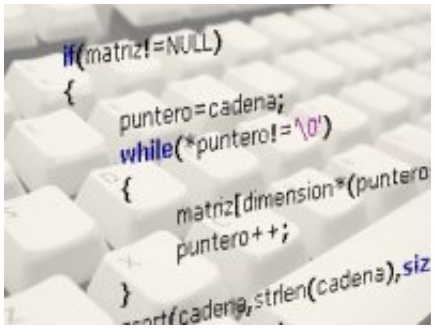
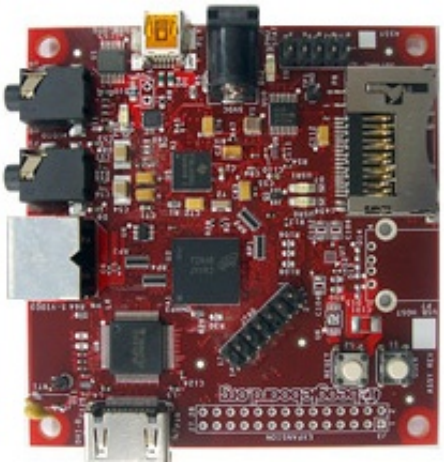
SISTEMAS O.R.P

Blog sobre informática, electrónica, robótica y otros temas

Crear un demonio en el linux embebido en placas

Ahora que están de moda las placas que contienen un linux embebido como las beagleboard, las raspberry pi o las foneras, he creido interesante escribir un artículo en el cual se explica cómo ejecutar una tarea por tiempo indefinido desde que arranca la placa y sin necesidad de acceder por shell a la misma.





Presupongo que tienes el compilador cruzado que genera ejecutables para la plataforma de la placa y que tienes acceso a ella mediante shell con el usuario root.

Nuestro objetivo es sencillo, crear lo que se denomina en entornos linux y unix un **demonio** (en windows es un servicio) lo más sencillo posible. Un demonio no es nada más que una aplicación que se ejecuta en segundo plano y no muestra directamente datos al usuario ni este puede interactuar con ella.

Como queremos que el ejemplo sea bastante sencillo nuestro demonio simplemente va a escribir en un fichero una cadena cada cierto tiempo. Vosotros podreis modificarlo para que haga lo que querais (leer y escribir datos en un puerto serie, analizar imágenes de una webcam, manejar GPIO, crear un servicio para internet, etc).

Scripts de demonios:

Normalmente cuando se arranca un sistema linux, después de cargar el kernel se ejecuta el proceso init, que es el proceso padre del que dependerán el resto de procesos que se ejecuten en el sistema. Este proceso lee el fichero */etc/inittab* para saber entre otras cosas en qué nivel de arranque (runlevel de 1 a 5) debe iniciar el sistema.

- 0 Detener o apagar el sistema
- 1 Modo monousuario, generalmente utilizado para mantenimiento del sistema
- 2 Modo multiusuario, pero sin soporte de red
- 3 Modo multiusuario completo, con servicios de red
- 4 No se usa, puede usarse para un inicio personalizado
- 5 Modo multiusuario completo con inicio gráfico (X Window)
- 6 Modo de reinicio (reset)

Dependiendo de ese nivel de arranque se ejecutarán unos scripts ubicados en la carpeta

correspondiente. Así por ejemplo si el nivel es 3, los scripts de la carpeta */etc/rc3.d* que empiecen por S (Start) se ejecutarían al arrancar. Estos scripts tienen un número que indica en qué orden se ejecutan (se puede repetir el número).

Pero… un momento… ¡¡¡ Si estos ficheros son en realidad son enlaces simbólicos !!!.

Efectivamente, en realidad son accesos directos que apuntan al directorio */etc/init.d* donde realmente están los scripts para los demonios. El proceso init lee de la carpeta */etc/rc3.d* los enlaces que empiezan por S y ejecuta el script al que apunta con el parámetro *start*. Igualmente cuando se apaga o se reinicia el sistema el proceso init lee de la carpeta */etc/rc0.d* o */etc/rc6.d* respectivamente los enlaces que empiezan por K y ejecuta el script al que apunta con el parámetro *stop*.

