Capítulo 6

Servicio de red

En este capítulo se describe el sistema utilizado para habilitar el software de control del helicoptero en el arranque del sistema como servicio de red.

6.1. Sistema de arranque de Debian GNU/Linux

El sistema de arranque de Debian está basado en el modelo System V que está estructurado en distintos niveles de ejecución. Estos niveles de ejución definen distintas configuraciones o modos de operación predeterminados de la máquina. A continuación se hace un listado del uso de los siete niveles de ejecución definidos en Debian:

- Nivel 0: Detiene el sistema.
- Nivel 1: Modo monousuario, utilizado para labores de administración del sistema.
- Niveles 2-5: Modos multiusuario, por defecto un sistema Debian arranca en el modo 2.
- Nivel 6: Reinicia el sistema.

Un sistema Debian al arrancar ejecuta el proceso *init* que se encarga de gestionar el resto de procesos que se cargan el sistema. El proceso *init* lee de su archivo de configuración */etc/inittab* qué procesos se ejecutan. En el arranque del sistema se ejecuta el archivo */etc/init.d/rcS* que se encarga de montar el sistema de archivos, cargar el núcleo e inicializar algunos servicios básicos como el sistema de red. Una vez terminado estos primeros pasos, el proceso *init* ejecuta los procesos asociados al nivel de ejecución al que va a pasar (en el caso de inicio del sistema por defecto los del nivel 2).

Cada nivel de ejecución tiene asociado un directorio en el que hay una estructura de archivos que definen los procesos que se inician o finalizan al entrar en ese nivel de ejecución. Estos directorios son del tipo */etc/rcN.d* donde *N* es el número del nivel de ejecución. El nombre de los archivos tienen la siguiente nomenclatura:

- La primera letra es una S para indicar que al entrar en el nivel de ejecución se le de al archivo la orden *start* o una K para que se le de la orden *stop*.
- Los siguientes dos caracteres se corresponden con un número que representa el orden en el que se ejecutan esas ordenes.
- Por último se añade un nombre descriptivo del servicio como *httpd* para el servidor de páginas web.

Así por ejemplo el archivo */etc/rc2.d/S89cron* indica que al entrar en el nivel de ejecución 2 se debe ejecutar ese archivo con la orden *start* como parámetro con lo que se inicializa el demonio cron. Una característica habitual en este tipo de sistemas es que ya que los niveles de ejecución comparten muchos de estos archivos, en vez de repetir los mismos por el sistema de directorios lo que se hace es guardar los mismos en el directorio */etc/init.d* y hacer enlaces simbólicos con el nombre adecuado en los directorios */etc/rcN.d*. Es decir usando el anterior ejemplo del demonio cron, existe el archivo */etc/init.d/cron* que sabe interpretar todas las órdenes como *start*, *stop* o *restart* y en los directorios */etc/rcN.d* hay enlaces simbólicos a este archivo con el nombre adecuado que lo activan (Ej. */etc/rc2.d/S89cron*) o desactivan (Ej. */etc/rc0.d/K11cron*) según el estado de ejecución.

6.2. Crear un servicio de red

Una vez explicado el sistema de arranque Debian vamos a explorar las dos opciones para crear un servicio de red que desde que se inicie la máquina esté disponible para el control del helicoptero:

- Para crear un servicio de red una alternativa es usar el superdemonio **inetd**. Este demonio correctamente configurado escucha en los puertos que se le indique y una vez que recibe una comunicación por uno de los puertos a los que está atendiendo, consulta su archivo de configuración */etc/inetd.conf* y ejecuta el proceso que se indica en el mismo conectando la entrada y salida estándar de ese proceso con el socket del puerto de internet. De esta forma los procesos que gestionan el servicio de red no tienen que preocuparse de las conexiones a los puertos del ordenador y gestionar las mismas. Una de las ventajas de inetd es que servicios poco utilizados no tienen que consumir recursos del sistema hasta el momento de necesitarlos ya que es inetd el que se encarga de iniciar bajo petición el ejecutable que atiende al servicio.

