UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

instituto de astronomía
unam

Instituto de astronomía

Publicaciones Técnicas

“Reporte Técnico” **RT-2009-05**

LA BIBLIOTECA COMOAN 0.10.

S. Zazueta.

LA BIBLIOTECA COMOAN 0.1.0

AUTOR:
Salvador Zazueta Rubio.
VERSIÓN DEL DOCUMENTO: 1.0

Tabla de Contenido

RESUMEN

Documento de la biblioteca COMOAN. La biblioteca implementa un protocolo de comunicación entre procesos orientado a los instrumentos del OAN. Las funciones de la biblioteca simplifican la programación de servidores que cumplen con una serie de requerimientos definidos previamente. La biblioteca se desarrolló en lenguaje "C" y se ejecuta en el operativo Linux.

INTRODUCCIÓN

En este trabajo se presenta una biblioteca en lenguaje "C" que implementa el protocolo de comunicación definido en el documento [1]. La finalidad del protocolo es definir un estándar para la intercomunicación entre los diferentes instrumentos integrados a la red de control del telescopio. La biblioteca se llama COMOAN. La COMOAN simplifica el desarrollo de programas servidores que cumplen con el protocolo [1].

También se presenta un programa muy sencillo para demostrar la aplicación de la biblioteca para implementar un programa servidor.

En otro apartado se documenta las funciones de aplicación de la biblioteca (API).

La biblioteca se basa en funciones de GLIB-2.0 para aprovechar la capacidad de manejo de estructuras, colas, listas y hebras de ejecución. La plataforma de desarrollo es Linux.

La COMOAN provee una interfaz sencilla de usar en el sentido de aislar el protocolo de comunicación de las funciones que se deben ejecutar por el programa.

El usuario define una función de llamada y la biblioteca se encarga de atenderla cuando se recibe la instrucción adecuada a través del canal de comunicación con otros procesos.

LA BIBLIOTECACOMOAN

REQUERIMIENTOS

Para compilar la biblioteca "**comoan**" así como el programa servidor de ejemplo se requieren las herramientas estándar usadas por los programas GNU, a saber, configure, gcc, make, etc.

También se requiere la biblioteca GLIB en su versión mayor o igual a 2.12. La documentación de la biblioteca GLIB se puede ver en la siguiente liga

http://library.gnome.org/devel/glib/

La mayoría de las distribuciones de Linux ya tienen instaladas las herramientas mencionadas. Los "scripts" de configure verifican si ya están instaladas las bibliotecas necesarias. A manera de ejemplo:

> ./configure --prefix=dir1

En general se usa a lo largo del trabajo al signo " >" seguido de itálicas como un indicador de una línea de instrucción de una terminal de consola linux (xterm, bash, sh).

ARCHIVOSFUENTE

Los archivos fuente de la biblioteca **comoan** y del programa de ejemplo son:

- libcomoan-0.1.0.tar.gz
- serv_ejem-0.1.0.tar.gz

y se pueden obtener de la dirección: http://www.astrosen.unam.mx/~chava/comoan

FUNCIONAMIENTODE UN PROGRAMASERVIDOR

A grandes rasgos el funcionamiento de un programa servidor basado en la biblioteca **comoan** es el siguiente:

- Inicio de variables.
- Agregar funciones de atención para las instrucciones recibidas por el canal de comunicación (socket[1]).
- Crear una o más hebras de ejecución para rutinas de control y supervisión del instrumento que maneja el servidor.
- Ejecutar la rutina de **g_main_loop_run** de GLIB para esperar y ejecutar las instrucciones que

llegan por el **socket**. Simultáneamente ejecutar las rutinas de control y comunicación con el instrumento manejado por el servidor.

Las instrucciones del servidor se reciben como cadenas ASCII. La biblioteca separa en "tokens" la cadena recibida y los pone en una cola (pila) de ejecución. Posteriormente la biblioteca inicia el proceso de decodificación y ejecución de las instrucciones.

Las instrucciones son definidas por el usuario de la biblioteca, a cada "token" definido se le asigna una función de atención.

PARA COMPILAR LA BIBLIOTECA

Para compilar la biblioteca se usan las llamadas estándar.

> ./configure --prefix=$HOME/instrumentacion LDFLAGS=-L\ $HOME/instrumentacion/lib

> make

> make install

Si el directorio $HOME/instrumentacion existe, entonces las instrucciones anteriores crearán e instalarán los archivos de la biblioteca COMOAN en los directorios $HOME/instrumentacion/include y $HOME/instrumentacion/lib.

