Trabajo Práctico 2: Base de datos de personas

Arana Andrés, *P. 86.203*
and2arana@gmail.com

Arias Damián, *P. 89.952*
arias.damian@gmail.com

Sergio Matias Piano, *P. 85.191*
smpiano@gmail.com

2do. Cuatrimestre de 2014
75.59 Técnicas de Programación Concurrente 1
Facultad de Ingenieria, Universidad de Buenos Aires

Resumen

Este informe sumariza el desarrollo del trabajo practico 2 de la materia Técnicas de Programación Concurrente I (75.59) dictada en el segundo cuatrimestre de 2014 en la Facultad de Ingenieria de la Universidad de Buenos Aires. El mismo consiste en la construcción de una aplicación de gestión de información de contacto de personas compuesta por un servidor de datos y varios clientes conectados a través de una cola de mensajes.

Índice

1. Análisis del problema

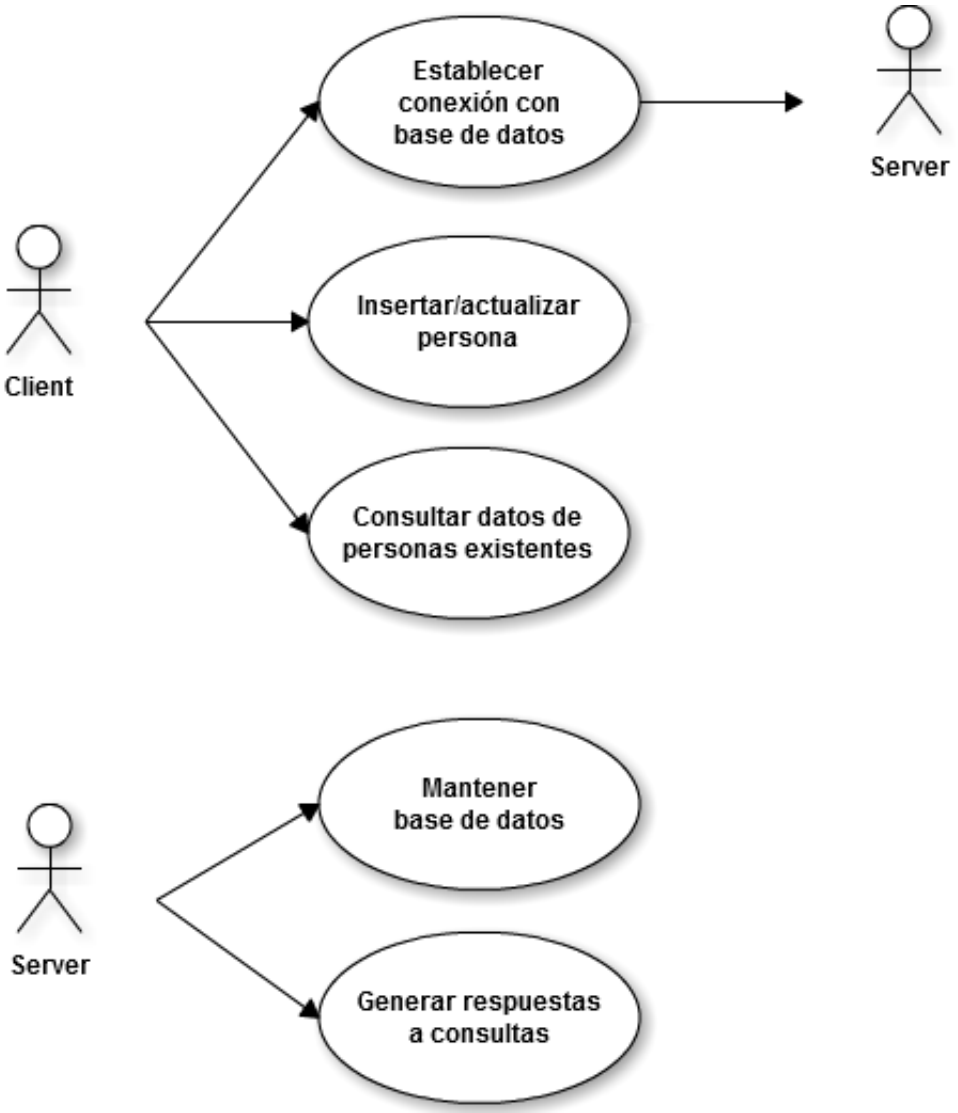
1.1. Descripción

El sistema a desarrollar se dividirá en dos aplicaciones, una será el gestor de base de datos que se encargará de realizar las inserciones y actualizaciones de los registros y la otra será el cliente que realizará pedidos de inserción, actualización y consulta de los registros.

Los campos de los registros tendrán que ser validados al momento de su ingreso para que su tamaño se matenga dentro de los límites impuestos.

Como requisitos adicionales la base de datos se deberá almacenar en un archivo cada vez que el gestor se cierre y, por otro lado, el gestor de la base deberá poder atender consultas de múltiples clientes al mismo tiempo.

1.2. Casos de uso



2. Desarrollo de la solución

2.1. Hipótesis y alcance

- El gestor de base de datos no soporta la operación de borrado de registros.
- No se pueden consultar personas por su dirección o su teléfono.
- Los clientes no reciben confirmación luego de modificar o insertar un registro en la base de datos.

2.2. Procesos

- **server**: este proceso se encarga de leer el archivo completo que representa a la base de datos y mapea todos sus registros en memoria. Todas las modificaciones y lecturas a la base de datos se harán directamente en memoria y se guardará su estado en disco recién cuando se cierre el servidor. Lo siguente que hará el servidor será crear una nueva cola de mensajes pidiendolé su identificador al kernel. Una vez finalizada la etapa de inicialización, el servidor escucha la cola de mensajes esperando la conexión de los clientes y respondiendo consultas de los ya conectados.
