

```
remove(MY_FIFO);
```

Consulte el manual de referencia de la libreria GNU C para más información (`/Software/Win32/gcc/docs/libc.pdf`).
Algunos de los headers que deberá utilizar para implementar este servidor:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <string.h>
#include <limits.h>

#include "myfifo.h"  // libreria propia del servidor de la Empresa X
```

Implemente y compile `tp6.1`. Ejecute `tp61` en background, puesto que quedará bloqueado en la función `read()`.

2. Estando en ejecución `tp61` verificar el directorio `/tmp`, ejecutar el comando `ls -l /tmp` y chequear si allí se puede ver el archivo asociado con el FIFO del servidor. Recuerde que `tp61` quedará bloqueado, deberá usar el comando `kill` para enviar la señal de terminación de proceso a `tp61`: `kill -SIGKILL <id de proceso tp61>` tambien puede probar: `kill -SIGINT <id de proceso tp61>`.

3. Desarrolle un programa que será el cliente que realice peticiones al servidor del punto anterior. Recuerde que siempre debe ejecutarse primero el servidor y luego el cliente. El diseño del cliente sería el siguiente:

```
main()
      abrir FIFO de W
      crear buffer de MAX_BUFFER caracteres
      para <suc> 0 hasta 9 hacer
            armar en buffer mensaje a enviar al servidor para <suc>
            grabar buffer en FIFO
            imprimir buffer indicado lo enviado al servidor
      fin para
      cerrar FIFO
fin main()
```

Según lo indicado en la introducción, debemos enviar como dato al servidor el id de proceso del cliente y el número de sucursal de la cual el cliente requiere su total de ventas. Para facilitar el *parsing* de este mensaje por parte del servidor, se podría optar por el siguiente formato:

"<id proceso cliente/sucursal>"

una forma posible de armar el buffer, sería:

```
...
   char buffer[MAX_BUFFER+1];
   ...
   int suc;
   for(suc=0;suc<10;suc++) { // solicito total ventas de sucursal suc

         snprintf(buffer,MAX_BUFFER,"<%d/%d>",(int) getpid(),suc);
         ...
   }
...
```

Implemente el programa `tp62.c` y compílelo como `tp62`. Ejecute `tp62` en foreground (recuerde que antes debe estar ejecutándose `tp61`). Recomendamos que utilice dos consolas, una para ejecutar el servidor y otra para ejecutar el cliente a modo de no confundirse con los mensajes. Observe los mensajes enviados por `tp62` y los recibidos por `tp61`:

```
consola servidor:

grchere@debian:~/gccwork/src/fifo/v1$ ./tp61 &
[1] 2813
grchere@debian:~/gccwork/src/fifo/v1$ main():inicio servidor FIFO!
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255
```

```
grchere@debian:~/gccwork/src/fifo/v1$
main():lei: [<2838/0>] de FIFO [/tmp/myfifo]
main():lei: [<2838/1>] de FIFO [/tmp/myfifo]
main():lei: [<2838/2>] de FIFO [/tmp/myfifo]
main():lei: [<2838/3>] de FIFO [/tmp/myfifo]
main():lei: [<2838/4>] de FIFO [/tmp/myfifo]
main():lei: [<2838/5>] de FIFO [/tmp/myfifo]
main():lei: [<2838/6>] de FIFO [/tmp/myfifo]
main():lei: [<2838/7>] de FIFO [/tmp/myfifo]
main():lei: [<2838/8>] de FIFO [/tmp/myfifo]
main():lei: [<2838/9>] de FIFO [/tmp/myfifo]

grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ ps
  PID TTY          TIME CMD
 2794 pts/0    00:00:00 bash
 2813 pts/0    00:00:00 tp61
 2839 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/fifo/v1$
[1]+  Terminated              ./tp61
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ ls -l /tmp
total 12
drwx------ 2 grchere grchere 4096 2009-06-25 14:17 gconfd-grchere
drwx------ 2 grchere grchere 4096 2009-06-25 14:16 keyring-SehzsZ
prwxrwxrwx 1 grchere grchere    0 2009-06-25 14:36 myfifo
drwx------ 2 grchere grchere 4096 2009-06-25 14:17 orbit-grchere
prwxr-xr-x 1 grchere grchere    0 2009-06-25 14:33 SciTE.2822.in
grchere@debian:~/gccwork/src/fifo/v1$


