Práctica 2: INTERACCIÓN ENTRE 2 SERVIDORES CON Sun RPC
(Será realizado en la Sala de Computo)

La empresa Centrales Eléctricas de Occidente (CEO) de Popayán, en asociación con la Universidad del Cauca (Grupos I+D GAI e IDIS) requiere implementar un sistema AMI basado en SAA (Sistema de Amarres Automático). SAA fue desarrollado por el grupo GAI del Depto de Automática y Control de la FIET. El objetivo de esta asociación es implementar un sistema AMI-SAA, que permita utilizar la red eléctrica de CEO-Popayán para la transmisión de datos usando los dispositivos desarrollados en SAA, con el fin de monitorear, gestionar y controlar de manera más eficiente el suministro de energía eléctrica en la ciudad de Popayán.

El sistema AMI-SAA está compuesto por los siguientes nodos:

PLC_TU: Son nodos instalados junto a los medidores inteligentes, y es responsable de transmitir información de consumo y configuración a los nodos PLC_MMS. La transmisión de información se realiza utilizando la red de distribución (uso del protocolo PLC)

PLC_MMS: Este tipo de nodo se encuentra instalado cerca a los transformadores de la red de distribución. Son encargados de recibir la información de consumo y configuración de los PLC_TU.

Gestor de la red SAA(GRSAA): Mediante GPRS (General Packet Radio Service), estos nodos se encargan de recibir la información consumo de energía recogido de un sector de la ciudad a través de los nodos PLC_MMS. Con esta información se realiza la facturación de consumo y se realizan estadísticas del consumo energético.

Se requiere implementar una aplicación distribuida utilizado el modelo de RPC de Sun para GNU/Linux, que permita implementar la parte inteligente de los nodos que conforman el sistema AMI-SAA. Cuando el sistema AMI-SAA se pone en marcha, un operador de la empresa CEO se desplaza a las residencias y mediante un menú, registra el dispositivo PLC_TU en los nodos PLC_MMS. Para llevar a cabo esta operación mediante un menú de texto podrá abrir sesión (ver Figura 1) y si las credenciales son correctas se debe mostrar los siguientes datos del operador (Id, Nombre completo y usuario) y posteriormente se desplegará un menú de texto Operador (ver Figura 2), mediante el cual podrá registrar los dispositivos PLC_TU.

```
=== Menu abrir sesion ===
1. Abrir sesion.
2. Salir.
=======================
Seleccione una opcion: 1
```

Figura 1. Menú session

```
=== Menu operador ===
1. Registrar dispositivo
2. Consultar dispositivo
3. Salir.
======================
Seleccione una opcion:
```

Figura 2. Menú operador

Para ingresar a la aplicación como **operador CE0** debe ingresar id, usuario y clave.
La primera opción permite registrar un dispositivo **PLC_TU**, para ello se debe ingresar el id_plctu, el nombre del propietario de la residencia, la dirección residencial e internamente se debe fijar un valor de consumo en cero. El **máximo número de dispositivos** que se podrán registrar será de **5 dispositivos**.
Si el registro del dispositivo **PLC_TU** es exitoso se debe emitir un mensaje:

** Usuario registrado exitosamente **.

Si el registro del dispositivo **PLC_TU** no es exitoso se debe emitir el mensaje:

** Dispositivo NO registrado, se alcanzó la cantidad máxima de dispositivos a registrar **

La segunda opción permite consultar los datos de registro de un dispositivo **PLC_TU**. Se debe ingresar el id_plctu y si la consulta es exitosa se debe mostrar la siguiente información: id_plctu, propietario, domicilio y consumo.
Si la consulta es no exitosa, se debe desplegar el mensaje:

** Dispositivo PCL_TU no encontrado **

Si al estar en el **Menú sesión** (ver Figura 1) se ingresan las credenciales incorrectas, se desplegará un **Menú usuario** (ver Figura 3), mediante el cual solo se podrá consultar los datos de un dispositivo que este ya registrado.

```
=== Menu usuario ===
1. Consultar dispositivo
2. Salir.
======================
```

