Índice

1. Enunciado

Segundo Proyecto

75.59 - Técnicas de Programación Concurrente I

Objetivo

Para el segundo proyecto se deberá realizar la implementación del ejercicio 6 de la guía de Mensajes, cuyo enunciado se copia a continuación:

> Escribir dos programas tales que uno se comportará como un gestor de una base de datos compuesta por una tabla de personas (`char(61) nombre, char(120) direccion, char(13) telefono`);
> El otro programa es un cliente de la base de datos. Este cliente puede leer el contenido de la base de datos y también puede agregar nuevos registros. Se deberá utilizar cola de mensajes para comunicar el programa cliente con el gestor de la base.

Como condiciones adicionales a las planteadas por el ejercicio, se deberán cumplir las siguientes:

1. La base de datos se deberá persistir en almacenamiento permanente (archivo) cuando el gestor se cierra.
2. Cuando el gestor se inicia, la base de datos se deberá leer desde el almacenamiento permanente a una memoria compartida, donde será manipulada por el gestor.
3. Se deberán poder ejecutar simultáneamente dos clientes que trabajen contra el gestor.

Requerimientos no Funcionales

Los siguientes son los requerimientos no funcionales de la aplicación:

1. El proyecto deberá ser desarrollado en lenguaje C o C++, siendo este último el lenguaje de preferencia.
2. Los clientes y el gestor de base de datos pueden no tener interfaz gráfica y ejecutarse en una o varias consolas de línea de comandos.
3. El proyecto deberá funcionar en ambiente Unix / Linux.
4. La aplicación deberá funcionar en una única computadora.

Tareas a Realizar

A continuación se listan las tareas a realizar para completar el desarrollo del proyecto:

1. Dividir el proyecto en procesos.
2. Una vez obtenida la división en procesos, establecer un esquema de comunicación entre ellos teniendo en cuenta los requerimientos de la aplicación. ¿Qué procesos se comunican entre sí? ¿Qué datos necesitan compartir para poder trabajar?

3. Diseñar y documentar el protocolo de comunicación que utilizarán los clientes con el programa gestor de la base de datos.
4. Realizar la codificación de la aplicación. El código fuente debe estar documentado.

Informe

Junto con el proyecto se deberá entregar un informe. El informe se deberá presentar en una carpeta o folio con el siguiente contenido:

1. Informe propiamente dicho
2. CD con el código fuente de la aplicación

El informe a entregar junto con el proyecto debe contener los siguientes ítems:

1. Breve análisis de problema, incluyendo una especificación de los casos de uso de la aplicación.
2. Detalle de resolución de la lista de tareas anterior.
3. Diagramas de clases de un cliente y del gestor de base de datos.

2. Requerimientos Funcionales

El sistema desarrollado permite realizar el alta, baja y modificación de los datos de una persona en la base de datos. El mismo cuenta con dos aplicaciones diferentes; una corresponde al cliente y otra al gestor de la base de datos.

3. Hipótesis

- Se utilizó como clave primaria de la base de datos el campo nombre de cada registro. Es por este motivo, que se supone que si dos personas se llaman de igual forma, son la misma persona.
- Se supone que el gestor siempre va a tener lugar disponible en la cola de mensajes, para enviar la respuesta a una petición de algún cliente; ya que en esta versión solo se ejecutan simultáneamente dos clientes.

4. Consideraciones de Diseño

- Se decidió desarrollar las aplicaciones en lenguaje C++, para utilizar el encapsulamiento de las clases.
- Cada cliente se ejecuta en una consola distinta, al igual que el gestor.
- Los campos permiten el ingreso de cadenas separadas por espacios, de forma tal de considerar los nombres compuestos, el ingreso de apellido y nombre de las personas, entre otras cosas.
- Para el acceso a la memoria compartida se utilizó un semáforo, de forma tal de lograr sincronizar el mismo. Este semáforo es único para la base de datos en su totalidad.
- Para acceder al archivo utilizado en la persistencia de la base de datos se utiliza un lockfile
- Al modificar una persona desde la aplicación cliente se le solicita al usuario que ingrese primero el nombre de la misma tal cual figura en la base de datos, y luego se le solicita que ingrese la nueva dirección y teléfono.
- Antes de realizar alguna modificación en la base de datos, se pide la confirmación del usuario, para evitar registrar datos erróneos.
- Se tomó como convención utilizar 100 registros por bloque de memoria. Dicho número puede alterarse modificando el valor de la constante MAX_REG_MEM en el archivo BloqueDeRegistros.h.

