PRACTICAS VOLUNTARIAS

DE

SISTEMAS OPERATIVOS II

Asignatura Obligatoria de 3er Curso (2º cuatrimestre)
Ingeniería Técnica en Informática de Sistemas
U.N.E.D

Curso 2010-2011

INFORMACIÓN GENERAL

Carácter de las prácticas

Las prácticas de la asignatura Sistemas Operativos II son **voluntarias**. Por ello, se recuerda al alumno/a la obligación e importancia de hacer las prácticas por si mismo **sin copiarlas** de otros compañeros, ya que ello repercutirá en perjuicio del propio alumno/a que no adquirirá el grado formativo adecuado.

Con el objetivo de no malgastar el tiempo de estudio de esta asignatura, se recomienda al alumno/a que intente hacer estas prácticas voluntarias siempre y cuando cumpla las siguientes condiciones:

- Se haya dado ya un **primer repaso** a los temas de la asignatura en los que se basa cada práctica y no haya tenido muchos problemas para entenderlos.
- Disponga de **tiempo suficiente** para realizarlas.

Objetivo de las prácticas

El objetivo de estas prácticas es que el alumno/a desarrolle su habilidad en el uso del **lenguaje C** aplicado a la creación de programas para el sistema operativo **Linux**. Además también se pretende que use de forma práctica algunos de los aspectos teóricos tratados en los apuntes de la asignatura.

Requisitos técnicos

Para poder realizar estas prácticas el alumno debe disponer de cualquiera de las múltiples distribuciones del **sistema operativo Linux**, y tener instalado en la misma un **compilador de lenguaje C**, como por ejemplo `gcc`.

Forma de entregar las prácticas

El alumno/a debe enviar los siguientes ficheros asociados con cada práctica:

* *Practica1*: `practica1.c explico_p1.pdf`
* *Practica 2*: `practica2.c explico_p2.pdf`
* *Practica 3*: `comun.h servidor.c cliente.c explico_p3.pdf`

Además debe incluir un fichero que lleve por nombre `alumno.txt` en el que debe incluir la siguiente información: *Nombre*, *apellidos*, *DNI*, *centro asociado en el que se ha matriculado*, y *teléfono de contacto*.

Estos 8 ficheros se deberán comprimir en un fichero ZIP que lleve por nombre

`SO2_[número del DNI del alumno].zip`.

Por ejemplo:

`SO2_32515294Z.zip`

Este fichero ZIP debe ser enviado a la siguiente dirección electrónica:

so2@dia.uned.es

En el tema del mensaje se debe poner PRACTICAS DE SO2, y en el cuerpo del mensaje se deben escribir el *nombre*, *apellidos*, *DNI*, *centro asociado en el que se ha matriculado* y *teléfono de contacto*.

A los pocos días recibirá un e-mail confirmándole que sus prácticas han sido recibidas. Sino recibe dicho mensaje no dude en ponerse en contacto con el equipo docente.

Fecha de entrega de las prácticas

El **último día** para entregar las prácticas es el **18 de mayo de 2011**. Esta fecha es **improrrogable**.

Evaluación de las prácticas

La nota de las prácticas son **Apto** o **No apto.** La correcta realización de las mismas (calificación de Apto) se verá reflejada en una **mejora de la nota** obtenida en el examen, siempre y cuando el alumno/a **tenga aprobado el examen**

¡¡Aviso importante!!, el equipo docente se reserva el derecho de ponerse en contacto con el alumno/a y realizarle diferentes cuestiones relativas a las practicas para verificar que efectivamente es el autor de las mismas y no las ha copiado. Si dicha verificación no fuese satisfactoria se bajará a modo de penalización la nota obtenida en el examen.

Consulta dudas relativas a las prácticas

La consulta de dudas relativas a las prácticas puede realizarse a través de la dirección de correo electrónico

so2@dia.uned.es

PRACTICA 1

OBTENCIÓN DE LA MASCARA DE MODO SIMBÓLICA DE UN FICHERO

1.1 OBJETIVOS

El objetivo de esta práctica es practicar con algunos de los conceptos teóricos estudiados en los Temas 2 y 3 de la asignatura relativos a ficheros tales como tipos, permisos, máscara de modo y máscara de modo simbólica

El alumno/a deberá escribir el programa en C `practica1.c` que haga todo lo que se pide en el enunciado. La explicación detallada de dicho programa se debe realizar en un fichero aparte que lleve por nombre `explico_p1.pdf`.