- La otra alternativa para crear un servicio de red es gestionar en el propio proceso la conexión a los puertos del sistema y escuchar las peticiones que van llegando. Para ello se utilizan en el código llamadas al sistema que nos permiten asociar sockets a los puertos y así dar un determinado servicio. Para configurar que un servicio de este tipo se ejecute al iniciar el sistema hay que utilizar un script que lo gestione como hemos explicado en la sección 6.1.

6.3. Implementación

En las pruebas de vuelo la placa Hércules no tendrá periférico alguno conectado, por ello se plantea la necesidad de que dicho programa principal sea capaz de arrancar de forma automática al iniciar el sistema. Dado que dicho sistema está diseñado exclusivamente para la ejecución de la aplicación de monitorización y gestión de sensores, que es el programa principal en el computador de vuelo, se diseña como un servicio de red que se inicia con el sistema y se asocia a una serie de puertos de forma exclusiva mientras esté en ejecución.

Por otra parte, dada la criticidad del sistema, y para integrarlo dentro del control de tierra, se ha diseñado una aplicación de monitorización accesoria que permite consultar el estado de ejecución del programa de la Hércules, así como activarlo o desactivarlo bajo petición del usuario.

6.3.1. Inicio con el sistema del programa principal.

Para introducir el programa en el proceso de arranque, Debian GNU/Linux proporciona un script tipo que se encuentra en */etc/init.d/skeleton* el cual copiamos con el nombre */etc/init.d/hercules* y modificamos de la siguiente forma:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct
    scripts to be
#                    placed in /etc/init.d.
### END INIT INFO

# Author: Patricia Rodríguez Díaz-Peñalver

# PATH should only include /usr/* if it runs after the mountnfs.
    sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
#Nombre y descripción del servicio.
DESC="Servicio de captura de datos de sensores en la placa
    Hércules"
NAME=hercules
#Ruta del programa a ejecutar.
DAEMON=/home/padur/Patricia/helicoptero/main
DAEMON_ARGS="--options args"
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.0-6) to ensure that this file is
    present.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
    # Return
    #   0 if daemon has been started
    #   1 if daemon was already running
    #   2 if daemon could not be started
```

```
    start-stop-daemon --start --quiet --pidfile $PIDFILE --exec
       $DAEMON --test > /dev/null \
       || return 1
#Añadimos el parámetro -b para que el programa sea ejecutado en
    background.
    start-stop-daemon --start -b --quiet --pidfile $PIDFILE --
       exec $DAEMON -- \
       $DAEMON_ARGS \
       || return 2
    # Add code here, if necessary, that waits for the process to
       be ready
    # to handle requests from services started subsequently which
        depend
    # on this one.  As a last resort, sleep for some time.
}

# Function that stops the daemon/service

do_stop()
{
    # Return
    #   0 if daemon has been stopped
    #   1 if daemon was already stopped
    #   2 if daemon could not be stopped
    #   other if a failure occurred
    start-stop-daemon --stop --quiet --retry=TERM/30/KILL/5 --
       pidfile $PIDFILE --name $NAME
    RETVAL="$?"
    [ "$RETVAL" = 2 ] && return 2
    # Wait for children to finish too if this is a daemon that
       forks
    # and if the daemon is only ever run from this initscript.
    # If the above conditions are not satisfied then add some
       other code
    # that waits for the process to drop all resources that could
        be
    # needed by services started subsequently.  A last resort is
       to
    # sleep for some time.
    start-stop-daemon --stop --quiet --oknodo --retry=0/30/KILL/5
        --exec $DAEMON
    [ "$?" = 2 ] && return 2
    # Many daemons don't␣delete␣their␣pidfiles␣when␣they␣exit.
␣␣␣␣rm␣-f␣$PIDFILE
␣␣␣␣return␣"$RETVAL"
}