- **client**: podrá haber varias instancias de este proceso corriendo en el sistema al mismo tiempo. En el momento de su creación se le pasará por parámetro el identificador de la cola de mensajes a la que se deba subscribir (el identificador es informado por el servidor al momento de su creación). Una vez subscripta a la cola de mensajes cada cliente otendrá un identificador único por parte del servidor con el cual podrán leer y escribir los mensajes.

2.3. Resolución de casos de uso

- **Establecer conexión con base de datos**: el establecimiento de la conexión implica que tanto el cliente como el servidor conocen el identificador de la cola de mensajes del sistema y que además los clientes tiene un id asignado por el servidor que se genera en forma incremental a medida que se van conectando clientes.
- **Insertar/actualizar persona**: el cliente puede pedir la inserción o actualización de un registro particular especificando en id que éste tiene en la base de datos, para esto el usuario deberá ingresar el comando **upsert**.
- **Consultar datos de personas existentes**: para consultar la base de datos se implementaron tres comandos:
  - **selall**: solicita el listado de todos los registros existentes en la base de datos.
  - **selname**: solicita los datos de la persona con el nombre especificado.
  - **selid**: solicita los datos de la persona con el id especificado.
- **Mantener base de datos**: el servidor actualiza y lee los registros que se encontrarán en memoria principal. Cuando lea un mensaje de inserción o actualización realizará las acciones correspondientes. Cuando el servidor se cierre de persistirá todo a disco.
- **Generar respuestas a consultas**: el servidor lee los pedidos de los clientes desde la cola y realiza las búsquedas generando los elementos para satisfacer la consulta y los envía como un mensaje a la cola.