Figura 3. Menú usuario

Las funcionalidades del **Menú usuario**, para esta práctica, no se deben implementar
La primera opción permite consultar los datos de registro de un dispositivo **PLC_TU**. Se debe ingresar el id_plctu y si la consulta es exitosa se debe mostrar la siguiente información: id_plctu, propietario, direccion y consumo.
Si la consulta es no exitosa, se debe desplegar el mensaje:

** Dispositivo PCL_TU no encontrado **

Cuando un **operador CEO** complete el registro de **5** dispositivos **PLC_TU**, desde el **servidor1**(nodo plc_tu) se debe enviar un mensaje de notificación al **servidor2** (nodo grsaa). En el **servidor2** se debe desplegar el siguiente mensaje:

*** El dispositivo PLC_MMS con Id:xx completo sus registros y esta activo en el sistema ***

Desarrollo de la aplicación

La aplicación debe implementarse usando **Sun RPC**, mediante el **lenguaje de programación C** en un

sistema operativo GNU/Linux. En la Figura 4 se observa un diagrama de contexto que muestra las operaciones que puede realizar un operador CEO y los usuarios. Las operaciones deben ser implementadas en un servidor de usuarios (servidor), sevidor de plc_tu(servidor1) y servidor de registro de saa(servidor2). Cada vez que se envie una petición en el nodo plc_tu y se reciba una petición en los nodos servidores, se deben crear ecos por medio de un mensaje en pantalla en el cual se describe cual función se está solicitando o ejecutando.. Cuando 5 dispositivos PLC_TU son registrados el servidor1, debe notificar este evento al servidor2.

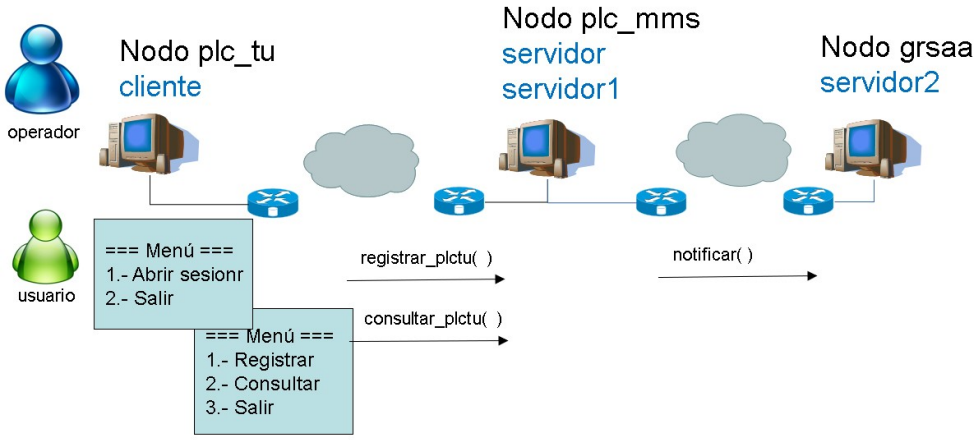

Figura 4. Diagrama de Contexto.

Antes de iniciar se debe crear la siguiente estructura de directorios:

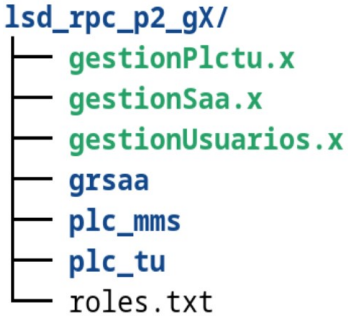

Figura 5. Estructura de directorios para desarrollar la práctica.

La carpeta, con la solución de este ejercicio, debe ser comprimida utilizando el comando:
tar cfvz directorioEmpaquetado.tar.gz directorioaComprimir , este archivo debe ser subido al sitio del curso en la plataforma Moodle. El nombre del archivo comprimido debe tener el siguiente formato lsd_rpc_p2_gX.tar.gz. Donde gX corresponde al nombre del grupo asignado en este curso. Dentro de la carpeta se debe establecer en un archivo denominado roles.txt, indicándose el nombre y apellido de los 2 estudiantes y su role correspondiente en la implementación de la solución (cliente y servidor2, servidor1).