1.2 ENUNCIADO DE LA PRÁCTICA

Escribir un programa en C de nombre `practica1.c` que sirva para visualizar en pantalla la máscara de modo simbólica de uno o varios ficheros. El programa para cada fichero especificado como argumento (bien como una ruta absoluta o relativa), debe consultar la máscara de modo que hay en el nodo-i del fichero y generar a partir de dicha información la máscara de modo simbólica.

El programa ejecutable que se compile a partir de `practica1.c` se debe llamar `mask_sim`. A continuación se muestran algunos ejemplos de la funcionalidad que debe tener este programa:

1) Supóngase que `prueba` es un directorio cuya máscara simbólica es `drwxr-xr-x`, entonces la orden `$ mask_sim prueba` debe mostrar en pantalla la salida `prueba: drwxr-xr-x`

2) Supóngase que `fichero` es un fichero ordinario cuya máscara simbólica es `-rws--srwt`, entonces la orden `$ mask_sim fichero` debe mostrar en pantalla la salida `fichero: -rws--srwt`

3) Supóngase que `apunta` es un enlace simbólico cuya máscara simbólica es `lrwxrwxrwx`, entonces la orden `$ mask_sim apunta` debe mostrar en pantalla la salida `apunta: lrwxrwxrwx`

4) Supóngase que `/dev/cua0` es un fichero de dispositivo modo carácter cuya máscara simbólica es `crw-rw-rw-`, que `/dev/fd0` es un fichero de dispositivo modo bloque cuya máscara simbólica es `brw-rw-rw-`, que `/dev/log` es un conector cuya máscara simbólica es `srw-rw-rw-` y que `/dev/printer` es una tubería con nombre cuya máscara simbólica es `prwxrwxrwx`, entonces la orden

```
$ mask_sim /dev/cua0 /dev/fd0 /dev/log /dev/printer
```

debe mostrar en pantalla la salida

```
/dev/cua0: crw-rw-rw-
/dev/fd0: brw-rw-rw-
/dev/log: srw-rw-rw-
/dev/printer: prwxrwxrwx
```

PRACTICA 2

CREACION Y SINCRONIZACION DE PROCESOS

2.1 OBJETIVOS

El objetivo de esta práctica es manejar algunos de los conceptos teóricos aprendidos en en el Tema 5 de los apuntes de la asignatura, relativos a la creación de procesos (uso de la llamada al sistema `fork`) y comunicación y sincronización de procesos mediante el usos de señales (uso de las llamadas al sistema `signal`, `kill`, `sleep`, `pause`, etc, ..).

El alumno/a deberá escribir el programa en C `practica2.c` que haga todo lo que se pide en el enunciado. La explicación detallada de dicho programa se debe realizar en un fichero que lleve por nombre `explico_p2.pdf`.

2.2 ENUNCIADO DE LA PRACTICA

Escribir un programa en C de nombre `practica2.c` y cuyo ejecutable se llame `creasin` que reciba a través de la línea de ordenes de la consola de un sistema Linux el número n de procesos hijo que debe crear ($1 \leq n \leq 15$). La forma de llamar al programa debe ser:

```
$ creasin n
```

El programa debe realizar las siguientes acciones:

A) *Creación de procesos.* El primer proceso o proceso patriarca (i=0) debe encargarse de crear un proceso hijo (i=1). A su vez el proceso (i=1) debe crear un proceso hijo (i=2), que será el nieto del proceso (i=0). El proceso (i=2) debe crear un proceso hijo (i=3), que será nieto del proceso (i=1) y biznieto del proceso (i=0). La creación de procesos se debe detener cuando se hayan creado un total de n+1 procesos (el proceso patriarca i=0 más sus n descendientes).

B) *Creación de un primer punto de sincronización.* Una vez que todos los n+1 procesos han sido creados empezarán a comunicarse entre si. Para ello es necesario introducir un primer punto de sincronización mediante el cual los procesos que se van a comunicar indican que están disponibles para iniciar la comunicación. Para esta primera sincronización cada proceso debe realizar las siguientes operaciones:

1) Esperar a que su proceso hijo le comunique que ha sido creado.