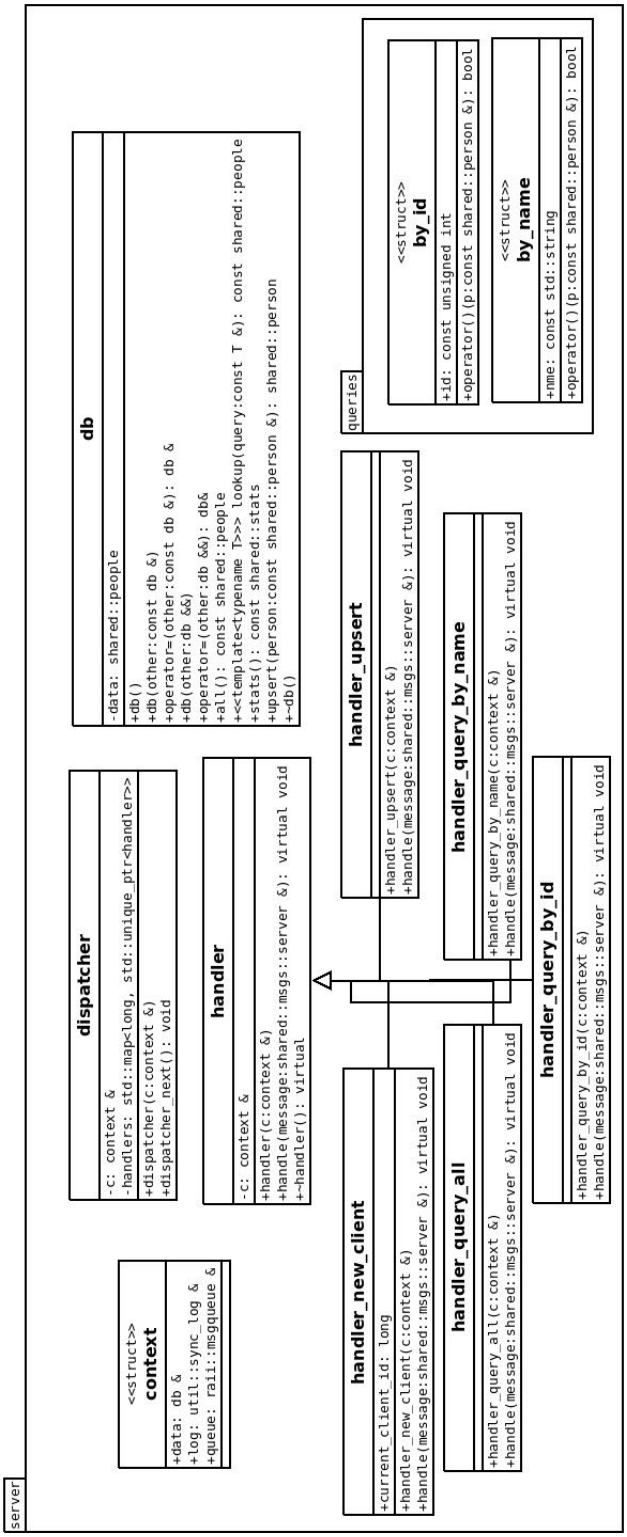
2.4. Protocolo de comunicación

Los datos escritos en la cola de mensajes podrán ser de tres tipos. Cada mensaje (de cualquier tipo) tendrá obligatoriamente el campo **long type** que será el identificador del proceso al cual va dirigido el mensaje.

