Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Tesis de Licenciatura

GNet Biblioteca de Red

Integrantes

Myriam Cinthia Ródenas
María Martha Rodríguez

Director

Lic. Roberto Bevilacqua

Agradecimiento

Myriam Cinthia Ródenas

Agradezco a Raúl, por la infinita paciencia que tuvo durante todas mis ausencias y porque siempre me brindó su apoyo incondicional, lo que hizo más llevadero mi estudio.

Agradezco a Hernán, Damián y Alan, porque sufrieron mi necesidad de estudio y finalización de todos los trabajos realizados durante mi carrera.

Agredezco a todos mis animalitos, por la callada compañía que me brindaron mientras estudiaba.

Agradezco a mis familiares y amigos, porque supieron comprender mis repetidas ausencias.

Y finalmente, agradezco a mis padres, porque me inculcaron desde muy chica la posibilidad que me brindaba el estudio para desarrollarme como persona.

María Martha Rodríguez

A mi esposo Robert le agradezco, su comprensión, su paciencia, su aliento y el estar siempre a mi lado....

A Sam, mi “negro” y a Jack, nuestro cocker, les adgradezco su compañía en cada uno de los momentos.

A mi hermana MariAle le agradezco, el haber comprendido mis ausencias y su apoyo.

A mis Amigos les agradezco porque entendieron mis inmumerables ausencias.

Y finalmente, no por eso menos importante, agradezco a mi Mamá y a mi Papá porque me brindaron todas las posibilidades para que yo sea lo que soy, en especial, una persona de bien.

Dedicatoria

Myriam Cinthia Ródenas

A Raúl, Hernán, Damián, Alan, mis animalitos, mis familiares y amigos, mis padres.

María Martha Rodríguez

A los que creen en mí... mi Mamá, mi Papá, mi hermana MariAle, mis Amigos, a la memoria de Sam, Jack y a mi amor Robert.

Indice

GNet Biblioteca de Red

Resumen

En la actualidad las redes están conformadas por diferentes arquitecturas y, por ende, la coexistencia de distintos sistemas operativos es inevitable. Es por ello que el desarrollo de software a nivel mundial no escapa a esta realidad y se hace imprescindible la implementación de código abierto y portable a las distintas plataformas.

Este es el motivo por el cual consideramos necesario implementar bibliotecas que encapsulen la plataforma utilizada brindando a los desarrolladores de software total transparencia.

En esta tesis presentamos y desarrollamos una biblioteca de protocolos de red con las características descriptas.

Abstract

Nowadays, networks are integrated by different architectures and coexistence among different operating systems should not be ignored. That is why software development around the world is not out of this scope and the implementation of open and portable sources on different platforms is essential.

Therefore, we consider necessary to implement libraries that encapsulate the platform use offering software developers total transparency.

In this thesis we present and develop a network library with the described characteristics.

CAPITULO I

Introducción

Debido a la tendencia mundial de desarrollo de software, actualmente se hace cada vez más imprescindible crear código abierto y portable a los distintos sistemas operativos y arquitecturas.

En la actualidad, las redes están conformadas por diferentes arquitecturas y, por ende, la coexistencia de distintos sistemas operativos es inevitable. La interconectividad entre estos sistemas hace cada vez más necesario permitir, en forma transparente, el uso de protocolos de red por parte de los programadores, para el correcto desarrollo de sus aplicaciones.

Por tal motivo, consideramos necesario implementar bibliotecas de protocolos de red existentes, para el uso de las mismas por parte de los desarrolladores de software.

CAPITULO II

Estado del Arte

Luego de haber realizado una búsqueda exhaustiva en internet, encontramos un proyecto denominado GNet – Biblioteca de Protocolos de Red.

El mismo consiste en una biblioteca de diferentes protocolos de red construida sobre GLib. Dicha biblioteca permite a los desarrolladores de software crear aplicaciones haciendo uso de la operatoria de red con total transparencia.

Actualmente soporta los siguentes protocolos:

- TCP "client" and "server" sockets
- UDP and IP Multicast
- Internet address abstraction
- Asynchronous socket IO
- Asynchronous DNS lookup
- Byte packing and unpacking
- URLs (Experimental)
- Server and Conn objects (Experimental)
- SOCKS support (Experimental)

A continuación se da una lista de las aplicaciones que ya utilizan GNet:

- Jungle Monkey - Distributed File Sharing Program

JUNGLE MONKEY

http://www.junglemonkey.net

Jungle Monkey (JM) es un programa para compartir archivos distribuidos. Los usuarios se unen a distintos canales donde reciben avisos sobre los recursos disponibles. Estos recursos pueden incluir archivos, grupos de chat, grupos de búsquedas y otros canales. Los usuarios pueden enviar avisos sobre sus propios archivos, grupos de chat, grupos de búsqueda, y canales que creen. Jungle Monkey (JM) es unicamente para Unix. Puede ser portado a Windows.

Jungle Monkey es parte de un proyecto de investigación en el Departamento de EECS de la Universidad de Michigan. En este momento se busca agregar 'end-host multicast', búsqueda de clave distribuída, y estructura de datos distribuída. El lider de éste desarrollo es David Helder de la Universidad de Michigan.

- Gabber - Messaging/Chat Program

http://gabber.sourceforge.net/

Gabber es un cliente GNOME (código abierto y gratuito) para un sistema de mensajes instantáneos llamado Jabber. Jabber (código abierto y gratuito) es un sistema de mensajes instantáneos distribuido. No se basa en un solo servidor y el protocolo está bien documentado. Jabber permite comunicación con diferentes sistemas de mensajes instantáneos, incluyendo ICQ y AIM. Hoy en día existen diferentes clientes Jabber, pero no otros clientes GNOME como aquí se describe. Afortunadamente Gabber sirve como un cliente GNOME Jabber robusto.

- Ximian Installer and Red Carpet - Software Package Management

http://www.ximian.com

Ximian™ Red Carpet™ es el software líder en soluciones de administración para desktops Linux. La organización intuitiva de los canales Red Carpet, la resolución de conflictos y dependencias automáticas, hacen que, con el mismo sea fácil instalar, modificar y mantener software a través de Internet desde Ximian. Específicamente diseñado para organizaciones que tienen ambientes heterogéneos, Red Carpet maneja una gran variedad de distribuciones Linux, incluyendo Red Hat, SuSE, Mandrake, Debian y otros.

- GnomeICU - Gnome ICU Client

http://gnomeicu.sourceforge.net

GnomeICU es uno de los programas de ICQ más populares en todo el mundo basados en UN*X.

El chat se ha convertido es uno de los usos más importantes de Internet después del email. La mensajería instantánea permite a los usuarios comunicarse en tiempo real con amigos, miembros de la familia y compañeros de trabajo en forma eficiente. GnomeICU soporta las siguientes características de ICQ:

- Server-side list storage
- Basic messaging
- File transfer (temporarily broken due to protocol change)
- Contact history
- Visible/Invisible/Ignore lists
- Ability to interface ICU from the command line
- Icon themes

- Gnome Chinese Checkers - board game

http://gchch.sourceforge.net/

Gnome Chinese Checkers es una implementación del juego Chinese Checkers. Incluye un server que soporta hasta 6 jugadores, cada uno de los cuáles se conecta desde su propio cliente. El juego incluye, rotación automática de jugadores y más.

La propuesta entonces es incorporar a la biblioteca GNet nuevos protocolos para permitir a los desarrolladores de software crear sus aplicaciones lo más amplia e independientemente posible con respecto al uso de redes. Incluimos a la biblioteca mencionada los siguientes protocolos de red: rexec, rsh, rcp, rlogin, telnet, tftp.

Esto formará parte del proyecto GNet actualmente en desarrollo en la Universidad de Michigan. David Helder es el lider de desarrollo del proyecto “GNet, a network library” en MESH (Michigan Engineering Software and Hardware).

CAPITULO III

GLib

GLib es una biblioteca de uso general que provee distintos tipos de datos, macros, conversión de tipos, facilidades para caracteres y archivos, etc.

Puede ser instalada sobre plataformas UNIX, Windows, OS/2 y BeOS.

Esta biblioteca fue desarrollada bajo “GNU Library General Public License (GNU LGPL)”.

Instalación de la biblioteca sobre UNIX

En UNIX, GLib utiliza el sistema standard GNU con:

- autoconf, para la configuración de paquetes y resolución de detalles de portabilidad.
- automake, para crear los makefiles que conforma con el código standard de GNU.
- libtool, para crear bibliotecas compartidas en múltiples plataformas.

La secuencia normal para compilación e instalación de la biblioteca GLib es:

```
./configure
 make
 make install
```

Las siguientes son listas de consulta dedicadas a GLib y bibliotecas relacionadas:

- http://mail.gnome.org

  Discusiones generales sobre GLib. También se puede ver los artículos sobre GTK+ y GLib o subscribirse en gtk-list@gnome.org
- gtk-devel-list@gnome.org

  Lista de discusión sobre desarrollo de GTK+ y GLib (parches, bugs, nuevas características, etc.)
- gtk-doc-list@gnome.org

  Lista de discusión sobre la documentación de GTK+ y GLib.

Tipos de Datos Básicos

Los siguientes son los tipos básicos de GLib, definidos para portabilidad y uso sencillo.

Síntesis:

```
#include <glib.h>
typedef     gboolean;
typedef     gpointer;
typedef     gconstpointer;
typedef     gchar;
typedef     guchar;

typedef     gint;
typedef     guint;
typedef     gshort;
typedef     gushort;
typedef     glong;
typedef     gulong;

typedef     gint8;
typedef     guint8;
typedef     gint16;
typedef     guint16;
typedef     gint32;
typedef     guint32;

#define     G_HAVE_GINT64
```

```
typedef    gint64;
typedef    guint64;
#define    G_GINT64_CONSTANT          (val)

typedef    gfloat;
typedef    gdouble;

typedef    gsize;
typedef    gssize;
```

Descripción:

GLib define los tipos de datos más usados. Pueden ser divididos en cuatro grupos:

- Tipos nuevos que no son parte del standard C: gboolean, gsize, gssize.
- Enteros, cuyo tamaño está garantizado sea el mismo en todas las plataformas: gint8, guint8, gint16, guint16, gint32, guint32, gint64, guint64.
- Tipos que pueden ser usados en forma mucho más fácil que sus correspondientes en standard C: gpointer, gconstpointer, guchar, guint, gushort, gulong.
- Tipos que corresponden a standard C: gchar, gint, gshort, glong, gfloat, gdouble.

Detalle de los tipos:

gboolean

typedef gint gboolean;

Tipo booleano standard. Valores posibles: TRUE o FALSE.

---

gpointer

typedef void* gpointer;

Puntero a void.

---

gconstpointer

typedef const void* gconstpointer;

Puntero a una constante.

---

gchar

typedef char gchar;

Corresponde al tipo standard de C caracter.

---

guchar

typedef unsigned char guchar;

Corresponde al tipo standard de C caracter sin signo.

---

gint

typedef int gint;

Corresponde al tipo standard de C entero. El rango de este tipo se encuentra entre G_MININT y G_MAXINT.

---

guint

typedef unsigned int guint;

Corresponde al tipo standard de C entero sin signo. El rango de este tipo se encuentra entre 0 y G_MAXUINT.

---

gshort

typedef   short   gshort;

Corresponde al tipo standard de C entero pequeño. El rango de este tipo se encuentra entre G_MINSHORT y G_MAXSHORT.

---

gushort

typedef   unsigned   short   gushort;

Corresponde al tipo standard de C entero pequeño sin signo. El rango de este tipo se encuentra entre 0 y G_MAXUSHORT.

---

glong

typedef   long   glong;

Corresponde al tipo standard de C entero grande. El rango de este tipo se encuentra entre G_MINLONG y G_MAXLONG.

---

gulong

typedef   unsigned   long   gulong;

Corresponde al tipo standard de C entero grande sin signo. El rango de este tipo se encuentra entre 0 y G_MAXULONG.

---

gint8

typedef signed char gint8;

Entero con signo de 8 bits en todas las plataformas. El rango de este tipo se encuentra entre -128 y 127.

---

guint8

typedef unsigned char guint8;

Entero sin signo de 8 bits en todas las plataformas. El rango de este tipo se encuentra entre 0 y 255.

---

gint16

typedef signed short gint16;

Entero con signo de 16 bits en todas las plataformas. El rango de este tipo se encuentra entre –32.768 y 32.767.

---

guint16

typedef unsigned short guint16;

Entero sin signo de 16 bits en todas las plataformas. El rango de este tipo se encuentra entre 0 y 65.535.

