# Índice

**1. Enunciado** **2**

**2. Requerimientos Funcionales** **4**

**3. Hipótesis** **4**

**4. Consideraciones de Diseño** **4**

**5. Herramientas de concurrencia utilizadas** **4**

**6. Comunicación entre procesos** **5**
6.1. Protocolo utilizado . . . . . 6

**7. Casos de Uso** **7**
7.0.1. Especificación de los casos de uso . . . . . 8

**8. Diagrama de clases** **12**
8.1. Diagrama de clases Cliente . . . . . 12
8.2. Diagrama de clases Gestor . . . . . 12

**9. Compilación** **14**

**10. Modo de Uso** **14**

**11. Conclusiones** **15**

# 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 sobre la base de datos. El mismo cuenta de dos aplicaciones diferentes, una aplicación cliente y una aplicación gestor.

## 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.

## 4. Consideraciones de Diseño

- 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 un sincronismo en el mismo. Este semáforo es único para la base de datos en su totalidad.
- La persistencia de la base de datos **TODO**
- 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. Siendo estos datos los almacenados a partir de dicho momento en la base de datos para la persona en cuestión. Es decir, al modificar un registro de una persona se deben modificar los campos dirección y teléfono de la misma. En caso de no querer modificar alguno, debe reingresarse la información ya existente para dicho campo.

## 5. 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 los mismos no posean relación entre ellos, como por ejemplo, en este caso, siendo 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 produce que no sea necesario sincronizar la escritura en la misma por mas que un proceso escriba en ella dado que no puede suspenderse la escritura de dicha estructura antes de finalizar, 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 alli. A su ves, 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.

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

```
 typedef struct {
        long mtype;
        pid_t id;
        T_PETICION tipo;
        Registro registro;
        char respuesta[256];
} mensaje;
```

A continuación se presenta una descripción de el significado y la utilización de cada uno de los campos que conforman dicha estructura.

- mtype: es un long obligatorio que se utiliza además para indicar quien es el destinatario del mensaje.
- id: es el número de proceso de quien envía el mensaje.
- tipo: es un enumerado que representa de que tipo de petición proviene el 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 de error o éxito desde el gestor hacia los clientes, informandoles a través de estos el resultado de la operación realizada.

## 5.1. Protocolo utilizado

En cuanto al protocolo de comunicación utilizado, el mismo 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 el formato de 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_de_la_operacion
- respuesta = “ ”

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

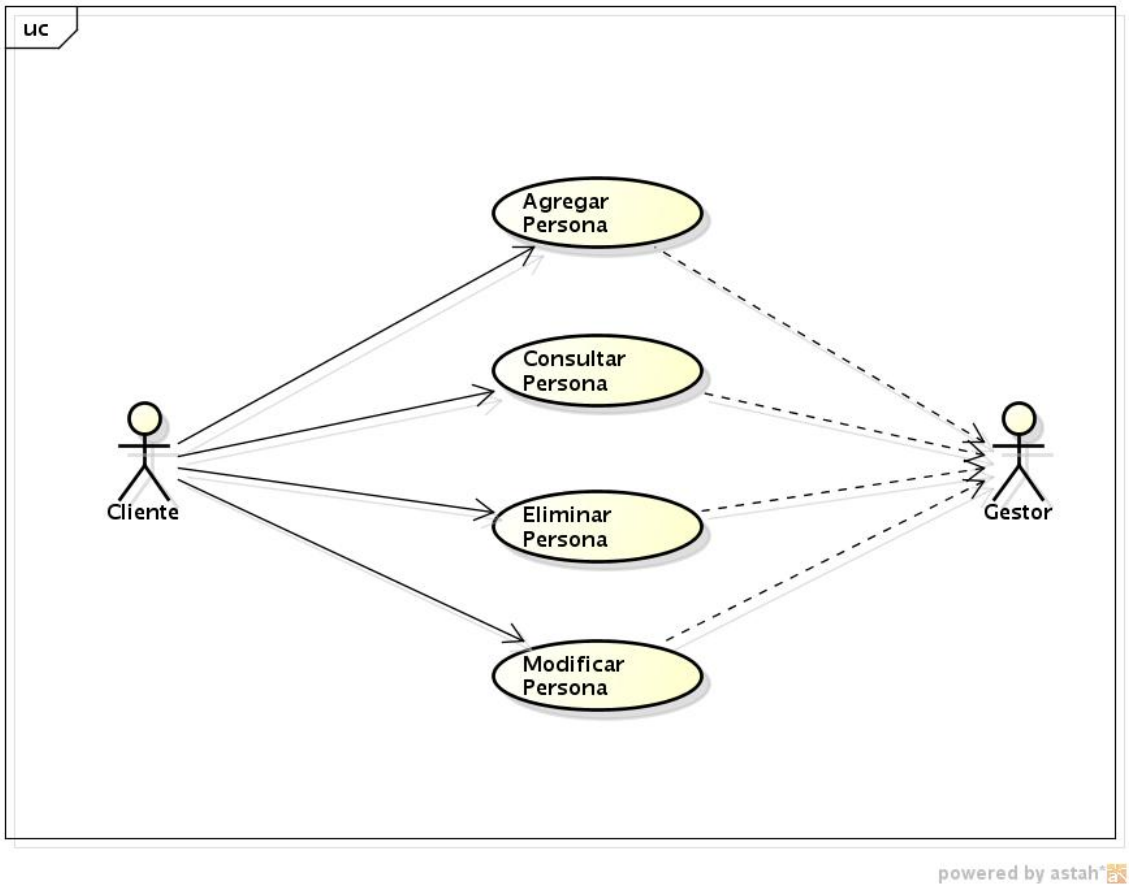
- mtype = pid_del_cliente_que_debe_recibirlo

