# Programando para inetd


## Juan J. Martínez

**reidrac@usebox.net**



Este artículo describe la programación de demonios de *inetd* mediante el ejemplo de un pequeño servidor HTTP.

Se ha empleado OpenBSD 3.3-RELEASE para la impementación, aunque lo que se explica es común a cualquier sistema BSD, salvo quizás algunos matices de la configuración de **inetd**.

*Juan J. Martínez tiene 26 años y trabaja como analista-programador en una empresa dedicada a los servicios web. Juanjo participa en la iniciativa wireless de su ciudad, donde promueve el uso de OpenBSD en los puntos de acceso.*


## Tabla de contenidos

# 1. Introducción: el 'super server'

**inetd** es un demonio que se encarga de proporcionar una serie de servicios mediante la invocación de otros programas. Es el 'super servidor de internet' porque se encarga de servir servidores.

El demonio **inetd** apareció en 4.3BSD y desde entonces ha cambiado mucho su uso debido a que los servicios que se proporcionan a través de él tienen poco rendimiento, sobretodo si deben manejar un alto número de peticiones concurrentes. Esto puede variar según la implementación de **inetd**, pero por lo general el super servidor no puede competir con un demonio especializado ya que **inetd** debe crear un nuevo proceso para que se encarge de procesar cada petición.

**inetd** está indicado finalmente para servicios que tienen que contestar a peticiones esporádicas o que soportan un tráfico de medio a bajo. No obstante la sencillez con la que se pueden programar los servicios hace muy interesante emplear **inetd** para implementaciones de referencia o prototipos rápidos, ya que el programador puede desentenderse de parte de la programación específica en red y centrarse en las tareas concretas del propio servicio.

# 2. Configurando inetd

**inetd** escucha esperando conexiones en ciertos puertos. Cuando una conexión se produce en uno de dichos puertos el super servidor se encarga de de invocar al programa adecuado para satisfacer el servicio.

Una vez arranca **inetd**, intenta leer su fichero de configuración, que por defecto es `/etc/inetd.conf`. El formato de este fichero es sencillo y soporta comentarios con #, ignorándose el resto de la linea.

Fragmento de configuración:

```
# un servicio de ejemplo
ident          stream  tcp     wait    identd  /usr/sbin/identd        identd
```

Cada linea de configuración consta de una serie de campos separados por espacios o tabuladores. Es necesario especificar todos los campos, que son:

```
nombre del servicio
tipo de socket
protocolo
espera/no espera[.max]
usuario[.grupo] o usuario[:grupo]
programa servidor
argumentos para el programa servidor
```

Vamos a estudiar cada uno de los campos.

**Nombre del servicio**: Este nombre debe estar listado en `/etc/services` e indica al super-servidor que puerto debe escuchar. En un servidor HTTP como el demonio de ejemplo que veremos más adelante sería 'www'.

**Tipo de socket**: Aquí puede ponerse uno de los siguientes valores: stream, dgram, raw, rdm, seqpacket; dependiendo del tipo de socket. En el ejemplo se va a emplear el tipo 'stream', que es el que se usa en el protocolo HTTP.

**Protocolo**: Se trata de un protocolo válido de los listados en `/etc/protocols`, como por ejemplo tcp o udp. Nuestro demonio va a emplear 'tcp'.

**Espera/no espera[.max]**: Indica si **inetd** debe esperar a que termine el proceso encargado de responder a la petición o, por el contrario, ponerse a esperar más peticiones inmediatamente después de ejecutar el programa encargado de atender el servicio. El campo [.max] es opcional e indica cuantas instancias lanzará **inetd** del proceso en 60 segundos (por defecto 256). En este caso se va a emplear 'nowait' ya que el demonio implementado solo es capaz de procesar una petición a la vez, por lo que necesitará que el super-servidor ejecute una instancia por petición.

**Usuario[.grupo] o usuario[:grupo]**: Indica el usuario (indicando opctionalmente el grupo) que debe ejecutar el servicio. Esto es muy importante porque, por lo general, emplearemos un usuario con los permisos necesarios para realizar el trabajo, y nada más. En este caso se emplea 'nobody'.

**Programa servidor**: El programa que se encargará del servicio, incluyendo la path completa.

**Argumentos para el programa servidor**: Estos son los argumentos que se pasarán al servidor, pasando siempre como primer argumento el nombre del programa. En el demonio de ejemplo no son necesarios argumentos, pero *sí* se debe pasar uno, que es el nombre del programa: **db-httpd**.

Resumiendo, así va a quedar la llamada al demonio **db-httpd**:

```
www stream tcp nowait nobody /usr/local/libexec/db-httpd/db-httpd db-httpd
```

# 3. Ejemplo de demonio: db-httpd

La programación con **inetd** es sencilla: lo que leamos de la entrada estándar corresponderá con lo que nos envían los clientes y lo que escribamos en en la salida estándar será lo que se envíe desde el servidor.