2) Transcurrido un segundo (llamada al sistema `sleep(1);`) comunicarle a su proceso padre que él ya ha sido creado.

Para realizar esta comunicación, cada proceso va a esperar la recepción de la señal SIGUSR1 y una vez recibida y se la va a enviar a su proceso padre. La señal SIGUSR1 actúa a modo de testigo único que se intercambian los distintos procesos.

C) *Comunicación y sincronismo entre procesos*. Una vez realizado el apartado b) todos los procesos están creados y listos para iniciar la comunicación, la cual va a ser iniciada por el proceso patriarca (i=0). La secuencia de sincronismo que debe realizar cada proceso es:

1) Esperar a que su proceso padre le envíe la señal SIGUSR1.

2) Una vez recibida la señal SIGUSR1 y transcurrido un segundo enviársela a su proceso hijo para pasarle el turno.

En esta secuencia hay dos excepciones:

1) El proceso patriarca (i=0) debe recibir la señal desde el proceso hijo final (i=n).

2) El último proceso hijo (i=n) no tiene descendencia por lo tanto le pasará la señal al proceso patriarca (i=0).

Se establece así una trayectoria circular en el paso de la señal que actúa de testigo, donde el último proceso creado se la pasa al primero. Este circuito se debe repetir un total de *tres veces* al final de las cuales se iniciará una secuencia de terminación.

D) *Secuencia de terminación*. La terminación se debe realizar en orden inverso al de creación. Así, el último proceso creado (i=n) debe ser el primero en terminar. La secuencia de sincronismo que se debe ejecutar es:

1) Cuando el ciclo de sincronismo del apartado c) se haya realizado tres veces, el proceso patriarca (i=0) le enviará la señal SIGTERM a su proceso hijo (i=1).

2) Cada proceso hijo debe esperar la recepción de la señal SIGTERM procedente de su proceso padre.

3) Una vez que reciba esta señal y transcurrido un segundo, se la enviará a su proceso hijo.

Con la secuencia anterior se consigue que la orden de terminar se transmita desde el proceso patriarca (i=0) hasta el hijo final (i=n), atravesando toda la estructura de procesos creada.

Cuando el último proceso hijo recibe la señal SIGTERM, se encarga de devolvérsela a su proceso padre, iniciándose así un retorno de la señal en sentido contrario. La secuencia de sincronismo que debe ejecutar cada proceso es:

1) Esperar a recibir la señal SIGTERM.

2) Una vez recibida la señal SIGTERM y transcurrido un segundo, enviársela a su proceso padre.

3) Terminar su ejecución con una llamada a `exit`.

Hay que tener cuidado de que el proceso patriarca (i=0) no le envíe la señal SIGTERM a su proceso padre, ya que al ser éste el interprete de órdenes, el resultado que se obtienen es la terminación de la sesión de trabajo.

Una vez se haya terminado de ejecutar `creasin` el usuario debe comprobar que no existe ningún proceso de los creados vivos, es decir, se realizado la terminación de los procesos correctamente. Para ello basta teclear desde la línea de ordenes el comando `ps`.

2.3 AYUDAS

2.3.1 Ayuda 1

Como se puede observar, en la transmisión de las señales SIGUSR1 y SIGTERM se pueden distinguir dos sentidos en el flujo de las mismas:

- Desde los procesos padre a los hijos. Este flujo está asociado a la transmisión de ordenes.

- Desde los procesos hijo a los padre. Este flujo está asociado a la comunicación de la aceptación de la orden.

A la hora de contemplar estos dos sentidos en el fluir de las señales resulta bastante cómodo utilizar dos manejadores distintos por cada señal. Uno de los manejadores se activa cuando las señales viajan en un sentido y el otro cuando las señales viajan en sentido opuesto.

Pero la pregunta que surge es: ¿Cómo distinguir un sentido de otro ?. Esa distinción la deben realizar los propios manejadores y está determinada por el enunciado del problema. A modo de ejemplo, vamos a ver cómo se realiza esta distinción para la señal SIGTERM

El primer flujo de esta señal tiene como origen los procesos padre y como destino los procesos hijo. Por lo tanto, al crear los procesos se va a instalar un manejador (`manejador_1`) que realice las siguientes operaciones:

1) Instalar el segundo manejador (`manejador_2`) de la señal SIGTERM.