---

gint32

typedef signed int gint32;

Entero con signo de 32 bits en todas las plataformas. El rango de este tipo se encuentra entre –2.147.483.648 y 2.147.483.647.

---

guint32

typedef   unsigned   int   guint32;

Entero sin signo de 32 bits en todas las plataformas. El rango de este tipo se encuentra entre 0 y 4.294.967.295.

---

g_have_gint64

#define   G_HAVE_GINT64   1   /* siempre verdadero */

Esta macro se define si enteros de 64-bits con y sin signo están disponibles en la plataforma.

---

gint64

G_GNUC_EXTENSION   typedef   signed   long   long   gint64;

Entero con signo de 64 bits en todas las plataformas donde está disponible. El rango de este tipo se encuentra entre –9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

---

guint64

G_GNUC_EXTENSION   typedef   unsigned   long   long   guint64;

Entero sin signo de 64 bits en todas las plataformas donde está disponible. El rango de este tipo se encuentra entre 0 y 18.446.744.073.709.551.615.

G_GINT64_CONSTANT()

#define G_GINT64_CONSTANT(val) (G_GNUC_EXTENSION (val##LL))

Esta macro se utiliza para insertar un entero de 64-bit en el código fuente.

val : valor entero ej: 0x1d636b02300a7aa7U.

---

gfloat

typedef   float   gfloat;

Corresponde al tipo standard de C real de punto flotante (simple precisión). El rango de este tipo se encuentra entre G_MINFLOAT a G_MAXFLOAT.

gdouble

```
typedef   double   gdouble;
```

Corresponde al tipo standard de C real doble precisión. El rango de este tipo se encuentra entre G_MINDOUBLE to G_MAXDOUBLE.

gsize

```
Typedef   unsigned   int   gsize;
```

Entero de 32 bits sin signo que representa el tamaño de las estructuras de datos.

gssize

```
typedef   signed   int   gssize;
```

Entero de 32 bits con signo que representa el tamaño de las estructuras de datos.

Límites de los Tipos Básicos:

Métodos portables para determinar el límite de los tipos standars.

Síntesis:

```
#include <glib.h>

#define    G_MININT
#define    G_MAXINT
#define    G_MAXUINT

#define    G_MINSHORT
#define    G_MAXSHORT
#define    G_MAXUSHORT

#define    G_MINLONG
#define    G_MAXLONG
#define    G_MAXULONG

#define    G_MININT64
#define    G_MAXINT64
#define    G_MAXUINT64

#define    G_MINFLOAT
#define    G_MAXFLOAT

#define    G_MINDOUBLE
#define    G_MAXDOUBLE
```

Descripción:

Estas macros proveen un método portable para determinar los límites de algunos de los tipos standard enteros y punto flotante.

Detalles:

G_MININT

#define G_MININT INT_MIN
Valor mínimo que puede tener un gint.

G_MAXINT

#define G_MAXINT INT_MAX
Valor máximo que puede tener un gint.

G_MAXUINT

#define G_MAXUINT UINT_MAX
Valor máximo que puede tener un guint.

G_MINSHORT

#define G_MINSHORT SHRT_MIN
Valor mínimo que puede tener un gshort.

G_MAXSHORT

#define G_MAXSHORT SHRT_MAX
El valor máximo que puede tener un gshort.

G_MAXUSHORT

#define   G_MAXUSHORT   USHRT_MAX

Valor máximo que puede tener un gushort.

G_MINLONG

#define   G_MINLONG   LONG_MIN

Valor mínimo que puede tener un glong.

G_MAXLONG

#define   G_MAXLONG   LONG_MAX

Valor máximo que puede tener un glong.

G_MAXULONG

#define   G_MAXULONG   ULONG_MAX

Valor máximo que puede tener un gulong.

G_MININT64

#define   G_MININT64   ((gint64) 0x8000000000000000)

Valor mínimo que puede tener un gint64.

G_MAXINT64

#define   G_MAXINT64   ((gint64) 0x7fffffffffffffff)

Valor máximo que puede tener un gint64.

G_MAXUINT64

#define   G_MAXUINT64   ((guint64) 0xffffffffffffffff)

Valor máximo que puede tener un guint64.

G_MINFLOAT

#define   G_MINFLOAT   FLT_MIN

Valor mínimo que puede tener un gfloat.

G_MAXFLOAT

#define G_MAXFLOAT FLT_MAX

Valor máximo que puede tener un gfloat.

G_MINDOUBLE

#define G_MINDOUBLE DBL_MIN

Valor mínimo que puede tener un gdouble.

G_MAXDOUBLE

#define G_MAXDOUBLE DBL_MAX

Valor máximo que puede tener un gdouble.

Macros Standards de Uso Frecuente

Síntesis:

```
#include <glib.h>


#define    GLIB_MAJOR_VERSION
#define    GLIB_MINOR_VERSION
#define    GLIB_MICRO_VERSION


#define    G_OS_WIN32
#define    G_OS_BEOS
#define    G_OS_UNIX


#define    GLIB_CHECK_VERSION   (major,minor,micro)


#define    G_DIR_SEPARATOR
#define    G_DIR_SEPARATOR_S
#define    G_SEARCHPATH_SEPARATOR
#define    G_SEARCHPATH_SEPARATOR_S


#define    TRUE
#define    FALSE


#define    NULL


#define    MIN    (a, b)
#define    MAX   (a, b)
```

```
#define    ABS    (a)
#define    CLAMP (x, low, high)

#define    G_STRUCT_MEMBER (member_type, struct_p, struct_offset)
#define    G_STRUCT_MEMBER_P   (struct_p, struct_offset)
#define    G_STRUCT_OFFSET    (struct_type, member)

#define    G_MEM_ALIGN

#define    G_CONST_RETURN
```

<u>Descripción:</u>

Estas macros proveen características de uso frecuente.

<u>Detalles:</u>

GLIB_MAJOR_VERSION

```
#define   GLIB_MAJOR_VERSION   2
```

La mayor versión de la biblioteca GLib.

GLIB_MINOR_VERSION

```
#define   GLIB_MINOR_VERSION   0
```

La menor versión de la biblioteca GLib.

GLIB_MICRO_VERSION

```
#define   GLIB_MICRO_VERSION   6
```

La versión micro de la biblioteca GLib.

G_OS_WIN32

#define G_OS_WIN32

Esta macro sólo está definida para Windows. Se puede utilizar para código específico de Windows de la siguiente forma: "ifdef G_OS_WIN32".

G_OS_BEOS

#define G_OS_BEOS

Esta macro sólo está definida para BeOS. Se puede utilizar para código específico de BeOS de la siguiente forma: "ifdef G_OS_BEOS".

G_OS_UNIX

#define G_OS_UNÍS

Esta macro sólo está definida para UNIX. Se puede utilizar para código específico de UNIX de la siguiente forma: "ifdef G_OS_UNIX".

GLIB_CHECK_VERSION()

#define GLIB_CHECK_VERSION(major,minor,micro)

Chequea la versión de la biblioteca GLib. Retorna TRUE si la versión de la biblioteca GLib es igual o mayor que la versión ingresada.

Ejemplo 1. Chequeo de la versión de la biblioteca GLib

major :	el mayor número de versión
minor :	el menor número de versión
micro :	el número de versión micro

G_DIR_SEPARATOR

#define G_DIR_SEPARATOR

Separador de directorio. Este es '/' en el sistema operativo UNIX y '\' en el sistema operativo Windows.

G_DIR_SEPARATOR_S

#define G_DIR_SEPARATOR_S

Separador de directorio como caracter. Este es '/' en el sistema operativo UNIX y '\' en el sistema operativo Windows.

G_SEARCHPATH_SEPARATOR

#define G_SEARCHPATH_SEPARATOR

Separador de ‘search path’ . Este es ':' en el sistema operativo UNIX y ';' en el sistema operativo Windows.

G_SEARCHPATH_SEPARATOR_S

#define G_SEARCHPATH_SEPARATOR_S

Separador de ‘search path’ como caracter. Este es ':' en el sistema operativo UNIX y ';' en el sistema operativo Windows.

TRUE

#define TRUE (!FALSE)

Define el valor TRUE como un tipo gboolean.

FALSE

#define FALSE (0)

Define el valor FALSE como un tipo gboolean.

NULL

#define NULL

Define el puntero a NULL.

MIN()

#define MIN(a, b) (((a) < (b)) ? (a) : (b))

Calcula el mínimo entre *a* y *b*.

a :	valor numérico
b :	valor numérico
Retorna :	Mínimo entre a y b

MAX()

#define MAX(a, b) (((a) > (b)) ? (a) : (b))

Calcula el máximo entre *a* y *b*.

a :	valor numérico
b :	valor numérico
Retorna :	Máximo entre a y b

ABS()

```
#define ABS(a)        (((a) < 0) ? -(a) : (a))
```

Calcula el valor absoluto de a.

Ejemplos:

ABS(-10) es 10.

ABS(10) es también 10.

a :	valor numérico
Retorna :	Valor absoluto de a

CLAMP()

```
#define CLAMP(x, low, high)  (((x) > (high)) ? (high) : (((x) < (low)) ? (low) :
(x)))
```

Asegura que x esté entre los límites seteados por low y high.

Ejemplos:

CLAMP(5, 10, 15) es 10.

CLAMP(15, 5, 10) es 10.

CLAMP(20, 15, 25) es 20.

x :	valor a utilizar
low :	mínimo valor permitido
high :	máximo valor permitido
Retorna :	Valor de x clamped al rango entre low y high

G_STRUCT_MEMBER()

#define G_STRUCT_MEMBER(member_type, struct_p, struct_offset)

Retorna un miembro de la estructura en un determinado offset, utilizando el tipo dado.

Member_type :	el tipo de la estructura del campo
struct_p :	puntero a la estructura
struct_offset :	el offset del campo desde el comienzo de la estructura en bytes
Retorna :	El miembro de la estructura

G_STRUCT_MEMBER_P()

#define G_STRUCT_MEMBER_P(struct_p, struct_offset)

Retorna un puntero sin tipo de un offset dado de una estructura

struct_p :	puntero a la estructura
struct_offset :	el offset desde el comienzo de la estructura en bytes
Retorna :	Un puntero sin tipo a *struct_p* más *struct_offset* en bytes

G_STRUCT_OFFSET()

#define G_STRUCT_OFFSET(struct_type, member)

Retorna el offset en bytes de un miembro de la estructura.

struct_type :	un tipo de estructura ej: GtkWidget.
member :	un campo en la estructura ej: *window*
Retorna :	El offset del miembro desde el comienzo de *struct_type*.

G_MEM_ALIGN

#define G_MEM_ALIGN

Indica el número de bytes donde la memoria se alinea en la arquitectura actual.

G_CONST_RETURN

#define G_CONST_RETURN

Si G_DISABLE_CONST_RETURNS está definido, esta macro no expande nada. Por defecto, esta macro expande a const. Esta macro debe ser usada en lugar de const para funciones que retornan una valor que no debe ser modificado. El objetivo de esta macro es permitir la opción de habilitar const para devolver una cadena de caracteres por defecto. Esta macro sólo debe ser usada para devolver valores y para parámetros de salida, no tiene sentido para parámetros de entrada.

Macros de Conversión de Tipos

Síntesis:

```
#include <glib.h>

#define    GINT_TO_POINTER          (i)
#define    GPOINTER_TO_INT          (p)
#define    GUINT_TO_POINTER         (u)
#define    GPOINTER_TO_UINT         (p)
#define    GSIZE_TO_POINTER         (s)
#define    GPOINTER_TO_SIZE         (p)
```

Descripción:

Muchas veces GLib, GTK+ y otras bibliotecas permiten pasar "datos del usuario" a un callback, en la forma de un puntero void. A veces es preferible pasar un entero en lugar de un puntero. Se puede alocar un entero en la forma:

```
int *ip = g_new (int, 1);
*ip = 42;
```

Pero el inconveniente de esto es que hay que liberar memoria más tarde.