5. Descripción de la división en procesos

Para desarrollar las aplicaciones se decidió dividirlas de la siguiente forma:

- Para el caso de los clientes:

  - Cada cliente es un proceso.

- Para el caso del gestor:

  - Se dividió el mismo en dos procesos diferentes; uno de ellos espera la señal para finalizar el gestor mientras que el otro proceso es el que recibe, procesa y responde las consultas de los clientes.

6. Comunicación entre procesos

Para realizar la comunicación entre los procesos gestor y cliente se utilizó una *Cola de Mensajes*, dado que la misma permite comunicarlos aunque no posean relación entre ellos, siendo en este caso dos aplicaciones diferentes. La ventaja que posee la cola de mensajes es que la misma permite enviar estructuras de datos completas a través de ella; en lugar de bytes como en los fifos. Esto implica que no sea necesario sincronizar la escritura de varios procesos en la misma, dado que no puede suspenderse la escritura de dicha estructura antes de finalizar la misma. Es decir, no puede darse el caso de que por cambiar el ipc, el uso de cpu a otro proceso escriba uno la mitad de su estructura en la cola y el otro comience desde allí. Al mismo tiempo, la utilización de colas de mensajes simplifica el protocolo de comunicación entre los procesos, dado que es posible utilizar los campos de la estructura como por ejemplo, el dato long obligatorio para identificar que proceso es destinatario de cada uno de los mensajes. Esto último posibilita que la cola de mensajes sea bidireccional.

La estructura utilizada para transmitir los mensajes a través de la cola es la siguiente:

```
1  typedef struct {
2          long mtype;
3          pid_t id;
4          T_PETICION tipo;
5          Registro registro;
6          char respuesta[256];
7  } mensaje;
```

A continuación se presenta una descripción de cada uno de los campos que conforman dicha estructura.

- mtype: es un long obligatorio que se utiliza para indicar el destinatario del mensaje.
- id: es el número de proceso de quien envía el mensaje.
- tipo: es un enumerado que representa el tipo de petición del mensaje en cuestión. Los valores posibles para este campo son los siguientes: “AGREGAR, ELIMINAR, CONSULTAR, MODIFICAR”
- registro: es una estructura que contiene las cadenas de caracteres con los datos de la persona que se desea agregar, modificar, eliminar o sobre la que se desea consultar.
- respuesta: es una cadena de caracteres que se utiliza para enviar mensajes desde el gestor hacia los clientes, informandoles el resultado de la operación realizada.

6.1. Protocolo utilizado

El protocolo de comunicación utilizado se desprende de observar la estructura anterior. A continuación se describe el formato de los mensajes que se transmiten mediante la cola, de forma tal de aclarar cada uno de los tipos de mensajes utilizados para establecer la comunicación entre los procesos.
Para los mensajes que parten desde los clientes hacia el gestor se conforman de la siguiente manera:

- mtype = PETICION
- id = pid_del_proceso_que_envia
- tipo = AGREGAR ó ELIMINAR ó CONSULTAR ó MODIFICAR
- registro = Datos_de_la_persona_requeridos_por_la_operacion
- respuesta = “ ”

Para los mensajes que parten desde el gestor hacia los clientes se conforman de la siguiente manera:

- mtype = pid_del_cliente_que_debe_recibirlo
- id = “ ”
- tipo = “ ”
- registro = Datos_consultados_de_la_persona
- respuesta = mensaje_informando_el_resultado_de_la_operación

El gestor lee los mensajes que tengan como mtype el valor PETICION y los clientes leen los mensajes cuyo mtype corresponda al id de su proceso. De esta manera la cola de mensajes es bidireccional y cada cliente va a obtener la respuesta que le corresponda, ya que el pid es único para cada proceso.

7. Herramientas de concurrencia utilizadas

- Cola de mensajes: utilizada para resolver la comunicación entre los procesos cliente y gestor.