LA VARIABLE "INSTRUMENTACION"

Algunas funciones de la biblioteca utilizan la variable de ambiente INSTRUMENTACION. Esta variable debe apuntar a la trayectoria donde reside el directorio "**com**" definido en [1].

Por omisión la biblioteca ajusta el valor de esta trayectoria a */usr/local/instrumentacion,* así pues, es necesario que el proceso que use la biblioteca tenga privilegios de lectura y escritura sobre este directorio, de lo contrario la biblioteca aborta el proceso con un error.

Para usar los ejemplos que se detallan a continuación es necesario ajustar la variable al directorio de HOME del usuario.

> mkdir $HOME/instrumentacion

> export INSTRUMENTACION=$HOME/instrumentacion

La localización del directorio puede ser cualquiera en el sistema de archivos, sólo está restringida por los permisos del usuario que ejecuta el programa sobre el directorio en cuestión. Por simplicidad se escogió $HOME/instrumentacion.

RECOMENDACIONES GENERALES PARA PROGRAMAS SERVIDOR

La biblioteca **comoan**, así como cualquier otro programa que use hebras de ejecución necesita usar herramientas como "mutexes" para restringir el acceso a las variables comunes y a las partes críticas del programa. En general se recomienda el uso de las herramientas de la biblioteca GLIB para realizar estas funciones.

Para el intercambio de mensajes entre hebras de ejecución se puede usar las funciones de colas asíncronas (GasyncQueue) del GLIB.

Otra recomendación importante es **no usar** funciones que bloqueen la ejecución del programa en las funciones de atención a instrucciones de los programas servidores. A manera de ejemplo, suponga que tiene las siguientes funciones de atención en su programa y atienden las instrucciones "EDOTEL" y "ACT" respectivamente.

fn_edo_telescopio

fn_mueve_telescopio

Es obvio que estas funciones interactuan con el instrumento y requieren comunicación con el mismo, las llamadas del canal de comunicación se pueden bloquear por diferentes razones: el socket no está listo, hay retransmisiones pendientes, etc. Si la función en cuestión hace uso de las rutinas de comunicación del operativo y esta rutina se bloquea, la hebra del programa servidor encargada de la atención de funciones de llamada se puede bloquear. El programa servidor no se bloqueará pero los resultados de las operaciones del instrumento son impredecibles debido a la pérdida de la secuencia en las comunicaciones servidor-instrumento, además de posibles retrasos en la ejecución del programa.

Así pues, es necesario aislar las rutinas que se pueden bloquear y colocarlas en una hebra de ejecución (Gthread) y, como se mencionó arriba utilizar las colas de mensajes entre hebras y banderas o "mutexes" para mantener la sincronización de las comunicaciones y/o las rutinas de control del instrumento.

PROGRAMA DE SERVIDOR DE EJEMPLO

PARA CREAR UN SERVIDOR DE EJEMPLO

El archivo **serv_ejem-0.1.0.tar.gz** sirve para crear un programa servidor muy sencillo que ejecuta algunas instrucciones y cumple con los requerimientos de [1].

El servidor puede interpretar una serie de instrucciones recibidas a través del canal de comunicación "**socket**"[1].

La definición de las instrucciones se puede ver en el archivo "src/instrucciones.c".

Previo a la ejecución del programa es necesario poner las variables de ambiente **LD_LIBRARY_PATH** e **INSTRUMENTACION**.

Suponiendo que el directorio de instalación de la biblioteca **comoan** es

prefix=$HOME/instrumentacion, las instrucciones que se detallan a continuación crearán y ejecutarán el programa de ejemplo "**serv_ejem**".

```
> tar xvfz serv_ejem-0.1.0.tar.gz
> cd serv_ejem-0.1.0
> export INSTRUMENTACION=$HOME/instrumentacion
> ./configure --prefix=$INSTRUMENTACION   LDFLAGS=$INSTRUMENTACION/lib
> make
> export LD_LIBRARY_PATH=$INSTRUMENTACION/lib
> src/serv_ejem --hostip=localhost
```

No es necesario hacer el "make install" puesto que se trata de un programa de prueba.

PARA MANDAR INSTRUCCIONES AL SERVIDOR

Para probar al servidor podemos abrir una terminal de consola y mandar instrucciones al programa por medio de la instrucción "**hose**" del paquete netpipes de Linux (Slackware).