Los punteros son siempre de por lo menos 32 bits en tamaño (en todas las plataformas que GLib intenta soportar). Aunque se pueda almacenar, por lo menos, valores enteros de 32-bit en un puntero, no es correcto:

```
gpointer p;
int i;
p = (void*) 42;
i = (int) p;
```

Otra vez, el ejemplo no es correcto. El problema es que en algunos sistemas es necesario realizar lo siguiente:

```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```

Entonces GPOINTER_TO_INT(), GINT_TO_POINTER(), etc. hacen lo correcto en la plataforma actual.

Importante
No se pueden almacenar punteros en enteros. Esto no es portable en ninguna forma. Estas macros solo permiten almacenar enteros en punteros, reservando únicamente 32 bits del entero. Valores fuera de ese rango van a ser truncados.

Detalles:

GINT_TO_POINTER()

```
#define   GINT_TO_POINTER(i)   ((gpointer)  (i))
```

Convertir un entero en un tipo puntero

i : entero convertido en un puntero

GPOINTER_TO_INT()

```
#define   GPOINTER_TO_INT(p)   ((gint)   (p))
```

Extrae un entero de un puntero. El entero debe haber sido almacenado en el puntero con GINT_TO_POINTER().

p : puntero que contenie el entero

GUINT_TO_POINTER()

```
#define   GUINT_TO_POINTER(u)   ((gpointer)  (u))
```

Convierte un entero sin signo a un tipo puntero

u : entero sin signo convertido en un puntero

GPOINTER_TO_UINT()

```
#define   GPOINTER_TO_UINT(p)   ((guint)  (p))
```

Extrae un entero sin signo de un puntero. El entero debe haber sido almacenado en el puntero con GUINT_TO_POINTER().

p : Puntero a extraer en la forma de un entero sin signo

GSIZE_TO_POINTER()

```
#define    GSIZE_TO_POINTER(s)   ((gpointer) (gsize) (s))
```

Convierte un gsize en un tipo puntero.

s : gsize convertido en un puntero

GPOINTER_TO_SIZE()

```
#define   GPOINTER_TO_SIZE(p)   ((gsize) (p))
```

Extrae un gsize de un puntero. El gsize debe haber sido almacenado en el puntero con GSIZE_TO_POINTER().

p : Puntero desde donde extraer un gsize

CAPITULO IV

GNet

GNet es una biblioteca de red. Está escrita en C, orientada a objetos, y basada en GLib. Fue creada con el objetivo de ser rápida, fácil de usar y portable. Esta biblioteca provee abstracción de alto nivel para cliente y servidor.

Características

- TCP "client" and "server" sockets
- UDP and IP Multicast
- Internet address abstraction
- Asynchronous Socket IO
- Asynchronous DNS lookup
- SHA and MD5 hashes
- Byte packing and unpacking

GNet está bajo licencia GNU Lesser General Public License.

Compilación de Programas con GNet

Los desarrolladores que deseen usar GNet en sus programas, pueden utilizar el script gnet-config para identificar cuáles son los flags que se necesitan setear para la compilación.

Por ejemplo:

```
# gcc main.c `gnet-config --cflags --libs`
```

Para compilar y linkear el programa main.c con GNet:

Si se utiliza autoconf y automake, se pueden usar las macros gnet.m4 para setear las variables apropiadas. Solamente se debe agreguar lo siguiente al archivo “configure.in”:

```
dnl Need GNet
AM_PATH_GNET(<número de versión>,
[LIBS="$LIBS $GNET_LIBS" CFLAGS="$CFLAGS $GNET_CFLAGS"],
AC_MSG_ERROR(No se puede encontrar GNet: Está gnet-config en el
path?))
```

Donde <número de versión> es la versión de GNet que se está utilizando (ej: 1.0.4).

Se puede obtener el número de versión de GNet ejecutando:

```
# gnet-config --version
```

Otros Conceptos

- GNet fue desarrollada de modo tal que su implementación está oculta al programador. No se deben incluir header files de red más que <gnet/gnet.h>.
- GLib incluye las funciones g_ntohs, g_htons, g_ntohl, y g_htonl. Se deben utilizar éstas en lugar de las incluídas en <netinet/in.h>.
- Las funciones "Non-blocking" retornan inmediatamente. Las funciones asincrónicas son non-blocking y hacen un callback cuando terminan. Operaciones de IO en socket IOChannels (ej: read, write) son non-blocking. Utilizar las funciones asincrónicas de GLib g_io_add_watch

en un IOChannel para setear un callback que será invocado cuando el GIOChannel puede ser leído (o escrito, o hay un error). Ver los ejemplos echoclient y echoserver.

GNet – Programas de Ejemplo

- echoclient and echoserver
- hfetch
- SDR

GNet incluye varios programas de ejemplo. La idea es que los desarrolladores los tomen como ejemplo para poder programar utilizando esta biblioteca.
Los programas ejemplo están en un directorio de ejemplos que viene con los fuentes. Para generar los ejemplos, instalar GNet, luego ir al directorio de los ejemplos y ejecutar make.
A continuación se describe lo que hace cada uno de los programas ejemplo.

- echoclient and echoserver

El echoclient se conecta al echoserver y envía la información que el usuario ingresa. El echoserver luego la envía nuevamente y el echocliente la imprime.
Estos programas demuestran cómo escribir un cliente y un servidor básicos, basados en TCP. Echoclient-udp y Echoserver-udp son UDP equivalentes.
Existen dos métodos para escribir un server: blocking y non-blocking.
El método blocking acepta una conexión, lee y escribe al sockect hasta que el socket es cerrado. Sólo un echoclient se puede conectar a la vez.

Un servidor bueno y robusto no usaría este método, pero es adecuado para muchas aplicaciones simples.

El segundo método es non-blocking. El servidor no se bloquea mientras lee o escribe el socket o cuando espera por una conexión. Como nunca se bloquea puede aceptar nuevas conexiones cuando está leyendo o escribiendo en otro socket. Solo puede bloquearse por unos instantes cuando lee o escribe pero, como lee únicamente cuando hay algo para leer o escribe cuando hay espacio para escribir en el buffer, es muy raro que esta situación ocurra.

La ventaja del método non-blocking es que múltiples clientes pueden ser atendidos al mismo tiempo. Es, generalmente, el mejor método para utilizar.

GNet utiliza el ‘event loop’ y GIOChannels de GLib para lograr esto.

Otro método, no implementado aún, es el uso de threads. La ventaja es que es más fácil para programar. La desventaja es que no es tan eficiente en equipos con un solo procesador o en un equipo que no tiene una buena implementación de threads.

- hfetch

Con hfetch se puede obtener un archivo por HTTP.

- SDR

SDR imprime avisos de sesión multimedia. Demuestra cómo usar ‘multicast sockets’ en GNet. SDR sólo funciona si la red soporta IP Multicast. SDR provee un mecanismo muy bueno para probar IP Multicast. Si la red soporta multicast, SDR imprimirá avisos después de unos segundos, de lo contrario, no imprime nada.

CAPITULO V

Implementación de Protocolos en GNet

Funciones que se exportan al programador:

Remote Shell - rsh

- Función GnetRsConnect

Definición: Realiza la conexión al servidor *host* con el usuario *user* para ejecutar el comando *command*.

Parámetros de entrada: gchar *user, gchar *host, gchar *command
user: usuario con el cual se va a realizar la conexión
host: servidor al cual se va a realizar la conexión
command: comando a ejecutar

Retorna: gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

- Función GnetRsInit

Definición: Inicializa las siguientes opciones:
usuario: por defecto utiliza el mismo usuario que genera el comando rsh (usuario local = usuario remoto)
input: por defecto no redirecciona el input desde el dispositivo /dev/null

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRsNullOff

Definición: Deshabilita la opción de redirección de input desde el dispositivo /dev/null.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRsNullOn

Definición: Habilita la opción de redirección de input desde el dispositivo /dev/null.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRsUserOff

Definición: Utiliza el mismo usuario que genera el comando rsh (usuario local = usuario remoto).

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRsUserOn

Definición: Setea el usuario para ejecutar el comando remoto.

Parámetros de entrada: gchar *username
username: usuario con el cual se va a realizar la conexión

Retorna: void

Remote Execute – rexec

- Función GnetRxConnect

Definición: Realiza la conexión al servidor *host* con el usuario *user* y el password *passwd* para ejecutar el comando *command*.

Parámetros de entrada: gchar *host, gchar *user, gchar *passwd, gchar *command
host: servidor al cual se va a realizar la conexión
user: usuario con el cual se va a realizar la conexión
passwd: password del usuario user
command: comando a ejecutar

Retorna: gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

- Función GnetRxInit

Definición: Inicializa las siguientes opciones:
usuario: por defecto utiliza el mismo usuario que genera el comando rexec (usuario local = usuario remoto)

password: por defecto utiliza el password del usuario que genera el comando rexec (usuario local = usuario remoto)
shell: por defecto no utiliza el shell BSD

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRxBSDOff

Definición: No utiliza el shell BSD

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRxBSDOn

Definición: Utiliza el shell BSD

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRxPassOff

Definición: Utiliza el password del usuario que genera el comando rexec (usuario local = usuario remoto).

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRxPassOn

Definición: Setea el password para ejecutar el comando remoto.

Parámetros de entrada: gchar *pass
pass: password del usuario con el cual se va a realizar la conexión

Retorna: void

- Función GnetRxUserOff

Definición: Utiliza el mismo usuario que genera el comando rexec (usuario local = usuario remoto).

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRxUserOn

Definición: Setea el usuario para ejecutar el comando remoto.

Parámetros de entrada: gchar *username
username: usuario con el cual se va a realizar la conexión

Retorna: void

Remote Copy Procedure - rcp

- Función GnetRcConnect

Definición: Realiza la conexión desde un servidor a otro servidor remoto (que puede ser el mismo), utilizando para conectarse los usuarios habilitados en los servidores mencionados para ejecutar una copia remota.

Parámetros de entrada: gint argc, gchar *argv[]
argc: cantidad de argumentos que recibe
argv: vector con los argumentos recibidos, el cual va a ser indexado de acuerdo al valor de argc (usuario1@host1:file1, usuario2@host2:file2)

Retorna: gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

- Función GnetRcInit

Definición: Inicializa las siguientes opciones:
acceso: por defecto no preserva el modo de acceso
recursión: por defecto deshabilita la opción de copia recursiva
directorio: por defecto deshabilita la opción de copia de directorios

copia hacia host remoto: por defecto deshabilita la opción de copia hacia un servidor remoto
copia desde host remoto: por defecto deshabilita la opción de copia desde un servidor remoto

<u>Parámetros de entrada:</u> no recibe parámetros de entrada

<u>Retorna:</u> void

- Función GnetRcAccessOff

<u>Definición:</u> No preserva el modo de acceso.

<u>Parámetros de entrada:</u> no recibe parámetros de entrada

<u>Retorna:</u> void

- Función GnetRcAccessOn

<u>Definición:</u> Preserva el modo de acceso.

<u>Parámetros de entrada:</u> no recibe parámetros de entrada

<u>Retorna:</u> void

- Función GnetRcDirectoryOff

Definición: Deshabilita la opción de copia de directorios.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcDirectoryOn

Definición: Habilita la opción de copia de directorios.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRecursiveOff

Definición: Deshabilita la opción de copia recursiva.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRecursiveOn

Definición: Habilita la opción de copia recursiva.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRemoteFromOff

Definición: Deshabilita la opción de copia desde servidor remoto.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRemoteFromOn

Definición: Habilita la opción de copia desde servidor remoto.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRemoteToOff

Definición: Deshabilita la opción de copia hacia servidor remoto.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRcRemoteToOn

Definición: Habilita la opción de copia hacia servidor remoto.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

<u>**Remote Login - rlogin**</u>

- Función GnetRlConnect

<u>Definición:</u> Realiza la conexión al servidor *host* con el usuario *user* para ejecutar el comando *command*.

<u>Parámetros de entrada:</u> gchar *user, gchar *host
user: usuario con el cual se va a realizar la conexión
host: servidor al cual se va a realizar la conexión

<u>Retorna:</u> gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

- Función GnetRlInit

<u>Definición:</u> Inicializa las siguientes opciones:
8-bits: por defecto deshabilita la trasnferencia en ocho bits
reconocimiento caracter de escape: por defecto deshabilita el reconocimiento de cualquier caracter como caracter de escape
redefinición caracter de escape: por defecto deshabilita la redefinición del caracter de escape

usuario: por defecto utiliza el mismo usuario que genera el logueo remoto (usuario local = usuario remoto)

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlEightBitOff

Definición: Deshabilita la transferencia en ocho bits.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlEightBitOn

Definición: Habilita la transferencia en ocho bits.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlEscapeCharOff

Definición: Deshabilita el reconocimiento de cualquier caracter como caracter de escape.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlEscapeCharOn

Definición: Habilita el reconocimiento de cualquier caracter como caracter de escape.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlNoEscapeOff

Definición: Deshabilita la redefinición del caracter de escape.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlNoEscapeOn

Definición: Habilita la redefinición del caracter de escape.

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlUserOff

Definición: Utiliza el mismo usuario que genera el logueo remoto (usuario local = usuario remoto).

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetRlUserOn

Definición: Setea el usuario para ejecutar el logueo remoto.

Parámetros de entrada: gchar *username
username: usuario con el cual se va a realizar la conexión

Retorna: void

Trivial File Transfer Protocol - tftp

- Función GnetTFInit

Definición: Inicializa las siguientes opciones:
mode: por defecto habilita el modo ascii de transmisión
máximo timeout: setea el máximo timeout
timeout: setea el timeout

Parámetros de entrada: no recibe parámetros de entrada

Retorna: void

- Función GnetTFMode

Definición: Inicializa el modo de transmisión

Parámetros de entrada: gchar *mode
mode: modo de transmisión

Retorna: gint
Código que indica el resultado de la operación:

  - 0: conexión correcta
  - <> 0: problema en la conexión

- Función GnetTFRexmt

Definición: Inicializa el máximo timeout

Parámetros de entrada: gchar *rexmt
rexmt: máximo timeout

Retorna: gint
Código que indica el resultado de la operación:

  - 0: conexión correcta
  - <> 0: problema en la conexión

- Función GnetTFTimeout

Definición: Inicializa el timeout

Parámetros de entrada: gchar *timeout
timeout: timeout de retransmisión

Retorna: gint
Código que indica el resultado de la operación:

  - 0: conexión correcta
  - <> 0: problema en la conexión

- Función GnetTFPut

<u>Definición:</u> Realiza la copia desde el servidor local al servidor remoto *host* de los archivos contenidos en *file.*

<u>Parámetros de entrada:</u> gchar *host, gchar *file
host: servidor al que se copia el archivo
file: contiene el path y el nombre del archivo local y el archivo remoto a copiar

<u>Retorna:</u> gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

- Función GnetTFGet

<u>Definición:</u> Realiza la copia desde el servidor remoto *host* al servidor local de los archivos contenidos en *file.*

<u>Parámetros de entrada:</u> gchar *host, gchar *file
host: servidor al que se copia el archivo
file: contiene el path y el nombre del archivo local y el archivo remoto a copiar

Retorna: gint

Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

Telnet

- Función GnetTnConnect

Definición: Realiza la conexión al servidor *host* con el usuario *user* para ejecutar el comando *command*.

Parámetros de entrada: gint argc, gchar *argv[]
argc: cantidad de argumentos que recibe
argv: vector con los argumentos recibidos, el cual va a ser indexado de acuerdo al valor de argc (telnet host)

Retorna: gint
Código que indica el resultado de la operación:

- 0: conexión correcta
- <> 0: problema en la conexión

Consideraciones Generales

1) Rsh, rexec, rlogin

   Por motivos de seguridad rexec, rsh y rlogin no están habilitados por defecto en Red Hat Linux 7.1.

   Para poder usarlos primero se debe ejecutar **ntsysv** para habilitar rsh, rexec y rlogin. Por último, se debe reiniciar xinetd con:

   # /sbin/service xinetd restart

   para activar los cambios.

   A partir de este momento todos los usuarios excepto root podrán usar rexec, rsh y rlogin.

   Las siguientes son las instrucciones para poder utilizar estas herramientas desde root. No es recomendado por un tema de seguridad, pero a veces es necesario.

   - Agregue los nombres de las herramientas que desea permitir al archivo /etc/securetty.
   - Si desea habilitar el inicio de sesión a nivel de root con rsh, rexec y rlogin, agregue las siguientes líneas a /etc/securetty:

     ```
     rsh
     rexec
     rlogin
     ```

   - Para permitir que root inicie una sesión usando estas herramientas por telnet agregue las siguientes líneas a /etc/securetty:

     ```
     pts/0
     pts/1
     ```

2) Tftp

   Modificar el archivo /etc/inet.d/tftp. Cambiar la última línea por:

   ```
   server_args= -c  -l /tftpboot
   ```

3) Una vez compilados los programas, para que cualquier usuario pueda ejecutarlos se deben cambiar sus protecciones agregando el "sticky bit" (seteo de usuario o grupo en tiempo de ejecución):

   Ejemplo para testrcp:

   ```
   # chown root testrcp
   # chmod u+s testrcp
   ```

4) Un mecanismo de autenticación utilizado por rsh, rexec, rcp y rlogin es el archivo .rhosts. Este archivo se ubica en el directorio de logueo del usuario y el mismo contiene los hosts y usuarios remotos desde los cuales se puede hacer uso de las herramientas anteriormente mencionadas.

   Ejemplo de .rhosts:

hostone * ←	cualquier usuario de hostone puede ejecutar comandos remotos
hosttwo test ←	sólo el usuario test de hosttwo puede ejecutar comandos remotos

CAPITULO VI

Conclusiones

En el presente trabajo agregamos a la biblioteca GNet existente los protocolos antes mencionados, a saber: rsh, rexec, rcp, rlogin, tftp y telnet.

Para la creación de los mismos, utilizamos código abierto y portable a los distintos sistemas operativos y arquitecturas.

Los programadores pueden hacer uso de esta biblioteca para el desarrollo de las distintas aplicaciones haciendo uso de las funciones exportadas por cada uno de los protocolos de red existentes en la misma, tanto los originales como los agregados por nosotras. El código de las funciones exportadas por cada uno de estos protocolos se encuentra en el Apéndice C del presente trabajo.

Las aplicaciones de los desarrolladores pueden hacer uso o no de la biblioteca GLib, no así de la biblioteca GNet. Es decir, el uso de la biblioteca GNet es necesario, ya que en la misma se encuentran las definiciones de los distintos protocolos, pero la biblioteca GLib es optativa, ya que la misma sólo se refiere a funciones y tipos de datos. Aclaramos que la biblioteca GNet hace uso de la biblioteca GLib, por lo tanto, esta última debe estar instalada aunque los desarrolladores no hagan uso de la misma en sus aplicaciones.

Como ejemplo de uso de los diferentes protocolos agregados a la biblioteca GNet existente, se incluyen en este trabajo los programas testrsh.c, testrexec.c, testrcp.c, testrlogin.c, testtftp.c, testtelnet.c, los cuales se encuentran en el Apéndice D.

Cabe mencionar que en el Apéndice B se encuentran las instrucciones paso a paso para la correcta instalación de las bibliotecas GLib y GNet, siendo esta última la biblioteca modificada con los nuevos protocolos, objetivo de esta tesis.

Por todo lo anteriormente mencionado, se concluye que los desarrolladores de aplicaciones pueden trabajar en forma transparente, haciendo uso de los distintos protocolos que se encuentran en la biblioteca GNet, conociendo únicamente las funciones exportadas por la misma. De esta forma se independizan de la implementación del protocolo en sí, fijando su mayor atención solamente en la funcionalidad de la aplicación a desarrollar.

CAPITULO VII

Propuestas de Continuación

Las siguientes son las distintas propuestas de continuación a la presente tesis que consideramos interesantes realizar.

Kerberos

Agregar Kerberos a la implementación realizada en esta tesis, a cada uno de los protocolos mencionados (rsh, rexec, rcp, rlogin, tftp, telnet).

Encriptación

Agregar encriptación a la implementación realizada en esta tesis, a cada uno de los protocolos mencionados (rsh, rexec, rcp, rlogin, tftp, telnet).

Agregar Otros Protocolos

Agregar otros protocolos a la biblioteca GNet, como ser: ftp, icmp, http, snmp, smtp, etc.

Adaptar a Otros Sistemas Operativos

Realizar la adaptación a otros sistemas operativos, por ejemplo Microsoft Windows, de los protocolos existentes en la biblioteca GNet.

APÉNDICES

APÉNDICE A - RFCs

TELNET

Network Working Group J. Postel
Request for Comments: 854 J. Reynolds
Obsoletes: NIC 18639 May 1983

TELNET PROTOCOL SPECIFICATION

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet are expected to adopt and implement this standard.

INTRODUCTION

The purpose of the TELNET Protocol is to provide a fairly general, bidirectional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

GENERAL CONSIDERATIONS

A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1. When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" hosts to keep information about the characteristics of each other's terminals and terminal handling conventions. All hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

NOTE: The "user" host is the host to which the physical terminal is normally attached, and the "server" host is the host which is normally providing some service. As an alternate point of view, applicable even in terminal-to-terminal or process-to-process communications, the "user" host is the host which initiated the communication.

2. The principle of negotiated options takes cognizance of the fact that many hosts will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but

structured within the TELNET Protocol are various "options" that will be sanctioned and may be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

3. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

a. Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.

b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged. This non-response is essential to prevent endless loops in the negotiation. It is required that a response be sent to requests for a change of mode -- even if the mode is not changed.

c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a host may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline web suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option.

However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the detailed control was no longer necessary.

It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options -- since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use “DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length -- such a "sub-negotiation" might include fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "TELNET Option Specifications," should be consulted for information about the procedure for establishing new options.

THE NETWORK VIRTUAL TERMINAL

The Network Virtual Terminal (NVT) is a bi-directional carácter device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

TRANSMISSION OF DATA

Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

1) Insofar as the availability of local buffer space permits, data should be accumulated in the host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs.
This signal could be generated either by a process or by a human user.

The motivation for this rule is the high cost, to some hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command.

This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinguish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local) computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a host plans to support terminal-to-terminal communication it is suggested that the host provide the user with a means of manually signaling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process.

Note that the symmetry of the TELNET model requires that there is an NVT at each end of the TELNET connection, at least conceptually.

STANDARD REPRESENTATION OF CONTROL FUNCTIONS

As stated in the Introduction to this document, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described below. These standard representations have standard, but not required, meanings (with the exception that the Interrupt Process (IP) function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other and, a system which does provide the function to a local user is obliged to provide the same function to a network user who transmits the standard representation for the function.

Interrupt Process (IP)

Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be required by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

Abort Output (AO)

Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal.
Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" carácter (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by server systems which provide this function, that there may be buffers external to the system (in the network and the user's local host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal (described below) to the user system.

Are You There (AYT)

Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

Erase Character (EC)

Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

*NOTE: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

Erase Line (EL)

Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

THE TELNET "SYNCH" SIGNAL

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms.
Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's host.

To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the TELNET command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the TELNET connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream.

The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode, a DM is a no operation; when in urgent mode, it signals the end of the urgent processing.

If TCP indicates the end of Urgent data before the DM is found, TELNET should continue the special handling of the data stream until the DM is found.

If TCP indicates more Urgent data after the DM is found, it can only be because of a subsequent Synch. TELNET should continue the special handling of the data stream until another DM is found.

"Interesting" signals are defined to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essentially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired.

For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the TCP Urgent notification is needed at the TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the TELNET IP character;

2. Send the TELNET SYNC sequence, that is:

   Send the Data Mark (DM) as the only character
   in a TCP urgent mode send operation.

3. Send the character string STOP; and

4. Send the other protocol's analog of the TELNET DM, if any.

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step 2 above to ensure that the TELNET IP gets through to the server's TELNET interpreter.

The Urgent should wake up the TELNET process; the IP should wake up the next higher level process.

THE NVT PRINTER AND KEYBOARD

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

NAME	CODE	MEANING
NULL (NUL)	0	No Operation
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or

		visible signal (which does NOT move the print head).
Back Space (BS)	8	Moves the print head one character position towards the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For

example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.)
Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational desition.

Note that "CR LF" or "CR NUL" is required in both directions (in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, nonetheless, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NULL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local carácter set mapping.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not required, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch

This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

Break (BRK)

This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)
Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Abort Output (AO)

Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)

Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)

The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)

The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local".
Just as the NVT data byte 68 (104 octal) should be mapped into whatever the local code for "uppercase D" is, so the EC carácter should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 124 (174 octal) is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for

format effectors: if the terminal actually does have a "Vertical Tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

TELNET COMMAND STRUCTURE

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

NAME	CODE	MEANING
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data Mark	242	The data stream portion of a Synch. This should always be accompanied

		by a TCP Urgent notification.
Break	243	NVT character BRK.
Interrupt Process	244	The function IP.
Abort output	245	The function AO.
Are You There	246	The function AYT.
Erase character	247	The function EC.
Erase Line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party

		to perform, the indicated option.
IAC	255	Data Byte 255.

CONNECTION ESTABLISHMENT

The TELNET TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving its port L and different user ports U.

Port Assignment

When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

TFTP

Network Working Group K. R. Sollins
Request for Comments: 783 MIT
June, 1981
Updates: IEN 133

THE TFTP PROTOCOL (REVISION 2)

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

ACKNOWLEDGEMENTS

The protocol was originally designed by Noel Chiappa, and was redesigned by him, Bob Baldwin and Dave Clark, with comments from Steve Szymanski. The current revision of the document includes modifications stemming from discussions with and suggestions from Larry Allen, Noel Chiappa, Dave Clark, Geoff Cooper, Mike Greenwald, Liza Martin, David Reed, Craig Milo Rogers (of UCS-ISI), Kathy Yellick, and the author.

The acknowledgement and retransmission scheme was inspired by TCP, and

the error mechanism was suggested by PARC's EFTP abort message.
This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

1. Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks implementing UDP. (This should not exlude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data. 1 2

Three modes of transfer are currently supported: netascii ; octet , raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. Additional modes can be defined by pairs of cooperating hosts.

1
This is ascii as defined in "USA Standard Code for Information Interchange" [1] with the modifications specified in "Telnet Protocol

Specification". Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.

2

This replaces the "binary" mode of previous versions of this.

2. Overview of the Protocol

Any transsfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes signals termination of a transfer. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data.

Most errors cause termination of the connection. An error is signalled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events: not being able to satisfy the request (e.g., file not found, access violation, or no such user), receiving a packet which cannot be explained by a delay or duplication in the network (e.g. an incorrectly formed packet), and losing access to a necessary resource (e.g., disk full or access denied during a transfer).

TFTP recognizes only one error condition that does not cause termination, the source port of a received packet being incorrect. In this case, an error packet is sent to the originating host.

This protocol is very restrictive, in order to simplify implementation. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reorder incoming data packets.

3. Relation to other Protocols

As mentioned TFTP is designed to be implemented on top of the Datagram protocol. Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in Figure 3-1, the order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's) used by TFTP are passed to the Datagram layer to be used as ports; therefore they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.

Figure 3-1: Order of Headers