consola cliente:
grchere@debian:~/gccwork/src/fifo/v1$ ./tp62
main():inicio cliente FIFO!
main():envie 8 bytes a servidor [<2838/0>]
main():envie 8 bytes a servidor [<2838/1>]
main():envie 8 bytes a servidor [<2838/2>]
main():envie 8 bytes a servidor [<2838/3>]
main():envie 8 bytes a servidor [<2838/4>]
main():envie 8 bytes a servidor [<2838/5>]
main():envie 8 bytes a servidor [<2838/6>]
main():envie 8 bytes a servidor [<2838/7>]
main():envie 8 bytes a servidor [<2838/8>]
main():envie 8 bytes a servidor [<2838/9>]
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ kill -SIGTERM 2813
grchere@debian:~/gccwork/src/fifo/v1$
```

4. Debemos encontrar un medio para que el servidor no quede permanentemente bloqueado en la función `read()`,a la espera -tal vez interminable- de un cliente, una forma posible sería multiplexando las operaciones de I/O (I/O multiplexing) , es una técnica avanzada de I/O que puede lograrse a través de la función `select()` o bien de la función `poll()`. Permitirá darle a la función `read()` cierto time-out, es decir, que no quede bloqueada por siempre. La idea sería invocar a la función `poll()`, ésta nos dirá cuando hay una operación de I (input) pendiente (devuelve 1) y recién allí invocaríamos a `read()` con un time-out preestablecido. Luego de atendido el cliente, volveríamos nuevamente a la función `poll()`, el esquema es el siguiente:

```
main()
      crear FIFO en sistema de archivos
      si pude crear FIFO Ok entonces
            otorgar permisos a FIFO en sistema de archivos
            abrir FIFO de R/W
            crear buffer de MAX_BUFFER caracteres
            mientras VERDAD (loop infinito)
                  polling de FIFO con 10 segundos de time-out
                  si hay lectura pendiente entonces
                        leer buffer de FIFO
                        si lectura FIFO Ok entonces
                              imprimir buffer
                        fin si
                  sino
                        error en polling
                  fin si
            fin mientras
            cerrar FIFO
            borrar FIFO del sistema de archivos
      sino
            error en creacion de FIFO!
      fin si
fin main()
```

¿Qué requiere la función `poll()`? Básicamente, hay que indicarle sobre cuáles descriptores de archivos (file descriptors) estamos interesados y sobre cuáles eventos le puedan ocurrir a esos descriptores. Esta información se proporciona a través de la estructura `pollfd`. Debemos agregar los headers: `<stropts.h>` y `<poll.h>` requeridos por esta función. Para nuestro caso, lo podemos hacer de la siguiente forma:

```
...
   // polling info
   int rpoll;
   struct pollfd fdpoll[1];
   fdpoll[0].fd = fdi;
   fdpoll[0].events = POLLIN | POLLPRI;
....
   while(1) { // loop infinito
      rpoll = poll(fdpoll,1L,10000);  // time out de 10 segundos para read()
...
   }
...
```

Copie a `tp61.c` como `tp63.c`, implemente el polling propuesto, compile como `tp63` y ejecute en background. Incluya varios mensajes en esta nueva versión del servidor para verificar si queda bloqueado como antes o no. Puede continuar usando el mismo cliente.

5. Si bien hemos avanzado en cuanto al bloqueo, no obstante el servidor continúa dentro de un bucle infinito. Como Ud. ya sabe sobre señales, se propone implementar una forma elegante de terminar este servidor utilizando la señal `SIGUSR2`, tal como ya lo ha hecho antes. La idea es que cuando el servidor reciba la señal `SIGUSR2`, éste salga del loop infinito y termine normalmente. Debemos capturar la señal `SIGUSR2`, y cuando ésta se produzca, en la función que manipula esta señal, debemos poner la variable global `salir` en `1` (true, verdad, activado) y ahora el loop inifito

pasaría a ser `while(!salir) { ... }` se puede declarar a `salir` como una variable entera fuera de la función `main()`, puesto que es global.

Copie `tp63.c` como `tp64.c`, implemente la manipulación de la señal, la variable global, cambie el loop infinito, Compile como `tp64`, pruebe el sistema, podría obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/fifo/v1$ ./tp64 &
main():inicio servidor FIFO!
main():para salir envie se?al SIGUSR2 a proceso 2847
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255
[1] 2847
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ main():lei:
[<2849/0><2849/1><2849/2><2849/3><2849/4><2849/5><2849/6><2849/7><2849/8><2849/9
>] de FIFO [/tmp/myfifo]

grchere@debian:~/gccwork/src/fifo/v1$


consola cliente:
grchere@debian:~/gccwork/src/fifo/v1$ ./tp62
main():inicio cliente FIFO!
main():envie 8 bytes a servidor [<2849/0>]
main():envie 8 bytes a servidor [<2849/1>]
main():envie 8 bytes a servidor [<2849/2>]
main():envie 8 bytes a servidor [<2849/3>]
main():envie 8 bytes a servidor [<2849/4>]
main():envie 8 bytes a servidor [<2849/5>]
main():envie 8 bytes a servidor [<2849/6>]
main():envie 8 bytes a servidor [<2849/7>]
main():envie 8 bytes a servidor [<2849/8>]
main():envie 8 bytes a servidor [<2849/9>]
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v1$
```