2) Enviar la señal SIGTERM a su proceso hijo.

3) Enviar la señal a su proceso padre. Esto sólo lo realiza el último proceso hijo que es quién debe encargarse de hacer que la señal inicie su camino de vuelta.

Se puede observar que la primera acción del `manejador_1` es instalar el `manejador_2`. Esto se hace porque sabemos que la próxima vez que se reciba la señal, el sentido de su viaje será el contrario. El `manejador_2` realiza las siguientes operaciones:

1) Enviar la señal SIGTERM a su proceso padre.

2) Terminar la ejecución del proceso mediante una llamada `exit`.

Para determinar las acciones que deben realizar los manejadores de la señal SIGUSR1 habrá que realizar un análisis parecido.

2.3.2 Ayuda 2

Es importante asegurarse de que los procesos permanecen vivos ya que de otra forma no podrán responder al envío de una señal por parte de otro proceso y el programa no funcionará adecuadamente.

2.3.3 Ayuda 3

A continuación se puede muestra a modo de ejemplo la salida en pantalla que produce la ejecución del programa `creasin` con n=3. (Nota: Obviamente los valores que se muestran de los pid de los procesos dependerán de cada ejecución del programa.)

```
$ creasin 3

*** PARTE A: Creación de procesos ***
Creado proceso hijo nº 1 (pid= 532, pid_proceso_padre=531)
Creado proceso hijo nº 2 (pid= 533, pid_proceso_padre=532)
Creado proceso hijo nº 3 (pid= 534, pid_proceso_padre=533)

*** PARTE B: Creación de un punto de sincronismo ***
Proceso hijo nº 3 (pid= 534) preparado
Proceso hijo nº 2 (pid= 533) preparado
Proceso hijo nº 1 (pid= 532) preparado
Proceso patriarca (pid= 531) preparado

*** PARTE C: Comunicación y sincronismo ***
Proceso patriarca. Contador=3
Proceso hijo nº 1
Proceso hijo nº 2
Proceso hijo nº 3
Proceso patriarca. Contador=2
Proceso hijo nº 1
Proceso hijo nº 2
Proceso hijo nº 3
Proceso patriarca. Contador=1
Proceso hijo nº 1
Proceso hijo nº 2
Proceso hijo nº 3
Proceso patriarca. Contador=0

*** PARTE D: Secuencia de terminación ***
Patriarca envía la señal SIGTERM al Hijo nº 1.
```

```
Hijo nº 1 recibe y envía la señal SIGTERM al Hijo nº 2.
Hijo nº 2 recibe y envía la señal SIGTERM al Hijo nº 3.
Hijo nº 3 recibe la señal SIGTERM.

Proceso hijo nº 3 terminado.
Proceso hijo nº 2 terminado.
Proceso hijo nº 1 terminado.
Proceso patriarca terminado.
```

PRACTICA 3

APLICACIÓN DE LAS COLAS DE MENSAJES: ARQUITECTURA CLIENTE/SERVIDOR

3.1 OBJETIVOS

El objetivo de esta práctica es manejar algunos de los conceptos teóricos aprendidos en el Tema 7 de la asignatura, relativos a los mecanismos IPC. En concreto se practicará con el uso de las colas de mensajes.

El alumno/a deberá escribir tres programas en C: `comun.h`, `servidor.c` y `cliente.c`. La explicación detallada de los programas implementados se debe realizar en un fichero que lleve por nombre `explico_p3.pdf`.

3.2 ENUNCIADO DE LA PRACTICA

Las arquitecturas cliente/servidor son actualmente muy utilizadas en el desarrollo de aplicaciones distribuidas. En esta practica se propone implementar una arquitectura de este tipo mediante colas de mensajes. Básicamente, se trata de desarrollar un programa que acepta peticiones por parte de otros programas; aquel que atiende las peticiones es el *servidor*, y el que realiza las peticiones es el *cliente*.

Si se quiere realizar una arquitectura cliente/servidor mediante colas de mensajes, se necesitan dos colas de mensajes (ver Figura 3.1):

- *Cola Cliente - Servidor* (CCS) para canalizar el paso de la información del cliente al servidor.
- *Cola Servidor - Cliente* (CSC) para canalizar el paso de la información del servidor al cliente.