```
 ---------------------------------------------------------
| Local Medium | Internet | Datagram | TFTP |
 ---------------------------------------------------------
```

4. Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system, or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, or the first data packet for read. In general an acknowledgment packet will contain the block number of the data packet being acknowledged. Each data packet has associated with it a block number; block numbers are consecutive and begin with one. Since the positive response to a write request is an acknowledgment packet, in this special case the block number will be zero. (Normally, since an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet, then the request has been denied.

In order to create a connection, each end of the connection chooses a TID for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it the two TID's of the ends of the connection, the source TID and the destination TID. These TID's are handed to the supporting UDP (or other datagram protocol) as the source and destination ports. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 decimal (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively. The appendix contains a similar example for reading a file.

1. Host A sends a "WRQ" to host B with source= A's TID, destination= 69.

2. Host B sends a "ACK" (with block number= 0) to host A with source= B's TID, destination= A's TID.

At this point the connection has been established and the first data packet can be sent by Host A with a sequence number of 1. In the next step, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in steps 1 and 2. If a source TID does not match, the packet should be discarded as erroneously sent from somewhere else. An error packet should be sent to the source of the incorrect packet, while not disturbing the transfer.

This can be done only if the TFTP in fact receives a packet with an incorrect TID. If the supporting protocols do not allow it, this particular error condition will not arise.

The following example demonstrates a correct operation of the protocol in which the above situation can occur. Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in response to the two requests. When the first response arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen for the two connections on host B and host A checks the source TID's of the messages it receives, the first connection can be maintained while the second is rejected by returning an error packet.

5. TFTP Packets

TFTP supports five types of packets, all of which have been mentioned above:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

The TFTP header of a packet contains the opcode associated with that packet.

Figure 5-1: RRQ/WRQ packet

```
 2 bytes    string    1 byte    string   1 byte
 ------------------------------------------------
| Opcode |  Filename  |   0  |    Mode    |   0  |
 ------------------------------------------------
```

RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in Figure 5-1. The file name is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "octet", or "mail" (or any comibnation of upper and lower case, such as "NETASCII", NetAscii", etc.) in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to its own format. Octet mode is used to transfer a file that is in the 8-bit format of the machine from which the file is being transferred. It is assumed that each type of machine has a single 8-bit format that is more common, and that that format is chosen. For example, on a DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with four bits of breakage. If a host receives a octet file and then returns it, the returned file must be identical to the original. Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode. The mail recipient string should be of the form "username" or "username@hostname". If the second form is used, it allows the option of mail forwarding by a relay computer.

The discussion above assumes that both the sender and recipient are operating in the same mode, but there is no reason that this has to be the case. For example, one might build a storage server. There is no reason that such a machine needs to translate netascii into its own form of text. Rather, the sender might send files in netascii, but the storage server might simply store them without translation in 8-bit format. Another such situation is a problem that currently exists on DEC-20 systems. Neither netascii nor octet accesses all the bits in a word. One might create a special mode for such a machine which read all the bits in a word, but in which the receiver stored the information in 8-bit format. When such a file is retrieved from the storage site, it must be restored to its original form to be useful, so the reverse mode must also be implemented. The user site will have to remember some information to achieve this. In both of these examples, the request packets would specify octet mode to the foreign host, but the local host would be in some other mode. No such machine or application specific modes have been specified in TFTP, but one would be compatible with this specification.
It is also possible to define other modes for cooperating pairs of hosts, although this must be done with care. There is no requirement that any

other hosts implement these. There is no central authority that will define these modes or assign them names.

Figure 5-2: DATA packet

```
 2 bytes     2 bytes      n bytes
 ----------------------------------
| Opcode |   Block #  |   Data     |
 ----------------------------------
```

Data is actually transferred in DATA packets depicted in Figure 5-2.
DATA packets (opcode = 3) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the end of the transfer. (See the section on Normal Termination for details.)

All packets other than those used for termination are acknowledged individually unless a timeout occurs. Sending a DATA packet is an acknowledgment for the ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and 1

Figure 5-3: ACK packet

```
 2 bytes     2 bytes
 ------------------------
| Opcode |   Block #  |
 -------------------------
```

ACK packets are acknowledged by DATA or ERROR packets. Figure 5-3 depicts an ACK packet; the opcode is 4. The block number in an ACK echoes the block number of the DATA packet being acknowledged. A WRQ is acknowledged with an ACK packet having a block number of zero.

Figure 5-4: ERROR packet

```
 2 bytes     2 bytes      string    1 byte
 ---------------------------------------------
| Opcode |  ErrorCode |   ErrMsg   |   0  |
 ---------------------------------------------
```

An ERROR packet (opcode 5) takes the form depicted in Figure 5-4. An ERROR packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error. A table of values and meanings is given in the appendix. (Note that several error codes have been added to this version of this document.) The error message is intended for human consumption, and should be in netascii.

Like all other strings, it is terminated with a zero byte.

6. Normal Termination

The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e. Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.

7. Premature Termination

If a request can not be granted, or some error occurs during the transfer, hen an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

I. Appendix

Order of Headers

```
                              2 bytes
 ------------------------------------------------------------
| Local Medium | Internet | Datagram | TFTP Opcode |
 ------------------------------------------------------------
```

TFTP Formats

```
Type   Op #     Format without header


     2 bytes    string   1 byte     string   1 byte
         -----------------------------------------------
RRQ/  | 01/02 |  Filename |   0  |    Mode    |   0  |
WRQ    -----------------------------------------------
     2 bytes    2 bytes       n bytes
     ---------------------------------------
DATA  | 03    |   Block #  |    Data    |
     ---------------------------------------
     2 bytes    2 bytes
     -------------------------
ACK   | 04    |   Block #  |
     --------------------------
     2 bytes  2 bytes        string    1 byte
     ----------------------------------------
ERROR | 05    |  ErrorCode |   ErrMsg   |   0  |
     ----------------------------------------
```

Initial Connection Protocol for reading a file

1. Host A sends a "RRQ" to host B with source= A's TID, destination= 69.

2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Internet User Datagram Header [2]

Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |      Destination Port         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Values of Fields

Source Port	Picked by originator of packet.
Dest. Port	Picked by destination machine (69 for RRQ or WRQ).
Length	Number of bytes in packet after Datagram header.
Checksum	Reference 2 describes rules for computing checksum. Field contains zero if unused.

Note: TFTP passes transfer identifiers (TID's) to the Internet User Datagram protocol to be used as the source and destination ports.

---

3 This has been included only for convenience. TFTP need not be implemented on top of the Internet User Datagram Protocol.

4

The implementor of this should be sure that the correct algorithm is used here.

References

[1] USA Standard Code for Information Interchange, USASI X3.4-1968.

[2] Postel, Jon., "User Datagram Protocol," RFC768, August 28, 1980.

[3] "Telnet Protocol Specification," RFC764, June, 1980.

RLOGIN

Network Working Group B. Kantor

Request for Comments: 1282 Univ. of Calif San Diego

Obsoletes: RFC 1258 December 1991

BSD Rlogin

Status of this Memo

This memo documents an existing protocol and common implementation that is extensively used on the Internet. This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Protocol Description

The rlogin facility provides a remote-echoed, locally flow-controlled virtual terminal with proper flushing of output [1]. It is widely used between Unix hosts because it provides transport of more of the Unix terminal environment semantics than does the Telnet protocol, and because on many Unix hosts it can be configured not to require user entry of passwords when connections originate from trusted hosts.

The rlogin protocol requires the use of the TCP. The contact port is 513. An eight-bit transparent stream is assumed.

Connection Establishment

Upon connection establishment, the client sends four null-terminated strings to the server. The first is an empty string (i.e., it consists solely of a single zero byte), followed by three non-null strings: the client username, the server username, and the terminal type and speed. More explicitly:

```
<null>
client-user-name<null>
server-user-name<null>
terminal-type/speed<null>
```

For example:

```
<null>
bostic<null>
kbostic<null>
vt100/9600<null>
```

The server returns a zero byte to indicate that it has received these strings and is now in data transfer mode. Window size negotiation may follow this initial exchange (see below).

From Client to Server (and Flow Control)

Initially, the client begins operation in "cooked" (as opposed to to "raw") mode. In this mode, the START and STOP (usually ASCII DC1,DC3) characters are intercepted and interpreted by the client to start and stop output from the remote server to the local terminal, whereas all other characters are transmitted to the remote host as they are received. (But see below for the handling of the local-escape character.)

In "raw" mode, the START and STOP characters are not processed locally, but are sent as any other character to the remote server.
The server thus determines the semantics of the START and STOP characters when in "raw" mode; they may be used for flow control or have quite different meanings independent of their ordinary usage on the client.

Screen/Window Size

The remote server indicates to the client that it can accept window size change information by requesting a window size message (as out of band data) just after connection establishment and user identification exchange. The client should reply to this request with the current window size.

If the remote server has indicated that it can accept client window size changes and the size of the client's window or screen dimensions changes, a 12-byte special sequence is sent to the remote server to indicate the current dimensions of the client's window, should the user process running on the server care to make use of that information.

The window change control sequence is 12 bytes in length, consisting of a magic cookie (two consecutive bytes of hex FF), followed by two bytes containing lower-case ASCII "s", then 8 bytes containing the 16-bit values for the number of character rows, the number of characters per row, the number of pixels in the X direction, and the number of pixels in the Y direction, in network byte order. Thus:

FF FF s s rr cc xp yp

Other flags than "ss" may be used in future for other in-band control messages. None are currently defined.

From Server to Client

Data from the remote server is sent to the client as a stream of characters. Normal data is simply sent to the client's display, but may be processed before actual display (tabs expanded, etc.).

The server can imbed single-byte control messages in the data stream by inserting the control byte in the stream of data and pointing the TCP "urgent-data" pointer at the control byte. When a TCP urgent-data pointer is received by the client, data in the TCP stream up to the urgent byte is buffered for possible display after the control byte is handled, and the control byte pointed to is received and interpreted as follows:

02 A control byte of hex 02 causes the client to discard all buffered data received from the server that has not yet been written to the client user's screen.

10 A control byte of hex 10 commands the client to switch to "raw" mode, where the START and STOP characters are no longer handled by the client, but are instead treated as plain data.

20 A control byte of hex 20 commands the client to resume interception and local processing of START and STOP flow control characters.

80 The client responds by sending the current window size as above.

All other values of the urgent-data control byte are ignored. In all cases, the byte pointed to by the urgent data pointer is NOT written to the client user's display.

Connection Closure

When the TCP connection closes in either direction, the client or server process which notices the close should perform an orderly shut-down, restoring terminal modes and notifying the user or processes of the close before it closes the connection in the other direction.

Implementation Notes

The client defines a client-escape character (customarily the tilde, "~"), which is handled specially only if it is the first character to be typed at the beginning of a line. (The beginning of a line is defined to be the first character typed by the client user after a new-line [CR or LF] character, after a line-cancel character, after resumption of a suspended client session, or after initiation of the connection.)

The client-escape character is not transmitted to the server until the character after it has been examined, and if that character is one of the defined client escape sequences, neither the client-escape nor the character following it are sent. Otherwise, both the client-escape character and the character following it are sent to the server as ordinary user input.

If the character following the client-escape character is the dot ".", or the client-defined end-of-file character (usually control-D), the connection is closed. This is normally treated by the server as a disconnection, rather than an orderly logout.

Other characters (client-defined, usually control-Z and control-Y) are used to temporarily suspend the rlogin client when the host has that ability. One character suspends both remote input and output; the other suspends remote input but allows remote output to continue to be directed to the local client's terminal.

Most client implementations have invocation switches that can defeat normal output processing on the client system, and which can force the client to remain in raw mode despite switching notification from the server.

A Cautionary Tale [2]

The rlogin protocol (as commonly implemented) allows a user to set up a class of trusted users and/or hosts which will be allowed to log on as himself without the entry of a password. While extremely convenient, this represents a weakening of security that has been successfully exploited in previous attacks on the internet. If one wishes to use the password-bypass facilities of the rlogin service, it is essential to realize the compromises that may be possible thereby.

Bypassing password authentication from trusted hosts opens ALL the systems so configured when just one is compromised. Just as using the same password for all systems to which you have access lets a villain in everywhere you have access, allowing passwordless login among all your systems gives a marauder a wide playing field once he has entered any of your systems. One compromise that many feel achieves a workable balance between convenience and security is to allow password bypass from only ONE workstation to the other systems you use, and NOT allow it between those systems. With this measure, you may have reduced exposure to a workable minimum.

The trusted host specification is ordinarily one of a host name. It is possible, by compromise of your organization's domain name server, or compromise of your network itself, for a villain to make an untrusted host masquerade as a trusted system. There is little that a user can do about this form of attack. Luckily, so far such attacks have been rare, and often cause enough disruption of a network that attempts are quickly noticed.

When the file containing a user's list of trusted logins is inadvertently left writeable by other users, untrustworthy additions may be made to it.

Secure authentication extensions to the rlogin protocol (Kerberos, et al) can greatly reduce the possibility of compromise whilst still allowing the convenience of bypassing password entry. As these become more widely deployed in the internet community, the hazards of rlogin will decrease.

References

[1] Stevens, W., "UNIX Network Programming", ISBN 0-13-949876-1.

[2] Garfinkel & Spafford, "Practical Unix Security",
ISBN 0-937175-72-2.

APÉNDICE B - Instalación

A continuación se detalla paso a paso la forma de instalación de las bibliotecas GNet y GLib.

Biblioteca GLib

1. Copiar del CD de instalación el archivo glib.tar al directorio raíz (/)