#
#␣Function␣that␣sends␣a␣SIGHUP␣to␣the␣daemon/service
␣␣␣␣␣␣␣␣
```

```
do_reload() {
    #
    # If the daemon can reload its configuration without
    # restarting (for example, when it is sent a SIGHUP),
    # then implement that here.
    #
    start-stop-daemon --stop --signal 1 --quiet --pidfile
        $PIDFILE --name $NAME
    return 0
}

case "$1" in
  start)
    [ "$VERBOSE" != no ] && log_daemon_msg "Starting␣$DESC" "
        $NAME"
    do_start
    case "$?" in
        0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
        2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
    esac
    ;;
  stop)
    [ "$VERBOSE" != no ] && log_daemon_msg "Stopping␣$DESC" "
        $NAME"
    do_stop
    case "$?" in
        0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
        2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
    esac
    ;;
  #reload|force-reload)
    #
    # If do_reload() is not implemented then leave this commented
        out
    # and leave 'force-reload' as an alias for 'restart'.
    #
    #log_daemon_msg "Reloading␣$DESC" "$NAME"
    #do_reload
    #log_end_msg $?
    #;;
  restart|force-reload)
    #
    # If the "reload" option is implemented then remove the
    # 'force-reload' alias
    #
    log_daemon_msg "Restarting␣$DESC" "$NAME"
    do_stop
    case "$?" in
      0|1)
```

```
        do_start
        case "$?" in
            0) log_end_msg 0 ;;
            1) log_end_msg 1 ; ; # Old process is still running
            *) log_end_msg 1 ; ; # Failed to start
        esac
        ;;
       *)
        # Failed to stop
        log_end_msg 1
        ;;
    esac
    ;;
  *)
    #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-
        reload}" >&2
    echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}"
        >&2
    exit 3
    ;;
esac
```

Código fuente 6.1: Código del archivo */etc/init.d/hercules*.

Las modificaciones sobre el archivo */etc/init.d/skeleton* se limitan al nombre y descripción del servicio (*hercules*), la ruta del programa que será ejecutado (*main*) y un parámetro para la ejecución en background del programa.

Como último paso, este archivo tiene que ser enlazado en los directorios */etc/rc?.d* con los nombres adecuados para se inicie y se pare según el nivel de ejecución. Se crean así enlaces simbólicos ejecutando las siguientes órdenes:

```
    ln -s /etc/init.d/hercules /etc/rc0.d/K01hercules
    ln -s /etc/init.d/hercules /etc/rc1.d/K01hercules
    ln -s /etc/init.d/hercules /etc/rc6.d/K01hercules
    ln -s /etc/init.d/hercules /etc/rc2.d/S99hercules
    ln -s /etc/init.d/hercules /etc/rc3.d/S99hercules
    ln -s /etc/init.d/hercules /etc/rc4.d/S99hercules
    ln -s /etc/init.d/hercules /etc/rc5.d/S99hercules
```

Código fuente 6.2: Órdenes para el enlazado del archivo 'hercules'.

6.3.2. Sistema de monitorización del programa.

Se crea un servicio de red mediante el demonio *inetd* que permite monitorizar, arrancar y detener el programa cuando recibe ciertos mensajes a través del puerto 25557. Para ello se siguen los siguientes pasos:

1. Modificar */etc/services* añadiendo el nuevo servicio para el puerto 25557, al que llamamos *hercules*.

```
# Network services, Internet style
...
tcpmux          1/tcp                   # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                  # message send protocol
msp             18/udp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
...

# Local services
hercules    25557/tcp
```

Código fuente 6.3: Definición del nuevo servicio *hercules* en /etc/services.

2. Se modifica */etc/inetd.conf* para activación del servicio en el superdemonio *inetd*.

```
# Internet server configuration database
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path
    > <args>
#
...
hercules        stream  tcp     nowait  root    /opt/hercules/
    supervisor.sh           supervisor.sh
...
```

Código fuente 6.4: Inclusión del servicio *hercules* en /etc/inetd.conf.

3. Crear el fichero */opt/hercules/supervisor.sh* con el siguiente código que permite gestionar las acciones a realizar según los mensajes que le lleguen desde tierra:

```
#!/bin/bash

read ENTRADA
/bin/sleep 50&

if [[ $ENTRADA = '0' ]]; then
   pgrep a.out >>/dev/null
   if [[ $? = 0 ]]; then
      /bin/echo "AS"
   else
      /bin/echo "AN"
   fi
   exit;
fi

if [[ $ENTRADA = '1' ]]; then
   pgrep a.out >>/dev/null
        if [[ $? = 0 ]]; then
                /bin/echo "EA"
        else
      nohup /home/paty/ProgramaPablo/a.out & >>/dev/null
                /bin/echo "AS"
        fi

   exit;
fi

if [[ $ENTRADA = '2' ]]; then
   PID=`pgrep a.out` >>/dev/null
        if [[ $? = 0 ]]; then
      kill -9 $PID
                /bin/echo "AN"
        else
                /bin/echo "EP"
        fi
        exit;
fi


