Publicaciones Técnicas
pubtecia@astro.unam.mx

Instituto de Astronomía
UNAM

Comunicación Interna CI-2014-05

aceptación: 02-06-2014
recepción: 20-08-2014

Procedimiento de configuración de la microcomputadora BeagleBone para que ejecute una aplicación dedicada desde memoria RAM .

S. Zazueta.
Instituto de Astronomía. Universidad Nacional Autónoma de México.
Km. 103 Carretera Tijuana-Ensenada, Ensenada, B.C., México.

RESUMEN:

El objetivo de este trabajo es mostrar la manera de configurar la microcomputadora BeagleBone para que ejecute una aplicación dedicada desde un disco de memoria RAM (*RAM disk*). El sistema está basado en el operativo Linux® y aprovechamos el mecanismo existente llamado "initrd". Se presentan los pasos necesarios para descomprimir una imagen "initrd" existente, modificar el archivo "init" y probar que el sistema ejecute la aplicación.

La aplicación de un sistema operativo que carga en memoria RAM todos sus programas es una característica deseable de un sistema de control dedicado, por lo que el trabajo sirve como referencia para su aplicación en los sistemas de control de los instrumentos del OAN.
Otra ventaja del uso de un disco de memoria RAM es que se maximiza la vida útil de la memoria flash, tarjeta microSD, de la BeagleBone.

Contenido

1. INTRODUCCIÓN

Una de las características principales de un sistema electrónico dedicado es la capacidad de iniciar y realizar sus funciones cuando se enciende. A todos nos ha pasado con una computadora con SO Linux o Windows® que el sistema no inicia por una falla en el sistema de archivos. Ya sea que el sistema de archivos esté "sucio" o que presente una corrupción tal que necesite la interacción con el usuario para poder tomar decisiones para repararlo.

La BeagleBone es una computadora de una sola tarjeta basada en un microprocesador ARM® que puede ejecutar una serie de sistemas operativos como Android, Windows o Linux.

En el OAN Ensenada hemos utilizado la BeagleBone para varias aplicaciones de control dedicado. Escogimos esta plataforma debido a su forma compacta, su conectividad, su bajo precio, la disponibilidad de herramientas de desarrollo de libre acceso y la facilidad con la que es posible desarrollar aplicaciones de control dedicado.

Una de las ventajas de la BeagleBone es que carga el operativo de una memoria flash tipo tarjeta microSD, lo que permite usarla como si tuviera instalado un disco duro. Sin embargo, las memorias flash tienen varias desventajas que afectan el desempeño de un sistema de control dedicado. Dos inconvenientes sobresalen: el primer problema es el número finito de ciclos de escritura de la memoria flash; esta condición a la larga causa que el sistema no inicie por daño en el disco duro. El segundo problema, que no es inherente a las memorias flash pero que es muy grave para un sistema de control dedicado, es la corrupción del sistema de archivos que se da con las posibles fallas de la energía eléctrica. Normalmente un corte pasajero de la energía eléctrica no debería causar que un sistema de control dedicado deje de funcionar. La corrupción del sistema de archivos y la posible reparación del mismo pueden requerir la intervención de un operador, lo que equivale a una falla del sistema.

Se puede mitigar el "desgaste" de la memoria flash limitando el número de accesos a la misma y configurando el operativo para que grabe sólo la información más valiosa. También es posible que el sistema inicie sin verificar el estado del sistema de archivos, el "fsck" en Linux o el "chkdsk" en Windows, pero a la larga se puede llegar a tener un sistema de archivos tan corrupto que no podrá iniciar.

Otra solución es utilizar una memoria flash con dos o más particiones. Una parte donde reside el sistema operativo y los programas de aplicación. Esta partición se monta con acceso de sólo lectura. Durante el proceso de inicio, el proceso de "bootstrap", la BeagleBone carga en memoria RAM un sistema de archivos compacto y ejecuta un "init" modificado donde se inician los programas del sistema de control dedicado.

La segunda partición de la memoria flash se usa como espacio de disco y se monta con acceso de lectura/escritura. El propósito de esta partición es guardar la información del funcionamiento del sistema de control dedicado. Si esta partición se corrompe sólo se pierden algunos archivos, como la última bitácora de funcionamiento del equipo o las últimas mediciones realizadas.

El objetivo de este trabajo es dar una descripción de los pasos necesarios para instalar una aplicación dedicada en la BeagleBone, que se ejecute desde la memoria RAM, basado en el mecanismo de un disco de memoria RAM y la creación de la imagen "initrd" del sistema de archivos que se cargará en el disco de memoria RAM.

El siguiente texto proviene de la documentación del núcleo del SO Linux.