Los scripts que se encuentran en la carpeta */etc/init.d* son en realidad scripts shell que dependiendo del parámetro que se le pase (start, stop, reload, etc) ejecuta el comando necesario para lo que se requiere. Ni que decir tiene que igualmente nosotros podemos ejecutar por nuestra cuenta el script con los parámetros mencionados para parar o arrancar el demonio desde una consola.

Por tanto vamos a crear un sencillo script que arranque o pare nuestro demonio. Lo crearemos dentro del directorio */etc/init.d* y lo llamaremos *nohacenada* con el siguiente contenido:

```
1  #!/bin/sh
2
3  case "$1" in
4          start)
5                 /usr/sbin/nohacenada
6                 ;;
7          stop)
8                 test -e /var/run/nohacenada.pid || exit 2
9                 kill `cat /var/run/nohacenada.pid`
10                ;;
11         *)
12                echo "El uso es: $0 {start|stop}"
13                exit 1
14                ;;
15 esac
16 exit 0
```

Este ejemplo muestra un sencillo script donde se espera como parámetro una cadena *start* o *stop*. Si es *start* simplemente arranca la aplicación. Si es *stop* comprueba si existe el fichero */var/run/nohacenada.pid* para saber si la aplicación está corriendo, después lee su contenido, que es el identificador del proceso, para posteriormente mandarle una señal kill y que este pueda cerrarse por si sóla limpiamente. Si no es ninguno de los dos parámetros anteriores es que el usuario intentó ejecutar el script por su cuenta sin poner *start* o *stop* como parámetro.

El script debe tener permisos 755 y pertenecer a root por lo que debemos dárselo de esta forma desde una consola shell con el usuario root:

```
chmod 755 /etc/init.d/nohacenada
chown root:root /etc/init.d/nohacenada
```

Además hay que crear los enlaces simbólicos en el directorio correspondiente para que init sepa que tiene que arrancarlo (niveles 2, 3, 4 y 5) y pararlo (niveles 0, 1 y 6):

```
ln -s  /etc/init.d/nohacenada /etc/rc1.d/S76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc2.d/S76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc3.d/S76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc4.d/S76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc5.d/S76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc0.d/K76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc1.d/K76nohacenada
ln -s  /etc/init.d/nohacenada /etc/rc6.d/K76nohacenada
```

o si te lo permite el linux que tienes instalado, simplemente:

```
update-rc.d nohacenada defaults 76
```

Con estos pasos ya hemos terminado nuestro script para que el proceso init pueda arrancar o parar nuestro demonio.

El demonio:

Para crear nuestro demonio usaremos el lenguaje C, que es el lenguaje estandar de linux. Nuestra aplicación se convertirá en demonio para quedar residente en el sistema permanentemente hasta que se termine mediante kill.

Programaremos nuestra aplicación para que informe de los errores en el log, genere un fichero pid, cree un hilo de ejecución que se convierta en demonio y escriba una cadena cada segundo mientras espera a que se termine la aplicación mediante una señal.

Dado que la aplicación no muestra datos al usuario este no sabe que está pasando por dentro, por eso habrá que informarle a través de los log. De esto se encargan las funciones openlog y syslog. Toda la información podrá ser consultada en el fichero */var/log/syslog*.

Muchos demonios generan en el directorio */var/run* un fichero con el nombre de la aplicación y extensión pid. Dentro guardan el PID o identificador de proceso en formato numérico. Con esto se obtienen dos ventajas: por un lado la misma aplicación sabe si ya hay una copia de ella misma ejecutandose para no duplicar procesos; por otro lado como guarda el pid dentro del fichero cualquier proceso puede leerlo para poder comunicarse con la aplicación (por ejemplo el comando *kill* que explicaré más adelante).