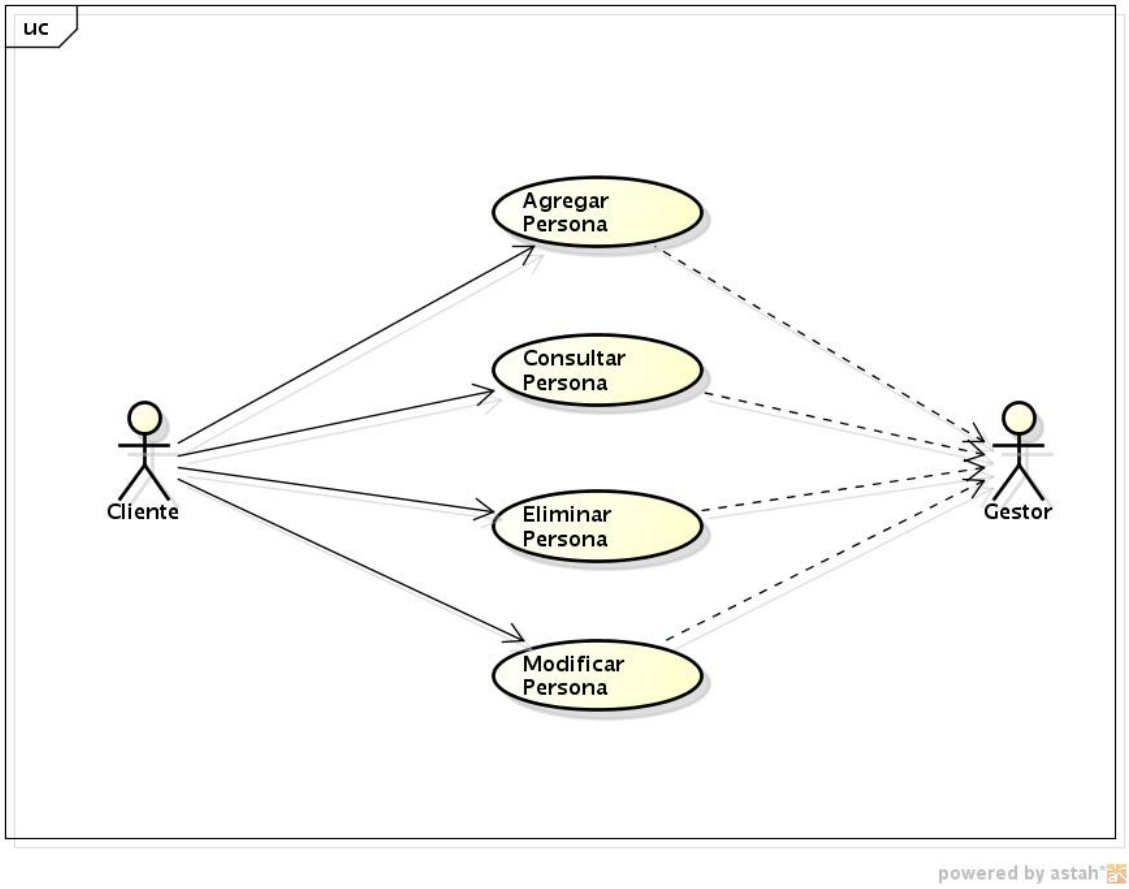
- Memoria compartida: utilizada para almacenar la base de datos. Permitiría, en caso de existir mas de un gestor, que todos los gestores compartan la información allí almacenada.

- Semáforo: utilizado para obtener sincronismo en el acceso a la memoria compartida que contiene la base de datos.

- LockFiles: utilizado para sincronizar la persistencia de la base de datos en un archivo, de forma tal de garantizar que solo un gestor acceda a este recurso a la vez, en caso de existir mas de uno.

- Señales: utilizada para detener la ejecución del proceso gestor que esta esperando una petición. Se utiliza la señal SIGINT en este caso para finalizar de forma correcta el gestor de base de datos.