/bin/echo "EO"
exit;
```

Código fuente 6.5: Código de la función de supervisión 'supervisor.sh' creada para el servicio *hercules*.

La función */opt/hercules/supervisor.sh* espera tres cadenas de caracteres posibles: '0', '1' o '2'. Con la primera opción indicará a tierra si el programa principal del computador de vuelo está activo o no. Esto puede llegar a ser muy útil si los canales de comunicación habilitados por la aplicación (puertos udp 4950-4954) fallan o por algún error del sistema no podemos saber si el programa en la Hércules

está realmente activo.

Las otras dos opciones serán para activar y desactivar *main* en el caso de que el usuario observase conveniente hacerlo utilizando este servicio.

Se ha diseñado una pequeña interfaz gráfica en Matlab para comunicación con el servicio de red definido.

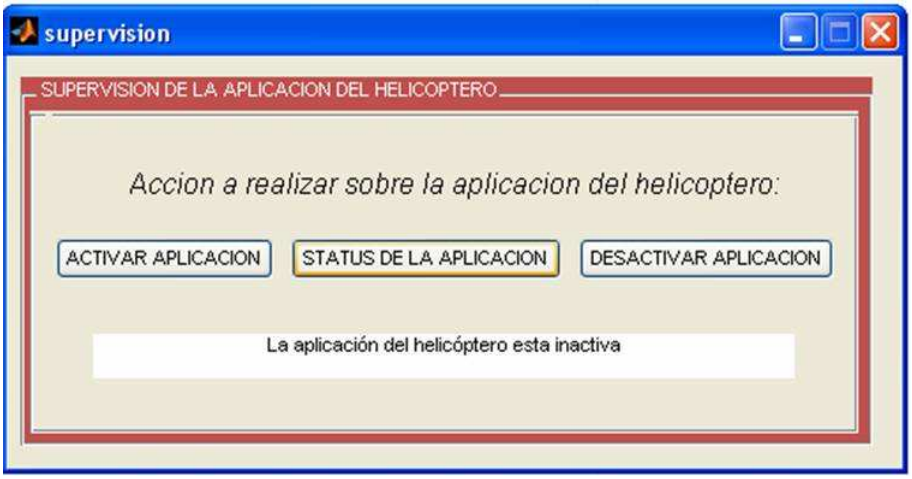

Figura 6.1: Ventana 'supervision.fig' para gestión del servicio de red.

Los callbacks pertenecientes a cada botón son sencillos envíos de las distintas cadenas de caracteres por el puerto udp 25557 y recepción de la correspondiente respuesta. Vemos como ejemplo el código perteneciente al botón 'ACTIVAR APLICACIÓN':

```
function pushbutton1_Callback(hObject, eventdata, handles)
 % Socket por el puerto del servicio 'hercules'.
p=tcpip('192.168.32.1',25557);
fopen(p);
 % Se envía solicitud de activación del programa 'main'.
fprintf(p,'1');
nrecepcion=50;
[datos,count]=fscanf(p,' %s',nrecepcion)
if count>0
 % El programa se ha activado con éxito.
        if datos=='AS'
            set(handles.text4,'string','La aplicación del helicóptero se ha activado');
 % El programa ya estaba activo.
        elseif datos=='EA'
            set(handles.text4,'string','Error: la aplicación del helicóptero ya estaba activada');
        end
else
 % Si no se recibe nada hay un error.
        set(handles.text4,'string','Error de recepcion');
end
fclose(p);
```

```
delete(p);
clear p;
```

Código fuente 6.6: Código Matlab para envío de mensajes al servicio de red 'hercules'.

Todo lo referente a servicios de red para arranque y supervisión de estado del sistema se ha probado con éxito fuera del laboratorio, es decir, no se han realizado las pruebas en la placa Hércules. Sin embargo, habiendo sido exitosos todos los ensayos en un PC servidor con sistema operativo Debian no se encuentran motivos para pensar que surgirán problemas especiales al crear servicios de red similares en el computador de vuelo.