```
# cp   <cd de instalacion>/glib/glib.tar   /
```

2. Abrir el archivo glib.tar

```
# cd   /
# tar xvf glib.tar
```

3. Ejecutar los siguientes comandos para la instalación de la biblioteca

```
# ./configure
# make
# make install
```

Biblioteca GNet

1. Copiar del CD de instalación el archivo gnet.tar al directorio raíz (/)

```
# cp   <cd de instalacion>/gnet/gnet.tar   /
```

2. Abrir el archivo gnet.tar

```
# cd   /
# tar xvf gnet.tar
```

3. Ejecutar los siguientes comandos para la instalación de la biblioteca

```
# ./configure
# make
# make install
```

APÉNDICE C - Código

Funciones que se exportan al programador:

Remote Shell – rsh

gint GnetRsConnect(gchar *user, gchar *host, gchar *command)

```
{
        struct passwd *pw;
        struct servent *sp;
        glong omask;
        gint one, pid = 0, rem, uid;
        gchar *args;
        gchar *null = NULL;
        gchar **saved_environ;
        gchar **argv;

        saved_environ = __environ;
        __environ = &null;

        one = 1;

        /* To many arguments or too few.  */
        if (!host)
           return 1; /* Invalid number of parameters */

        /* If no command, must have been called as rlogin. */
        if (!command)
        {
           setuid(getuid());
           argv[0] = "rlogin";
           execve(_PATH_RLOGIN, argv, saved_environ);
           return 2;   /* Can't exec _PATH_RLOGIN */
        }
        if (!(pw = getpwuid(uid = getuid())))
           return 3;   /* Unknown user id */

        if (!rs_user_flag)
           user = pw->pw_name;


        /* Get the server */
        sp = NULL;
        if (sp == NULL)
           sp = getservbyname("shell", "tcp");
        if (sp == NULL)
           return 4;   /* Unknown service: shell/tcp */

        rem = rcmd(&host, sp->s_port, pw->pw_name, user, command, &rfd2);

        if (rem < 0)
           return 5; /* Connection failed */

        if (rfd2 < 0)
           return 6;   /* Can't establish stderr */

        setuid(uid);
omask=sigblock(sigmask(SIGINT)|sigmask(SIGQUIT)|sigmask(SIGTERM));
        if (signal(SIGINT, SIG_IGN) != SIG_IGN)
                signal(SIGINT, GnetRsSendsig);
        if (signal(SIGQUIT, SIG_IGN) != SIG_IGN)
                signal(SIGQUIT, GnetRsSendsig);
        if (signal(SIGTERM, SIG_IGN) != SIG_IGN)
                signal(SIGTERM, GnetRsSendsig);

        if (!rs_null_flag)
        {
           pid = fork();
           if (pid < 0)
                return 7; /* Fork error */
        }

        {
           ioctl(rfd2, FIONBIO, &one);
```

```
            ioctl(rem, FIONBIO, &one);
        }

        GnetRsTalk(omask, pid, rem);
        if (!rs_null_flag)
            kill(pid, SIGKILL);
        return 0;
}
```

void GnetRsInit(void)

```
{
    GnetRsUserOff();
    GnetRsNullOff();
}
```

void GnetRsUserOn(gchar *username)

```
{
    rs_user_flag = TRUE;
    user = username;
}
```

void GnetRsUserOff(void)

```
{
    rs_user_flag = FALSE;
}
```

void GnetRsNullOn(void)

```
{
    rs_null_flag = TRUE;
}
```

void GnetRsNullOff(void)

```
{
    rs_null_flag = FALSE;
}
```

Remote Execute - rexec

```
gint GnetRxConnect(gchar *host, gchar *user, gchar *passwd, gchar
*command)
{

  /* Other variables. */
  gchar *prog_name = "rexec";


  struct servent *service; /* Returned from service datata base, give port. */
  gint port_exec; /* Rexec port to use. */
  gint sock; /* Rexec socket. */
  gint sock_open, stdin_open, shut_down; /* Open file descriptor flags. */


  /* To many arguments or too few.  */
  if (!host)
     return 1; /* Invalid number of parameters - No hostname */

  /* If no command, must have been called as rlogin. */
  if (!command)
     return 2; /* Invalid number of parameters - No command */


  /* Get the server */
  service = getservbyname("exec","tcp");
  if ( (port_exec = service->s_port) >= MAX_PORT )
      port_exec = htons(DEFAULT_PORT);


  if (!rx_user_flag)
     user = getenv("REXEC_USER");

  if ( user == NULL )
  {
     uid_t uid = getuid();
     struct passwd *pw = getpwuid(uid);
```

```
    if (!(pw && pw->pw_name))
      return 3; /* "Can't lookup uid" */

    user = strdup(pw->pw_name);
}

if (!rx_pass_flag)
    passwd = getenv("REXEC_PASS");

if ( passwd == NULL )
     passwd = getpass("Password: ");


if (!(user && passwd))
    return 4; /* Can't use without supplying a user and password */

if ( (sock = rexec(&host, port_exec, user, passwd, command,
                      NULL)) < 0 )
     return 5 ;  /* Error in rexec system call */


if ( rx_sig_flag )
{
  gint sig;

  if (rx_bsd_flag)
  {
      for (sig = 0; sig < sizeof(bsd_signals)/sizeof(gint); ++sig)
             GnetRxSetSignal(bsd_signals[sig],GnetRxEchoSig);
  }
  else
  {
      for (sig = 1; sig < NSIG; ++sig)
             GnetRxSetSignal(sig,GnetRxEchoSig);
  }
}

sock_open = stdin_open = 1;
shut_down = 0;
while (sock_open) /* echo stdin -> remote host
```

```
                                   remote host -> stdout
                              until the remote host closes the socket. */
 {
   fd_set read_set;
   gint select_ret;

   FD_ZERO(&read_set);
   if (stdin_open)
      FD_SET(STDIN_FILENO, &read_set);
   if (sock_open)
      FD_SET(sock, &read_set);


   /* Using an infinite timeout in select (last parameter = NULL). */
   do
   {
    errno =0;
    select_ret=select(FD_SETSIZE, &read_set, NULL, NULL, NULL);
   } while ( errno == EINTR );

   if ( select_ret < 0 )
      return 6; /* Error in select system call */

   if ( FD_ISSET(sock, &read_set) ) /* Input available from remote host. */
      sock_open = GnetRxEchoFd(STDOUT_FILENO, sock, prog_name);

   if ( FD_ISSET(STDIN_FILENO, &read_set) )  /* Input available from
stdin. */
      stdin_open = GnetRxEchoFd(sock, STDIN_FILENO, prog_name);

   if ( ! stdin_open && ! shut_down )
   {
    if (shutdown(sock, 1) < 0)
         return 7; /* Error from shutdown */
    shut_down = 1;
   }

 }
 return 0;
}
```

```
void GnetRxInit(void)
{
    GnetRxUserOff();
    GnetRxPassOff();
    GnetRxBSDOff();
}


void GnetRxUserOn(gchar *username)
{
    rx_user_flag = TRUE;
    user = username;
}


void GnetRxUserOff(void)
{
    rx_user_flag = FALSE;
}


void GnetRxPassOn(gchar *pass)
{
    rx_pass_flag = TRUE;
    passwd = pass;
}


void GnetRxPassOff(void)
{
    rx_pass_flag = FALSE;
}


void GnetRxBSDOn(void)
{
    rx_bsd_flag = TRUE;
}
```

```
void GnetRxBSDOff(void)
{
    rx_bsd_flag = FALSE;
}
```

Remote Copy Procedure - rcp

```
gint GnetRcConnect (gint argc, gchar *argv[])
{
        struct servent *sp;
        gchar *targ;
        const gchar *shell;
        gchar *null = NULL;
        gint status = 0;

        saved_environ = __environ;
        __environ = &null;

        /* File1 */
        if (*argv[0] == 0)
           return 1;   /* Invalid file */

        /* File2 */
        if (*argv[1] == 0)
           return 1;   /* Invalid file */

        /* Get the server */
        sp = getservbyname("shell", "tcp");
        if (sp == NULL)
           return 3;   /* Unknown service: shell/tcp */

        rc_port = sp->s_port;

      /* User */
      if (!(pwd = getpwuid(uid = getuid())))
         return 2;   /* Unknown user */

        if (rc_fflag)
        {
           /* Follow "protocol", send data */
           (void)GnetRcResponse();
           (void)setuid(uid);
            GnetRcSource(margc, margv);
            return errs;
        }

        if (rc_tflag)
        {
           /* Receive data */
           (void)setuid(uid);
            GnetRcSink(argc, argv);
            return errs;
        }

        rem = -1;
        /* Command to be executed on remote system using "rsh" */
        (void)snprintf(cmd, sizeof(cmd), "rcp%s%s%s",
           rc_iamrecursive ? " -r" : "", rc_pflag ? " -p" : "",
           rc_targetshouldbedirectory ? " -d" : "");

        (void)signal(SIGPIPE, GnetRcLostConn);

        if ((targ = GnetRcColon(argv[argc - 1]))!=NULL)
      {
           /* destination is remote host */
           *targ++ = 0;
           status = GnetRcToRemote(targ, argc, argv);
        }
        else
      {
           status = GnetRcToLocal(argc, argv);        /* destination is local host
*/
           if (rc_targetshouldbedirectory)
                GnetRcVerifyDir(argv[argc - 1]);
        }
        return status;
}
```

void GnetRcInit(void)

```
{
    GnetRcAccessOff();
    GnetRcRecursiveOff();
    GnetRcDirectoryOff();
    GnetRcRemoteFromOff();
    GnetRcRemoteToOff();
}
```

void GnetRcAccessOn(void)

```
{
    rc_pflag = TRUE;
}
```

void GnetRcAccessOff(void)

```
{
    rc_pflag = FALSE;
}
```

void GnetRcRecursiveOn(void)

```
{
    rc_iamrecursive = TRUE;
}
```

void GnetRcRecursiveOff(void)

```
{
    rc_iamrecursive = FALSE;
}
```

void GnetRcDirectoryOn(void)