8. Casos de Uso



8.0.1. Especificación de los casos de uso

Caso de uso: Agregar Persona	
Descripción: Alta de los datos de una persona en la base de datos	
Actores participantes: Cliente	
Pre-condiciones: Se deben haber ejecutado las aplicaciones cliente y gestor	
Flujo Principal	
1	El actor cliente ingresa a la aplicación.
2	El actor indica la acción a realizar Si la acción es el alta de una nueva persona entonces ejecutar subflujo Alta de persona (S1)
Flujos Alternativos	
S1	Alta de persona:
S1.1	El sistema solicita el ingreso del nombre de la persona a dar de alta.
S1.2	El cliente ingresa el nombre de la persona
S1.3	El sistema valida que el nombre de dicha persona no exista ya en la base de datos (E1)
S1.4	El sistema solicita el ingreso de la dirección de dicha persona
S1.5	El cliente ingresa la dirección de la persona
S1.6	El sistema solicita el ingreso del teléfono de dicha persona
S1.7	El cliente ingresa el teléfono de la persona.
S1.8	El sistema solicita que el usuario confirme que desea agregar esa persona.
S1.9	El cliente confirma su deseo de agregar dicha persona (E2).
S1.10	El sistema procesa el registro, arma la petición y se la envía al gestor.
S1.11	El sistema da de alta la persona en la base de datos.
S1.12	El caso de uso comienza nuevamente desde el paso 2.
Flujos de Excepción	
E1	Ya existe una persona con dicho nombre, se le informa al cliente y se vuelve al menú anterior. El caso de uso finaliza.
E2	El cliente no confirma su deseo de agregar a la persona, se vuelve al menú anterior. El caso de uso finaliza.
Post-condiciones: Se agregaron los datos de la nueva persona a la base de datos y desde este momento está disponible dicha información para el resto de los clientes	

Caso de uso: Modificar Persona	
Descripción: Modificación de los datos de una persona en la base de datos	
Actores participantes: Cliente	
Pre-condiciones: Se deben haber ejecutado las aplicaciones cliente y gestor	
Flujo Principal	
1	El actor cliente ingresa a la aplicación.
2	El actor indica la acción a realizar Si la acción es la modificación de los datos de una persona entonces ejecutar subflujo Modificación de persona (S1)
Flujos Alternativos	
S1	Modificación de persona:
S1.1	El sistema solicita el ingreso del nombre de la persona a modificar.
S1.2	El cliente ingresa el nombre de la persona
S1.3	El sistema valida que el nombre de dicha persona ya exista en la base de datos (E1)
S1.4	El sistema solicita el ingreso de la dirección de dicha persona
S1.5	El cliente ingresa la dirección de la persona
S1.6	El sistema solicita el ingreso del teléfono de dicha persona
S1.7	El cliente ingresa el teléfono de la persona.
S1.8	El sistema solicita que el usuario confirme que desea modificar los datos de esa persona.
S1.9	El cliente confirma su deseo de modificar los datos dicha persona (E2).
S1.10	El sistema procesa el registro, arma la petición y se la envía al gestor.
S1.11	El sistema modifica los datos de la persona en la base de datos.
S1.12	El caso de uso comienza nuevamente desde el paso 2.
Flujos de Excepción	
E1	Si no existe una persona con dicho nombre, se le informa al cliente y se vuelve al menú anterior. El caso de uso finaliza.
E2	El cliente no confirma su deseo de modificar los datos de la persona, se vuelve al menú anterior. El caso de uso finaliza.
Post-condiciones: Se modificaron los datos de la persona en la base de datos	

Caso de uso: Eliminar Persona	
Descripción: Baja de los datos de una persona en la base de datos	
Actores participantes: Cliente	
Pre-condiciones: Se deben haber ejecutado las aplicaciones cliente y gestor	
Flujo Principal	
1	El actor cliente ingresa a la aplicación.
2	El actor indica la acción a realizar Si la acción es la baja de una persona entonces ejecutar subflujo Baja de persona (S1)
Flujos Alternativos	
S1	Baja de persona:
S1.1	El sistema solicita el ingreso del nombre de la persona a dar de baja.
S1.2	El cliente ingresa el nombre de la persona
S1.3	El sistema valida que el nombre de dicha persona exista en la base de datos (E1)
S1.4	El sistema solicita que el usuario confirme que desea eliminar los datos de esa persona.
S1.5	El cliente confirma su deseo de modificar los datos dicha persona (E2).
S1.6	El sistema procesa el registro, arma la petición y se la envía al gestor.
S1.7	El sistema elimina los datos de la persona en la base de datos.
S1.8	El caso de uso comienza nuevamente desde el paso 2.
Flujos de Excepción	
E1	No existe una persona con dicho nombre, se le informa al cliente y se vuelve al menú anterior. El caso de uso finaliza.
E2	El cliente no confirma su deseo de modificar los datos de la persona, se vuelve al menú anterior. El caso de uso finaliza.
Post-condiciones: Se eliminaron los datos de la persona en la base de datos	