1. Crear el IDL

En Sun RPC se definen las interfaces usando la notación XDR (External Data Representation) en un archivo con extensión .x. De acuerdo con el enunciado se requiere la interacción con 2 servidores, por lo tanto, es necesario el uso de 2 interfaces, la interface gestionUsuarios.x, que permitirá la interacción entre el cliente y el servidor susuario (ver figura 4) y la interface gestionJuego.x, que permitirá la interacción entre el cliente y el servidor sjuego y las interface registroJuego.x (ver figura 5).

```
/*Declaración de datos a transferir entre el cliente y el servidor*/
/*Declaración de constantes*/
const MAXCAD = 20;
const MAXDAT = 12;
/*Declaración de la estructura que permite almacenar los datos para una
sesion*/
struct datos_sesion{
        int id;
        char usuario[MAXDAT];
        char clave[MAXDAT];
};
/*Declaración de la estructura que permite almacenar los datos de un
usuario*/
struct datos_usuario{
        int id;
        char nombreCompleto[MAXDAT];
        char usuario[MAXDAT];
        char clave[MAXDAT];
};
/*Definición de las operaciones que se pueden realizar*/
program gestion_usuarios{
        version gestion_usuarios_version{
                int abrirsesion(datos_sesion)=1;
                datos_usuario consultarusuario(int id)=2;
        }=1;
}=0x30000013;
```

Figura 6. Interface gestionUsuarios.x

Mediante un editor de texto cree la definición de la interface incluida en la Figura 6.

```
/*Declaración de datos a transferir entre el plc_tu y el plc_mms*/
/*Declaración de constantes*/
const MAXCAD = 20;
const MAXDAT = 12;
/*Declaración de la estructura que permite almacenar los datos de un
plc_tu*/
struct datos_plctu{
        char id_plctu[MAXDAT];
        char propietario[MAXCAD];
        char direccion[MAXCAD];
         int consumo;
};

/*Definición de las operaciones que se pueden realizar*/
program gestion_dispositivos{
        version gestion_dispositivos_version{
                bool registrar_plctu(datos_plctu dplctu)=1;
                datos_plctu consultar_plctu(int id)=2;
        }=1;
}=0x30000015;
```

Figura 7. Interface gestionPlctu.x

Mediante un editor de texto cree la definición de la interface incluida en la Figura 7.

```
/*Declaración de datos a transferir entre el plc_mms y el plc_mms*/

program gestion_saa{
version gestion_saa_version{
                void notificarPlcmms(int id_usuario)=1;
        }=1;
}=0x30000014;
```

Figura 8. Interface gestionSaa.x

Mediante un editor de texto cree la definición de la interface incluida en la Figura 8.

2. Procesamiento de la Interface gestionUsuarios.x

Ubicados en el directorio de trabajo, utilice el compilador de interfaces 'rpcgen' para procesar la definición de interfaces creada en el punto anterior. Desde una consola debe introducir el siguiente comando:

```
rpcgen gestionUsuarios.x
```

Mediante el compilador de interfaces rpcgen se generan siguientes archivos:

gestionUsuarios_clnt.c: Contiene la funcionalidad del client stub.
gestionUsuarios.h: En este archivo de cabecera se definen estructuras declaradas en la interface remota.
gestionUsuarios_svc.c: Contiene la funcionalidad del server stub.
gestionUsuarios_xdr.c: Contiene las rutinas de aplanamiento o codificación de los datos.

Los archivos generados se organizan en las carpetas de plc_tu (cliente) y plc_mms (servidor) de la siguiente forma:

plc_tun(cliente):
gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_clnt.c
gestionUsuarios_xdr.c

plc_mms (servidor):
gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_svc.c
gestionUsuarios_xdr.c

3. Procesamiento de la Interface gestionPlctu.x y gestionSaa.x

Ubicados en el directorio de trabajo, utilice el compilador de interfaces 'rpcgen' para procesar la definición de interfaces. Desde una consola debe introducir los siguientes comandos:

```
rpcgen gestionPlctu.x
rpcgen gestionSaa.x
```

Mediante el compilador de interfaces rpcgen al procesar la interfaz gestionPlctu.x, se generan los siguientes archivos:
gestionPlctu_clnt.c: Contiene la funcionalidad del client stub.
gestionPlctu.h: En este archivo de cabecera se definen estructuras declaradas en la interface remota.
gestionPlctu_svc.c: Contiene la funcionalidad del server stub.
gestionPlctu_xdr.c: Contiene las rutinas de aplanamiento o codificación de los datos.