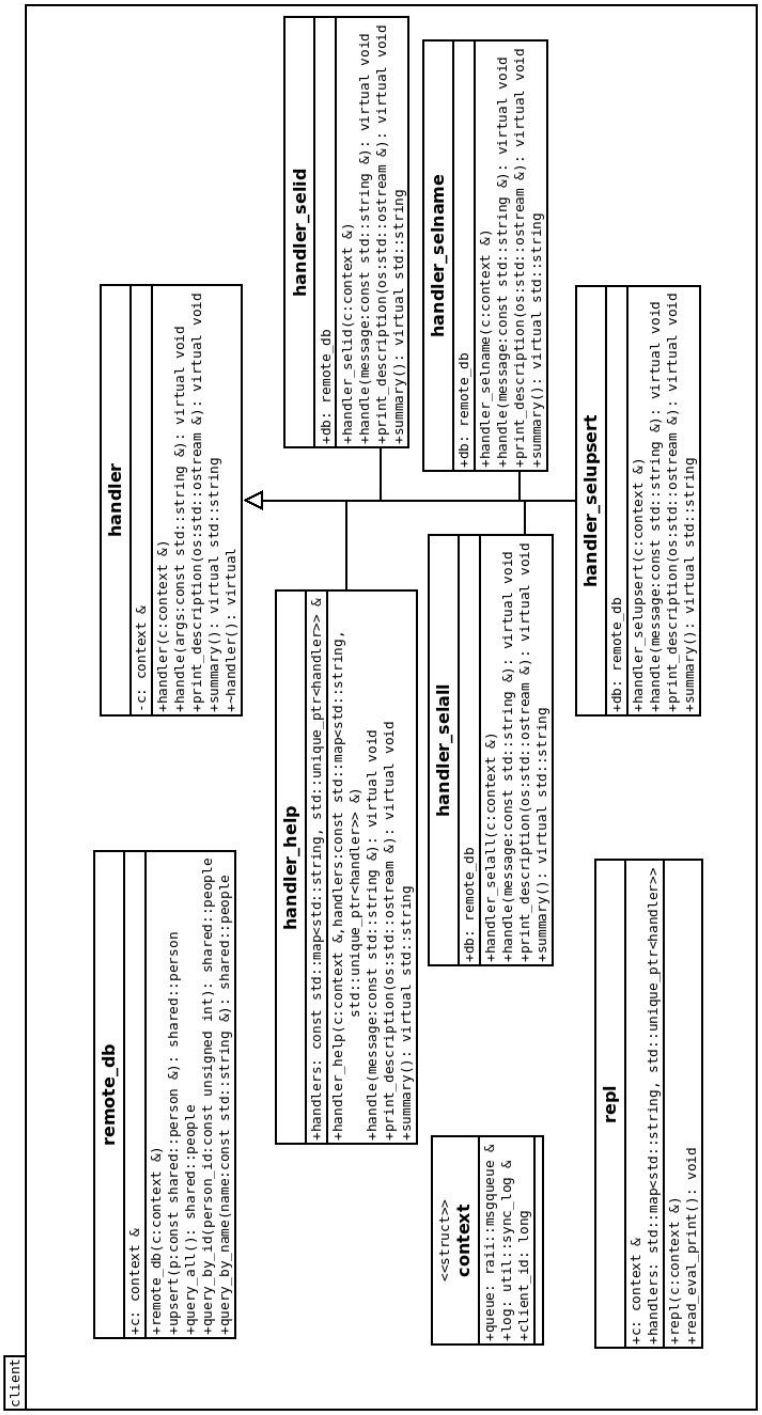
- **mensaje para el server**: tendrá el identificador 1. Este tipo de mensajes tiene un campo 'subtype' que identificará los cuatro tipos de mensajes que pueden ser procesados por el servidor.
- **mensaje para un cliente**: tendrá el identificador del cliente (>1). Este tipo de mensaje será una respuesta por parte del servidor a una consulta del cliente en cuestion. En general contendrá un registro leído desde la base de datos.
- **mensaje de broadcast**: este mensaje especial (con un identificador único) es enviado por el servidor a todos los clientes y solo es leído por aquellos clientes que no tengan asignado aún un identificador. El contenido de este mensaje es el identificador generado por el servidor.