Por ejemplo para pedir el estado se puede hacer:

```
> export SOCKET1=$HOME/instrumentacion/com/serv_ejem/com/socket
> echo "ESTADO" | hose -unix- $SOCKET1 --slave
```

Nótese que el valor del campo "host" del programa "hose" es -unix-, esto permite crear una conexión de tipo socket AF_LOCAL con el programa servidor [1].

Un ejemplo de la función suma:

> echo "SUMA 123.45 6.78" | hose -unix- $SOCKET1 --slave

El servidor debe regresar el resultado de la suma de los dos números.

Más adelante se detalla el funcionamiento y las instrucciones de la biblioteca para implementar la función SUMA del servidor.

EXPLICACIONDEL PROGRAMADE EJEMPLO(SERV_EJEM)

A continuación se presenta una explicación de las partes más relevantes de los listados "src/serv_ejem.c" y "src/instrucciones.c" del paquete serv_ejem.
La función "**main**" utiliza la definición estándar de un programa basado en la biblioteca **glib,** a este listado básico se le añaden algunas rutinas donde se definen las tareas del programa de ejemplo. A continuación se presenta el listado de la función **main**.

```
int main(int argc, char *argv[])
{
        int iRes;
        GMainLoop *mainLoop;
        GThread *thread;

        // lee las opciones de la línea de mando (ver listado en serv_ejem.c)
        decod_opciones( argc, argv);

        // para iniciar las hebras de ejecución (del glib)
        g_thread_init(NULL);
        mainLoop = g_main_loop_new( NULL, TRUE);   // (del glib)

        // Crea la hebra que maneja la comunicación con el instrumento
        // en este ejemplo sólo se ejecuta  indefinidamente "fn_comunicacion".
        thread = g_thread_create(fn_comunicacion, GINT_TO_POINTER(900),
                                 TRUE, NULL);

        // inicio de las funciones de aplicación (ver listado en serv_ejem.c)
        agrega_fuentes_eventos(mainLoop);

        g_main_loop_run( mainLoop ); // (del glib)
        return 0;
}
```

Otras funciones importantes de "serv_ejem.c" son: **agrega_fuentes_eventos** y **serv_ejem_instrucciones_inicia**.

```
void
agrega_fuentes_eventos(GMainLoop *loop)
{
        char *pc;
        char pcSock[1024];
        char pcAux[1024];

        // Se crea la trayectoria serv_ejem/com
        crea_archivos_dir_com(pcProgNombre);

        // agrega las funciones que debe atender el servidor
        // definidas en (instrucciones.c)
        serv_ejem_instrucciones_inicia();

        // crea el servidor de instrucciones a través del socket
        pc = dame_trayectoria_dir_com();
        strcpy( pcAux, pc);
        strcat( pcAux, "socket");
        strcpy( pcSock, pcAux );
        inicia_servidor(pcSock);

        // manejador de eventos y actualización del archivo de estado
        g_timeout_add( 100, maneja_eventos , NULL);
}
```

La función "**serv_ejem_instrucciones_inicia**" se encuentra en el archivo "src/instrucciones.c" junto con las definiciones de las funciones que atiende el programa servidor. El listado de la función se presenta a continuación.

```
void
serv_ejem_instrucciones_inicia(void)
{
        // se debe llamar para iniciar la cola de "tokens"
        ins_decod_inicia();

        ins_decod_agrega_instruccion("SUMA",fn_suma );
        ins_decod_agrega_instruccion("ESTADO",fn_edo );
        ins_decod_agrega_instruccion("HABILITA",fn_hab );
        ins_decod_agrega_instruccion("DESHABILITA",fn_deshab );
        ins_decod_agrega_instruccion("SALIR",fn_salir );
}
```

La llamada:
ins_decod_agrega_instruccion("SUMA",fn_suma);
agrega a la lista de instrucciones la función "fn_suma" que debe ser ejecutada cuando se

reciba en el "socket" la palabra "SUMA".
El prototipo de función que se debe pasar a la función **ins_decod_agrega_instruccion** debe cumplir con el siguiente formato:

```
void
fn_suma(gpointer data)
{
  // ejec. la fn. suma
}
```