Esta sencillez nos permitirá realizar servidores con lenguajes de programación que, sin disponer de funciones para trabajar en red, sí disponen mecanismos para manipular la entrada y la salida estándar.

En este caso vamos a realizar un pequeño servidor de páginas web un tanto especial. Nuestro **db-httpd** será capaz de servir recursos mediante el protocolo HTTP, pero estos recursos se encontrarán en una base de datos y no el sistema de ficheros como es más habitual.

Además se ha implementado en unas 80 líneas de código shell, junto a un sencillo interface en C para la base de datos.

## 3.1. Diseño del demonio

La mayor parte del trabajo realizado por el servidor HTTP se lleva a cabo empleando herramientas disponibles en cualquier entorno de shell.

Las herramientas empleadas son: `date(1)`, `echo(1)`, `test(1)`, `printf(1)`, `sed(1)` y `tr(1)`.

La interfaz para la bases de datos es simple. Se trata de un programa que dada una clave se encarga de devolver el recurso asociado para una base de datos concreta.

Para aumentar la funcionalidad del servidor se han definido dos bases de datos: `httpd-index` y `httpd-db`.

La primera base de datos se encarga de indexar los recursos asociando pares CAMINO/RECURSO, donde RECURSO puede ser el camino a un fichero en disco u otra clave con la que obtener el recurso propiamente dicho de la segunda base de datos.

De esta forma podemos servir páginas directamente desde disco o desde la base de datos, empleando unicamente llamadas al programa auxiliar.

## 3.2. Implementación

Para la implementación se ha empleado la shell `ksh` para el servidor HTTP y C para las herramientas que manejan la base de datos.

Como *backend* de la base de datos se ha elegido `ndbm` por su sencillez y disponibilidad, aunque podría emplear algún sistema de gestión de bases de datos como MySQL respetando el sencillo interface sin ninguna dificultad.

### 3.2.1. NDBM

Se han implementado dos herramientas para manipular las bases de datos con `ndbm(3)`, aunque solo una de ellas será invocada por el servidor, quedando la otra relegada a tareas de mantenimiendo, como generar las bases de datos.

#### *3.2.1.1. mkhttpd-db*

Esta es la herramienta administrativa que se encarga de leer de la entrada estándar los pares CAMINO/RECURSO para generar las bases de datos `httpd-index.db` y `httpd-db.db`.

Su funcionamiento es sencillo: se introduce en la base de datos `httpd-index.db` el valor RECURSO empleando como clave CAMINO. Si RECURSO comienza por `db:`, entonces se emplea RECURSO como clave para almacenar el fichero completo en la base de datos `httpd-db.db`.

Se exige que el fichero exista si debe alojarse en la base de datos, sino no es necesario.

Por ejemplo, si disponemos un fichero llamado `httpd-index` con el siguiente contenido:

```
/ db:/var/www/index.html
/index.html db:/var/www/index.html
/imagen.jpg /var/www/imagen.jpg
```

Y ejecutamos:

```
# mkhttpd-db < httpd-index
httpd-index.db: 3 line(s) processed
```

Tendremos en `httpd-index.db` entradas para `/`, `/index.html` e `/index.jpg`, con las tres primeras apuntando cada una a su recurso correspondiente en `index-db.db` y con la última entrada apuntando al fichero en disco `/var/www/imagen.jpg`.

El código de esta herramienta puede parecer algo complicado debido a que se hace control de errores:

```
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#include"ndbm.h"

int
main(int argc, char *argv[])
{
char  idx[1024],
file[1024];

FILE *fd;
char *buffer;
struct stat fs;

DBM  *db_idx,
*db_db;
datum  key,
data;

int  line;

db_idx=dbm_open(SERVER_INDEX, O_CREAT | O_TRUNC | O_RDWR, 0664);
if(!db_idx)
{
perror(argv[0]);
return 1;
}

db_db=dbm_open(SERVER_DB, O_CREAT | O_TRUNC | O_RDWR, 0664);
if(!db_db)
{
perror(argv[0]);
return 1;
}

line=1;
while(!feof(stdin))
{
if(fscanf(stdin,"%1024[^ ] %1024[^\n]\n", idx, file)==2)
{
key.dptr=idx;
key.dsize=strlen(idx);

data.dptr=file;
data.dsize=strlen(file)+1;
```

```
dbm_store(db_idx, key, data, DBM_REPLACE);

if(!strncmp(file,"db:",3))
{
if(strlen(file)>3) {
if(stat(file+3,&fs)!=-1)
{
buffer=(char *)malloc(fs.st_size);
if(buffer)
{
fd=fopen(file+3,"r");
if(fd)
{
fread(buffer,fs.st_size,1,fd);
data.dptr=buffer;
data.dsize=fs.st_size;

key.dptr=file;
key.dsize=strlen(file);

dbm_store(db_db, key, data, DBM_REPLACE);

fclose(fd);
} else
fprintf(stderr,"%s: failed to add file arround line %i\n", argv[0], line);


free(buffer);
} else
fprintf(stderr,"%s: failed to add file arround line %i\n", argv[0], line);
} else
fprintf(stderr,"%s: file not found arround line %i\n", argv[0], line);
} else
fprintf(stderr,"%s: null file arround line %i\n", argv[0], line);
}
} else
fprintf(stderr,"%s: error arround line %i\n",
argv[0], line);
line++;
}

printf("%s.db: %i line(s) processed\n", SERVER_INDEX, line-1);

dbm_close(db_idx);
dbm_close(db_db);

return 0;
}
```