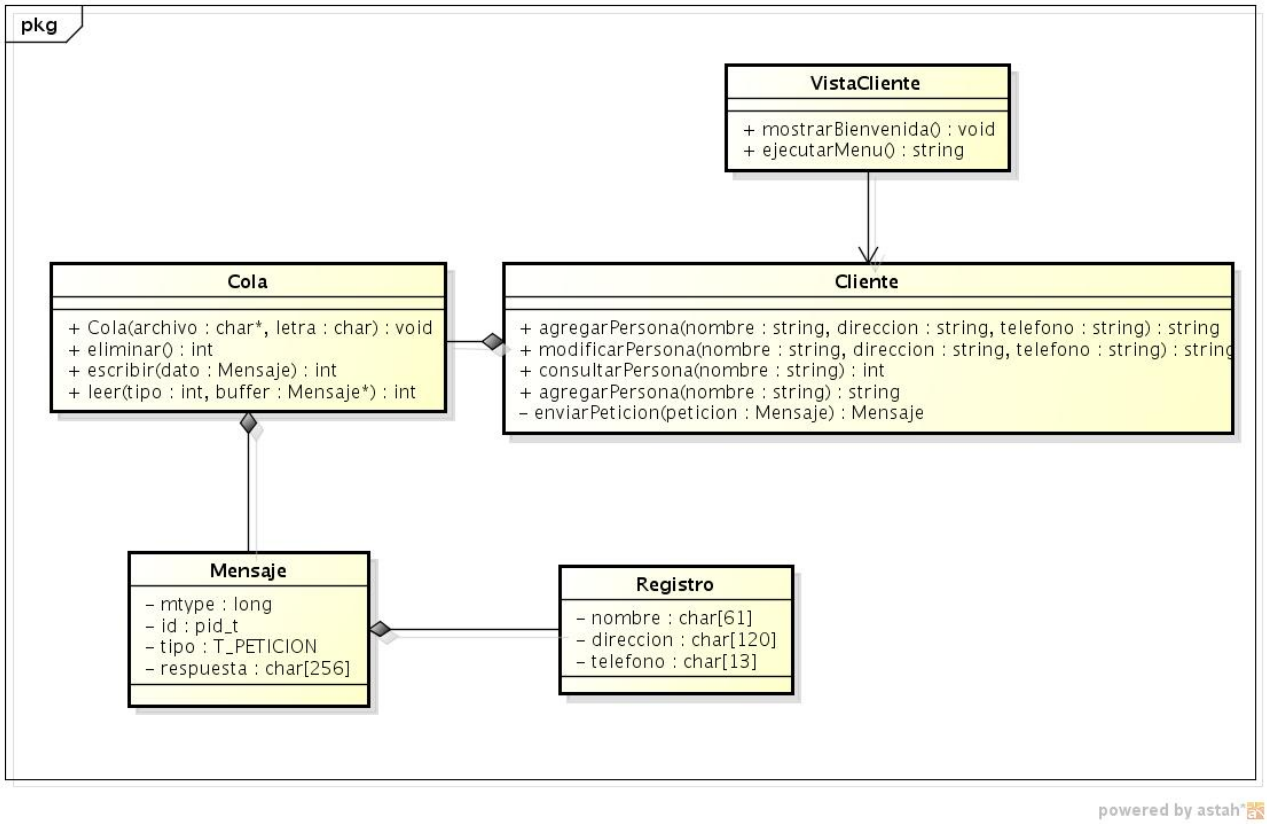
Caso de uso: Consultar Persona	
Descripción: Consulta de los datos de una persona en la base de datos	
Actores participantes: Cliente	
Pre-condiciones: Se deben haber ejecutado las aplicaciones cliente y gestor	
Flujo Principal	
1	El actor cliente ingresa a la aplicación.
2	El actor indica la acción a realizar Si la acción es la consulta de los datos de una persona entonces ejecutar subflujo Consulta de persona (S1)
Flujos Alternativos	
S1	Consulta de persona:
S1.1	El sistema solicita el ingreso del nombre de la persona a consultar.
S1.2	El cliente ingresa el nombre de la persona
S1.3	El sistema valida que el nombre de dicha persona exista en la base de datos (E1)
S1.6	El sistema procesa el registro, arma la petición y se la envía al gestor.
S1.7	El sistema consulta sobre los datos de la persona en la base de datos.
S1.5	El sistema informa nombre, dirección y teléfono de la persona consultada
S1.6	El caso de uso comienza nuevamente desde el paso 2.
Flujos de Excepción	
E1	No existe una persona con dicho nombre, se le informa al cliente y se vuelve al menú anterior. El caso de uso finaliza.
Post-condiciones: Se obtuvieron los datos de la persona consultada de la base de datos	

9. Diagrama de clases

En esta sección se presentan los diagramas de clase para cada una de las aplicaciones.