Al enviar la señal `SIGUSR2` al proceso `tp64` debería suceder:

```
consola cliente:
grchere@debian:~/gccwork/src/fifo/v1$ kill -SIGUSR2 2847
grchere@debian:~/gccwork/src/fifo/v1$

consola server:
grchere@debian:~/gccwork/src/fifo/v1$ main():recibi se?al de salida
main():fin servidor FIFO! retorno=0

[1]+  Done                    ./tp64
grchere@debian:~/gccwork/src/fifo/v1$
```

6. Ahora debemos hacer que el servidor pueda obtener un total de ventas de la sucursal para cada sucursal pedida por los clientes, para ello, vamos a simular los valores, de la misma forma que lo hicimos en `tp51.c`. Podríamos tener un arreglo global para los totales: `double ventas[10];` y agregar la función `init()` que se llamaría la comienzo del servidor, inicializando el vector:

```
void init() {
   srandom(time(0));
   int i;
   for(i=0;i<10;i++) ventas[i]=((double) random())/10000000.0;
}
```

Otro problema es que el servidor debe interpretar las peticiones de los clientes (*parsing*) para ello implementamos la función `parse()` y luego, una vez que sabemos lo que el cliente quiere, podemos llamar a la función `proceso()`:

```
// analiza buffer recibido de los clientes para determinar id de proceso cliente
// y nro de sucursal solicitada
// formato de buffer: "<id proceso/nro sucursal>...<id proceso/nro sucursal>"
void parse(char *buffer) {
   char id[MAX_BUFFER];
   char suc[MAX_BUFFER];
   int iid,isuc;
   char *p;
   while(*buffer) {
      memset(id,0,MAX_BUFFER);
      memset(suc,0,MAX_BUFFER);
      while( *buffer && *buffer != '<') buffer++;
      if (  *buffer == '<' ) buffer++;
      p=id;
      while( *buffer && *buffer != '/') *p++=*buffer++;
      if (  *buffer == '/' ) buffer++;
      p=suc;
      while( *buffer && *buffer != '>') *p++=*buffer++;
      if (  *buffer == '>' ) buffer++;
      // final de parsing, ¿que obtuve?
      printf("parse(): id proceso [%s] sucursal [%s]\n",id,suc);
      iid=atoi(id);
      isuc=atoi(suc);
      printf("parse(): id proceso [%d] sucursal [%d]\n",iid,isuc);
      proceso(iid,isuc);
   }
}

// luego de parsing, proceso id de proceso, id de sucursal
void proceso(int id,int suc) {
   printf("proceso(): id proceso [%d] sucursal [%d] total $
%lf\n",id,suc,ventas[suc]);
}
```

Copie `tp64.c` como `tp65.c`, implemente las funciones propuestas, llame a estas funciones en el lugar indicado, Compile como `tp65`, pruebe el sistema, podría obtener una salida como esta:

```
consola servidor:
grchere@debian:~/gccwork/src/fifo/v1$ ./tp65 &
main():inicio servidor FIFO!
main():para salir envie se?al SIGUSR2 a proceso 2858
main():inicializo total de ventas de sucursales
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255
[1] 2858
grchere@debian:~/gccwork/src/fifo/v1$ main():lei: [<2860/0>] de FIFO
```

```
[/tmp/myfifo]
parse(): id proceso [2860] sucursal [0]
parse(): id proceso [2860] sucursal [0]
proceso(): id proceso [2860] sucursal [0] total $  138.479047
main():lei: [<2860/1>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [1]
parse(): id proceso [2860] sucursal [1]
proceso(): id proceso [2860] sucursal [1] total $  55.640795
main():lei: [<2860/2>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [2]
parse(): id proceso [2860] sucursal [2]
proceso(): id proceso [2860] sucursal [2] total $  99.352217
main():lei: [<2860/3>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [3]
parse(): id proceso [2860] sucursal [3]
proceso(): id proceso [2860] sucursal [3] total $  138.450279
main():lei: [<2860/4>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [4]
parse(): id proceso [2860] sucursal [4]
proceso(): id proceso [2860] sucursal [4] total $  10.434312
main():lei: [<2860/5>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [5]
parse(): id proceso [2860] sucursal [5]
proceso(): id proceso [2860] sucursal [5] total $  84.119174
main():lei: [<2860/6>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [6]
parse(): id proceso [2860] sucursal [6]
proceso(): id proceso [2860] sucursal [6] total $  45.279634
main():lei: [<2860/7>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [7]
parse(): id proceso [2860] sucursal [7]
proceso(): id proceso [2860] sucursal [7] total $  212.639716
main():lei: [<2860/8>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [8]
parse(): id proceso [2860] sucursal [8]
proceso(): id proceso [2860] sucursal [8] total $  54.377959
main():lei: [<2860/9>] de FIFO [/tmp/myfifo]
parse(): id proceso [2860] sucursal [9]
parse(): id proceso [2860] sucursal [9]
proceso(): id proceso [2860] sucursal [9] total $  82.784439
main():recibi se?al de salida
main():fin servidor FIFO! retorno=0

[1]+  Done                    ./tp65
grchere@debian:~/gccwork/src/fifo/v1$
```

