CAPÍTULO 3. INSTALACIÓN DE *QNX* EN EL *PC-104*.

3.1. Introducción.

Como se ha dicho anteriormente, se pretende portar el *software* del *PC-104* de Linux a *QNX*. Resulta evidente que la mencionada máquina debe tener instalado el sistema operativo *QNX*. Esta instalación ha de incluir *drivers* para la red, el disco duro y el puerto serie. Además debe tener demonios para *telnet* y *ftp*, imprescindibles para la comunicación con el *PC* de tierra, y un editor de textos y un compilador *C* - *C++* para poder variar los programas en el campo de pruebas en caso de ser necesario.

3.2. Instalación desde CD.

La manera más sencilla de instalar *QNX* es mediante el *CD* de instalación. Basta insertarlo en el dispositivo de *CD-ROM* del ordenador y seguir las instrucciones que el asistente va dando. Se puede instalar en una partición de Linux o Windows sin que esto afecte a los citados *SO*, se puede arrancar directamente desde *CD*, o bien instalar *QNX* en una partición vacía.

Esta última opción no está bien desarrollada, y genera dos problemas. El primero es que el programa no hace diferencia entre espacio sin particionar y particiones vacías. Por ejemplo, dará lo mismo tener 256 *megas* sin formato que 128 *megas* formateados en *ext2* y 128 en *FAT32*, ya que, si en estas particiones no hay nada guardado, *QNX* verá de todas maneras 256 *megas* vacíos. Una forma de evitar este problema es guardar cualquier archivo en cada una de las particiones vacías. El segundo problema es que las nuevas particiones que se pueden generar han de ocupar o toda la memoria libre o una fracción potencia de dos de la misma. Es decir, si hay *X megas* libres, la partición para *QNX* podrá ocupar *X*, *X/2*, *X/4*, *X/8*, etc. del espacio libre.

Aparte de los problemas con el particionado, en otras instalaciones hechas anteriormente se observó otro: la falta de modularidad. No se pueden seleccionar los paquetes a instalar. Se instalan todos. Al principio se instalan los paquetes básicos, y al arrancar por primera vez el sistema, él mismo preguntará si se quieren instalar el resto de programas (compilador, entorno de desarrollo...). Ésos son los dos únicos módulos. Después está el *3rd Party CD*, en el que vienen aplicaciones libres desarrolladas por usuarios, y que no está soportado por la *QSSL*. Aparte de estos paquetes, que sí están separados, el resto habría que instalarlos todos, con el gasto inútil de recursos que ello conlleva.

Aún así, y debido a la simplicidad del proceso de instalación, se optó por intentar esta opción. Se trató de hacer la instalación completa, pero no fue posible debido a la escasa memoria disponible en la *compact flash* (*CF* en adelante), por lo que hubo que conformarse con la básica. Ésta incluye no sólo los comandos y librerías esenciales, sino otros instrumentos superfluos como el entorno gráfico, un navegador para internet, programas de configuración para el entorno gráfico, instalador de paquetes, etc. A pesar de todo se comprobó que esta instalación era posible hacerla en una partición de 60 *megas*, por lo que quedaba libre la mayor parte de *CF*.

Sin embargo la instalación básica desde *CD* dejaba mucho que desear. Para empezar no incluye ningún compilador, si bien el problema mayor es que no arrancaba bien. Para arrancar el sistema era necesario recurrir al arranque alternativo, que deshabilita el *DMA*. Se hacía necesario recurrir a otro tipo de instalación.

3.3. Instalación mediante imagen del sistema.

Quizás resulte extraño que un sistema operativo orientado a sistemas empotrados y de tiempo real sea tan poco modulable en su instalación. Esto es debido a que la instalación desde *CD* está pensada únicamente para ordenadores de sobremesa. Para los sistemas empotrados es necesario recurrir a otro tipo de instalación. En ella se hace una imagen del sistema que contiene las librerías, ejecutables y ficheros necesarios. Para ello es necesario tener una instalación completa en un *PC* de sobremesa, con el que se pasará la imagen al *PC* empotrado. La idea es tener un sistema operativo mínimo dentro del *PC* empotrado, de manera que se ejecute la aplicación requerida automáticamente consumiendo los mínimos recursos posibles.