- id = “ ”
- tipo = “ ”
- registro = “ ”
- respuesta = mensaje_informando_éxito_o_error_al_realizar_la_operación

## 6. Herramientas de concurrencia utilizadas

- Cola de mensajes: utilizada para resolver la comunicación entre los procesos cliente y el 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(a la base de datos).
- LockFiles: utilizado para la persistencia de la base de datos en un archivo(el lockfile) de forma de garantizar que solo 1 gestor pueda escribir a la vez, en caso de existir mas de uno.
- Señales: utilizada para detener la ejecución del gestor, se utiliza la señal SIGTERM en este caso para finalizar de forma correcta el gestor de base de datos.

# 7. Casos de Uso



### 7.0.1. 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<br>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 sistema procesa el registro, arma la petición y se la envía al gestor. |
| S1.8 | 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. |
| | |
| 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<br>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 sistema procesa el registro, arma la petición y se la envía al gestor. |
| S1.8 | 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. |
| | |
| 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<br>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 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 eliminaron los datos de la persona en la base de datos | |

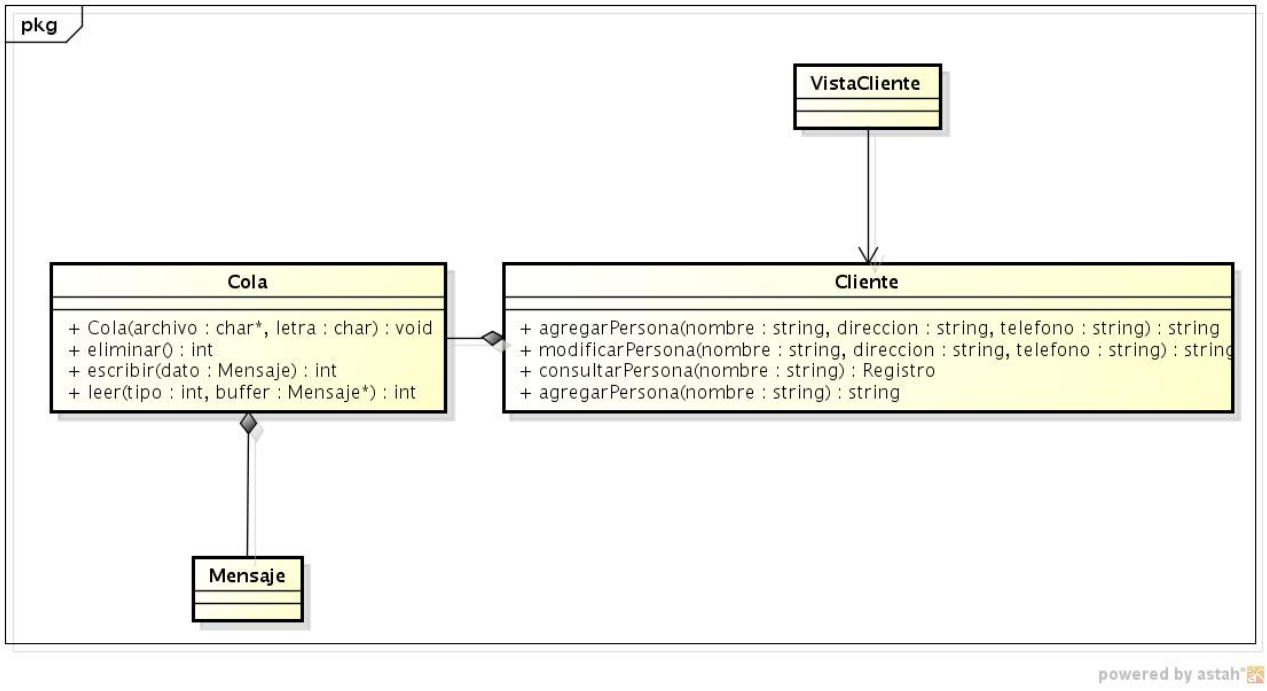
| **Caso de uso: Consultar Persona** | |
|---|---|
| | |
| **Descripción:** Consulta de los datos de una persona en la base de datos | |
| Actores participantes: Cliente | |
| Pre-condiciones: Se deben haber ejecutado las aplicaciones cliente y gestor | |
| | |
| Flujo Principal | |
| 1 | El actor cliente ingresa a la aplicación. |
| 2 | El actor indica la acción a realizar<br>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) |
| S1.4 | El sistema procesa la petición |
| 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 | |