consola cliente:

```
grchere@debian:~/gccwork/src/fifo/v1$ ./tp62
main():inicio cliente FIFO!
main():envie 8 bytes a servidor [<2860/0>]
main():envie 8 bytes a servidor [<2860/1>]
main():envie 8 bytes a servidor [<2860/2>]
main():envie 8 bytes a servidor [<2860/3>]
main():envie 8 bytes a servidor [<2860/4>]
main():envie 8 bytes a servidor [<2860/5>]
main():envie 8 bytes a servidor [<2860/6>]
main():envie 8 bytes a servidor [<2860/7>]
main():envie 8 bytes a servidor [<2860/8>]
main():envie 8 bytes a servidor [<2860/9>]
```

```
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ kill -SIGUSR2 2858
grchere@debian:~/gccwork/src/fifo/v1$
```

7. Modificar el servidor para grabar en un FIFO propio para cada cliente el total de ventas de la sucursal, con el siguiente formato: <sucursal/total ventas> . El FIFO en donde grabará la respuesta el servidor es un FIFO previamente creado por el cliente y su nombre tiene el formato: `MY_FIFO.<id proceso cliente>`. Por lo tanto, cliente y servidor, ambos, crean FIFO's, aquí tenemos código que puede compartirse y es repetitivo. Entonces vamos a crear una biblioteca de funciones compartidas `myfifo.c` allí implementaremos la función `crearfifo()`:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <string.h>
#include <limits.h>

#include "myfifo.h"

// crea pipe, devuelve true si pudo hacerlo, caso contrario devuelve false
int crearfifo(char *pipe) {
      int mf = mkfifo(pipe,O_CREAT);
      if ( mf == 0 || errno == EEXIST ) {
         chmod(pipe,0777);
         return 1;
      } else return 0;
}
```

Ahora la función `proceso()`, debería contestarle al cliente el total de ventas de sucursal pedido:

```
// luego de parsing, proceso id de proceso, id de sucursal
void proceso(int id,int suc) {
   char fifo[MAX_BUFFER],buffer[MAX_BUFFER];
   snprintf(fifo,MAX_BUFFER,"%s.%d",MY_FIFO,id);
   printf("proceso(): id proceso [%d] sucursal [%d] total $  %lf
fifo:[%s]\n",id,suc,ventas[suc],fifo);
   //crearfifo(fifo); //se supone previamente creado por el cliente
   int fdo = open(fifo,O_RDWR,0777);
   snprintf(buffer,MAX_BUFFER,"<%d/%lf>",suc,ventas[suc]);
   write(fdo,buffer,strlen(buffer));
   close(fdo);
}
```

Copie `tp65.c` como `tp66.c`, implemente los cambios propuestos, compile como: `gcc -Wall -o tp66 tp66.c myfifo.c`

8. Modificar el cliente para crear FIFO cliente y recibir respuestas del servidor. Ahora el diseño del cliente sería:

```
main()
      obtengo id de proceso
      armo nombre de fifo cliente con MY_FIFO+id de proceso
      si pude crearfifo() entonces
            abrir FIFO cliente de RW
            abrir FIFO server de W
            crear buffer de MAX_BUFFER caracteres
            para <suc> 0 hasta 9 hacer
                  armar en buffer mensaje a enviar al servidor para <suc>
                  grabar buffer en FIFO server
                  imprimir buffer indicado lo enviado al servidor
                  leer buffer de FIFO cliente
                  imprimir buffer indicado lo recibido del servidor
            fin para
            cerrar FIFO server
            cerrar FIFO cliente
            borrar FIFO cliente del sistema de archivos
      sino
            error creando FIFO cliente!
      fin si
fin main()
```

Copie `tp62.c` como `tp67.c`, implemente los cambios propuestos, compile como: `gcc -Wall -o tp67 tp67.c myfifo.c` . Ejecute server, cliente, pruebe el sistema, podrá obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/fifo/v1$ ./tp66 &
main():inicio servidor FIFO!
main():para salir envie se?al SIGUSR2 a proceso 2863
main():inicializo total de ventas de sucursales
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255
[1] 2863
grchere@debian:~/gccwork/src/fifo/v1$ main():lei: [<2866/0>] de FIFO
[/tmp/myfifo]
parse(): id proceso [2866] sucursal [0]
parse(): id proceso [2866] sucursal [0]
proceso(): id proceso [2866] sucursal [0] total $  159.773684
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/1>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [1]
parse(): id proceso [2866] sucursal [1]
proceso(): id proceso [2866] sucursal [1] total $  53.234007
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/2>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [2]
parse(): id proceso [2866] sucursal [2]
proceso(): id proceso [2866] sucursal [2] total $  101.434253
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/3>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [3]
parse(): id proceso [2866] sucursal [3]
proceso(): id proceso [2866] sucursal [3] total $  209.114185
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/4>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [4]
```

```
parse(): id proceso [2866] sucursal [4]
proceso(): id proceso [2866] sucursal [4] total $  4.183333
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/5>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [5]
parse(): id proceso [2866] sucursal [5]
proceso(): id proceso [2866] sucursal [5] total $  183.813549
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/6>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [6]
parse(): id proceso [2866] sucursal [6]
proceso(): id proceso [2866] sucursal [6] total $  67.664777
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/7>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [7]
parse(): id proceso [2866] sucursal [7]
proceso(): id proceso [2866] sucursal [7] total $  118.434233
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/8>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [8]
parse(): id proceso [2866] sucursal [8]
proceso(): id proceso [2866] sucursal [8] total $  173.126028
fifo:[/tmp/myfifo.2866]
main():lei: [<2866/9>] de FIFO [/tmp/myfifo]
parse(): id proceso [2866] sucursal [9]
parse(): id proceso [2866] sucursal [9]
proceso(): id proceso [2866] sucursal [9] total $  104.360250
fifo:[/tmp/myfifo.2866]

grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ main():recibi se?al de salida
main():cierro FIFO server [/tmp/myfifo]
main():elimino FIFO server [/tmp/myfifo]
main():fin servidor FIFO! retorno=0

[1]+  Done                    ./tp66
grchere@debian:~/gccwork/src/fifo/v1$
```