3.3.1. Disco de inicio.

Para hacer un disco de inicio se utiliza el comando *dinit*. Éste será el primer paso a dar para la instalación de *QNX*. Además de en disquetes, este comando puede utilizarse en discos duros, mediante la opción *-h*. Se crearán en el dispositivo cuatro archivos: *.boot*, *.altboot*, *.bitmap* y *.inodes*. De éstos los dos primeros son imágenes del sistema. En particular *.boot* es la imagen que se carga por defecto, y *.altboot* es la del arranque alternativo, que se ejecuta si el usuario pulsa la tecla Escape cuando se le pregunta (*Press Esc for .altboot.....*). Para el *PC* de control las imágenes *.boot* y *.altboot* serán idénticas, salvo en la configuración de la red. Así, la primera tendrá la configuración para la red del helicóptero, y la segunda para la del laboratorio.

```
# dinit /dev/fd0
Using loader /usr/qnx630/target/qnx6/x86/boot/sys/ipl-diskpc2-flop
Disk '/dev/fd0' contains 2880 blocks (1440K).
# mount -t qnx4 /dev/fd0 /floppy
# ls /floppy
.            .altboot     .boot
..           .bitmap      .inodes
# _
```

Figura 3-1. Creación de un disco de inicio.

3.3.2. Imagen del sistema.

Aunque una vez utilizado *dinit* el ordenador arranca, no lo hace de la manera en que quiere el usuario. Tiene un sistema operativo con muy pocas posibilidades, sin apenas librerías, *drivers* o aplicaciones, y por supuesto sin los programas que haya diseñado el propio usuario. Es evidente que esa imagen hay que cambiarla, y aquí llega el problema más complicado de todos, pues hacer una imagen personalizada no es nada fácil.

El sistema expande los archivos de la imagen en la memoria *RAM*, y ejecuta aquéllos que ha de ejecutar, y en la manera en que ha de hacerlo. Evidentemente el ordenador no sabe por sí mismo todas estas cosas, y el usuario tendrá que crear la imagen de manera que el *PC* se arranque como él quiere.

3.3.3. *Buildfile*.

Un *buildfile* es un fichero de texto que describe la imagen que se quiere instalar en el ordenador. Este fichero se compone de tres partes: *bootstrap script*, *startup script* y listado de ficheros.

3.3.3.1. *Bootstrap script*.

Este *script* le dice a la *CPU* qué programa de inicio (*startup program*) debe utilizar. En el *bootstrap script* se indica el tipo de máquina que se va a utilizar y la manera de arrancar en sistema operativo que se desea. Evidentemente cada tipo de *CPU* usará un programa diferente, por lo que el *bootstrap script* dependerá directamente de la mencionada unidad.

Para el *PC-104* el *bootstrap script* es:

```
[virtual=x86,bios +compress] .bootstrap={
startup-bios -s 64k -A
PATH=/proc/boot:/bin:/usr:/usr/bin:/usr/lib/gcc-lib/ntox86/2.95.3QNX-nto
LD_LIBRARY_PATH=/proc/boot:/lib:/usr/lib:/usr/lib/gcc-lib/ntox86/2.95.3QNX-nto
procnto-smp
}
```

x86:

Tipo de procesador.

bios:

Arranque por *bios*.

compress:

Se comprime el archivo imagen. Esto es especialmente útil, ya que el tamaño de este archivo está limitado (aproximadamente a tres *megas*).

.bootstrap:

Indica que va a comenzar el *bootsrap script*.

startup-bios:

Programa que se encarga de descomprimir la imagen y situarla adecuadamente en la *RAM*, hacer determinadas configuraciones, y finalmente ejecutar el kernel.

-*s 64k*: copia 64k de la ROM de la tarjeta de vídeo en la *RAM*, que es más rápida.
-*A*: hace que se reinicie el sistema en caso de terminación anormal del kernel.

PATH:

Ruta por defecto para la búsqueda de programas.

LD_LIBRARY_PATH:

Idem librerías. El directorio */proc/boot* es el predefinido por el sistema para guardar los archivos que se incluyen en la imagen. Llama la atención la ruta */usr/lib/gcc-lib/ntox86/2.95.3QNX-nto*. El porqué de la inclusión de este directorio se detallará más adelante, cuando se explique la instalación del compilador *gcc*.

procnto-smp:

Contiene el microkernel de *QNX* 6 y las gestiones de procesos, memoria, y nombres de rutas.

3.3.3.2. *Startup script*.

Llegados a este punto el proceso de *bootstrap* ha finalizado, y el ordenador ya ha arrancado. Sin embargo el sistema todavía no está completamente operativo. Como se dijo en el primer capítulo (apartado 1.5.3), el microkernel no se encarga de todas las

operaciones del sistema operativo. Por lo tanto será necesario lanzar los procesos que realicen las mencionadas tareas. Además el usuario puede necesitar la ejecución de otros programas, como *drivers*, aplicaciones comunes o incluso aplicaciones desarrolladas por él mismo. Es evidente que el sistema por sí solo no puede saber qué programas ejecutar y las opciones para cada uno de ellos. Se hace necesario un nuevo *script* que le indique al sistema tal información. Esta lista de instrucciones recibe el nombre de *startup script*.