En el parámetro "data" el servidor (la biblioteca) pone un apuntador al canal de comunicación, este apuntador es del tipo "GIOChannel". Es necesario que la función verifique que se trate de un canal válido, es decir, distinto del apuntador "NULL" como se verá más adelante. Este es el funcionamiento por omisión de la biblioteca, siguiendo como base a la función **ins_decod_ejecuta_con_source** el usuario puede implementar modificaciones que permitan otras capacidades.
La función suma ejecuta la suma de los dos números y regresa el resultado por el canal de comunicación. El formato de la instrucción es:

SUMA n1 n2

donde n1 y n2 son los números reales a sumar.
A continuación se explica el listado de la función suma:

```
void
fn_suma(gpointer data)
{
        char pc[100];
        unsigned int ui;
        double d=0.0;
        double d1=0.0;

        g_print("fn suma r\n");
        if( ins_decod_checa_token() != TOKEN_ES_NUMERO){
                g_print("faltan parametros\n");
                return;
        }
        d = ins_decod_saca_token_numero();

        if( ins_decod_checa_token() != TOKEN_ES_NUMERO){
                g_print("faltan parametros\n");
                return;
        }
        d1 = ins_decod_saca_token_numero();

        g_snprintf(pc,sizeof(pc)-1,"SUMA=%f\n", d+d1);
        g_printf("%s",pc);

        if( data != NULL ){
                g_io_channel_write((GIOChannel *)data,pc,strlen(pc),&ui );
        }
}
```

En el listado de la fn_suma se puede ver otras llamadas importantes.

```
if( ins_decod_checa_token() != TOKEN_ES_NUMERO){
                g_print("faltan parametros\n");
                return;
  }
  d = ins_decod_saca_token_numero();
```

Este juego de instrucciones sirve para verificar si el siguiente parámetro de la pila de ejecución es un número y posteriormente sacarlo de la lista para usarlo en el programa.

El siguiente conjunto de instrucciones sirve para enviar la respueta a través del canal de comunicación (socket).

```
  if( data != NULL ){
        g_io_channel_write((GIOChannel *)data,pc,strlen(pc),&ui );
  }
```

DESCRIPCIÓNDE LAS FUNCIONESDE LA BIBLIOTECACOMOAN

En esta sección se describe la operación y la sintaxis de las funciones de la biblioteca.

Los encabezados con los prototipos de las funciones se encuentran en el archivo "$prefix/include/comoan.h".

FUNCIÓNcrea_archivos_dir_com

Prototipo:

void
crea_archivos_dir_com(char *inst);

DESCRIPCIÓN:
Crea la trayectoria $INSTRUMENTACION/com/(*inst) de conformidad con [1].
Usa la variable de ambiente INSTRUMENTACION, si la variable no está asignada el programa utiliza por omisión INSTRUMENTACION=/usr/local/instrumentacion. Si el usuario no tiene privilegios de lectura, escritura sobre este directorio el programa termina su ejecución con un error.
También verifica el archivo de "candado"[1] y ejecuta la función "flock" para verificar si se está ejecutando otra instancia del programa servidor.

Esta función es la primera que se debe llamar para implementar un programa servidor. Vea el ejemplo en la sección "**PARA CREAR UN SERVIDOR DE EJEMPLO**".

FUNCIÓNdame_trayectoria_dir_com

Prototipo:

char *
dame_trayectoria_dir_com(void);

DESCRIPCIÓN:
Regresa una cadena de caracteres con el contenido de la trayectoria "com".
Se debe llamar previamente a la función "**crea_archivos_dir_com**" para poder llamar a esta función.

FUNCIÓNinicia_servidor

Prototipo:
void
inicia_servidor(char *arch);

DESCRIPCIÓN:
Inicia el servidor.
Abre el canal de comunicación "arch".
Una vez ejecutado **g_main_loop_run**(mainLoop) empieza a recibir instrucciones
a través del canal "arch" y a llamar las funciones de atención que se agregaron a la lista de ejecución.

Por ejemplo:

```
crea_archivos_dir_com("ruca");
ruca_instrucciones_inicia();

// crea el servidor de instrucciones a través del socket
pc = dame_trayectoria_dir_com();
strcpy( pcAux, pc);
strcat( pcAux, "socket");
strcpy( pcSock, pcAux );
inicia_servidor(pcSock);
```

pone el canal $INSTRUMENTACION/com/ruca/socket a recibir instrucciones en cuanto se empiece a ejecutar la función **g_main_loop_run.**
Ver programa de ejemplo en la sección "**PARA CREAR UN SERVIDOR DE EJEMPLO**".

