Segundo Proyecto
75.59 Técnicas de Programación Concurrente 1

Integrantes:

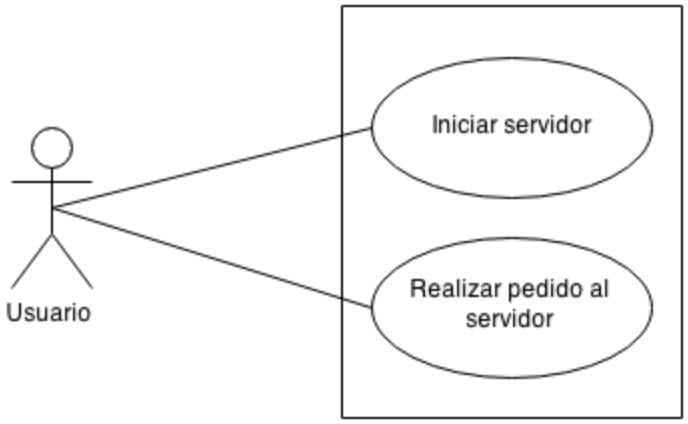
- Rodrigo Burdet 93440
- Nicolas Cisco 94173

El objetivo del trabajo práctico es desarrollar un gestor y cliente de de base de datos. Esta base de datos constará de una única tabla. La comunicación se deberá implementar mediante la cola de mensajes de System V. El cliente realizará peticiones de datos , ya sea, SELECT, UPDATE, DELETE o INSERT que el gestor atenderá. Además, se deberá persistir esta información.

Se implementaron dos procesos:

- Cliente: es el proceso que se encarga de realizar pedidos al servidor, este no tienen ningún conocimiento acerca del funcionamiento de la base de datos y/o de cómo se guarda la información. Solo conoce el protocolo de comunicación con el servidor.
- Servidor (gestor): es el proceso que se encarga de gestionar la base de datos y de atender los pedidos del cliente.

Casos de uso



Caso de uso:	Iniciar servidor
Descripción:	Se ejecuta el servidor
Actores:	Usuario
Pre-condiciones	El servidor no esta corriendo
Flujos :	
Flujo principal	1 | El usuario ejecuta el proceso servidor. 2 | Crea la cola de mensajes e imprime su id 3 | El servidor se encarga de atender todos los pedidos de los clientes
Flujo alternativo	
Post-condiciones	Se eliminó la cola de comunicación.

Caso de uso:	Realizar pedido al servidor
Descripción:	Hacer un pedido al servidor, ya sea para consultar, agregar, editar o eliminar datos.
Actores:	Usuario
Pre-condiciones	El servidor debe estar corriendo
Flujos :	
Flujo principal	1 | El usuario ejecuta el proceso cliente. 2 | Se controla que el usuario haya introducido los datos necesarios correctamente. 3 | Se genera el mensaje y se envía a través de la cola de mensajes. 4 | Se espera la respuesta del servidor y se muestra en pantalla.
Flujo alternativo	
Post-condiciones	Se cerró el cliente.

Ejemplo de ejecución

Servidor

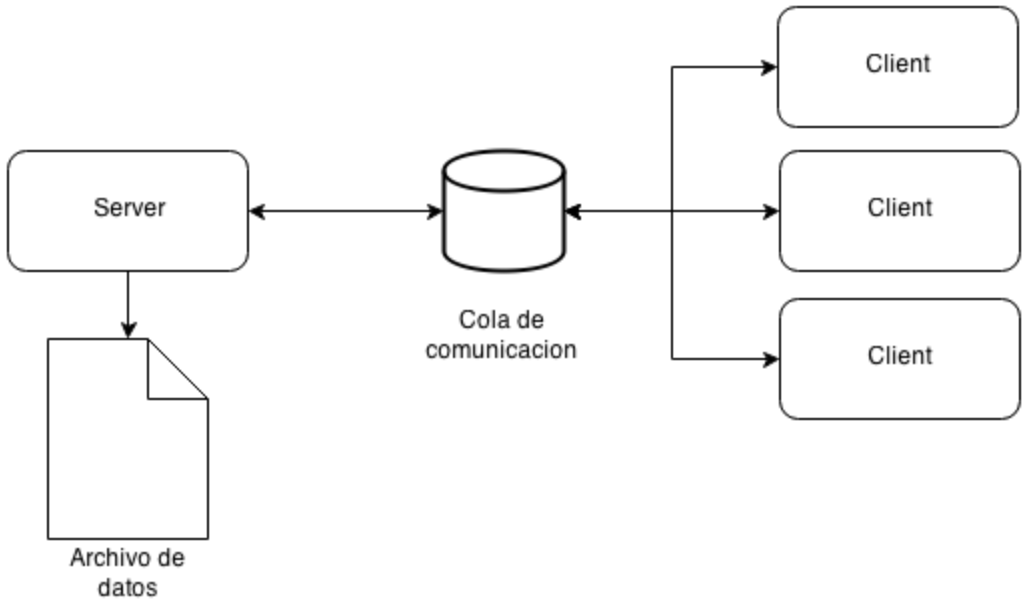
```
[user@myhost concurrente_msg]$ ./bin/server
Server running msgid: 196608
```