A continuación se expone y se explica el *startup script* del *PC-104*.

```
[+script] .script={
display_msg "  "
display_msg "Arranque, Dedalo"
display_msg "  "
display_msg "  "

display_msg "Arrancando..."
seedres
pipe &
mqueue &
devc-pty &
pci-bios &
display_msg "Esperando /dev/pci"
waitfor /dev/pci

display_msg "Configurando puerto serie"
devc-ser8250 -b 115200 3f8,4 -b 19200 2f8,3 &

display_msg "Puerto serie configurado:"
display_msg "COM1 115200 baudios (ser1); COM2 19200 baudios(ser2)"


display_msg "Iniciando io-net"
io-net -d speedo -p tcpip -v
display_msg "Esperando /dev/socket"
waitfor /dev/socket
waitfor /dev/io-net/ip0

display_msg "Configurando red"
#ifconfig en0 192.168.100.253 netmask 0xfffffc00 up
#route add -net default 192.168.100.1
netmanager &
display_msg "Demonio"
inetd &

random -t -p &
display_msg "Consolas"
devc-con -n2 &

display_msg "Detectando dispositivos EIDE"
devb-eide blk automount=hd0t79:/ cam quiet eide &

#devb-eide blk automount=hd0t79:/,automount=hd0t131:/linux cam quiet eide &

display_msg "Esperando /dev/hd0"
waitfor /dev/hd0

display_msg "Revisando el sistema de archivos"
chkfsys -qPr /

display_msg "Iniciando sesiones"
```

```
reopen /dev/con1
[+session] HOME=/ TERM=qansi-m esh &
reopen /dev/con2
[+session] HOME=/ TERM=qansi-m esh &

}
```

.script:

Indicador de comienzo del *startup script*.

Términos que se repiten:

display_msg: Orden para escribir mensajes por pantalla.

#xxxxxx: Comentarios.

&: Hace que el programa se ejecute en segundo plano.

Programas de uso general:

seedres: Reparte los recursos del sistema. Necesario para evitar conflictos en las asignaciones de recursos de los *drivers*.

pipe &: Administrador de *pipes*. Necesario para el uso de las mismas. Las *pipes* se utilizan principalmente para conectar la salida de un proceso con la entrada de otro.

mqueue &: Administrador de colas de mensajes (*POSIX 1003.1b*) y semáforos con nombre.

devc-pty &: Administrador de comunicaciones *Pty*.

pci-bios &: Este servidor proporciona el soporte a la *PCI-BIOS*. Necesario para cualquier sistema con una *PCI-BIOS*.

waitfor /dev/pci:

Se espera a la creación de */dev/pci* antes de continuar. *waitfor* se utiliza cuando un programa necesita de la finalización de otro para su ejecución. Típico para *drivers*. En esta ocasión antes de configurar la red será necesario que actúe el *driver* de la misma.

Puerto serie:

devc-ser8250: Controlador del puerto serie.

***-b 115200 3f8,4 -b 19200 2f8,3*:** Configura /dev/ser1 a 115200 baudios y /dev/ser2 a 19200. No obstante cada aplicación podrá modificar la velocidad de transferencia según sus necesidades.

Red:

io-net: Gestiona la red.

-d speedo**:** Ordena a io-net que utilice el controlador speedo.
-p tcpip**:** Protocolo tcp/ip.
-v**:** Mostrar información por pantalla.
waitfor /dev/socket
waitfor /dev/io-net/ip0

Se espera a la creación de */dev/socket* y */dev/io-net/ip0* antes de continuar.

Existen varias maneras de configurar la red. En este caso se utilizan dos, dependiendo de si el *PC* está conectado al *software* del helicóptero o a la red del laboratorio. Evidentemente habrá que cambiar estas líneas del *buildfile* cada vez que se quiera construir una u otra imagen. Bastará con añadir y quitar almohadillas (#) en las líneas que corresponda.

Red del helicóptero:

ifconfig: Proporciona los parámetros del interfaz de la red.

en0**:** Dispositivo *ethernet* 0.
192.168.100.253: IP.
netmask 0xffffffc00: Máscara de red.
up: Habilitado.

route: Manipula manualmente las tablas de ruta de la red.

add: Añade una ruta.

-net**:** Fuerza a interpretar lo que siga como *'network'* y no como *'host'*.
default 192.168.100.1**:** Gateway.

Red del laboratorio:

La red del laboratorio podría configurarse de la misma manera. Sin embargo en ella se utilizan servidores de nombres, y se hace más cómoda esta otra manera de introducir los parámetros.

netmanager:

Este programa configura la red según los datos que haya en el fichero */etc/net.cfg*. Consecuentemente habrá que incluir en la imagen este archivo, modificándolo de manera que tenga los parámetros adecuados para el *PC-104*.