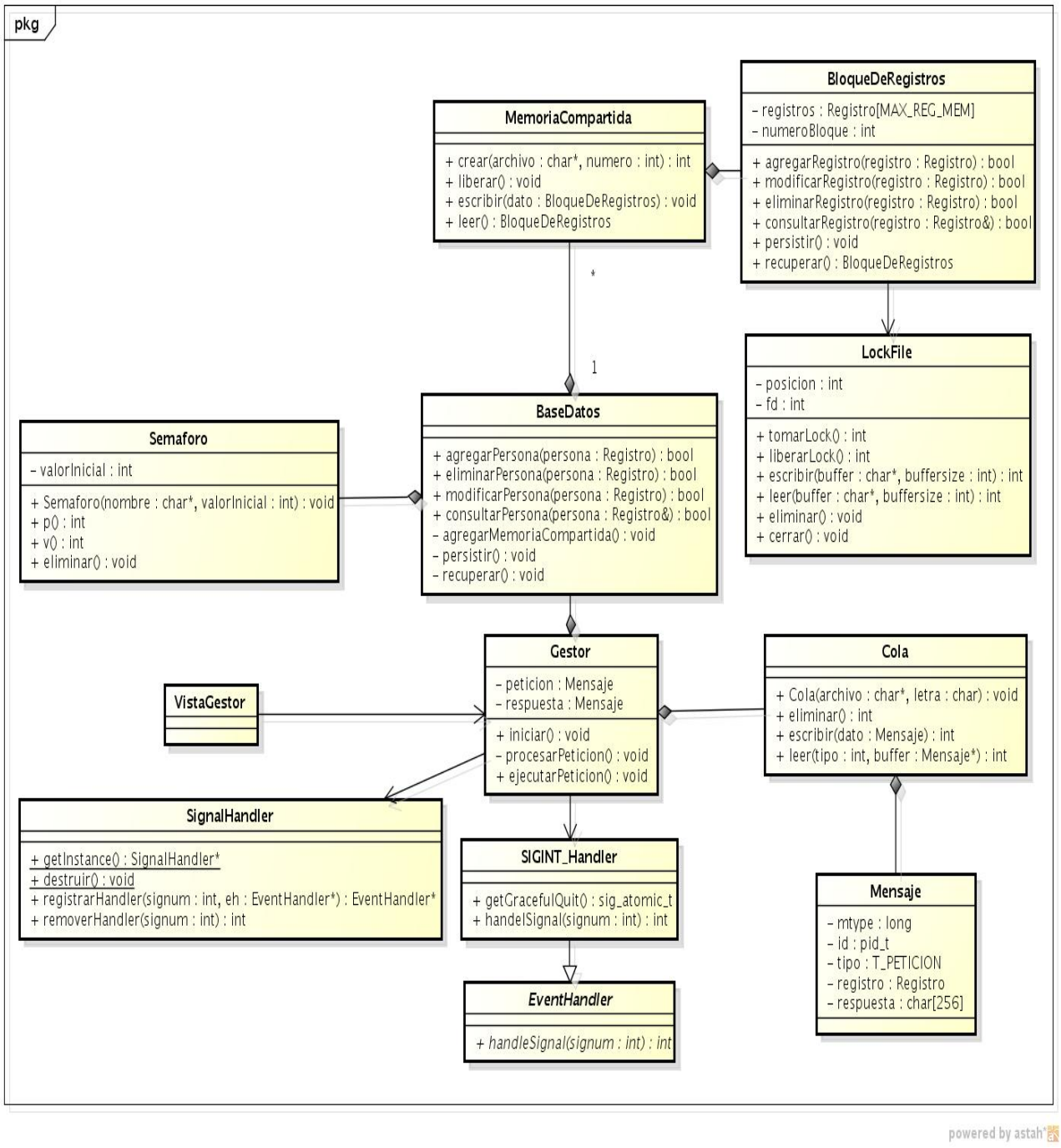
9.1. Diagrama de clases Cliente

A continuación se observa el diagrama de clases correspondiente a la aplicación cliente. Es importante aclarar que se incluye la clase Mensaje, siendo en realidad una estructura. Esta última se muestra en el diagrama dado que la clase Cola es de tipo template, y en este caso, es utilizada con dicha estructura.