3. Diagramas de clase

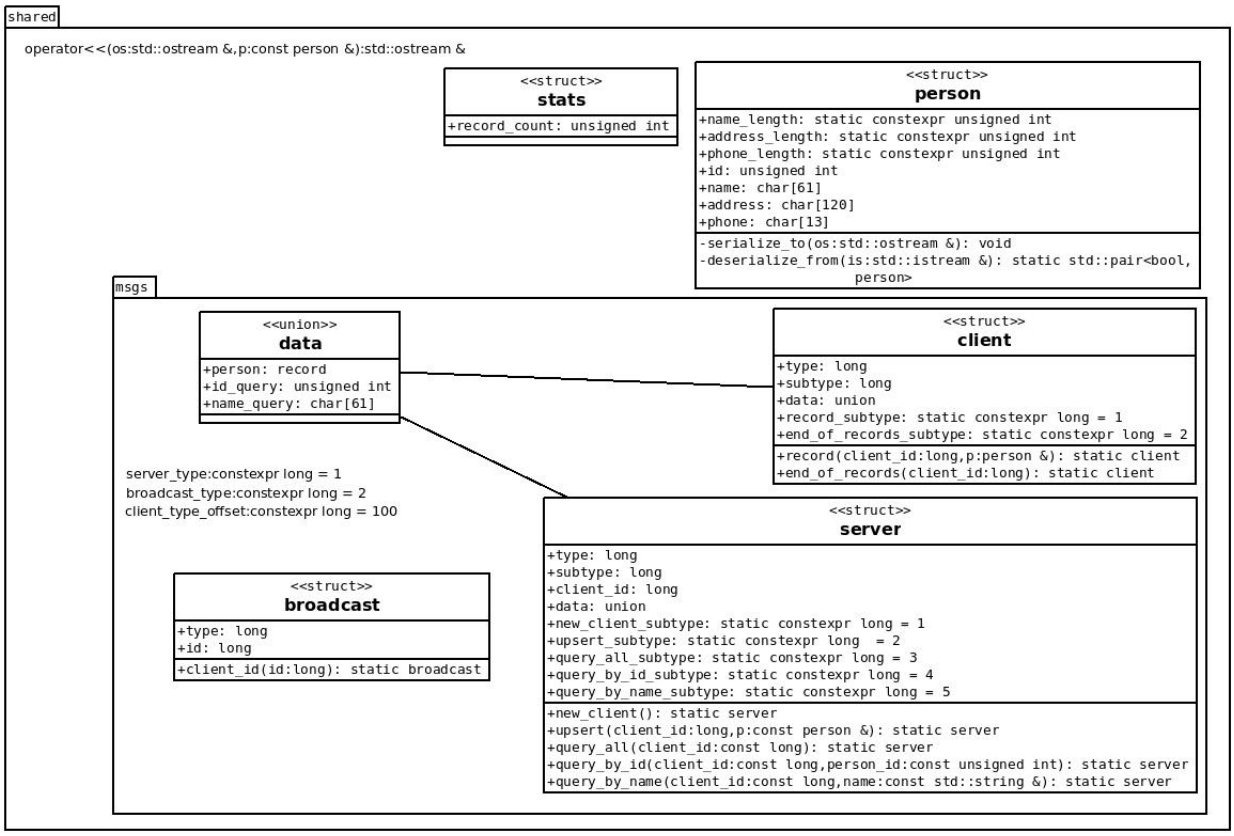
3.1. Server



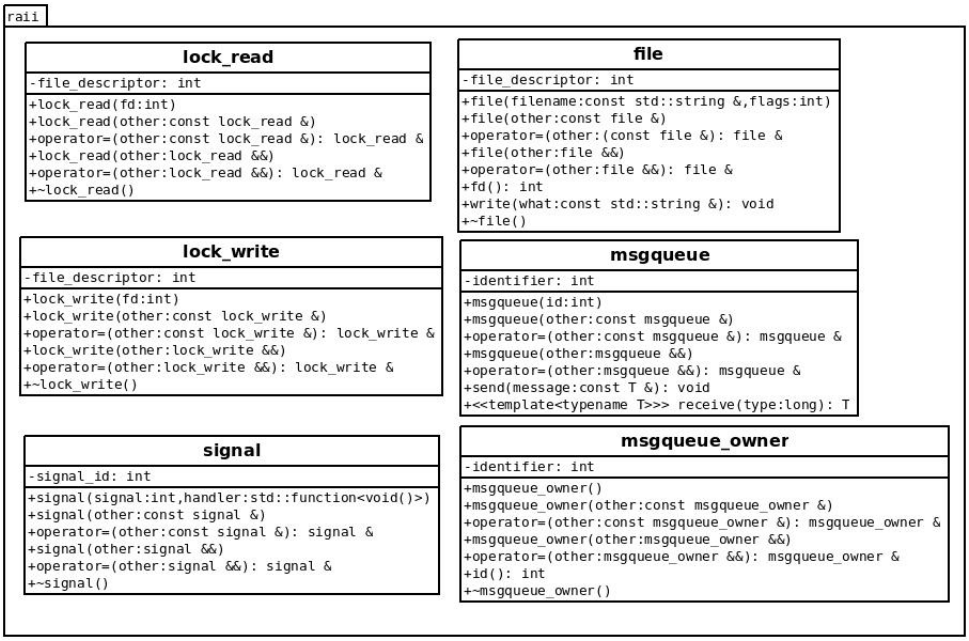
3.2. Client


client
remote_db
+c: context &
+remote_db(c:context &)
+upsert(p:const shared::person &): shared::person
+query_all(): shared::people
+query_by_id(person_id:const unsigned int): shared::people
+query_by_name(name:const std::string &): shared::people
handler
-c: context &
+handler(c:context &)
+handle(args:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
+~handler(): virtual
handler_selid
+db: remote_db
+handler_selid(c:context &)
+handle(message:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
handler_selname
+db: remote_db
+handler_selname(c:context &)
+handle(message:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
handler_help
+handlers: const std::map<std::string, std::unique_ptr<handler>> &
+handler_help(c:context &,handlers:const std::map<std::string,
std::unique_ptr<handler>> &)
+handle(message:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
handler_selall
+db: remote_db
+handler_selall(c:context &)
+handle(message:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
handler_selupsert
+db: remote_db
+handler_selupsert(c:context &)
+handle(message:const std::string &): virtual void
+print_description(os:std::ostream &): virtual void
+summary(): virtual std::string
<<struct>>
context
+queue: raii::msgqueue &
+log: util::sync_log &
+client_id: long
repl
+c: context &
+handlers: std::map<std::string, std::unique_ptr<handler>>
+repl(c:context &)
+read_eval_print(): void