/etc/net.cfg:

```
# nto network config file v1.2
version v1.2

[global]
hostname dedalo
domain us.es
nameserver 150.214.186.69
nameserver 150.214.130.15
nameserver 150.214.130.10
route 172.16.0.13 0.0.0.0 0.0.0.0
lookup file bind
[en0]
type ethernet
mode manual
manual_ip 172.16.1.253
manual_netmask 255.255.252.0
```

Como se observa, el archivo incluye todos los parámetros necesarios para configurar la red.

Demonios:

inetd: Arranca los demonios instalados. En este caso el servidor *ftp* (*ftpd*) y el servidor *telnet* (*telnetd*).

Random:

random: Proporciona fuentes seguras de datos aleatorios.

***-t*:** Usa el *'high-performance clock'* como fuente de datos aleatorios.
***-p*:** Revisa la información del sistema de */proc* para los datos aleatorios.

Interfaz E/S:

devc-con: *Driver* para las consolas, el teclado y la pantalla.

***-n2*:** Dos consolas.

Disco duro:

devb-eide: *Driver* para dispositivos *IDE*.

***blk automount=hd0t79:/*:** Opciones *blk*. La partición *hd0t79* (partición de *QNX*) se monta en el directorio raíz. Se podría añadir *'automount=hd0t131:/linux'*, de manera que la partición de Linux se montase automáticamente en el directorio */linux*. Sin embargo esto podría ocasionar problemas, ya que un apagado erróneo del *PC* podría producir daños en el sistema de archivos de Linux. Además al arrancar Linux revisaría el mencionado sistema, operación que se hace bastante tediosa. Por otra parte no hay necesidad de acceder a otra partición que no sea la de *QNX*, y en caso de necesitarlo, siempre se podrá montar de manera manual.

***cam quiet*:** Opciones *cam*. No mostrar mensajes por pantalla.
***eide*:** Opciones *eide*. En este caso no han sido necesarias.
waitfor /dev/hd0: Se espera a que termine *devb-eide*.
chkfsys: Revisión del sistema de archivos. Al no tener demasiados archivos, esta revisión no lleva mucho tiempo. Antes de hacerla ha de estar montado el disco duro, y de ahí la espera a */dev/hd0*.
***-q*:** No sacar comentarios por pantalla.
***-P*:** Preguntar antes de empezar.
***-r*:** Reconstruir el *'bitmap'* sin sacar mensajes por pantalla.

Sesiones:

reopen /dev/con1
[+session] HOME=/ TERM=qansi-m esh &
reopen /dev/con2
[+session] HOME=/ TERM=qansi-m esh &

De esta manera se abren las dos sesiones iniciando en el directorio raíz, con el terminar *qansi-m* (propio de *QNX*), y *'esh'* como *shell*.

3.3.3.3. Lista de archivos.

Como se dijo antes, en la lista de archivos se escriben las librerías y los programas que se van a necesitar. Como se ha montado el disco duro, tan sólo será necesario incluir en la imagen aquellos archivos que se usen en el *script*, más alguno necesario en las configuraciones. Los ficheros irán comprimidos en la imagen, y se expandirán en la *RAM*. El sistema los coloca por defecto en el directorio */proc/boot*. Algunos archivos necesitan de una ubicación determinada dentro del árbol de directorios del sistema del *PC* empotrado. En ese caso bastará con escribir: */ruta/del/archivo/target = /ruta/del/archivo/host*.

Por ejemplo, el programa *netmanager* buscará el archivo *net.cfg* en el directorio */etc*. *net.cfg*, como se vio en el apartado 3.3.3.2, es un fichero de texto con la configuración de la red. Partiendo del *net.cfg* del *PC* de mesa se creó uno nuevo para el *PC* empotrado, guardándose en la carpeta */root* (todavía del *PC* de mesa). Por lo tanto se pretende que el fichero */root/net.cfg* del *PC* de mesa cambie su ruta en el sistema empotrado a: */etc/net.cfg*. Para ello se escribirá en el *buildfile*: */etc/net.cfg=/root/net.cfg*.