Mediante el compilador de interfaces rpcgen al procesar la interfaz gestionSaa.x, se generan los siguientes archivos:
gestionSaa_clnt.c: Contiene la funcionalidad del client stub.
gestionSaa.h: En este archivo de cabecera se definen estructuras declaradas en la interface remota.
gestionSaa_svc.c: Contiene la funcionalidad del server stub.

Los archivos generados se organizan en las carpetas de plc_tu/, plc_mms/ y grsaa/ de la siguiente forma:

Carpeta plctu (cliente)

gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_clnt.
gestionUsuarios_xdr.

Estos archivos se agregaron en el paso 2

gestionPlctu.x
gestionPlctu.h
gestionPlctu_clnt.c
gestionPlctu_xdr.c

Carpeta plcmms (servidor, servidor1)

gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_svc.
gestionUsuarios_xdr.

Estos archivos se agregaron en el paso 2

gestionPlctu.x
gestionPlctu.h
gestionPlctu_svc.c
gestionPlctu_xdr.c

gestionSaa.x
gestionSaa.h
gestionSaa_clnt.c

Carpeta grsaa (servidor2)

gestionSaa.x
gestionSaa.h
gestionSaa _svc.c

4. Crear el código del cliente.

Sun RPC no posee un servicio binding de red global. En lugar de esto proporciona un servicio binding local llamado *portmapper*. El procesador de interfaces rpcgen permite generar templates (plantillas) del cliente y el servidor, las cuales crean las funcionalidades que logran que un cliente pueda comunicarse con un servidor utilizando el servicio de binding local *portmapper*. Para generar el código del cliente vamos a utilizar esta facilidad.

Pasos a seguir:

a. Ubicar en la carpeta plc_tu/, mediante el comando:

```
cd cliente
```

b.- Generar la plantilla cliente.c, mediante el comando:

```
rpcgen -Sc gestionUsuarios.x > cliente.c
```

c.- Generar la plantilla cliente2.c, mediante el comando:

```
rpcgen -Sc gestionPlctu.x > cliente2.c
```

c. Ubicados en la carpeta plc_tu/, cambiar los permisos para escribir en cliente.c, si es el caso, luego modificar el código generado (archivo cliente.c), introduciendo la lógica de control del programa, y los mensajes guía para orientar al usuario. En la Figura 9 se observa un ejemplo de las variables de entrada y salida de los procedimientos, y en la Figura 10, el llamado a los procedimientos que se encuentran dentro del archivo cliente.c.

```
CLIENT *clnt;
typedef enum{false,true} bool_t;
int  *result_1;
datos_sesion  abrirsesion_1_arg;
datos_usuario  *result_2;
int  consultarusuario_1_arg;
```

Figura 9. Variables de entrada y salida a los procedimientos en el archivo cliente.c

```
result_1 = abrirsesion_1(&abrirsesion_1_arg, clnt);
if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
}
```

Figura 10. Ejemplo del llamado a un procedimiento en el archivo cliente.c

d. Ubicados en la carpeta plc_tu/, se debe tomar las variables y la lógica generada en la función gestión_dispositivos_1 (char *host) del archivo cliente2.c e incluirla en el código del cliente.c, en el caso donde un operador CEO selecciona la opción uno u opción dos del Menú Operador, con el fin de que se realice una conexión con el nodo plc_mms (servidor1). En la Figura 11 se observa el bloque de código y las variables de entrada y salida, y el llamado del procedimiento que se encuentran dentro del archivo cliente2.c. y que debe ser copiado en el cliente.c

```
        CLIENT *clnt;
        bool_t  *result_1;
        datos_plctu  registrar_plctu_1_arg;
        datos_plctu  *result_2;
        int  consultar_plctu_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, gestion_dispositivos, gestion_dispositivos_version, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = registrar_plctu_1(&registrar_plctu_1_arg, clnt);
        if (result_1 == (bool_t *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        result_2 = consultar_plctu_1(&consultar_plctu_1_arg, clnt);
        if (result_2 == (datos_plctu *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
```