consola cliente:

```
grchere@debian:~/gccwork/src/fifo/v1$ ./tp67
main():inicio cliente FIFO!
main():creando FIFO [/tmp/myfifo.2866]
main():abro FIFO [/tmp/myfifo] para grabar
main():abro FIFO [/tmp/myfifo.2866] para leer y grabar
main():envie 8 bytes a servidor [<2866/0>]
main():recibi 14 bytes de servidor [<0/159.773684>]
main():envie 8 bytes a servidor [<2866/1>]
main():recibi 13 bytes de servidor [<1/53.234007>]
main():envie 8 bytes a servidor [<2866/2>]
main():recibi 14 bytes de servidor [<2/101.434253>]
main():envie 8 bytes a servidor [<2866/3>]
main():recibi 14 bytes de servidor [<3/209.114185>]
main():envie 8 bytes a servidor [<2866/4>]
main():recibi 12 bytes de servidor [<4/4.183333>]
main():envie 8 bytes a servidor [<2866/5>]
main():recibi 14 bytes de servidor [<5/183.813549>]
main():envie 8 bytes a servidor [<2866/6>]
main():recibi 13 bytes de servidor [<6/67.664777>]
main():envie 8 bytes a servidor [<2866/7>]
```

```
main():recibi 14 bytes de servidor [<7/118.434233>]
main():envie 8 bytes a servidor [<2866/8>]
main():recibi 14 bytes de servidor [<8/173.126028>]
main():envie 8 bytes a servidor [<2866/9>]
main():recibi 14 bytes de servidor [<9/104.360250>]
main():cierro FIFO's
main():elimino FIFO cliente [/tmp/myfifo.2866]
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ ls -l /tmp
total 12
drwx------ 2 grchere grchere 4096 2009-06-25 14:17 gconfd-grchere
drwx------ 2 grchere grchere 4096 2009-06-25 14:16 keyring-SehzsZ
prwxrwxrwx 1 grchere grchere    0 2009-06-25 14:48 myfifo
drwx------ 2 grchere grchere 4096 2009-06-25 14:17 orbit-grchere
prwxr-xr-x 1 grchere grchere    0 2009-06-25 14:33 SciTE.2822.in
grchere@debian:~/gccwork/src/fifo/v1$
grchere@debian:~/gccwork/src/fifo/v1$ kill -SIGUSR2 2863
grchere@debian:~/gccwork/src/fifo/v1$
```

9. Analicemos el código y la funcionalidad de ambos programas (servidor y cliente), seguramente el cliente también necesitará hacer un *parsing* de lo recibido por parte del servidor, cuyo formato es idéntico al mensaje enviado por cliente, sólo cambia su contenido. Aquí tenemos otra oportunidad de generalizar código, la función `parse()`. El problema es que `parse()`, desde su interior, llama a la función `proceso()` y ésta es propia del servidor. Necesitamos hacer que esta función pueda ser "intercambiable" desde el exterior de `parse()`. La idea sería que el cliente use su propia función `proceso()`, distinta a la del servidor. ¿Cómo puede lograrse esto en `parse()`? Pasándole a `parse()` un puntero a la función `proceso()` a utilizar!. En C/C++, de la misma forma que exiten punteros a variables, también existen punteros a funciones, muy útiles en casos como este. El prototipo de la función parse sería:

```
void parse(char *buffer,void (*proceso)(char *campo1,char *campo2));
```

El primer argumento, es el buffer a parsear, el segundo es un puntero la función `proceso()` a usar, a su vez, la función `proceso()` tiene el prototipo:

```
void proceso(char *campo1,char *campo2);
```

En donde campo1 será id de proceso o sucursal, según sea el caso.
En donde campo2 será sucursal o total de ventas, según sea el caso.