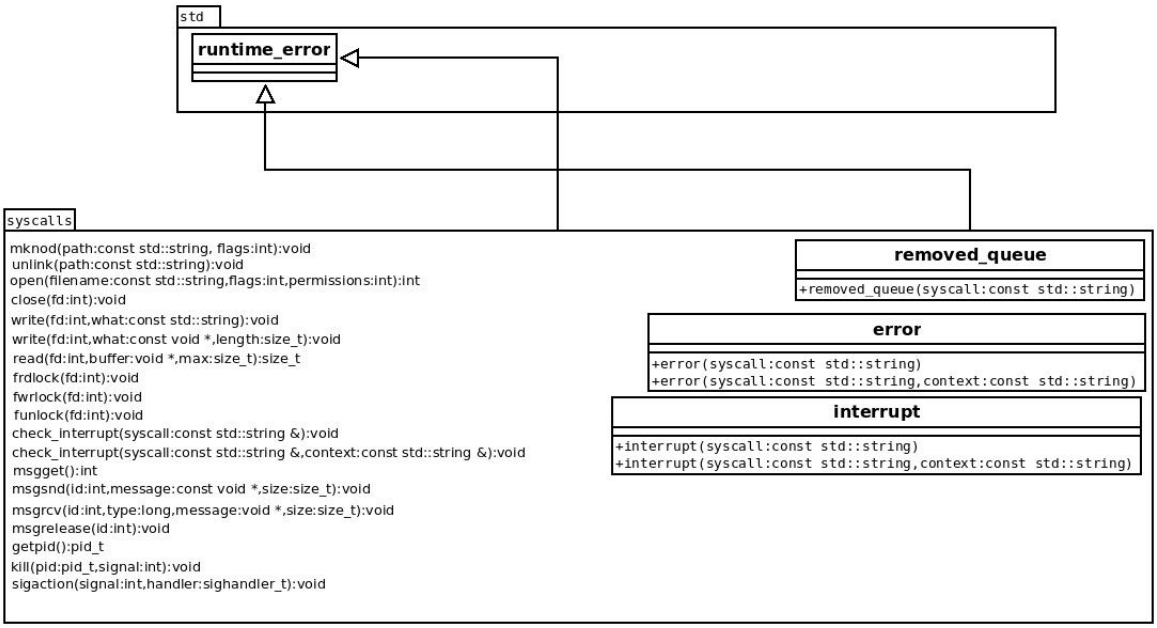
3.3. Shared



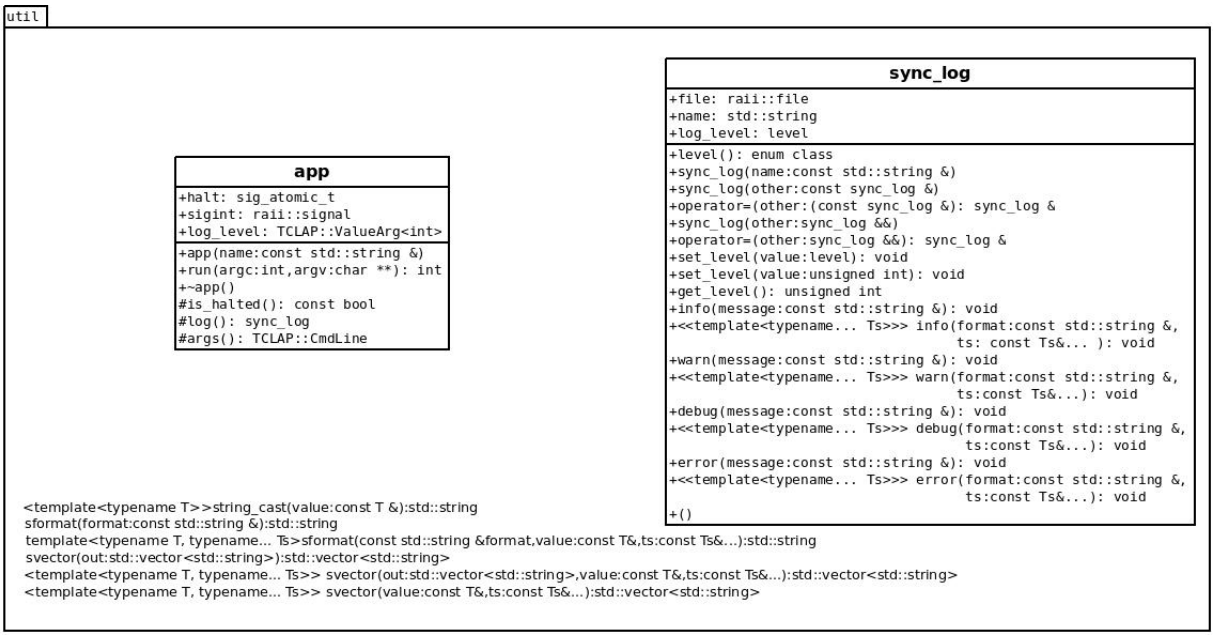
3.4. Raii



3.5. Syscalls



3.6. Util



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. Se deberán poder ejecutar simultáneamente más de 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. El código fuente de la aplicación se entregará mediante el campus, por lo cual no será necesario entregar el CD

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.