FUNCIÓNdir_com_abre_estado

Prototipo:

FILE *
dir_com_abre_estado(void);

DESCRIPCIÓN:
Regresa un apuntador de tipo FILE* (stdio.h), este apuntador sirve como parámetro a la función que agrega campos al archivo de "com"/estado.

Así pues, una secuencia para la actualización del archivo de **estado** puede ser:

```
f = dir_com_abre_estado(void);
dir_com_agrega_campo_estado(f, "AR", "10 10 10");
dir_com_agrega_campo_estado(f, "MASA_AIRE", masa_aire_str() );
dir_com_cierra_estado(f);
```

FUNCIÓNdir_com_agrega_campo_estado

Prototipo:

```
void
dir_com_agrega_campo_estado(FILE *f,
                            char *pcCampo,
                            char *pcValor);
```

DESCRIPCIÓN:
Agrega al archivo de estado (FILE*) el campo con el valor adecuado. El formato que genera la función es de tipo XML.
Por ejemplo:

dir_com_agrega_campo_estado(f, "AR", "10 10 10");

le agrega al archivo "f" la siguiente cadena:

<AR> 10 10 10 </AR>

Vea la descripción de **dir_com_abre_estado.**

FUNCIÓNdir_com_cierra_estado

Prototipo:

```
void
dir_com_cierra_estado(FILE *f);
```

DESCRIPCIÓN:
Cierra el archivo FILE*. Es una función que encapsula la función "fclose".

Se usa en conjunto con las funciones **dir_com_abre_estado** y **dir_com_agrega_campo_estado.** Ver la descripción de la primera.

FUNCIÓNdir_com_agrega_a_bitacora

Prototipo:

```
int
dir_com_agrega_a_bitacora(char *pcMsg );
```

DESCRIPCIÓN:
Agrega al archivo de bitácora[1] la cadena pcMsg.

Regresa 0 (cero) si no hubo error.

FUNCIÓNins_decod_inicia

Prototipo:

```
void
ins_decod_inicia(void);
```

DESCRIPCIÓN:
Se debe llamar para iniciar las cola de "tokens" de instrucciones del servidor y la lista de instrucciones que atiende el servidor.
Esta función se debe llamar antes que la instrucción para agregar "instrucciones", valga la redundancia, al servidor (**ins_decod_agrega_instruccion**).
Ver ejemplo en el listado src/instrucciones.c del programa serv_ejem.

FUNCIÓNins_decod_agrega_instruccion

Prototipo:

```
void
ins_decod_agrega_instruccion(char *pcInstruccion, void (*pfn)() );
```

DESCRIPCIÓN:
Agrega a la lista de instrucciones la función "pfn". La función será atendida cuando el servidor reciba la instrucción "pcInstruccion" a través del canal de comunicación (socket).
Es necesario haber llamado previamente a la función **ins_decod_inicia** para usarla.
Ejemplo:

```
ins_decod_agrega_instruccion("SUMA", fn_suma );
```

La función "fn_suma" se ejecutará cuando el servidor reciba la palabra "SUMA" a través del

canal de comunicación (socket).

La forma de la función debe ser:

```
void fn_suma(gpointer data)
{
      // instrucciones ...
}
```

En el apuntador "data" el servidor coloca un apuntador al canal de comunicación. Asi pues, para contestar algo al "cliente" la función debe usar la siguiente llamada:

```
if( data != NULL ){
     g_io_channel_write((GIOChannel *)data,pc,strlen(pc),&ui );
}
```

donde "pc" es un apuntador de tipo (char *) que contiene la respuesta para el "cliente".

FUNCIÓNins_decod_saca_token

Prototipo:

```
char *
ins_decod_saca_token(void);
```

DESCRIPCIÓN:
Saca el siguiente token de la cola de ejecución.
Puede servir para obtener parámetros de tipo (STRING, NUMERO) o simplemente para desechar el siguiente "token" en la lista.
Regresa un apuntador a una cadena de caracteres con el valor del siguiente "token".