#### *3.2.1.2. httpd-fetch*

Esta es la herramienta que llamará el servidor para acceder a los elementos almacenados en la bases de datos.

Su funcinamiento es el siguiente: requiere de 2 argumentos, el nombre de la base de datos (sin la extensión .db) y la clave del recurso a obtener.

Si la clave se encuentra en la base de datos, el programa devuelve 0 y escribe en la salida estándar el recurso asociado. En caso de no encontrar la clave, el programa devuelve un 1.

Con este comportamiento tan sencillo el servidor se encarga de hacer una primera llamada preguntando por el CAMINO en la base de datos `httpd-index`. Si se devuelve un 1 el servidor muestra un error *HTTP 404*.

Si el CAMINO está en la base de datos comprueba si el recurso devuelto comienza por *db:*. En caso negativo simplemente se envía el fichero apuntado por ese recurso (en caso de existir en disco, sino se devuelve un error *HTTP 404*). Si por el contrario el recurso comienza por *db:*, el servidor realiza otra llamada a **httpd-fetch**, esta vez empleando la base de datos `httpd-db` como primer argumento y el recurso obtenido de la llamada anterior como clave para la búsqueda. El resultado de esta petición se envía directamente al cliente como respuesta.

El código de esta herramienta es muy simple, pese a la gran funcionalidad que aporta al servidor:

```
#include<fcntl.h>
#include<stdio.h>

#include"ndbm.h"

int
main(int argc, char *argv[])
{
DBM  *db;
datum  key,
data;

if(argc<3)
{
printf("%s db key\n", argv[0]);
return 1;
}

db=dbm_open(argv[1], O_RDONLY, 0);
if(!db)
{
printf("%s: failed to fetch from %s.db\n", argv[0], argv[1]);
return 1;
}

key.dptr=argv[2];
key.dsize=strlen(argv[2]);
        data=dbm_fetch(db, key);

if(data.dptr)
fwrite(data.dptr,data.dsize,1,stdout);
```

```
dbm_close(db);
return 0;
}
```

### 3.2.2. db-httpd

Esta es la parte principal del servidor. Solo procesa peticiones **GET** e implementa un mínimo del protocolo.

Se ha extraido del fuente la identificación del tipo MIME de cada recurso al fichero `mime.types` por comodidad, aunque no sería necesario.