```
{
    rc_targetshouldbedirectory = TRUE;
}
```

void GnetRcDirectoryOff(void)

```
{
    rc_targetshouldbedirectory = FALSE;
}
```

```
void GnetRcRemoteFromOn(void)
{
    rc_iamremote = TRUE;
    rc_fflag = TRUE;
}

void GnetRcRemoteFromOff(void)
{
    rc_iamremote = FALSE;
    rc_fflag = FALSE;
}


void GnetRcRemoteToOn(void)
{
    rc_iamremote = TRUE;
    rc_tflag = TRUE;
}


void GnetRcRemoteToOff(void)
{
    rc_iamremote = FALSE;
    rc_tflag = FALSE;
}
```

Remote Login - rlogin

```
gint GnetRlConnect(gchar *user, gchar *host)
{
  struct passwd *pw;
  struct servent *sp;
  sigset_t smask;
  uid_t uid;
  gint res_doit;


  /* To many arguments or too few.  */
  if (!host)
      return 1; /* Invalid number of parameters */


  /* We must be uid root to access rcmd().  */
  if (geteuid ())
     return 2;  /* Must be setuid root */


  /* Get the name of the user invoking us: the client-user-name.  */
  if (!(pw = getpwuid (uid = getuid ())))
      return 3; /* Unknown user id */


  if (!rl_user_flag)
      user = pw->pw_name;

  /* Get the port number for the rlogin service.  */
  sp = NULL;
  if (sp == NULL)
      sp = getservbyname ("login", "tcp");

  if (sp == NULL)
     return 4;  /* Unknown service: login/tcp */

  GnetRlGetTerminal();
  GnetRlGetWindowSize (0, &winsize);

  setsig (SIGPIPE, GnetRlLostPeer);

  /* Block SIGURG and SIGUSR1 signals.  This will be handled by the
     parent and the child after the fork.  */
  /* Will use SIGUSR1 for window size hack, so hold it off.  */
  sigemptyset (&smask);
  sigaddset (&smask, SIGURG);
  sigaddset (&smask, SIGUSR1);
  sigprocmask (SIG_SETMASK, &smask, &smask);

  /*
   * We set SIGURG and SIGUSR1 below so that an
   * incoming signal will be held pending rather than being
   * discarded. Note that these routines will be ready to get
   * a signal by the time that they are unblocked below.
   */
  setsig (SIGURG, GnetRlCopyToChild);
  setsig (SIGUSR1, GnetRlWriterOob);


  rem = rcmd (&host, sp->s_port, pw->pw_name, user, term, 0);

  if (rem < 0)
     return 5; /* Connection failed */

#if defined (IP_TOS) && defined (IPPROTO_IP) && defined
(IPTOS_LOWDELAY)
  {
    gint one = IPTOS_LOWDELAY;
    gint igsetsock = 0;
    igsetsock = setsockopt (rem, IPPROTO_IP, IP_TOS, (gchar *)&one,
sizeof(gint));
  }
#endif

  /* Now change to the real user ID.  We have to be set-user-ID root
     to get the privileged port that rcmd () uses,  however we now want to
     run as the real user who invoked us.  */
  seteuid (uid);
  setuid (uid);

  res_doit = GnetRlDoit (&smask);
  if (res_doit == 1)
     return 6; /* Connection close */

  /*NOTREACHED*/
  return 0;

}
```

void GnetRlInit(void)

```
{
    GnetRlEightBitOff();
    GnetRlEscapeCharOff();
    GnetRlNoEscapeOff();
    GnetRlUserOff();
}
```

void GnetRlEightBitOn(void)

```
{
    rl_eight = TRUE;
}
```

void GnetRlEightBitOff(void)

```
{
    rl_eight = FALSE;
}
```

void GnetRlNoEscapeOn(void)

```
{
    rl_noescape = TRUE;
}
```

```
void GnetRlNoEscapeOff(void)
{
    rl_noescape = FALSE;
}


void GnetRlEscapeCharOn(void)
{
    GnetRlNoEscapeOff();
    rl_escapechar_flag = TRUE;
    rl_escapechar = GnetRlGetEscape(optarg);
}


void GnetRlEscapeCharOff(void)
{
    rl_escapechar_flag = FALSE;
    rl_escapechar = '~';
}


void GnetRlUserOn(gchar *username)
{
    rl_user_flag = TRUE;
    user = username;
}


void GnetRlUserOff(void)
{
    rl_user_flag = FALSE;
}
```

Trivial File Transfer Protocol - tftp

```
void GnetTFInit(void)
{
        strcpy(mode, "netascii");
        maxtimeout = 5 * TIMEOUT;
        rexmtval = TIMEOUT;
}


gint GnetTFMode(gchar *mode)
{
      strcpy(tf_line, "mode ");
      strcat(tf_line, mode);
      return (GnetTFCommand());
}


gint GnetTFPut(gchar *host, gchar *file)
{
        gint code = 0;

      strcpy(tf_line, "connect ");
      strcat(tf_line, host);
      code = GnetTFCommand();

        if (code != 0)
                return code;

      strcpy(tf_line, "put ");
      strcat(tf_line, file);
      return GnetTFCommand();
}
```

```
gint GnetTFGet(gchar *host, gchar *file)
{
	gint code = 0;

	strcpy(tf_line, "connect ");
	strcat(tf_line, host);
	code = GnetTFCommand();

	if (code != 0)
		return code;

	strcpy(tf_line, "get ");
	strcat(tf_line, file);
	return GnetTFCommand();
}


gint GnetTFRexmt(gchar *rexmt)
{
	strcpy(tf_line, "rexmt ");
	strcat(tf_line, rexmt);
	return GnetTFCommand();
}


gint GnetTFTimeout(gchar *timeout)
{
	strcpy(tf_line, "timeout ");
	strcat(tf_line, timeout);
	return GnetTFCommand();
}
```

Telnet

gint GnetTnConnect(gint argc, gchar *argv[])

```
{
#if !defined(__linux__)
	extern gchar *optarg;
	extern gint optind;
#endif
	gchar *user, *alias;
	  gint ch;

	GnetTnInit();		/* Clear out things */

	GnetTnTerminalSaveState();

	if ((tn_prompt = strrchr(argv[0], '/')))
		++tn_prompt;
	else
		tn_prompt = argv[0];

	user = alias = NULL;

	rlogin = (strncmp(tn_prompt, "rlog", 4) == 0) ? '~' :
_POSIX_VDISABLE;

	autologin = -1;

	while ((ch = getopt(argc, argv, "78EL:ab:ce:l:r")) != -1) {
		switch(ch) {
		case '8':
			eight = 3;	/* binary output and input */
			break;
		case '7':
			eight = 0;
			break;
		case 'E':
			rlogin = escape = _POSIX_VDISABLE;
```

```
                break;
            case 'L':
                eight |= 2;     /* binary output only */
                break;
            case 'a':
                autologin = 1;
                break;
            case 'c':
                skiprc = 1;
                break;
            case 'e':
                GnetTnSetEscapeChar(optarg);
                break;
            case 'l':
                autologin = -1;
                user = optarg;
                break;
            case 'b':
                alias = optarg;
                break;
            case 'r':
                rlogin = '~';
                break;
            }
        }

    if (autologin == -1)
            autologin = (rlogin == _POSIX_VDISABLE) ? 0 : 1;

    argc -= optind;
    argv += optind;

      if (argc)
        {
            gchar *args[7], **argp = args;

            *argp++ = tn_prompt;
            if (user)
                {
                *argp++ = "-l";
                *argp++ = user;
            }
            if (alias)
                {
                *argp++ = "-b";
                *argp++ = alias;
            }
            *argp++ = argv[0];          /* host */
            if (argc > 1)
                *argp++ = argv[1];      /* port */
            *argp = 0;

            if (sigsetjmp(toplevel, 1) != 0)
                GnetTnExit(0);
               return (GnetTnTn(argp - args, args));
      }
}
```

APÉNDICE D - Programas de Ejemplo

Testrsh - rsh

```
/* Programa para prueba de rsh */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{

   gchar hostname[80], *hst;
   gchar usuario[80], *usr;
   gchar comando[80], *com;
   gint  code;

   gnet_init ();

   GnetRsInit();

   hst = &hostname[0];
   usr = &usuario[0];
   com = &comando[0];
   *hst = *usr = *com = '\0';

   printf ("Enter remote username: ");
   (void) fflush(stdout);
   (void) fgets(usuario, sizeof(usuario) - 1, stdin);
   usuario[strlen(usuario) - 1] = '\0';

   if (*usr != '\0')
      GnetRsUserOn(usuario);
```

```
printf ("\n\n\n");
printf ("Enter hostname: ");
(void) fflush(stdout);
(void) fgets(hostname, sizeof(hostname) -1, stdin);
hostname[strlen(hostname) - 1] = '\0';

if (*hst == '\0')
  exit(1);

printf ("\n\n\n");
printf ("Enter command: ");
(void) fflush(stdout);
(void) fgets(comando, sizeof(comando) -1, stdin);
comando[strlen(comando) - 1] = '\0';

printf ("\n\n\n");

code = GnetRsConnect(usr, hst, com);

switch (code)
{
   case 0:
       printf ("Connection closed - Normal termination\n\r");
       break;
   case 1:
       printf ("Invalid number of parameters\n\r");
       break;
   case 2:
       printf ("Can't exec _PATH_RLOGIN\n\r");
       break;
   case 3:
       printf ("Unknown user id\n\r");
       break;
   case 4:
       printf ("Unknown service: shell/tcp\n\r");
       break;
   case 5:
       printf ("Connection failed\n\r");
       break;
```

```
        case 6:
            printf ("Can't establish stderr\n\r");
            break;
        case 7:
            printf ("Fork error\n\r");
            break;
        default:
            break;
    }

    exit(EXIT_SUCCESS);
}
```

Testrexec - rexec

```
/* Programa para prueba de rexec */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>


gint main(gint argc, gchar *argv[])
{

   gchar hostname[80], *hst;
   gchar usuario[80], *usr;
   gchar comando[80], *com;
   gchar password[80], *pwd;
   gint  code;

   gnet_init ();

   GnetRxInit();

   hst = &hostname[0];
   usr = &usuario[0];
   com = &comando[0];
   pwd = &password[0];
   *hst = *usr = *com = *pwd = '\0';

   printf ("Enter remote username: ");
   (void) fflush(stdout);
   (void) fgets(usuario, sizeof(usuario) - 1, stdin);
   usuario[strlen(usuario) - 1] = '\0';

   if (*usr != '\0')
   {
      GnetRxUserOn(usuario);
        printf ("\n\n\n");
        printf ("Enter remote username password: ");
        (void) fflush(stdout);
        (void) fgets(password, sizeof(password) -1, stdin);
        password[strlen(password) - 1] = '\0';

        if (*pwd == '\0')
            exit(1);

        GnetRxPassOn(password);
    }

    printf ("\n\n\n");
    printf ("Enter remote hostname: ");
    (void) fflush(stdout);
    (void) fgets(hostname, sizeof(hostname) -1, stdin);
    hostname[strlen(hostname) - 1] = '\0';

    if (*hst == '\0')
      exit(1);

    printf ("\n\n\n");
    printf ("Enter command: ");
    (void) fflush(stdout);
    (void) fgets(comando, sizeof(comando) -1, stdin);
    comando[strlen(comando) - 1] = '\0';

    printf ("\n\n\n");

    code = GnetRxConnect(hst, usr, pwd, com);

    switch (code)
    {
       case 0:
           printf ("Connection closed - Normal termination\n\r");
           break;
       case 1:
           printf ("Invalid number of parameters - No hostname\n\r");
           break;
       case 2:
           printf ("Invalid number of parameters - No command\n\r");
```

```
            break;
        case 3:
            printf ("Can't lookup uid\n\r");
            break;
        case 4:
            printf ("Can't use without supplying a user and password\n\r");
            break;
        case 5:
            printf ("Error in rexec system call\n\r");
            break;
        case 6:
            printf ("Error in select system call\n\r");
            break;
        case 7:
            printf ("Error from shutdown\n\r");
            break;
        default:
            break;
    }

    exit(EXIT_SUCCESS);
}
```

Testrcp - rcp

```
/* Programa para prueba de rcp */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{

   gchar hostname[80], *hst;
   gchar usuario[80], *usr;
   gchar archivo[80], *arch;
   gchar *file;
   char *pline;
   gint  code;

   gnet_init ();

   GnetRcInit();

   file = NULL;