Figura 11. Lógica generada en el archivo cliente2.c

Una vez copiado el código se puede eliminar el archivo cliente2.c. El código copiado debe ser ubicado adecuadamente dentro de la lógica que ya se tiene en el archivo cliente.c

5. Crear los códigos del servidor y servidor1 en el nodo plc_mms/.

De acuerdo con el requerimiento inicial es responsabilidad del programador del servicio implementar el código del servidor, para lo cual se utilizarán las plantillas de las Rutinas de Servicio.

Pasos que seguir:

Crear el código del servidor.

a. Ubicar el directorio servidor, mediante el comando:

```
cd plc_mms
```

b. Generar la plantilla del servidor.c, mediante el comando:

```
rpcgen -Ss gestionUsuarios.x > servidor.c
```

c. Implementar la lógica de las operaciones indicadas en el servidor.c

Crear el código del servidor1.

a. Generar la plantilla servidor1.c, mediante el comando:

```
rpcgen -Ss gestionPlctu.x > servidor1.c
```

b. Generar la plantilla cliente3.c, mediante el comando:

```
rpcgen -Sc gestionSaa.x > cliente3.c
```

c. Abrir el archivo cliente3.c y copiar el siguiente fragmento de código (ver Figura 12):

```
        CLIENT *clnt;
        void  *result_1;
        int  notificarplcmms_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, gestion_saa, gestion_saa_version, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = notificarplcmms_1(&notificarplcmms_1_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
```

Figura 12. Código del cliente3.c que se debe copiar

d. Abrir el archivo servidor1.c y pegar el fragmento de código de la siguiente manera(ver Figura 13):

```
int *
abrirsesion_1_svc(datos_sesion *argp, struct svc_req *rqstp)
{
        static int  result;

        CLIENT *clnt;
        void  *result_1;
        int  notificarplcmms_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, gestion_saa, gestion_saa_version, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = notificarplcmms_1(&notificarplcmms_1_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */

        return &result;
}
```

Figura 13. Código que se debe copiar y adaptar.

Incluir la lógica de control para que este código trabaje adecuadamente.

e. Eliminar el archivo cliente3.c mediante el comando:

```
rm -r cliente3.c
```

6.- Crear el código del servidor2

De acuerdo con el enunciado del problema es responsabilidad del programador del servicio implementar el código del servidor2, para lo cual se utilizarán las plantillas de las Rutinas de Servicio.

Pasos a seguir:

a. Ubicarse en la carpeta grsaa/, mediante el comando:
```
cd grsaa
```

b. Generar la plantilla servidor2.c, mediante el comando:
```
rpcgen -Ss gestionSaa.x > servidor2.c
```

Un ejemplo de la implementación del procedimiento remoto del servidor2 es el siguiente (ver Figura 13):

```c
#include "gestionSaa.h"

void *
notificarplcmms_1_svc(int *argp, struct svc_req *rqstp)
{
        static char * result;

        /*
         * insert server code here
         */

        return (void *) &result;
}
```

Figura 13. Código de implementación en el archivo servidor2.c

Implementar la lógica de notificarplcmms(), que básicamente es imprimir el siguiente mensaje:
*** El dispositivo PLC_MMS con Id:xx completo sus registros y esta activo en el sistema ***

Donde xx es un código que debe ser quemado en el servidor1.c.