Para crear un demonio la aplicación principal crea un hilo de ejecución con fork y la aplicación principal

se saldrá. Esto provoca que el hilo de ejecución se quede huerfano. Después se ejecuta umask para no heredar los permisos de los ficheros del proceso padre. A continuación se ejecuta setsid para que el proceso huerfano tenga su propio SID y sea un proceso independiente para el sistema operativo. Por otro lado cambiaremos de directorio de ejecución para apuntar a la raiz con un chdir ya que es el único directorio seguro que existe en linux. Finalmente cerraremos con un close la entrada estandar, la salida estandar y la salida de errores ya que no las necesitamos.

Con la función signal haremos que cuando el usuario (o init) nos envíe un comando *kill* (sin el -9) salgamos de la aplicación limpiamente. Esto es debido a que cuando se ejecuta *kill <pid>* lo que se está haciendo realmente es enviar una señal SIGTERM a la aplicación. La aplicación recupera la señal y ejecuta la función asociada mediante la función *signal*. En el ejemplo de más abajo veremos que se asocia a la función *adios* que simplemente cambia una variable para que se salga del bucle principal. Mientra se espera a recibir la señal, cada segundo se está escribiendo en el fichero */tmp/nohacenada.txt* una cadena.

Al salir del bucle se escribirá en el fichero anterior otra cadena y cerramos ese fichero. Igualmente cerraremos el log y se borrará el fichero pid para que se pueda volver a ejecutar la aplicación nuevamente (en un arranque del sistema o nosotros mismos).

Dejo el código fuente de la aplicación llamado *nohacenada.c* para que os hagais una idea y poder hacer vuestros demonios basados en este:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <syslog.h>
#include <signal.h>

#define FICHERO_PID "/var/run/nohacenada.pid"

volatile char estado = 0;

/* Función llamada cuando se recibe una señal SIGTERM */
void adios( int signum )
{
    estado = 1;
}

int main( void )
{
    struct stat st;
    FILE *fichero_pid;
    FILE *fichero_prueba;
    pid_t pid;
    pid_t sid;

    /* Activamos el manejo de mensajes al demonio de syslogd */
    openlog( "nohacenada", LOG_CONS | LOG_PID, LOG_DAEMON );
```

```
29
30      /* Comprobamos si existe le fichero PID */
31      if( stat( FICHERO_PID, &st ) == 0 )
32      {
33          syslog( LOG_ERR, "Ya existe un proceso similar cargado" );
34          exit( EXIT_FAILURE );
35      }
36
37      /* Creamos el fichero PID */
38      fichero_pid = fopen( FICHERO_PID, "w" );
39      if( fichero_pid == NULL )
40      {
41                  syslog( LOG_ERR, "No se pudo crear el fichero PID" );
42                  exit( EXIT_FAILURE );
43      }
44
45      /* Creamos un hilo de ejecucion */
46      pid = fork();
47
48      /* Si no se pudo crear el hilo */
49      if( pid < 0 )
50      {
51          syslog( LOG_ERR, "No se pudo crear el proceso hijo" );
52          fclose( fichero_pid );
53          exit( EXIT_FAILURE );
54      }
55      /* Si se pudo crear el hilo */
56      else if( pid > 0 )
57      {
58              /* Escribimos en el fichero PID el identificador del proceso */
59              fprintf( fichero_pid , "%d\n", pid );
60              fclose( fichero_pid );
61          exit( EXIT_SUCCESS );
62      }
63
64      /* Se evita el heredar la máscara de ficheros */
65      umask( 0 );
66
67      /* Convertimos el proceso hijo a demonio */
68      sid = setsid();
69
70      /* Si no se pudo convertir en demonio */
71      if( sid < 0 )
72      {
73          syslog( LOG_ERR, "No se pudo crear el demonio" );
74          unlink( FICHERO_PID );
75          exit( EXIT_FAILURE );
76      }
77
78      /* Cambiamos al directorio raiz */
79      if( chdir( "/" ) < 0 )
80      {
81          syslog( LOG_ERR, "No se pudo cambiar el directorio" );
82          unlink( FICHERO_PID );
83          exit( EXIT_FAILURE );
84      }
85
86      /* Cerramos los descriptores de archivo que no usaremos */
87      close( STDIN_FILENO );
88      close( STDOUT_FILENO );
89      close( STDERR_FILENO );
```