Además de cambios de rutas se pueden hacer enlaces, gestionar los permisos, etc.

```
#Librerías:

libc.so
libsocket.so
libc.so.2
libsocket.so.2
libcam.so.2
cam-disk.so
```

```
io-blk.so
libm.so
libm.so.2
libz.so
libz.so.2
/lib/dll/npm-tcpip.so
devn-speedo.so
fs-QNX4.so
fs-ext2.so

#Archivos de configuración:

/etc/hosts=/etc/hosts
/etc/termcap=/etc/termcap
/etc/inetd.conf=/etc/inetd.conf
/etc/ftpusers=/etc/ftpusers
/etc/net.cfg=/root/net.cfg

#Enlace para librería libc.so:

[type=link] /usr/lib/ldQNX.so.2=/proc/boot/libc.so


#Programas:

[code=uip data=copy perms=+r,+x]
seedres
random
pipe
mqueue
esh
chkfsys
devc-con
devb-eide
devc-ser8250
devc-pty
io-net
ifconfig
/sbin/route
pci-bios
inetd
netmanager
/usr/sbin/ftpd=/usr/sbin/ftpd
/usr/sbin/telnetd=/usr/sbin/telnetd
/bin/login=/bin/login
/bin/logout=/bin/logout
/bin/sh=/bin/sh
/bin/su=/bin/su
shutdown
```

Librerías:

Las librerías que se incluyen en la imagen son las que necesitan los programas que hay en la misma. No es fácil saber qué librerías utiliza cada programa. Para saber las librerías de uso general que necesita cada programa se usan en tubería las aplicaciones *objdump* y *grep*. Sin embargo hay veces en que los programas utilizan más librerías, como los *drivers*, que además dependen del hardware.

En la figura 3-2 se muestra un ejemplo de uso de *objdump* y *grep*. Con la invocación mostrada se intenta hallar las librerías que necesita *netmanager*. Éstas resultan ser *libsocket.so.2* y *libc.so.2*.

```
# objdump -x /bin/netmanager | grep "NEEDED"
objdump: /bin/netmanager: no symbols
  NEEDED      libsocket.so.2
  NEEDED      libc.so.2
# _
```

Figura 3-2. Uso de objdump y grep.

Archivos de configuración:

/etc/hosts: Contiene información sobre los *hosts* conocidos de la red.

/etc/termcap: Información sobre el terminal.

/etc/inetd.conf: Contiene información para que *inetd* arranque los demonios, en este caso los servidores *telnet* y *ftp*.

/etc/ftpusers: Lista de usuarios con acceso negado a *ftp*.

/root/net.cfg: Configuración de la red.

Programas:

La mayor parte de los programas ya se han explicado anteriormente al ejecutarse gracias al *startup script*. El resto de programas incluidos se justifican a continuación:

/bin/login: Programa para la identificación del usuario. Necesario para las conexiones de red.

/bin/logout: No es estrictamente necesario, pero es lógico que ya que puede entrar, el usuario pueda salir.

/bin/sh: Otro *shell* aparte de *esh*. *sh* es el *shell* que suele utilizarse. No obstante ha dado algún problema, por lo que se ha decido cambiar por *esh*, que va mucho mejor. Sin embargo cuando se abren terminales remotos, éstos piden *sh*, por lo que ha sido necesario introducirlo en la imagen, siendo el *shell* predefinido en *telnet*.

/bin/su: Al ejecutar este programa se pedirá la contraseña del súper usuario. Necesario, ya que en *telnet* está prohibido entrar como tal, y puede ser necesario tocar algún archivo con permisos restringidos.

shutdown: Programa para reiniciar o apagar el equipo con seguridad.

3.4. Creación e instalación de la imagen.

Una vez se tiene el *buildfile* sólo hace falta utilizar el programa *mkifs* para construir la imagen. Este proceso se hace en el *PC* de sobremesa. Construidas las imágenes éstas se pasan al *PC-104* vía *ftp*, habiendo arrancado el ordenador con el CD de instalación. El último paso es cambiar los archivos *.boot* y *.altboot* por las nuevas imágenes. La siguiente vez que se arranque *QNX* ya lo hará de la manera que se ha diseñado.

3.5. *mkifs* vs *IDE*.

Existe otra forma de construir la imagen del sistema, que es haciendo uso del entorno de desarrollo integrado (*IDE*) que trae la edición profesional que *QNX*. En teoría esta segunda forma de construir la imagen es más intuitiva y cómoda. Sin embargo en la realidad no resulta así.

3.5.1. Inconvenientes del *IDE*.

- La parte más complicada del *buildfile* es la composición de los *script*s, y éstos han de escribirse a mano igualmente utilizando el *IDE*.

- A la hora de seleccionar los programas se hace todo más tedioso, ya que hay que ir navegando en un explorador entre las carpetas y añadir uno por uno los archivos que se quieran añadir. Resulta mucho más cómodo irlos añadiendo en el *buildfile* escribiendo sus nombres.

- El *IDE* genera un *buildfile* mucho más largo y menos intuitivo que el hecho a mano. Además en la cabecera del mismo avisa de que no debe ser modificado, por lo que todos los cam*bios* han de hacerse con el *IDE*.