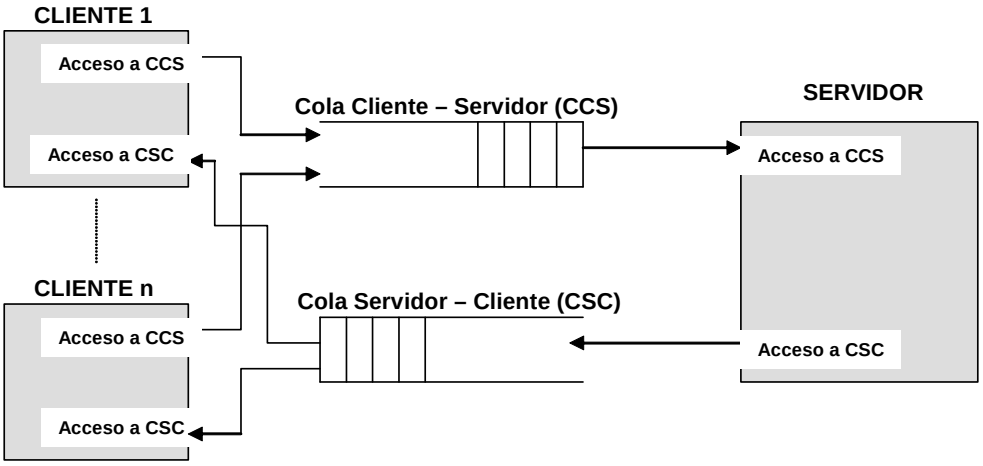

Figura 3.1: Arquitectura Cliente/Servidor mediante colas de mensajes

De esta manera, el servidor no gestionará su propia información y los clientes no escucharán sus propios mensajes. Esto es similar a realizar una comunicación bidireccional mediante el uso de tuberías.

Para poder desarrollar el sistema compuesto por dos procesos independientes (el servidor y el cliente), éstos deben compartir el tipo de datos que van a enviar/recibir por la cola de mensajes, el nombre de las colas y el conjunto de órdenes y estados. Esto configura el nivel de aplicación del *protocolo de comunicación*. Los mensajes que se intercambian deben incluir un tipo que permita identificar qué proceso es el que se ha conectado al servidor y realiza las peticiones, así como qué proceso debe recibir la información que proporciona el servidor. Este tipo va a ser el PID del proceso.

Teniendo en cuenta esto, se pretende desarrollar una aplicación con una estructura de cliente/servidor que gestione un listado de datos sobre los alumnos/as de una clase. En los siguientes subapartados se detallan las características de los tres programas que hay que desarrollar.

3.2.1 Implementación del protocolo de comunicación.

El protocolo de comunicación se debe implementar a través del archivo de cabecera `comun.h`. Este archivo debe contener las definiciones de las variables y de las constantes comunes, es decir, aquellas que vayan a ser utilizadas tanto por el programa que implemente al servidor como por el que implemente al cliente.

Como mínimo este archivo debe contener las siguientes constantes:

- Las relativas a las ordenes que se van a transmitir entre los clientes y el servidor. Existen cuatro posibles ordenes: PEDIR_DATOS, INSERTAR_DATO, FIN_DE_COMUNICACIÓN y ERROR.

- La longitud en bytes de un mensaje (LON_MENSAJE).

- El nombre del fichero (FICHERO) que se va utilizar en la creación de las llaves asociadas a CSC y CCS.

- Las claves de acceso (CLAVE_CCS y CLAVE_CSC) que se usarán en la creación de las llaves asociadas a CSC y CCS.

Además como mínimo este archivo debe contener las siguientes definiciones de estructuras:

- La *estructura de un registro de datos de un alumno/a*., que debe constar de un identificador numérico, el nombre, los apellidos y la nota. El identificador numérico debe ser un número entero positivo (comprendido entre 1 y 100) y la nota un número decimal (comprendido entre 0.0 y 10.0).

- La *estructura de un mensaje*, debe contener el pid del proceso, la orden que se transmite y un registro con los datos de un alumno/a. El tipo del mensaje será el pid del proceso y el cuerpo estará formado por la orden que se transmite y el registro con los datos de un alumno/a. Para la definición del pid del proceso se recomienda usar un número entero del tipo `long`.