```
90
91      /* Abrimos el fichero de prueba */
92      fichero_prueba = fopen( "/tmp/nohacenada.txt", "w" );
93      if( fichero_prueba == NULL )
94      {
95          syslog( LOG_ERR, "No se pudo crear el fichero de prueba" );
96          unlink( FICHERO_PID );
97          exit( EXIT_FAILURE );
98      }
99
100     /* Registramos la función que recibe la señal SIGTERM */
101     signal( SIGTERM, adios );
102
103     /* Bucle principal que se ejecuta hasta que se reciba un SIGTERM (ejecución del
104     while ( estado == 0 )
105     {
106         /* Escribimos la cadena, nos aseguramos que se vuelque al disco y esperamos
107         fprintf( fichero_prueba, "prueba\n" );
108         fflush( fichero_prueba );
109         sleep( 1 );
110     }
111
112     fprintf( fichero_prueba, "me voy\n" );
113     /* Cerramos el fichero de prueba */
114     fclose( fichero_prueba );
115     /* Borramos el fichero PID */
116         unlink( FICHERO_PID );
117     /* Cerramos el log */
118     closelog();
119 }
```

Para compilarlo depende del toolchain para cada plataforma, pero sería algo parecido a esto:

```
1 gcc -o nohacenada nohacenada.c
```

Como seguramente hayais instalado un compilador cruzado en vuestro ordenador de sobremesa, habrá que subir el fichero ejecutable a la placa. Yo recomiendo que useis *scp* en linux o winscp en windows si la placa tiene acceso a la red ethernet o wifi ya que hace uso del demonio sshd (el que se usa para las sesiones de consola remotas por ssh).

Una vez subido el fichero a la placa debemos darle permisos de ejecución, asignarle como propietario al usuario root y moverlo a la ruta */usr/sbin* a través de la consola shell con el usuario root de la siguiente manera:

```
1 chmod 775 nohacenada
2 chown root:root nohacenada
3 mv nohacenada /usr/sbin
```

Ahora podeis probar si funciona realmente simplemente ejecutando:

```
1 /etc/init.d/nohacenada start
```

Podeis comprobar en la carpeta */tmp* si hay un fichero llamado *nohacenada.txt* y consultarlo, también

podeis mirar en la carpeta */var/run* si existe un fichero llamado *nohacenada.pid* y visualizarlo. Si no veis nada podeis mirar el fichero */var/log/syslog* para ver si hubiese algún error de la aplicación nohacenada.

Y para pararlo:

```
1 /etc/init.d/nohacenada stop
```

Con esto espero que le saqueis bastante provecho a vuestras placas con linux embebido ya que así conseguis que nada más encenderla se ejecute el programa sin tener que hacer vosotros nada por vuestra cuenta. Quizá en el futuro explique cómo programar módulos para el kernel y extender la funcionalidad de vuestra placa aún más lejos (drivers para hardware, sistemas de ficheros, llamadas al sistema, etc).

Esta entrada fue publicada en Linux, Programación el 5 octubre 2011 [http://www.sistemasorp.es/2011/10/05/crear-un-demonio-en-el-linux-embebido-en-placas/] .

12 pensamientos en “Crear un demonio en el linux embebido en placas”

carlos
26 octubre 2012 en 20:44

buenas amigo muchas gracias por el aporte… pero una pregunta como puedo hacer para que cuando el S.O arranque el se siga ejecutando eh intentado algunas cosas pero no veo como hacerlo… Gracias

Oscar Autor
26 octubre 2012 en 21:25

El truco esta en no salir del bucle while, en el código: `while ( estado == 0 )`

carlos
26 octubre 2012 en 21:53

mmmm… si pero un cosa.. cuando reinicio la pc el demonio nohacenada no deberia volver a ejecutarse… eso es lo que quiero yo le cambie la dirección de /tmp/ a /home/ para que ahi se guarde el txt… pero como hago para que yo apenas prende la pc ese demonio se ejecute..

Oscar Autor
27 octubre 2012 en 1:42

¿Has creado los enlaces simbólicos al script de init.d?

carlos
29 octubre 2012 en 20:13

introduciendo los paquetes a mano no me ejecuta el demonio… lo que hice fue ejecutar el comando update-rc.d nohacenada defaults 76

Pero para que el comando pudiera ejecutarse correctamente coloque en el demonio:

#!/bin/sh
BEGIN INIT INFO
Provides : nohacenada
Required-Start: $syslog
Required-Stop: $syslog
Default-Start: 2 3 4 5
Default-Stop: 0 1 6
Short-Description: Crear un log

Description: activar el documentador de errores
#
END INIT INFO

case “$1” in
start)
/usr/sbin/nohacenada
;;
stop)
test -e /var/run/nohacenada.pid || exit 2
kill `cat /var/run/nohacenada.pid`
;;
*)
echo “El uso es: $0 {start|stop}”
exit 1
;;
esac
exit 0

porque sin los comentarios el comando no se ejecuta…
bueno de nuevo muchas gracias por su ejemplo me a servido de mucha ayuda

---

Oscar
29 octubre 2012 en 20:43

Gracias a ti por compartir tu experiencia con el resto.

Rafa
5 enero 2013 en 0:57

Hola nuevamente, ya funciono pero cree sólo un enlace simbólico asi;

ln -s /etc/init.d/nohacenada /etc/rc.d/S76nohacenada

no sé, sí sea correcto. El contenido de mi directorio /etc es:

mx27# cd /etc
mx27# ls
adjtime hotplug.d passwd
busybox.conf inetd.conf printcap
devfsd.conf init.d profile
dropbear inittab protocols
exports inputrc rc.d
fdprm issue resolv.conf
fstab issue.net securetty
fstab~ ltib-release services
group mime.types shadow
host.conf modprobe.conf shells
hosts modprobe.conf.dist udev
hosts.allow modules.devfs udhcpd.conf
hosts.deny mtab vsftpd.conf
hotplug nsswitch.conf xinetd.d
mx27# cd rc.d
mx27# ls
S76nohacenada rc.conf rc.local rc.serial
init.d rc.conf.orig rc.modules rcS
mx27# pwd
/etc/rc.d

Oscar Autor
5 enero 2013 en 7:43

Puede ser, aunque en tu caso ya no hace falta que el enlace simbólico sea S76nohacenada, con que se llama también noahcenada te sirve.

jorge Diego
19 agosto 2013 en 15:19

Buenos días Oscar,

Muchas Gracias por excelente aporte, muy claro y funciono muy bien.

Saludos

Oscar Autor
19 agosto 2013 en 15:20

Me alegro. Gracias por el comentario.

Mario
24 abril 2014 en 14:06

Hola estoy bastante perdido y no domino mucho del tema,estaría muy agradecido si alguien pudiese ayudarme.

Tengo una beaglebone (con Debian), y lo que quiero es que al alimentarla (sin que este conectada a la red ni a un ordenador ni nada), se habilite los permisos para un fichero.
Algo asi como: chmod 666 /sys/class/leds/beaglebone::usr2/brigtness.

Y no se muy bien que tengo que hacer, ¿si necesito crear el script 'nohacenada' y ya esta o necesito también de un demonio?…

Ayudadme por favor y muchas gracias.

Mario.

Oscar Autor
24 abril 2014 en 16:03

Para ese caso es mejor que lo añadas al fichero /etc/rc.local