3.5.2. Ventajas frente a *mkifs*.

- Se puede ver en todo momento el árbol de directorios de la imagen.

- Tiene una opción para añadir librerías automáticamente, de manera que se avisa al usuario de si hay librerías que faltan o sobran. Esta función está algo limitada ya que, como se dijo antes, hay librerías que dependen del hardware, y ésas hay que incluirlas manualmente en la imagen.

- El entorno de desarrollo posee una opción llamada *The Dietitian* (el dietista), que selecciona de cada librería únicamente los módulos que la imagen va a necesitar, instalando sólo los que se requieran. Hay que tener cuidado con esta función, ya que es posible que en el disco duro se instalen posteriormente más

programas, y éstos podrían necesitar los módulos excluidos. Como en este caso hay muchas aplicaciones aparte de las que lleva la imagen, y además hay memoria de sobra, se ha desestimado el uso del dietista.

- El entorno de desarrollo no sólo reconoce los *buildfiles* que él mismo ha generado. También es capaz de interpretar los *buildfiles* escritos manualmente por un usuario. De esta manera el usuario puede aprovecharse de las ventajas del *IDE* y revisar de una manera más cómoda el sistema de archivos de la imagen que está diseñando.

3.5.3. Instalación de programas en el disco duro.

Como se dijo al explicar el *startup script* del *buildfile*, el *PC* al arrancar ejecuta el *driver* para dispositivos *IDE* (devb-eide), y monta el disco duro en el directorio raíz. Por lo tanto se pueden instalar programas y librerías en el disco duro. Eso sí, es importante que estos archivos no se necesiten al arrancar. El tamaño de la imagen está limitado, por lo que es conveniente reducir al máximo el número de archivos que ésta contiene. Por otra parte cuanto menos ocupe la imagen, menos *RAM* consumirá.

La mayor parte de los programas son pequeños, y basta con copiar el ejecutable y las librerías que necesite en los directorios con las rutas por defecto (los elegidos con *PATH* y *LD_LIBRARY_PATH* en el *buildfile*). Por ejemplo, para instalar la aplicación de listado de archivos y carpetas (*ls*) bastaría con copiar el archivo *ls* en */bin*, y la librería *libc* en */lib*. Incluso este último paso no sería necesario, ya que esta librería la necesita la imagen, por lo que ya está instalada en */proc/boot*.

Sin embargo este proceso se complica mucho más cuando el programa no es tan sencillo. En ocasiones los programas necesitan archivos de configuración para poder funcionar, y los mensajes de error no son siempre clarificadores. Este ha sido el caso de programas como el editor *vim*, o el compilador *gcc*. En el caso del primero, al tratar de abrir el programa éste no reconocía el terminal (*Unrecognized TERM type*). Se probó a utilizar varios terminales diferentes infructuosamente. La solución pasaba por copiar el directorio */usr/lib/terminfo*. Más complicada fue la instalación de *gcc*, que se detalla a continuación.

- Copiar los ejecutables (*gcc* y *g++*) en la carpeta */bin*.

- Los archivos que se compilan suelen llamar a varios archivos de cabecera. Éstos tendrán que copiarse en la misma ubicación que en el sistema original, que es */usr/include*. Hay determinados archivos de cabecera que probablemente nunca llamarán las aplicaciones de este proyecto, como los del entorno gráfico. Sin embargo ante la duda, y teniendo en cuenta que no hay problemas de espacio en la *compact flash*, se ha decidido incluir todas. No obstante en caso de necesidad siempre pueden eliminarse.

- *gcc* invoca a una serie de programas y objetos, como *cc1* o *collect2*, que están en el directorio */usr/lib/gcc-lib/ntox86/2.95.3QNX-nto*. Se copia este directorio del sistema original.

- *gcc* necesita el programa *as*. Se instala en */bin*, y se soluciona el error.

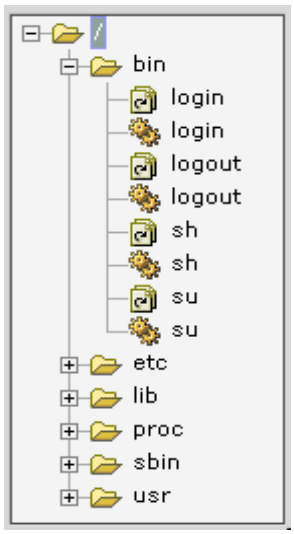
- *gcc* necesita el programa *ld*, que combina objetos y archivos necesarios para el programa que se está compilando. Aquí viene el primer problema serio, ya que *ld* se copió en los lugares en los que el sistema original estaba, y a pesar de todo seguía apareciendo el mensaje de error: *collect2: cannot find 'ld'*. Tras múltiples intentos fallidos, se optó por copiar el archivo *ld* en la misma carpeta que *collect2*, es decir, */usr/lib/gcc-lib/ntox86/2.95.3QNX-nto*. En el sistema original no hay copias ni enlaces de *ld* en tal directorio. Sorprendentemente el error desapareció al proceder de esta manera, por lo que se dejó así.