La función `parse()` debería ser más genérica, simplemente obtener los dos campos del mensaje y llamar (para cada mensaje parseado) a `proceso()` pasando los campos como argumentos. Los campos son de tipos de datos distintos, puesto que el id de proceso es un entero, mientras que el total de ventas es un double. Para compatibilizar esto, podríamos pensar en una función `proceso()` genérica que reciba dos cadenas de caracteres y que cada "proceso()" haga las conversiones pertinentes.
Esta nueva versión de `parse()` deberá ser puesta dentro de `myfifo.c`, puesto que pasaría a ser código genérico, reutilizable por cliente y servidor:

```
// analiza buffer recibido de los clientes para determinar
// id de proceso cliente y nro de sucursal solicitada
// formato de buffer: "<campo1/campo2><campo1/campo2>....<campo1/campo2>"
void parse(char *buffer,void (*proceso)(char *campo1,char *campo2)) {
   char id[MAX_BUFFER];
   char suc[MAX_BUFFER];
   //int iid,isuc;
   char *p;
   while(*buffer) {
      memset(id,0,MAX_BUFFER);
      memset(suc,0,MAX_BUFFER);
      while( *buffer && *buffer != '<') buffer++;
      if (  *buffer == '<' ) buffer++;
      p=id;
      while( *buffer && *buffer != '/') *p++=*buffer++;
      if (  *buffer == '/' ) buffer++;
      p=suc;
      while( *buffer && *buffer != '>') *p++=*buffer++;
      if (  *buffer == '>' ) buffer++;
      // final de parsing, ¿que obtuve?
      printf("parse(): campo1 [%s] campo2 [%s]\n",id,suc);
      //iid=atoi(id);
      //isuc=atoi(suc);
      //printf("parse(): id proceso [%d] sucursal [%d]\n",iid,isuc);
      (*proceso)(id,suc);  // hago uso del puntero, ejecuto funcion!!
   }
}
```

¿Cómo sería la invocación de `parse()` desde el cliente o desde el servidor? simplemente:

```
parse(buffer,proceso);
```

Donde `proceso`, es el nombre de la función a ser invocada por `parse()` cada vez que obtiene el par: `campo1` y `campo2`!!.

*Dentro de la carpeta `fifo`, crear la carpeta `v2` para almacenar la versión 2 de este proyecto.
*Dentro de `v2` copie todo el contenido actual de `v1`.
*Modifique `myfifo.c`, mueva la función `parse()` de `tp66.c` a `myfifo.c`
*Agregue dentro de `myfifo.h` el prototipo de la función `parse()`.
*Cambie los prototipos de las funciones `proceso()` tanto en `tp66` (servidor) como en `tp67` (cliente).
*Implemente la función `proceso()` en `tp67` como:

```
// luego de parsing, proceso sucursal y total de venta recibido
void proceso(char *s,char *v) {
   int suc=atoi(s);
   double vta=atof(v);
   printf("proceso(): sucursal [%d] total $  %lf\n",suc,vta);
}
```

*Implemente la función `proceso()` en `tp66` como:

```
// luego de parsing, proceso id de proceso, id de sucursal
void proceso(char *sid,char *ssuc) {
   int id=atoi(sid),suc=atoi(ssuc);
   char fifo[MAX_BUFFER],buffer[MAX_BUFFER];
   snprintf(fifo,MAX_BUFFER,"%s.%d",MY_FIFO,id);
   printf("proceso(): id proceso [%d] sucursal [%d] total $  %lf
fifo:[%s]\n",id,suc,ventas[suc],fifo);
   snprintf(buffer,MAX_BUFFER,"<%d/%lf>",suc,ventas[suc]);
   int fdo = open(fifo,O_RDWR,0777);
   write(fdo,buffer,strlen(buffer));
   close(fdo);
}
```

Entonces, `parse()` es compartida, esta en `myfifo.c`, su prototipo esta en `myfifo.h`, es distribuible.
Entonces, `proceso()` es local, propia de cada programa server o cliente, esta en `tp66.c` y en `tp67.c`. No hace falta que se llame `proceso()` en ambos programas, puede tener cualquier nombre nombre, sólo debe tener el prototipo que requiere `parse()`.

*Modifique `tp66.c` y `tp67.c` para que hagan la llamada a la función `parse()` de la forma indicada previamente.