9.2. Diagrama de clases Gestor

A continuación se observa el diagrama de clases para la aplicación gestor. Se incluye la clase Mensaje por las mismas razones enunciadas anteriormente. Para simplificar la apreciación del diagrama se decidió no mostrar en el mismo los métodos y atributos privados de las clases en cuestión.


pkg
MemoriaCompartida
+ crear(archivo : char*, numero : int) : int
+ liberar() : void
+ escribir(dato : BloqueDeRegistros) : void
+ leer() : BloqueDeRegistros
BloqueDeRegistros
- registros : Registro[MAX_REG_MEM]
- numeroBloque : int
+ agregarRegistro(registro : Registro) : bool
+ modificarRegistro(registro : Registro) : bool
+ eliminarRegistro(registro : Registro) : bool
+ consultarRegistro(registro : Registro&) : bool
+ persistir() : void
+ recuperar() : BloqueDeRegistros
*
1
LockFile
- posicion : int
- fd : int
+ tomarLock() : int
+ liberarLock() : int
+ escribir(buffer : char*, buffersize : int) : int
+ leer(buffer : char*, buffersize : int) : int
+ eliminar() : void
+ cerrar() : void
Semaforo
- valorInicial : int
+ Semaforo(nombre : char*, valorInicial : int) : void
+ p() : int
+ v() : int
+ eliminar() : void
BaseDatos
+ agregarPersona(persona : Registro) : bool
+ eliminarPersona(persona : Registro) : bool
+ modificarPersona(persona : Registro) : bool
+ consultarPersona(persona : Registro&) : bool
- agregarMemoriaCompartida() : void
- persistir() : void
- recuperar() : void
Gestor
- peticion : Mensaje
- respuesta : Mensaje
+ iniciar() : void
- procesarPeticion() : void
+ ejecutarPeticion() : void
Cola
+ Cola(archivo : char*, letra : char) : void
+ eliminar() : int
+ escribir(dato : Mensaje) : int
+ leer(tipo : int, buffer : Mensaje*) : int
VistaGestor
SignalHandler
+ getInstance() : SignalHandler*
+ destruir() : void
+ registrarHandler(signum : int, eh : EventHandler*) : EventHandler*
+ removerHandler(signum : int) : int
SIGINT_Handler
+ getGracefulQuit() : sig_atomic_t
+ handelSignal(signum : int) : int
Mensaje
- mtype : long
- id : pid_t
- tipo : T_PETICION
- registro : Registro
- respuesta : char[256]
EventHandler
+ handleSignal(signum : int) : int
powered by astah*

10. Compilación

Con el fin de facilitar la compilación de las aplicaciones se provee un script bajo el nombre **compilar.sh**. El mismo se encuentra ubicado en el directorio en el cuál se encuentra el código fuente, y se utiliza de la siguiente forma:

- Ubicarse en el directorio en el cuál se encuentra el código fuente de las aplicaciones. Para ello puede utilizarse el siguiente comando desde un ambiente UnixLinux:
  $:cd Directorio_del_trabajo_practico
- Ejecutar el siguiente comando:
  $:./compilar.sh

  * El script **compilar.sh** debe contar con permisos de ejecución para poder realizar la compilación.
- Una vez finalizada la compilación del paso anterior, se observarán, en dicho directorio, dos archivos ejecutables: *cliente* y *gestor*. Estos archivos representan respectivamente a las aplicaciones cliente y gestor, y para ejecutarlos debe utilizarse **./cliente** y **./gestor**.

11. Modo de Uso

Para utilizar la aplicación **gestor** debe considerarse que la misma finaliza al presionar la tecla *'s'* en la consola en la cuál esta corriendo; esto es informado en la misma mediante un mensaje.

Para utilizar la aplicación **cliente** se debe respetar la interfaz por consola que la misma presenta.