- El mismo problema surgió a la hora de llamar a todas las librerías, por lo que también se copiaron todas en el directorio de *collect2*.

Una vez hechas todas esas operaciones, *gcc* funcionaba correctamente. Para que lo hiciera también *g++* tan sólo hubo que añadir algún archivo más, como *_G_config.h* o la librería *libstdc++.so*.

3.5.4. Árbol de directorios.

Árbol de la imagen sola (expandida en la *RAM*):



Árbol de la ROM:

Figura 3-3. Árboles de directorios de RAM y ROM.

Ambos árboles, como se puede ver en la figura, están replegados, ya que si se desarrollaran por completo la imagen sería demasiado grande. El árbol resultante en el sistema es la suma de los dos que se muestran.

3.5.5. Programas instalados en la ROM.

Programas necesarios para compilar:

- Para construir:
make.
automake.
automake1.6.

- Compiladores:
gcc.
g++.

- Auxiliares:
as.
ld.

Visores de texto:

cat.
more.

Editores de texto:

elvis.
vi.
vim.

Buscadores:

find.
grep.

Exploración/gestión de archivos y directorios:

ls.
clear.
cp.
mkdir.
pwd.
rm.
mv.
ln.

Montado de dispositivos:

mount.
umount.

Comunicaciones:

- Red:
ftp.
ping.
telnet.

- Puerto serie:
qtalk.

Información del sistema:

ps.
pidin.
sin.

Ayuda:

use.

Herramientas de compresión/descompresión:

tar.
gzip.

3.6. Dificultades.

Uno de los mayores problemas a la hora de instalar el sistema operativo ha sido la búsqueda de ayuda. Como se explicará en las conclusiones, la instalación que se necesitaba en este proyecto era un tanto especial, ya que no se podía hacer por *CD*, pero tenía que incluir más funciones de las que suele tener un sistema embebido. Aun obviando este hecho, sigue siendo complicado encontrar información acerca de cómo se construye e instala una imagen en una máquina empotrada.

Aunque existen varias publicaciones sobre *QNX*, éstas son difíciles de conseguir y habría que pedirlas por correo al extranjero; por lo que la documentación electrónica se convierte en la fuente de información más importante. El primer sitio donde mirar es la ayuda propia de *QNX*. Esta ayuda en general es excepcionalmente buena. No se limita a los programas que trae la instalación, sino que describe también todos los comandos *POSIX* e incluso los de *C* estándar. A la hora de programar resulta un apoyo muy grande.

Sin embargo la ayuda no describe bien el proceso de creación de una imagen. Sí lo hace a grandes rasgos, pero a la hora de profundizar hay que mirar en varios documentos. Por otra parte en la ayuda sólo puede buscarse por nombres, y no por temas. Un ejemplo de este desorden es que en ciertos documentos se dice que hay que construir una imagen, e incluso se describe cómo es un *buildfile*; pero no se explica qué hay que hacer con la imagen una vez generada.

Para encontrar un tutorial completo sobre creación e instalación de imágenes es necesario remitirse la web oficial de *QNX* (*www.qnx.com*. Ver anexo 4). Creado por *Akhilesh Mritunjai*, el tutorial es una referencia imprescindible para todo aquel que se enfrente a este problema. Pero pese a lo completo de su contenido y al estilo coloquial en el que está escrito, sigue sin resolver muchos de los problemas que van surgiendo. Además hay algún que otro error en los *buildfile*s que expone, y que hacen que determinados programas que añade a la imagen no funcionen.

Otra ayuda importante ha sido *www.openqnx.com*, que es el único foro verdaderamente activo sobre *QNX* en internet. La mayor parte de las dudas de un neófito están ya planteadas y respondidas, y gracias al buscador de la web se puede llegar a ellas con facilidad. Aún así se siguen haciendo muchas preguntas nuevas sobre la creación e instalación de imágenes, lo cual vuelve a demostrar la falta de buena información sobre el tema, a pesar a la importancia del mismo.

Por último existen una serie de pequeñas webs que explican diversos aspectos de *QNX*, de mayor o menos utilidad. El fabricante de hardware *Versalogic* tiene una página en la que explican por encima el uso general de *QNX*, y cómo hacer para instalarlo en sus productos (ver anexo 5). Por extraño que resulte, pese a que para este proyecto no se ha usado en ningún momento material de este fabricante, su web ha sido una de las que más ha ayudado a comprender el complicado proceso de instalación.