# 8. Diagrama de clases

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

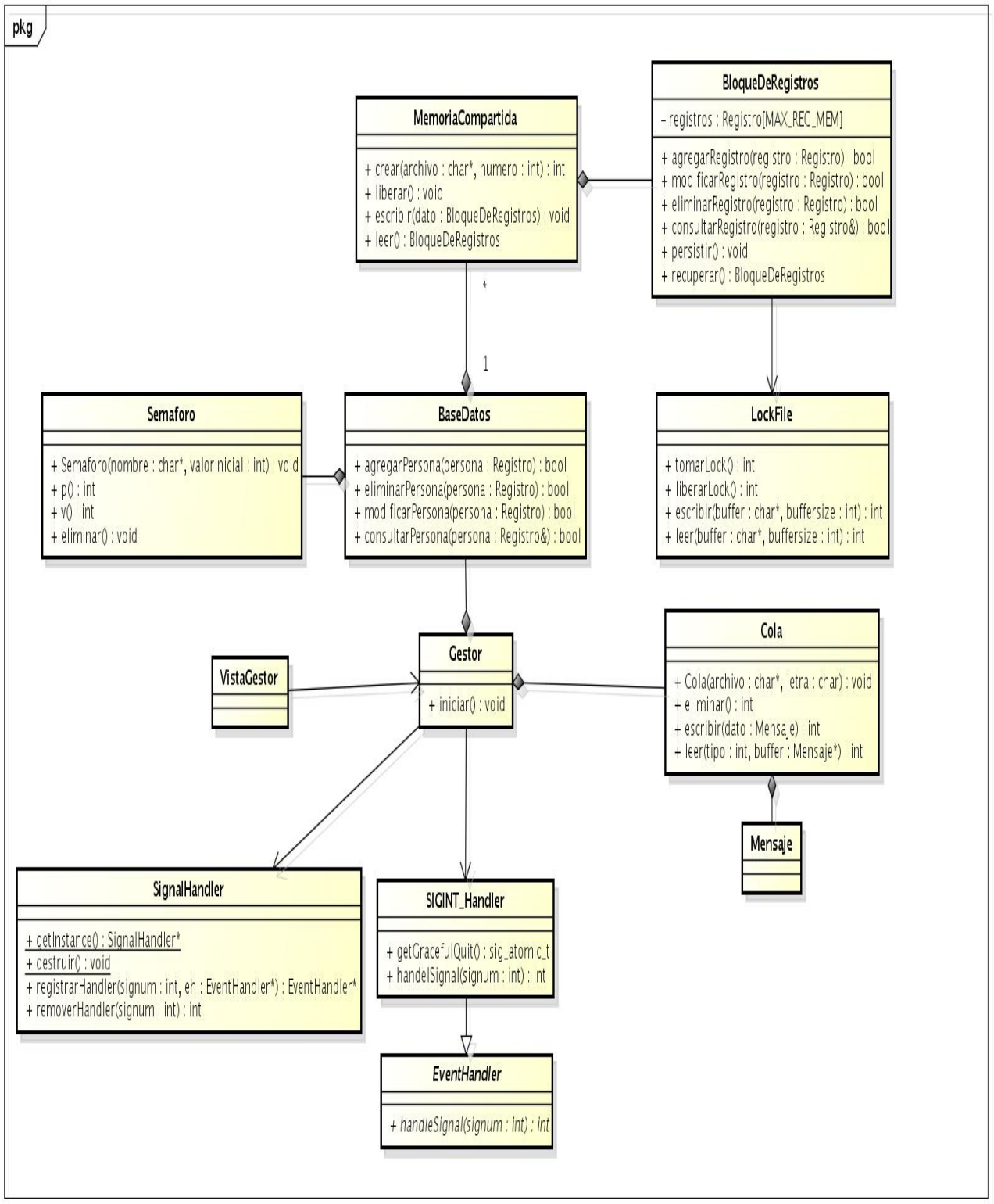
## 8.1. Diagrama de clases Cliente

A continuación se observa el diagrama de clases correspondiente a la aplicación cliente. En el mismo se observa la “clase Mensaje”. Esta “clase” es en realidad una estructura pero se muestra en el diagrama dado que la clase Cola es de tipo template, que en este caso, es utilizada con Mensajes. Es para remarcar esto que se incluye la “clase Mensaje” en el diagrama.




## 8.2. 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 cuestón.


pkg
MemoriaCompartida
+ crear(archivo : char*, numero : int) : int
+ liberar() : void
+ escribir(dato : BloqueDeRegistros) : void
+ leer() : BloqueDeRegistros
BloqueDeRegistros
- registros : Registro[MAX_REG_MEM]
+ agregarRegistro(registro : Registro) : bool
+ modificarRegistro(registro : Registro) : bool
+ eliminarRegistro(registro : Registro) : bool
+ consultarRegistro(registro : Registro&) : bool
+ persistir() : void
+ recuperar() : BloqueDeRegistros
*
1
Semaforo
+ 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
LockFile
+ tomarLock() : int
+ liberarLock() : int
+ escribir(buffer : char*, buffersize : int) : int
+ leer(buffer : char*, buffersize : int) : int
VistaGestor
Gestor
+ iniciar() : void
Cola
+ Cola(archivo : char*, letra : char) : void
+ eliminar() : int
+ escribir(dato : Mensaje) : int
+ leer(tipo : int, buffer : Mensaje*) : int
Mensaje
SignalHandler
+ getInstance() : SignalHandler*
+ destruir() : void
+ registrarHandler(signum : int, eh : EventHandler*) : EventHandler*
+ removerHandler(signum : int) : int
SIGINT_Handler
+ getGracefulQuit() : sig_atomic_t
+ handleSignal(signum : int) : int
EventHandler
+ handleSignal(signum : int) : int
powered by astah*

# 9. 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.
El mismo 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( para ejecutar los clientes debe utilizarse **./cliente**) y gestor(**./gestor**).

# 10. 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.
Al momento de querer modificar una persona desde la aplicación se debe ingresar el nombre de la misma tal cuál figura en la base de datos, dado que el mismo es la clave primaria.

# 11. Conclusiones