*Compile cliente y servidor como se indicó en el punto anterior. Ejecute server, cliente, pruebe el sistema, podrá obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/fifo/v2$ ./tp66 &
[1] 2877
grchere@debian:~/gccwork/src/fifo/v2$ main():inicio servidor FIFO!
main():para salir envie se?al SIGUSR2 a proceso 2877
main():inicializo total de ventas de sucursales
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255

grchere@debian:~/gccwork/src/fifo/v2$
grchere@debian:~/gccwork/src/fifo/v2$ main():lei: [<2883/0>] de FIFO
[/tmp/myfifo]
parse(): campo1 [2883] campo2 [0]
proceso(): id proceso [2883] sucursal [0] total $  127.815824
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/1>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [1]
proceso(): id proceso [2883] sucursal [1] total $  124.333209
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/2>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [2]
proceso(): id proceso [2883] sucursal [2] total $  104.190440
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/3>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [3]
proceso(): id proceso [2883] sucursal [3] total $  9.342347
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/4>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [4]
proceso(): id proceso [2883] sucursal [4] total $  25.764340
```

```
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/5>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [5]
proceso(): id proceso [2883] sucursal [5] total $  102.006486
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/6>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [6]
proceso(): id proceso [2883] sucursal [6] total $  124.583868
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/7>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [7]
proceso(): id proceso [2883] sucursal [7] total $  89.308833
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/8>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [8]
proceso(): id proceso [2883] sucursal [8] total $  172.868113
fifo:[/tmp/myfifo.2883]
main():lei: [<2883/9>] de FIFO [/tmp/myfifo]
parse(): campo1 [2883] campo2 [9]
proceso(): id proceso [2883] sucursal [9] total $  76.690872
fifo:[/tmp/myfifo.2883]
main():recibi se?al de salida
main():cierro FIFO server [/tmp/myfifo]
main():elimino FIFO server [/tmp/myfifo]
main():fin servidor FIFO! retorno=0

[1]+  Done                    ./tp66
grchere@debian:~/gccwork/src/fifo/v2$
```

consola cliente:

```
grchere@debian:~/gccwork/src/fifo/v2$ ./tp67
main():inicio cliente FIFO!
main():creando FIFO [/tmp/myfifo.2883]
main():abro FIFO [/tmp/myfifo] para grabar
main():abro FIFO [/tmp/myfifo.2883] para leer y grabar
main():envie 8 bytes a servidor [<2883/0>]
parse(): campo1 [0] campo2 [127.815824]
proceso(): sucursal [0] total $  127.815824
main():envie 8 bytes a servidor [<2883/1>]
parse(): campo1 [1] campo2 [124.333209]
proceso(): sucursal [1] total $  124.333209
main():envie 8 bytes a servidor [<2883/2>]
parse(): campo1 [2] campo2 [104.190440]
proceso(): sucursal [2] total $  104.190440
main():envie 8 bytes a servidor [<2883/3>]
parse(): campo1 [3] campo2 [9.342347]
proceso(): sucursal [3] total $  9.342347
main():envie 8 bytes a servidor [<2883/4>]
parse(): campo1 [4] campo2 [25.764340]
proceso(): sucursal [4] total $  25.764340
main():envie 8 bytes a servidor [<2883/5>]
parse(): campo1 [5] campo2 [102.006486]
proceso(): sucursal [5] total $  102.006486
main():envie 8 bytes a servidor [<2883/6>]
parse(): campo1 [6] campo2 [124.583868]
proceso(): sucursal [6] total $  124.583868
main():envie 8 bytes a servidor [<2883/7>]
parse(): campo1 [7] campo2 [89.308833]
proceso(): sucursal [7] total $  89.308833
```

```
main():envie 8 bytes a servidor [<2883/8>]
parse(): campo1 [8] campo2 [172.868113]
proceso(): sucursal [8] total $  172.868113
main():envie 8 bytes a servidor [<2883/9>]
parse(): campo1 [9] campo2 [76.690872]
proceso(): sucursal [9] total $  76.690872
main():cierro FIFO's
main():elimino FIFO cliente [/tmp/myfifo.2883]
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v2$
grchere@debian:~/gccwork/src/fifo/v2$ kill -SIGUSR2 2877
grchere@debian:~/gccwork/src/fifo/v2$
```

10. Analizando la biblioteca `myfifo.c` podemos observar que hay codigo repetitivo y poco reutilizable dentro de la funcion `parse()`, básicamente, ésta necesita el auxilio de otra función que pueda copiar un string que se encuentra dentro de otro string y delimitado por dos caracteres, se podría reutilizar una función tal como:

```
  char *strcpyentre(char *to,char *from,char desde,char hasta);
```

Esta función devuelve un puntero al desplazamiento ocurrido dentro de la cadena `from` a medida que se fue haciendo el *parsing*.

*Dentro de la carpeta `fifo`, crear la carpeta `v3` para almacenar la versión 3 de este proyecto.
*Dentro de `v3` copie todo el contenido actual de `v2`.
*Agregar el prototipo de `strcpyentre()` en `myfifo.h`, implementar la función `strcpyentre()` dentro de `myfifo.c`, para luego reutilizarla desde `parse()`, el código de la función `strcpyentre()` sería:

```
// to es un cadena que se encuentra dentro de from y esta delimitada
// por dos caracteres: desde y hasta
// si no encuentra la subcadena, devuelve una cadena vacia (inicializada a NULL)
// to es una cadena que tiene previamente asignado -al menos-
// strlen(from)+1 caracteres.
// devuelve el desplazamiento realizado en from durante el parsing
char *strcpyentre(char *to,char *from,char desde,char hasta) {
   memset(to,0,strlen(from)+1);
   while( *from && *from != desde) from++;
   if (  *from == desde ) from++;
   while( *from && *from != hasta) *to++=*from++;
   return from;
}
```

dentro de `myfifo.c`, el código de la función `parse()` ahora pasaría a ser:

```
// analiza buffer recibido de los clientes para determinar id de proceso
// cliente y nro de sucursal solicitada
// formato de buffer: "<campo1/campo2><campo1/campo2>....<campo1/campo2>"
void parse(char *buffer,void (*proceso)(char *campo1,char *campo2)) {
   char id[MAX_BUFFER];
   char suc[MAX_BUFFER];
   while(*buffer) {
      buffer=strcpyentre(id,buffer,'<','/');
      buffer=strcpyentre(suc,buffer,'/','>');
```

```
      if ( *id && *suc ) { // algo obtuve
         // final de parsing, ¿que obtuve?
         //printf("parse(): campo1 [%s] campo2 [%s]\n",id,suc);  // trace
         (*proceso)(id,suc);
      }
   }
}
```

*El manejo de pipes está vinculado con la señal `SIGPIPE` (también lo están los sockets tcp). Si un proceso pretende grabar sobre un pipe en donde el lector de dicho pipe ha terminado, se generará esta señal. Implementar una función de manipulación de esta señal para que emita el mensaje de error correspondiente en caso de recibir esta señal. Implemente esta señal tanto en cliente como en servidor.
*Compile cliente y servidor como se indicó en el punto anterior. Ejecute server, cliente, pruebe el sistema, la salida a obtener sería la misma, pero con una librería más reutilizable y mejor estructurada!.
*Elimine todos los mensajes innecesarios enviados a la consola, recompile, una salida posible sería:

consola server:

```
grchere@debian:~/gccwork/src/fifo/v3$ ./tp66 &
[1] 2890
grchere@debian:~/gccwork/src/fifo/v3$ main():inicio servidor FIFO!
main():para salir envie se?al SIGUSR2 a proceso 2890
main():inicializo total de ventas de sucursales
main():Tama?o maximo de operaciones atomicas sobre PIPES: 4096
main():Tama?o maximo de buffer de servidor: 255
proceso(): proceso [2891] suc [0] total $ 97.467573 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [1] total $ 29.156740 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [2] total $ 182.218157 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [3] total $ 197.326834 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [4] total $ 41.050789 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [5] total $ 76.399383 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [6] total $ 149.939370 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [7] total $ 127.515767 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [8] total $ 104.658367 fifo [/tmp/myfifo.2891]
proceso(): proceso [2891] suc [9] total $ 142.723383 fifo [/tmp/myfifo.2891]
main():recibi se?al de salida
main():cierro FIFO server [/tmp/myfifo]
main():elimino FIFO server [/tmp/myfifo]
main():fin servidor FIFO! retorno=0

[1]+  Done                    ./tp66
grchere@debian:~/gccwork/src/fifo/v3$
```

consola cliente:

```
grchere@debian:~/gccwork/src/fifo/v3$ ./tp67
main():inicio cliente FIFO!
main():creando FIFO [/tmp/myfifo.2891]
main():abro FIFO [/tmp/myfifo] para grabar
main():abro FIFO [/tmp/myfifo.2891] para leer y grabar
proceso(): suc [0] total $  97.467573
proceso(): suc [1] total $  29.156740
proceso(): suc [2] total $  182.218157
proceso(): suc [3] total $  197.326834
proceso(): suc [4] total $  41.050789
proceso(): suc [5] total $  76.399383
```

```
proceso(): suc [6] total $  149.939370
proceso(): suc [7] total $  127.515767
proceso(): suc [8] total $  104.658367
proceso(): suc [9] total $  142.723383
main():cierro FIFO's
main():elimino FIFO cliente [/tmp/myfifo.2891]
main():fin cliente FIFO!
grchere@debian:~/gccwork/src/fifo/v3$ kill -SIGUSR2 2890
grchere@debian:~/gccwork/src/fifo/v3$
```

En cuanto a los FIFO's

Habrá notado que los FIFO's que pudieron haber sido abiertos de solo lectura se abrieron como R/W (read/write, lectura-escritura). Esto está en relación con el hecho de no obtener un EOF cuando el servidor ya no tenga ningún cliente que atender o bien cuando el cliente ya no tenga más mensajes que recuperar del servidor.

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 named pipes o FIFO's como otra forma de comunicacion entre procesos provista por el sistema operativo.