3.7. Conclusiones.

QNX es un sistema operativo hecho para poder operar en un sistema embebido y consumiendo los mínimos recursos posibles. Para que la instalación sea mínima ha de haber tantas posibilidades para personalizar la instalación que resulta complejo hacerla. Como se ha visto hay que instalar manualmente aplicaciones tan básicas como *cp* o *ls*. Muchos de los programas que se utilizan normalmente vienen instalados con los paquetes básicos de cualquier sistema operativo, por lo que el usuario ya cuenta con ellos. Sin embargo no están, lo cual en más de una ocasión lleva a la confusión.

Los creadores de *QNX* han pensado por una parte en ordenadores de sobremesa en los que programar y trabajar, y un sistema empotrado en el que simplemente corran las aplicaciones creadas y sus programas auxiliares.

En cuanto a la instalación completa, el sistema operativo funciona muy bien. De hecho en la documentación del CD de instalación se pedía un ordenador Pentium III a 600 MHz. Por contra, el CD se instaló en un Pentium II a 233 MHz, y en ningún momento ha habido problema alguno. *QNX* tiene una gran estabilidad, y al estar basado en *UNIX* no resulta complicado el uso de los comandos.

Respecto al uso de *QNX* como sistema operativo, la instalación incluye programas básicos como el *shell*, editores de texto sencillos, un navegador de internet, visores de imágenes, etc. A esto hay que añadir las contribuciones de muchos usuarios que han creado o portado a *QNX* muchos programas de *software* libre tan útiles como un visor de archivos *pdf*, navegadores y clientes de correo de calidad (*Mozilla*), un editor para código *C/C++* muy cómodo (*Workspace*), aplicaciones de fotografía, y otras que hacen de *QNX* un sistema mucho más completo. Sin embargo *QNX* no está preparado para ser utilizado como sistema operativo principal. Para empezar le faltan herramientas y utilidades multimedia. Aunque no es necesario, también ayudaría mucho el que hubiera una consola con *scroll*. Pero la gran carencia de *QNX* es la de un *software* de herramientas de oficina. Tan sólo se puede instalar un editor de texto con formato, que es la versión portada del programa de distribución libre *Abiword*. Pero el editor no funciona nada bien, y lamentablemente al final resulta inútil tratar de escribir un documento serio con él. Por otra parte *Abiword* utiliza su propio formato de archivo, siendo incompatible con otros editores más conocidos, como *Microsoft Word* u *OpenOffice.org*. Un portado de esta última herramienta podría ser la solución definitiva a este grave problema. Hoy por hoy es necesario escribir la documentación sobre el trabajo hecho en *QNX* en otro sistema operativo, lo cual en muchas ocasiones resulta bastante engorroso.

En cuanto a la instalación mediante imagen, lo normal es que se incluya todo en la misma y se escriba en el *script* la ejecución de las aplicaciones pertinentes. La máquina se colocaría en su lugar de trabajo, se arranca y se deja funcionando. En este sentido el funcionamiento de *QNX* es excelente, ya que el aparato empotrado puede estar funcionando con una ROM en la que se han usado menos de seis *megas* y una *RAM* en la que se ha volcado la imagen, que descomprimida es probable que nunca vaya a llegar a los cinco *megas*. La creación de la imagen no es sencilla, y menos para un no iniciado. No sólo se necesita saber qué programas se ejecutan al arrancar, sino que hay que saber utilizarlos y adaptarlos al hardware que se tenga. Sin embargo si se quiere ajustar tanto la instalación, no es posible hacerlo de una forma más fácil.

De todas maneras los mayores problemas surgen a la hora de hacer una instalación intermedia; es decir, personalizar el sistema operativo sin una gran instalación, pero que incluya las funciones más comunes. Ya se dijo anteriormente que ése es el caso del presente proyecto, ya que se necesitaba una instalación pequeña pero en la que se tuviera la posibilidad de navegar por las carpetas, editar ficheros, compilar código y más utilidades que se han ido necesitando; como la compresión de archivos, las búsquedas, etc. Para ello hay que ir instalando cada programa que se vaya necesitando, y además saber qué librerías, *drivers*, archivos de configuración y ficheros auxiliares requiere la función. Quizás los desarrolladores de *QNX* debieran facilitar algo más este punto.

En definitiva, *QNX* es un sistema operativo que resulta óptimo para su uso en aplicaciones de tiempo real tanto por su estructura de microkernel por su estabilidad y su capacidad para instalarse y funcionar empleando los mínimos recursos necesarios. Flaquea en el uso general como sistema operativo de escritorio, si bien este punto está en proceso de mejora.