Práctica 1: PROCESO DE DESARROLLO EN XDR 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 sesión

```
=== 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. Las funcionalidades del Menú operador, para esta práctica, no se deben implementar. Si el usuario elige las opciones 1 o 2, se debe desplegar el mensaje:

En construcción!

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. Si el usuario elige las opciones 1, se debe desplegar el mensaje:

En construcción!

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 donde se muestran los nodos y usuarios que interactúan con esta aplicación. Las operaciones abrir_sesion() y consultar_usuario(), deben ser implementadas en un servidor denominado plc_mms. Cada vez que se envíe una petición desde el nodo plc_tu y se reciba una petición en el nodo plc_mms, **se deben crear ecos por medio de un mensaje en pantalla** en el cual se describe cual función se está solicitando o ejecutando.

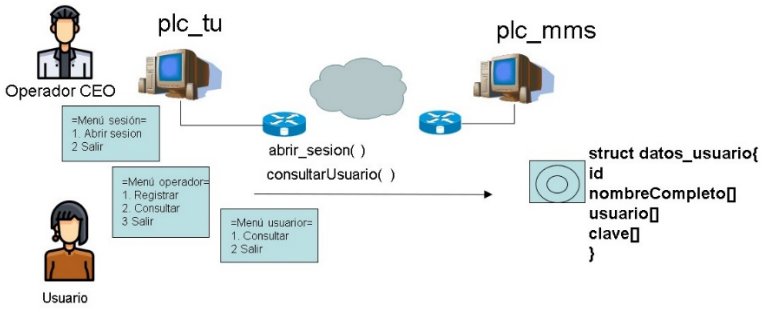

Figura 4. Diagramacde Contexto.

Antes de iniciar se debe crear la siguiente estructura de directorios (ver Figura 5):

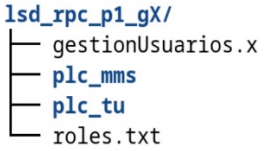

Figura 5. Estructura de directorios para desarrollar la práctica.

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_p1_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 (role cliente, role servidor).

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 el cliente debe recurrir a un servicio que le permita realizar 2 operaciones especiales. Por lo tanto, el proceso Servidor deberá publicar, mediante una definición de interface, el procedimiento que atenderá la solicitud del proceso cliente.
Para cumplir con los requerimientos del problema se recurre a la definición de 2 procedimientos, en un archivo denominado gestionUsuarios.x, como se muestra en la Figura 6.

```
/*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{
     bool abrirsesion(datos_sesion)=1;
     datos_usuario consultarusuario(int id)=2;
   }=1;
}=0x30000013;
```

Figura 6. Interface gestionUsuarios.*x*

Mediante un editor de texto cree un archivo denominado gestionUsuarios.x, e incluya el contenido que se muestra en la Figura 6, la cual corresponde a la definición de la interface remota.

2. Procesamiento de la Interface

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 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 la rutinas de aplanamiento o codificación de los datos.

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

Cliente:
gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_clnt.c
gestionUsuarios_xdr.c

Servidor:
gestionUsuarios.x
gestionUsuarios.h
gestionUsuarios_svc.c
gestionUsuarios_xdr.c

3. 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 el directorio cliente, mediante el comando:

```
cd plc_tu
```

b. Generar la plantilla del cliente, mediante el comando:

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

c. Ubicados en el subdirectorio 'plc_tu', editar el código generado (archivo cliente.c), introduciendo la lógica de control del programa, y los mensajes guía para orientar al usuario. En la Figura 7 se observa un ejemplo de las variables de entrada y salida de los procedimientos, y en la Figura 8, el llamado a los procedimientos que se encuentran dentro del archivo cliente.c.

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

Figura 6. Variables de entrada y salida a los procedimientos en el archivo cliente.c

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

Figura 7. Ejemplo del llamado a un procedimiento en el archivo cliente.c

4. Crear el código del Servidor.

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:

a. Ubicar el directorio del servidor, mediante el comando:

```
cd plc_mms
```

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

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

c. Ubicados en el subdirectorio servidor, cambiar los permisos para escribir en servidor.c (si es necesario hacerlo), luego modificar el código generado (archivo servidor.c), introduciendo la lógica

de cada uno de los procedimientos, un ejemplo de uno de los procedimientos se observa en la Figura 8.

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

    /*
     * insert server code here
     */

    return &result;

}
```

Figura 8. Procedimiento abrirsesion( ).

5. Compilar códigos

Pasos que seguir:

a. Estando en la carpeta **plc_mms,** desde la consola generar el archivo **makefile** 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 > mkS
```

Estando en la carpeta **plc_tu,** desde la consola generar el archivo **makefile** 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
```

b. Luego de haber generado los anteriores archivos, se edita el **makefile** generado para el servidor y el 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

Nota: Para entornos Debian 11 o Debian 12, adicionar las siguientes opciones de compilación:

```
# Compiler flags

CFLAGS += -g -I/usr/include/tirpc
LDLIBS += -lnsl -ltirpc
RPCGENFLAGS =

CC=gcc
```

Editar y modificar los siguientes parámetros en el archivo makeS

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

Ver nota sobre entornos Debian 11 o Debian 12.

c. 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
```

6. Ejecutar el cliente y el servidor

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 plc_mms. El programa Servidor, se puede ejecutar localmente o en otra máquina.

```
./servidor
```

b. Ubicados desde la carpeta plc_tu. El programa Cliente requiere como parámetro de entrada el nombre de la máquina (o dirección IP) donde está el NS.

```
./cliente localhost (o dirección IP)
```