Cliente

```
[user@myhost concurrente_msg]$ ./bin/client -h
Usage: ./bin/client [OPTION] NOMBRE [DIRECCION] [TEL]
  -m, --msgid=MSGID             Especificar el msgid a usar, si no se generara.
  -s, --select          Select, solo se debe especificar nombre
  -d, --delete          Delete, solo se debe especificar nombre
  -i, --insert          Insert, se debe especificar nombre direccion y telefono
  -u, --update          Update, se debe especificar nombre direccion y telefono
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -i "nombre" "direccion"
"123456"
Success
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -s "nombre"
Nombre: 'nombre'
Direccion: 'direccion'
Telefono: '123456'
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -u "nombre" "nueva direccion"
"654321"
Success
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -s "nombre"
Nombre: 'nombre'
Direccion: 'nueva direccion'
Telefono: '654321'
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -d "nombre"
Success
[user@myhost concurrente_msg]$ ./bin/client -m 196608 -s "nombre"
Error
```

Comunicación

Diagrama de comunicación entre procesos:



Protocolo de comunicación:

Los procesos se comunican escribiendo mensajes del tipo *MsgStruct* en la cola de mensajes. Esta estructura de datos está definida en el archivo *util/defines.h* y es la detallada a continuación:

```
typedef struct {
        long mtype;
        MsgData data;
} MsgStruct ;
```

El primer miembro de la estructura, *mtype*, es un *long* que identifica el tipo de mensaje, la implementación de la cola de mensajes de System V obliga a su existencia. En este protocolo se usará el valor de *SEVER_MSGTYP* (cuyo valor es *1*), definido en *util/defines.h*, para enviar mensajes hacia el servidor y el pid del proceso receptor para mensajes que envíe el servidor hacia el cliente.

La estructura *data* del tipo *MsgData* lleva la información del mensaje. Se encuentra definida en el archivo *util/defines.h*:

```
typedef struct {
        MsgType type;
        pid_t pid;
```

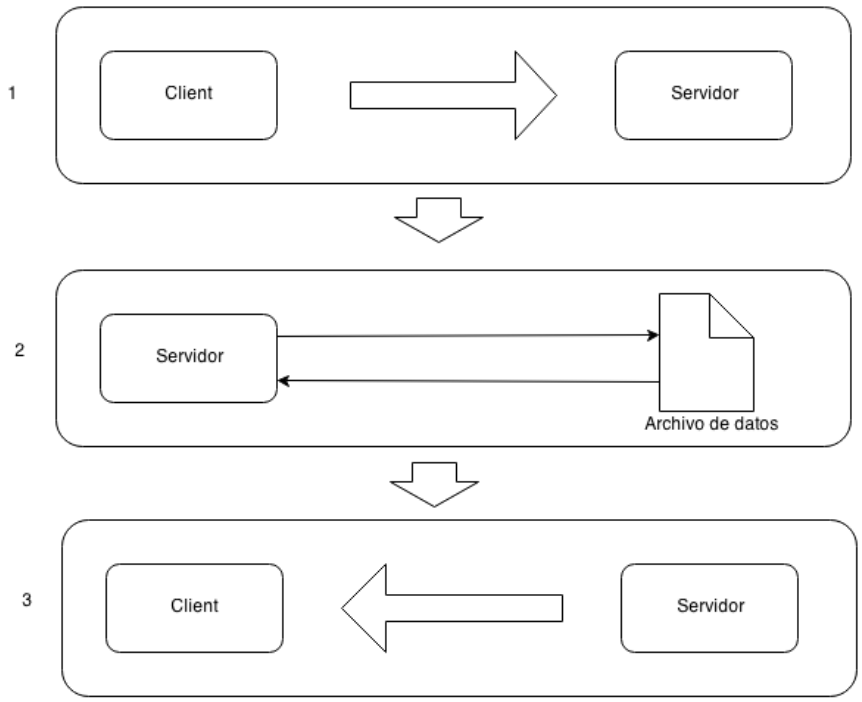
```
        Register reg;
} MsgData;
```

El miembro *type* indica el tipo de mensaje. *MsgType* es un enumerador cuyos valores son:

- SELECT: Utilizado para hacer un pedido de datos
- INSERT: Utilizado para hacer una inserción de datos
- DELETE: Utilizado para hacer un borrado de datos
- UPDATE: Utilizado para actualizar datos
- ERROR: Utilizado para dar aviso de error en la operación
- SUCCESS: Utilizado para dar aviso de que la operación se realizó satisfactoriamente.