FUNCIÓNins_decod_checa_token

Prototipo:

```
int
ins_decod_checa_token(void);
```

DESCRIPCIÓN:
Regresa un entero con el tipo del próximo "token" en la cola de ejecución.
El valor de regreso puede ser:

- TOKEN_NULL 0
- TOKEN_ES_NUMERO 1
- TOKEN_ES_STRING 2

FUNCIÓNins_decod_saca_token_numero

Prototipo:

```
double
ins_decod_saca_token_numero();
```

DESCRIPCIÓN:

Saca de la cola de "tokens" un número doble. Antes de llamar esta función se debe asegurar que el siguiente "token" es un número. Por ejemplo:

```
if( ins_decod_checa_token() == TOKEN_ES_NUMERO){
    d = ins_decod_saca_token_numero();
}
```

Normalmente los programas servidores usan esta secuencia para obtener parámetros que se usarán en el programa.

FUNCIONESAUXILIARES

Este grupo de funciones no se usan directamente, es decir, son utilizadas internamente por la biblioteca. Sin embargo, el usuario puede implementar otras facilidades o preproceso de las instrucciones utilizando como base estas funciones y los listados de la biblioteca.

FUNCIÓNins_decod_ejecuta_con_source

Prototipo:

```
void
ins_decod_ejecuta_con_source(char *pc, GIOChannel *source);
```

DESCRIPCIÓN:

No se debe llamar directamente.

Esta función se encarga de ejecutar las instrucciones contenidas en la cadena "**pc**". Genera una cola de tokens y luego los va procesando para llamar a las funciones de atención que el usuario agregó con la función **ins_decod_agrega_instruccion.**

EL parámetro **source** sirve para pasar el canal de comunicación a las funciones de atención.

FUNCIONES PARA IMPLEMENTAR UN CLIENTE

Las funciones siguientes sirven para implementar un programa cliente de un servidor. Es decir, establecer comunicación con un servidor y enviar y recibir mensajes.

Estas funciones pueden servir de base para la implementación de un servidor de servidores.

FUNCION dir_com_inicia_cliente

Prototipo:

int

dir_com_inicia_cliente(char *inst);

DESCRIPCIÓN:

Esta función sirve para iniciar las variables de un cliente de comunicación con un programa

servidor. La cadena "inst" debe contener el nombre de un servidor. Por ejemplo: "serv_ejem" que es el servidor de ejemplo.

Esta función usa la variable de ambiente INSTRUMENTACION para la trayectoria "com", si no está asignada la variable, el programa usa por omisión la trayectoria

/usr/local/instrumentacion.

La función regresa 0 si no hay error.

FUNCIONdir_com_manda_msg_servidor

Prototipo:

```
int
dir_com_manda_msg_servidor(char *pcMsg,
                           char *pcRespuesta,
                           int nbytes);
```

Envía un mensaje a un programa servidor. Previamente se debe haber establecido el servidor al que se debe enviar el mensaje por medio de la función **dir_com_inicia_cliente.**

La instrucción a enviar se debe poner en la cadena **pcMsg**.

En el apuntador a una cadena "**pcRespuesta**" se regresan como máximo "**nbytes**" con la respuesta del servidor si la hay.

La función regresa 0 si no hay error y un número diferente de cero para indicar lo contrario.

Ejemplo:

```
dir_com_inicia_cliente("serv_ejem");
dir_com_manda_msg_servidor("ESTADO", respuesta, 100);
g_print("EDO:%s", respuesta);
dir_com_manda_msg_servidor("SUMA 10.5 20.5", respuesta, 100);
g_print("SUMA=%s", respuesta);
```

En el ejemplo, se establece la identidad del servidor al que hay que mandar mensajes y posteriormente se le envian los mensajes "ESTADO" y "SUMA". Cabe mencionar que se pudo mandar ambos mensajes en una sola cadena, sin embargo, de esta manera se ejemplifica que sólo hay que llamar a **dir_com_inicia_cliente** una sola vez en los programas o cuando se necesite establecer comunicación con otro servidor.

REFERENCIAS

[1] Zazueta Rubio S. , "PROTOCOLO DE PROGRAMAS SERVIDORES DE INSTRUMENTACION", Reporte Técnico del IA UNAM RT-2009-04, 2009

[2] "GLIB Reference Manual", http://library.gnome.org/devel/glib/

Comité Editorial de Publicaciones Técnicas
Instituto de Astronomía
UNAM

M.C. Urania Ceseña
Dr. Carlos Chavarria
M.C. Francisco Murillo

Observatorio Astronómico Nacional
Km. 103 Carretera Tijuana-Ensenada
22860 Ensenada B.C.
editorial@astrosen.unam.mx