3.2.2 Implementación del Servidor.

El servidor se debe implementar a través del programa `servidor.c`. El servidor trabajará con el fichero sin formato `clase.dat` que contendrá los registros con los datos de los alumnos de una clase.

Las acciones que debe realizar el servidor son:

1) Abrir las colas CCS y CSC.

2) Implementar la etapa de comunicación con los clientes:

2.1) Debe examinar continuamente la CCS para detectar la presencia de mensajes enviados por los clientes. Se extraerá siempre el primer mensaje que haya independientemente de su tipo.

2.2) El servidor realiza la acción que le especifica el cliente en el mensaje.

- Si la orden es PEDIR_DATOS entonces el servidor enviará a la cola CSC tantos mensajes como registros existan en el fichero `clase.dat`.. Cada mensaje enviado por el servidor contendrá por tanto la siguiente información:
  - Tipo: el pid del proceso cliente que ha solicitado la información.
  - Orden: PEDIR_DATOS
  - Un registro con los datos de un alumno/a leído en el fichero `clase.dat`.

  Se debe tratar adecuadamente el caso en que el fichero `clase.dat` todavía no haya sido creado.

- Si la orden es INSERTAR_DATO entonces el servidor añadirá un registro al fichero `clase.dat` con los datos del alumno que se especifiquen en el mensaje extraído de la CCS. Además enviará un mensaje a la CSC con la siguiente información:
  - Tipo: el pid del proceso cliente que ha dado la orden de insertar el dato.
  - Orden: INSERTAR_DATO

- Si la orden es FIN_DE_COMUNICACIÓN entonces el servidor cierra las colas CCS y CSC, muestra por pantalla el texto "`Servidor cerrado`" y el programa finaliza.

- Si la orden no es ninguna de las anteriores, entonces el servidor enviará un mensaje a la CSC con la siguiente información:
  - Tipo: el pid del proceso cliente.

- Orden: ERROR

2.3) Cuando el servidor finaliza de atender una orden del cliente del tipo PEDIR_DATOS o INSERTAR_DATOS, cierra el fichero `clase.dat` y envía además un mensaje a la cola CSC con la siguiente información:

- Tipo: el pid del proceso cliente que ha dado la orden.
- Orden: FIN_DE_COMUNICACION

3.2.3 Implementación del Cliente.

El cliente se debe implementar a través del programa `cliente.c`. Las acciones que debe realizar este programa son:

1) Abrir las colas CCS y CSC.

2) Presentar por pantalla un menú con las cuatro opciones disponibles: insertar dato en el servidor, pedir datos al servidor, cerrar el servidor y salir del programa. El usuario debe elegir una.

- Si la orden seleccionada es “Insertar dato en el servidor” entonces el usuario debe introducir un registro de datos de un alumno/a, es decir, el identificador numérico, el nombre, los apellidos y la nota. A continuación se enviará un mensaje a la CCS con la siguiente información:
  - Tipo: el pid del proceso cliente.
  - Orden: INSERTAR_DATO
  - El registro de datos introducido por el usuario.
- Si la orden seleccionada es “Pedir datos en el servidor” entonces se enviará un mensaje a la CCS con la siguiente información:
  - Tipo: el pid del proceso cliente.
  - Orden: PEDIR_DATOS
- Si la orden seleccionada es “Cerrar el servidor” entonces se enviará un mensaje a la CCS con la siguiente información:

- Tipo: el pid del proceso cliente.
- Orden: FIN_DE_COMUNICACION

- Si la orden seleccionada es “Salir del programa” entonces se escribe en la pantalla el texto “`Cliente finalizado`”

3) Si la orden seleccionada ha sido “Insertar dato en el servidor” o “Pedir datos al servidor” entonces el cliente recibe información procedente del servidor, es decir debe leer la CSC en espera de la llegada mensajes. Se pueden dar las siguientes situaciones en función de la orden que especifique en el mensaje el servidor:

   - INSERTAR_DATO, el programa mostrará por pantalla el texto “`Registro de datos insertado con éxito`”.
   - PEDIR_DATOS, el programa mostrará por pantalla la información del registro de datos del alumno/a recibida en el mensaje.
   - FIN_DE_COMUNICACIÓN, el programa mostrará por pantalla el texto “`Fin de la operación`”.
   - ERROR, el programa mostrará por pantalla el texto “`Error de comunicación`”.
   - Si la orden no es ninguna de las anteriores el programa mostrará por pantalla el texto “`Mensaje desconocido`”.