5. Compilar códigos

Pasos que seguir:

a. Estando en la carpeta grsaa/, desde la consola generar el archivo mkS2 a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm gestionSaa.x > mkS2
```

b. Estando en la carpeta plc_mms/, desde la consola generar el archivo mkS a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm gestioUsuarios.x > mkS
```

c. Estando en la carpeta plc_mms/, desde la consola generar el archivo mkS1 a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm gestionPlctu.x > makeS1
```

d. Estando en la carpeta plc_tu/**,** desde la consola generar el archivo mkC a través de la opción que posee la aplicación rpcgen para facilitar la compilación de los archivos fuente. Esto se logra mediante el siguiente comando:

```
rpcgen -Sm gestionUsuarios.x > mkC
```

Cambiar los permisos (en caso de que sea necesario hacerlo) para escribir en mkC.

e. Luego de haber generado los anteriores archivos, editar el mkC generado para obtener el ejecutable cliente:

Editar y modificar los siguientes parámetros en el archivo mkC

CLIENT= cliente
SERVER= Borrar valor (Debe quedar en blanco)
SOURCES_CLNT.c = cliente.c
SOURCES_CLNT.h = gestionUsuarios.h gestionPlctu.h
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = gestionUsuarios.x

TARGETS_SVC.c = gestionUsuarios_xdr.c gestionPlctu_xdr.c
TARGETS_CLNT.c = gestionUsuarios_clnt.c gestionUsuarios_xdr.c gestionPlctu_clnt.c gestionPlctu_xdr.c
TARGETS = gestionUsuarios.h gestionUsuarios_xdr.c gestionUsuarios_clnt.c gestionPlctu.h gestionPlctu_xdr.c gestionPlctu_clnt.c

Los nombres de los archivos de color rojo deben colocarlos manualmente. Esto permitirá compilar la lógica de los 2 clientes (cliente de servidor y cliente de servidor1)

f. Editar el archivo mkS generado para obtener el archivo ejecutable servidor
Ubicarse en la carpeta plc_mms/.
Editar y modificar los siguientes parámetros en el archivo mkS

CLIENT= Borrar valor (Debe quedar en blanco)
SERVER= servidor
SOURCES_SVC.c = servidor.c
SOURCES_SVC.h = gestionUsuarios.h

g. Editar el mkS1 generado para obtener el ejecutable servidor1
Ubicarse en la carpeta plc_mms/.
Editar y modificar los siguientes parámetros en el archivo makeS1

CLIENT= Borrar valor (Debe quedar en blanco)
SERVER= servidor1
SOURCES_SVC.c = servidor1.c
SOURCES_SVC.h = gestionPlctu.h gestionSaa.h

TARGETS_SVC.c = gestionPlctu_svc.c gestionPlctu_xdr.c gestionSaa_clnt.c
TARGETS_CLNT.c = gestionPlctu_xdr.c
TARGETS = gestionPlctu.h gestionPlctu_xdr.c gestionPlctu_svc.c gestionSaa_clnt.c

Los nombres de los archivos de color rojo deben colocarlos manualmente.

h. Editar el archivo mkS2 generado para el ejecutable servidor2
Ubicarse en la carpeta grsaa/.
Editar y modificar los siguientes parámetros en el archivo mkS2

CLIENT= Borrar valor (Debe quedar en blanco)
SERVER= servidor2
SOURCES_SVC.c = servidor2.c
SOURCES_SVC.h = gestionSaa.h

b. Compilar los archivos fuentes:

Para compilar los códigos fuentes del cliente desde la carpeta plc_tu/:
make -f mkC

Para compilar los códigos fuentes del servidor desde la carpeta plc_mms/:
make -f mkS

Para compilar los códigos fuentes del servidor1 desde la carpeta plc_mms/:
make -f mkS1

Para compilar los códigos fuentes del servidor2 desde la carpeta grsaa/:
make -f makeS2

6. Ejecutar el cliente y los servidores

Cuando se compilan los archivos fuente se generan dos ejecutables, cuyo nombre depende de los parámetros fijados en la plantilla del archivo makefile.

a. Ubicados en la carpeta grsaa/. El programa ejecutable servidor2, se puede ejecutar localmente o en otra máquina.

./servidor2

b. Ubicados en la carpeta plc_mms/. El programa ejecutable servidor1, se puede ejecutar localmente o en otra máquina.

./servidor1

c. Ubicados en la carpeta plc_mms/. El programa ejecutable servidor, se puede ejecutar localmente o en otra máquina.

./servidor

d. Ubicados desde la carpeta plc_tu/. El programa ejecutable cliente requiere como parámetro de entrada el nombre de la máquina (o dirección IP) donde está el Servidor.

./cliente localhost (o dirección IP)