Ya que este es el servidor propiamente dicho, merece la pena estudiarlo linea a linea:

```
#!/bin/sh

# variables de configuración
# se generan en la instalación, no tocar
LANG=en_EN
SERVER_NAME="db-http"
SERVER_URL="http://www.usebox.net/jjm/db-httpd/"
SERVER_FETCH="/usr/local/libexec/db-httpd/httpd-fetch"
SERVER_INDEX="/usr/local/etc/db-httpd/httpd-index"
SERVER_DB="/usr/local/etc/db-httpd/httpd-db"
SERVER_MIME="/usr/local/etc/db-httpd/mime.types"

#
# funcion auxiliar que envía las cabeceras HTTP
#
# entrada: ESTADO DESCRIPCION_CORTA TIPO_MIME
#
function send_headers
{
NOW=`date +"%a, %d %b %Y %H:%M:%S GMT"`

echo -e "HTTP/1.0 $1 $2\r"
echo -e "Server: $SERVER_NAME\r"
echo -e "Date: $NOW\r"

if [ $3 ]; then
echo -e "Content-Type: $3\r"
fi

echo -e "Connection: close\r\n\r"
}

#
# funcion auxiliar que envia una página HTML de error
#
```

```
# entrada: ESTADO DESCRIPCION_CORTA DESCRIPCION_LARGA
#
function send_error
{
send_headers "$1" "$2" "text/html"

echo -e "<html>\n<head><title>HTTP ERROR $1</title></head>\n"
echo -e "<body bgcolor=#f0e0a0>\n<h2>$1 - $2</h2>\n<p><b>$3</b>\n"
echo -e "<hr>\n<small>$SERVER_NAME, $SERVER_URL</small>\n</body>\n"
echo -e "</html>\n"
}

# *** Comienza la ejecución ***

# leemos 3 cadenas: método, el recurso que nos piden y el protocolo
read METHOD RESOURCE_INDEX PROTOCOL

# es necesario leer el resto de la petición, aunque no la usamos para nada
ENDL=`printf "\r\n"`
read DUMMY
while [ $? = 0 -a "$DUMMY" != "" -a "$DUMMY" != "$ENDL" ]; do
read DUMMY
done

# si el método no es 'get' devolvemos una página de error
METHOD=`echo -n $METHOD | tr [:upper:] [:lower:]`
if [ "$METHOD" != "get" ]; then
send_error "501" "Not implemented" "That method is not implemented."
exit 1
fi

# primera llamada a base de datos
RESOURCE_PATH=`$SERVER_FETCH $SERVER_INDEX $RESOURCE_INDEX`

# si no se puede consultar la base de datos,
# mostramos una página de error
if [ $? = 1 ]; then
send_error "500" "Internal error" "$RESOURCE_PATH"
exit 1
fi

# incluimos el fuente que procesa el recurso para obtener
# el tipo MIME - el tipo mime se almacena en la variable MIME
. $SERVER_MIME

# si el recurso comienza por 'db:', se realiza la segunda llamada
# a base de datos devolviendo al cliente el recusto obtenido esta vez
if [ "yes" = "`echo "$RESOURCE_PATH" | sed 's/db:.*/yes/'`" ]; then
send_headers "200" "OK" $MIME
$SERVER_FETCH $SERVER_DB $RESOURCE_PATH

exit 0
fi
```

```
# si no es un recurso almacenado, enviamos al cliente el fichero
# correspondiente si es que existe
if [ -f "$RESOURCE_PATH" ]; then
send_headers "200" "OK" $MIME

cat $RESOURCE_PATH
exit 0
fi

# sino existe, mostramos una página de error
send_error "404" "Not found" "The requested resource is not available."

exit 1

# EOF
```

El fichero `mime.types`:

```
case "$RESOURCE_PATH" in

*.html | *.htm) MIME="text/html";;
*.jpg | *.jpeg | *.jpe) MIME="image/jpeg";;
*.gif) MIME="image/gif";;
*.png) MIME="image/png";;

*) MIME="text/plain";;

esac

# EOF
```

## 4. Conclusiones

El protocolo elegido para el demonio de ejemplo es relativamente sencillo, al menos la implementación de una pequeña parte. Pero el factor determinante para que el código del demonio fuera simple ha sido el uso de **inetd** como herramienta.

Pero no todo son ventajas. En este caso el lenguaje utilizado para programar el servidor no permite disponer de mecanismos para evitar ataques de denegación de servicio.

Si un cliente malintencionado quisiera atacar un servidor empleando **db-httpd**, solo tendría que hacer conexiones y no realizar ninguna petición. El demonio quedaría entoces bloqueado esperando leer de la entrada estándar. Es cierto que la configuración del super-servidor nos permite minimizar el impacto de estos ataques, dista mucho de ser una solución óptima para un sistema en producción.

Una buena mejora para el demonio de ejemplo sería posible empleando un shell como **BASH**, que permite dar un parámetro extra a las llamadas **read** con el que especificar un tiempo máximo de espera (*timeout*), evitando así el bloqueo indefinido.

Por último hay que mencionar que, junto a los posibles problemas que se pueden dar debido a la posibilidad de emplear lenguajes poco habituales en la programación de servicios de red, **inetd** posee sus propias limitaciones, como es la posibilidad de llevar un registro de las peticiones (logs).

# A. Documentación adicional.

La página de manual de inetd(8) (http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8&arch=i386&apropos=0&manpath=OpenBSD+Current).

La sección de inetd en el handbook (http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/network-inetd.html) de FreeBSD.

El RFC1945: Hypertext Transfer Protocol -- HTTP/1.0.

Hay varios demonios HTTP implementados con inetd en los más pintorescos lenguajes:

- bash-httpd: http://lug.umbc.edu/~mabzug1/bash-httpd.html
- AWKhttpd: http://awk.geht.net:81/README.html
- ps-httpd: http://www.pugo.org:8080/
- micro_httpd: http://www.acme.com/software/micro_httpd/
- weblet: http://leaf.sourceforge.net/devel/cstein/Packages/weblet.htm

También se ha empaquetado **db-httpd** y se puede descargar desde: http://www.usebox.net/jjm/db-httpd/.