4) Salvo en el caso en que el usuario haya seleccionado salir del programa, una vez finalizada una operación se volverá a mostrar el menú de opciones para que el usuario pueda insertar una nueva orden. Debe tratarse adecuadamente el caso en que el servidor haya sido ya cerrado.

3.3 AYUDA

3.3.1 Ayuda 1

El fichero de cabecera `comun.h` debe comenzar con las siguientes sentencias:

```
#ifndef _COMUN_
#define _COMUN_
```

Y finalizar con la sentencia:

```
#endif
```

Por otra parte, este fichero de cabecera se llamará en el programa del servidor y del cliente usando la sentencia:

```
#include “comun.h”
```

3.3.2 Ayuda 2

En el archivo de cabecera `comun.h`, la definición de la constante LON_MENSAJE debe expresar el número de bytes que ocupa únicamente el cuerpo del mensaje (orden y registro de datos de un alumno). Además recuerde que para determinar el tamaño en bytes de una variable o de un tipo de datos puede usar el operador `sizeof()`.

3.3.3 Ayuda 3

Los ficheros de cabecera necesarios para poder hacer uso de las llamadas al sistema relativas al mecanismo IPC de colas de mensajes son:

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

3.3.4 Ayuda 4

Para simplificar los programas a realizar se deben tener en cuenta las siguientes consideraciones:

1) Los nombres de los alumnos/as se considerarán que son simples: es decir, Pedro, Juan, etc. Si se desea introducir un nombre compuesto se hará de la siguiente forma: Juan_Manuel.

2) Se consideran únicamente el primer y el segundo apellido de cada alumno/a, además se supondrá que son apellidos simples (ver 1).

3) Cuando se dé la orden PEDIR_DATOS estos se mostrarán en el orden en que fueron insertados. Es decir, no es necesario ordenarlos ni alfabéticamente ni por su número identificador.

3.3.5 Ayuda 5

Supóngase que `server` y `client` son los archivos ejecutables asociados a los programas `servidor.c` y `cliente.c`, respectivamente. Supóngase además que el archivo `clase.dat` con los registros de datos de los alumnos/as de una clase no ha sido creado todavía.

A continuación se va presentar una posible traza de ejecución de estos archivos ejecutables ($ es el prompt del sistema):

```
$ server &
$ client

********OPCIONES DISPONIBLES*********************
     - Insertar datos                 (Pulsar i)
     - Pedir datos al servidor        (Pulsar p)
     - Cerrar el servidor             (Pulsar c)
     - Salir de esta aplicación       (Pulsar s)
*********************************************

Pulse una opción (i, p, c o s):
p

Servidor: No puedo abrir el fichero clase.dat

Pulse una opción (i, p, c o s):
i

***Insertar registro **
Identificador: 1
Nombre: Pedro
Apellido1: Martinez
Apellido2: Morales
Nota: 6.9

Registro de datos insertado con éxito.
Fin de la operación.

Pulse una opción (i, p, c o s):
i

***Insertar registro **
Identificador: 1
Nombre: Julian
Apellido1: Cao
Apellido2: García
Nota: 9.5
```

```
Registro de datos insertado con éxito.
Fin de la operación.

Pulse una opción (i, p, c o s):
s

Cliente finalizado
$
$client

********OPCIONES DISPONIBLES*********************
     - Insertar datos                 (Pulsar i)
     - Pedir datos al servidor        (Pulsar p)
     - Cerrar el servidor             (Pulsar c)
     - Salir de esta aplicación       (Pulsar s)
*************************************************

Pulse una opción (i, p, c o s):
p

  1  Pedro Martinez Morales   6.9
  25 Juan  Cao      García    9.5
  Fin de la operación

Pulse una opción (i, p, c o s):
c

Servidor cerrado

Pulse una opción (i, p, c o s):
s

Cliente finalizado


$
```