El servidor recibe mensajes del tipo *SELECT*, *INSERT*, *DELETE* y *UPDATE*. Ignora otro tipo de mensajes. El cliente envía los tipos de mensajes mencionados y recibe mensajes del tipo *ERROR* y *SUCCESS*.
El miembro *pid* es completado por el cliente y lleva el valor del id del proceso *(su pid)*.
Finalmente, el miembro *reg* del tipo *Register*, definido en dicho archivo, lleva los datos del registro.

```
typedef struct {
        char nombre[61];
        char direccion[120];
        char telefono[13];
} Register;
```



Paso 1: El cliente se comunica con el servidor. Este en el mensaje especifica que acción quiere realizar (*SELECT*, *INSERT*, *DELETE* o *UPDATE*), *mtype* = *SEVER_MSGTYP*, su pid y

completa con el registro con los datos. Estos datos varían según la acción a realizar, para *SELECT* y *DELETE* solo hace falta completar el nombre, para *INSERT* y *UPDATE* hace falta completar todos los campos.

Paso 2: El servidor recibe el mensaje y lo procesa.

Paso 3: El servidor le contesta al cliente. En *mtype* se escribe el pid recibido. Dependiendo del tipo de mensaje que se recibió se enviará distinto tipo de mensajes, para *INSERT*, *DELETE* y *UPDATE* que se realizaron satisfactoriamente, es decir, si el servidor no tuvo problemas internos, para el caso de *INSERT* no existía otro registro con el mismo nombre y para el caso de *DELETE* y *UPDATE* existía un registro con ese nombre se devolverá el tipo *SUCCCESS*. Si se recibió un mensaje del tipo *SELECT* y no hubo problemas al procesarlo, se completará el registro con la información sacada de archivo y colocará el tipo *SELECT*. De haber algún error al procesar el pedido, ya sea por error interno del servidor, que en SELECT, UPDATE y DELETE que no exista un registro con el nombre especificado o que en INSERT ya exista un registro con el nombre especificado se devolverá el tipo *ERROR*.

Estructura del proyecto

```
.
├── LICENSE
├── Makefile
├── README.md
└── src
    ├── client
    │   └── main.cpp
    ├── server
    │   └── main.cpp
    └── util
        └── defines.h
```

Dentro de la carpeta *src* se encuentra el código fuente del proyecto. Ahí se agrupa por carpetas teniendo en cuenta funcionalidades específicas para cada proceso y funcionalidades que son genéricas a todos o varios procesos. Todo lo particular se encuentra dentro de una subcarpeta que lleva el nombre del proceso. Mientras que lo genérico está bajo la carpeta *util.*

Compilación

El proyecto se desarrolló utilizando la herramienta *GNU Make*. Para compilar basta con hacer *make*. Que sería equivalente a hacer *make all*. Los parámetros que se pueden pasar son:

- **all**: compila todos los ejecutables
- **clean**: borra carpeta de ejecutables, carpeta de documentación y todos los archivos objeto

Ejemplo:

```
[user@myhost concurrente_msg]$ make -j3
mkdir ./bin/
gcc -c -Wall -Wextra -g -o "src//server/main.o" "src//server/main.c"
gcc -c -Wall -Wextra -g -o "src//client/main.o" "src//client/main.c"
gcc ./src//server/main.o  -Wall -Wextra -g -o bin//server
gcc ./src//client/main.o  -Wall -Wextra -g -o bin//client
```

Después de compilar tendremos la siguiente estructura de carpetas:

```
.
├── bin
│   ├── client
│   └── server
├── LICENSE
├── Makefile
├── README.md
└── src
    ├── client
    │   ├── main.c
    │   └── main.o
    ├── server
    │   ├── main.c
    │   └── main.o
    └── util
        └── defines.h
```

Configuración

El trabajo práctico permite configurar, en tiempo de compilación, el archivo, carácter a usar para generar el *key* de la cola de mensajes y el archivo de persistencia para los datos. El servidor no permite ninguna otra configuración adicional. Esta configuración se realizará editando el archivo *util/defines.h*.

El cliente permite configurar el id de la cola de mensajes en tiempo de ejecución. Esto se realiza pasando la opción *-m <msg_id>* al ejecutarlo. De lo contrario el cliente generará la *key* utilizando el nombre de archivo y el carácter especificado en *util/defines.h* en tiempo de compilación.