   /* Construccion path archivo 1 */

   hst = &hostname[0];
   usr = &usuario[0];
   arch = &archivo[0];
   *hst = *usr = *arch = '\0';

   printf ("Enter username (first filename): ");
   (void) fflush(stdout);
   (void) fgets(usuario, sizeof(usuario) - 1, stdin);
   usuario[(int)strlen(usuario) - 1] = '\0';
```

```
printf ("\n\n\n");
printf ("Enter hostname (first filename): ");
(void) fflush(stdout);
(void) fgets(hostname, sizeof(hostname) -1, stdin);
hostname[(int)strlen(hostname) - 1] = '\0';

printf ("\n\n\n");
printf ("Enter path/file (first filename): ");
(void) fflush(stdout);
(void) fgets(archivo, sizeof(archivo) -1, stdin);
archivo[(int)strlen(archivo) - 1] = '\0';

if (*arch == '\0')
      exit(1);

printf ("\n\n\n");

file = GnetRcConstFile(usr, hst, arch);

pline = &line[0];
strcat(pline,file);
strcat(pline," ");

/* Construccion path archivo 2 */

printf ("Enter username (second filename): ");
(void) fflush(stdout);
(void) fgets(usuario, sizeof(usuario) - 1, stdin);
usuario[(int)strlen(usuario) - 1] = '\0';

printf ("\n\n\n");
printf ("Enter hostname (second filename): ");
(void) fflush(stdout);
(void) fgets(hostname, sizeof(hostname) -1, stdin);
hostname[(int)strlen(hostname) - 1] = '\0';

printf ("\n\n\n");
printf ("Enter path/file (second filename): ");
(void) fflush(stdout);
(void) fgets(archivo, sizeof(archivo) -1, stdin);
```

```
   archivo[(int)strlen(archivo) - 1] = '\0';

   if (*arch == '\0')
        exit(1);

   printf ("\n\n\n");

   file = GnetRcConstFile(usr, hst, arch);

   strcat(pline,file);

   GnetRcMakeArgv();
   argc = margc;
   argv = margv;

   /* Remote copy */
   code = GnetRcConnect(argc, argv);

   switch (code)
   {
      case 0:
          printf ("Connection closed - Normal termination\n\r");
          break;
      case 1:
          printf ("Invalid file\n\r");
          break;
      case 2:
          printf ("Unknown user\n\r");
          break;
      case 3:
          printf ("Unknown service: shell/tcp\n\r");
          break;
      default:
          break;
   }

   exit(EXIT_SUCCESS);
}
```

Testrlogin - rlogin

```
/* Programa para prueba de rlogin */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{

   gchar hostname[80], *hst;
   gchar usuario[80], *usr;
   gint  code;

   gnet_init ();

   GnetRlInit();

   hst = usr = NULL;
   hst = &hostname[0];
   usr = &usuario[0];
   *hst = *usr = '\0';

   printf ("Enter remote username: ");

   (void) fflush(stdout);
   (void) fgets(usuario, sizeof(usuario) - 1, stdin);
   usuario[strlen(usuario) - 1] = '\0';

   if (*usr != '\0')
      GnetRlUserOn(usuario);

   printf ("\n\n\n");

   printf ("Enter hostname: ");
   (void) fflush(stdout);
```

```
    (void) fgets(hostname, sizeof(hostname) -1, stdin);
    hostname[strlen(hostname) - 1] = '\0';

    if (*hst == '\0')
      exit(1);

    printf ("\n\n\n");

    code = GnetRlConnect(usr, hst);

    switch (code)
    {
       case 0:
           printf ("Connection closed - Normal termination\n\r");
           break;
       case 1:
           printf ("Invalid number of parameters\n\r");
           break;
       case 2:
           printf ("Must be setuid root\n\r");
           break;
       case 3:
           printf ("Unknown user id\n\r");
           break;
       case 4:
           printf ("Unknown service: login/tcp\n\r");
           break;
       case 5:
           printf ("Connection failed\n\r");
           break;
       case 6:
           printf ("Connection close\n\r");
           break;
       default:
           break;
    }

    exit(EXIT_SUCCESS);
}
```

Testtftp – tftp

```
/* Programa para prueba de tftp: comando mode */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{
       gint  code;
         gchar modo[80], *md;


         gnet_init ();

         md = NULL;
         md = &modo[0];

         printf ("\n\n\n");
         printf ("Enter mode: ");
         (void) fflush(stdout);
         (void) fgets(modo, sizeof(modo) -1, stdin);
         modo[strlen(modo) - 1] = '\0';

         if (*md == '\0')
                 exit(1);

       GnetTFInit();

       code=GnetTFMode(md);

       printf ("\n\n\n");

       switch (code)
       {
         case 0:
                printf ("Normal termination\n\r");
                break;
             case 1:
                    printf ("Invalid mode\n\r");
                    break;
           default:
                    break;
           }

        exit(EXIT_SUCCESS);
}
```

```
/* Programa para prueba de tftp: comando put */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{
        gint  code;
          gchar hostname[80], *hst;
        gchar archivo[80], *arch;

          gnet_init ();

          hst = arch = NULL;
          hst = &hostname[0];
          arch = &archivo[0];

          printf ("\n\n\n");
          printf ("Enter hostname: ");
          (void) fflush(stdout);
          (void) fgets(hostname, sizeof(hostname) -1, stdin);
          hostname[strlen(hostname) - 1] = '\0';

          if (*hst == '\0')
                    exit(1);

        printf ("\n\n\n");
        printf ("Enter file: ");
        (void) fflush(stdout);
        (void) fgets(archivo, sizeof(archivo) -1, stdin);
        archivo[strlen(archivo) - 1] = '\0';

        if (*arch == '\0')
                exit(1);

        GnetTFInit();
        code=GnetTFPut(hst, arch);

        printf ("\n\n\n");

        switch (code)
        {
          case 0:
               printf ("Normal termination\n\r");
               break;
            case 1:
                    printf ("Invalid file\n\r");
                    break;
          case 2:
               printf ("Invalid host\n\r");
               break;
          case 3:
               printf ("No target machine specified\n\r");
               break;
          default:
                    break;
         }

        exit(EXIT_SUCCESS);
}
```

```
/* Programa para prueba de tftp: comando get */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{
        gint  code;
          gchar hostname[80], *hst;
        gchar archivo[80], *arch;

          gnet_init ();

          hst = arch = NULL;
          hst = &hostname[0];
          arch = &archivo[0];

          printf ("\n\n\n");
          printf ("Enter hostname: ");
          (void) fflush(stdout);
          (void) fgets(hostname, sizeof(hostname) -1, stdin);
          hostname[strlen(hostname) - 1] = '\0';

          if (*hst == '\0')
                    exit(1);

        printf ("\n\n\n");
        printf ("Enter file: ");
        (void) fflush(stdout);
        (void) fgets(archivo, sizeof(archivo) -1, stdin);
        archivo[strlen(archivo) - 1] = '\0';

        if (*arch == '\0')
                exit(1);

        GnetTFInit();
        code=GnetTFGet(hst, arch);

        printf ("\n\n\n");

        switch (code)
        {
           case 0:
                printf ("Normal termination\n\r");
                break;
             case 1:
                    printf ("Invalid file\n\r");
                    break;
           case 2:
                printf ("Invalid host\n\r");
                break;
           case 3:
                printf ("No target machine specified\n\r");
                break;
           default:
                    break;
           }

        exit(EXIT_SUCCESS);
}
```

```
/* Programa para prueba de tftp: comando rexmt */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{
       gint  code;
         gchar mrexmt[80], *mrmt;

         gnet_init ();

         mrmt = NULL;
         mrmt = &mrexmt[0];

         printf ("\n\n\n");
         printf ("Enter rexmt value: ");
         (void) fflush(stdout);
         (void) fgets(mrexmt, sizeof(mrexmt) -1, stdin);
         mrexmt[strlen(mrexmt) - 1] = '\0';

         if (*mrmt == '\0')
                 exit(1);

       GnetTFInit();

       code=GnetTFRexmt(mrmt);

       printf ("\n\n\n");

       switch (code)
       {
         case 0:
              printf ("Normal termination\n\r");
              break;
           case 1:
                 printf ("Bad value\n\r");
```

```
                        break;
                default:
                        break;
                }

        exit(EXIT_SUCCESS);
}
```

```
/* Programa para prueba de tftp: comando timeout */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>

gint main(gint argc, gchar *argv[])
{
       gint  code;
         gchar mtimeout[80], *tm;

         gnet_init ();

         tm = NULL;
         tm = &mtimeout[0];

         printf ("\n\n\n");
         printf ("Enter timeout value: ");
         (void) fflush(stdout);
         (void) fgets(mtimeout, sizeof(mtimeout) -1, stdin);
         mtimeout[strlen(mtimeout) - 1] = '\0';

         if (*tm == '\0')
                  exit(1);

       GnetTFInit();

       code=GnetTFTimeout(tm);

       printf ("\n\n\n");

       switch (code)
       {
          case 0:
              printf ("Normal termination\n\r");
              break;
            case 1:
                  printf ("Bad value\n\r");
```

```
                        break;
                default:
                        break;
                }

        exit(EXIT_SUCCESS);
}
```

Testtelnet - telnet

```
/* Programa para prueba de telnet */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <gnet.h>


gchar tn_line[256];
gint margc;
gchar *margv[20];


gint main(gint argc, gchar *argv[])
{
   gchar hostrmt[80], *hst;
   gint  code=1, fhst;
   gchar *pline;
   gchar *pargv;

   gnet_init ();

   pline = &tn_line[0];
   strcat(pline,argv[0]);
   strcat(pline," ");

   hst = NULL;
   hst = &hostrmt[0];
   *hst = '\0';

   if (argc > 1)
   {
     fhst = FALSE;
     while ( code < argc)
     {
      if (*argv[code] == '-')
        {
                pargv = argv[code];
                pargv++;
                if ( (*pargv == 'b') || (*pargv == 'e') || (*pargv == 'l') )
                        code++;
        }
      else
        {
          if (fhst == FALSE)
               fhst = TRUE;
          else
             break;
        }
        code++;
       }
       if (fhst)
          goto conexion;
   }

   printf ("Enter hostname: ");
   (void) fflush(stdout);
   (void) fgets(hostrmt, sizeof(hostrmt) -1, stdin);
   hostrmt[strlen(hostrmt) - 1] = '\0';
   printf ("\n\n\n");

   if (*hst == '\0')
   {
     printf ("No hostname!!!\n\n");
     exit(1);
   }

   strcat(pline,hst);

   GnetTnMakeArgv();
   argc = margc;
   argv = margv;

conexion:
   code = GnetTnConnect(argc, argv);
```

```
    switch (code)
    {
        case 0:
            printf ("Connection closed - Normal termination\n\r");
            break;
        case 1:
            printf ("Already connected to hostname\n\r");
            break;
        case 2:
            printf ("Parameters error\n\r");
            break;
        case 3:
            printf ("Error (IP_OPTIONS && IPPROTO_IP)\n\r");
            break;
        case 4:
            printf ("Bad source route option (IP_OPTIONS && IPPROTO_IP)\n\
r");
            break;
        case 5:
            printf ("Error host\n\r");
            break;
        case 6:
            printf ("Bad port number\n\r");
            break;
        case 7:
            printf ("Unknown service: tcp/telnet\n\r");
            break;
        case 8:
            printf ("Socket error\n\r");
            break;
        case 9:
            printf ("Alias error\n\r");
            break;
        case 10:
            printf ("Unable to connect to remote host\n\r");
            break;
        default:
            break;
    }
```

```
   exit(EXIT_SUCCESS);
}
```

BIBLIOGRAFÍA

[1] Redes de Computadoras, Andrew S. Tanenbaum, Ed Prentice Hall, Tercera Edición

[2] The Desing of The Unix Operating System, Bach, Prentice Hall Unix Network Programming, Stevens, Prentice Hall

[3] Internetworking with TCP/IP, Comer-Stevens, Prentice Hall

[4] Distributed Operating Systems. The logical Desing, Goschinski, Addison Wesley

[5] Modern Operating Systems, Tanenbaum, Prentice Hall

[6] Distributed Operating Systems, Tanenbaum, Prentice Hall

[7] Distributed Operating Systems, Sinha, IEEE-Press

SITIOS EN INTERNET

GNet

- http://www.gnetlibrary.org
- http://www.gtk.org
- http://mesh.eecs.umich.edu/projects.phtml

Aplicaciones que utilizan GNet

- http://gabber.sourceforge.net/
- http://gchch.sourceforge.net/
- http://gnomeicu.sourceforge.net/
- http://www.junglemonkey.net/
- http://www.ximian.com/

RFCs

- http://www.rfc-editor.org/