Initrd provee la capacidad de cargar un disco en memoria RAM por medio del programa de inicio "boot loader". Esta capacidad está diseñada para que el sistema inicie en dos etapas: la primera donde el núcleo inicia con un juego de manejadores precompilados, y la segunda etapa donde se cargan módulos adicionales del disco en RAM "initrd".

Usaremos esta capacidad del núcleo y el programa de inicio del SO Linux para crear una imagen "initrd" que use la partición normal del operativo en modo de sólo lectura y otra partición de trabajo donde se almacenarán los documentos de trabajo e inclusive programas de aplicación.

2. PRIMEROS PASOS

Se supone que se tiene acceso a una BeagleBone blanca o negra y se cuenta con una memoria flash SD-CARD de al menos 4 GB.

El ambiente de trabajo es una PC con SO Linux con una distribución Ubuntu o Debian.

Empezaremos por instalar la distribución Debian en la Beaglebone. Si ya cuenta con una memoria microSD con el operativo Debian para la BeagleBone, hay que referirse al PASO 2.

2.1. PASO 1

La imagen actualizada de Debian para la BeagleBone se puede encontrar en la dirección:

http://beagleboard.org/latest-images

En este caso usaremos la imagen:

http://debian.beagleboard.org/images/bone-debian-7.4-2014-04-23-2gb.img.xz

Existe un buen tutorial para la instalación en:

http://beagleboard.org/Getting%20Started#update

Lo que sigue es un pequeño resumen de la liga anterior.

Abrimos una terminal y descomprimimos el archivo *.xz con:

$xzcat -d bone-debian-7.4-2014-04-23-2gg.img.xz > bone-debian.img

Para escribir la imagen en la memoria microSD se usa la utilería del Ubuntu "imagewriter". Se debe insertar una memoria microSD en la computadora y después ejecutar la utilería.

chpc$sudo imagewriter

El programa "imagewriter" abre una ventana de dialogo indicando los pasos a seguir. Nos pide seleccionar la imagen a grabar. En este caso grabaremos la imagen que acabamos de descomprimir arriba, "bone-debian.img".

2.2 PASO 2

Puesto que la imagen original es para una memoria de 2GB nos queda una parte de la memoria microSD sin utilizar. Con la utilería "gparted" haremos dos cambios a las particiones de la memoria microSD:

Hacemos más grande la partición 2, donde está instalado el sistema de archivos de Debian. Le agregaremos un "gigabyte". Esto es para poder instalar más paquetes útiles al sistema de archivos. Dejaremos un espacio libre de la memoria microSD de unos 800 megabytes.

Creamos una partición de tipo "ext4" en la parte libre de la memoria. Le pondremos el nombre "trabajo". El nombre de la partición es para ser congruentes con la nomenclatura usada en las siguientes secciones.

En caso de que la memoria microSD de la BeagleBone ya tenga instalado Debian, hay que modificar la tabla de particiones con la utilería "gparted" para crear la partición 3 de trabajo.

2.3. PASO 3

Una vez creada la partición de "trabajo", expulsamos de la computadora la memoria microSD y la insertamos en la BeagleBone, después lo conectamos con el cable USB a nuestra computadora.

Luego de un tiempo, la BeagleBone cargará el operativo y veremos aparecer el disco removible llamado BEAGLE_BONE en el manejador de archivos de nuestra computadora.

Podemos acceder a la BeagleBone por medio de una terminal de ssh para verificar que todo esté en orden.

$ssh root@192.168.7.2

En este punto el mensaje de inicio de la BeagleBone debe aparecer en la terminal del ssh.

```
Debian GNU/Linux 7

BeagleBoard.org BeagleBone Debian Image 2014-04-23

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
Last login: Wed Apr 23 20:21:03 2014 from chavapcgw2.local
root@beaglebone:~#
```

Para instalar el paquete "dropbear" es necesario que la Beaglebone tenga acceso a Internet.

Instalamos el paquete "dropbear" en la BeagleBone con las instrucciones normales:

apt-get install dropbear

El paquete "dropbear" es un servidor de ssh que usa muy pocos recursos y se presta para uso en aplicaciones dedicadas.

Una vez instalado el "dropbear" es necesario agregar una contraseña a la cuenta del súper usuario con la instrucción "passwd".

root@beaglebrootone:~# passwd

2.4. PASO 4

El siguiente paso es poblar el sistema de archivos que usaremos como disco de memoria RAM. El sistema debe contener la estructura de archivos estándar del SO Linux y contener las bibliotecas necesarias para ejecutar los programas que se van a utilizar. Además, debe contener un archivo ejecutable "init" o "linuxrc" en la raíz. Suena complicado pero usaremos la imagen que viene por omisión en el Debian y modificaremos sólo lo necesario.

La imagen del sistema de archivos que vamos a modificar se llama "initrd.img" y se encuentra en la primera partición tipo VFAT de la memoria microSD. Esta imagen es la que usa la BeagleBone para iniciar.

Usaremos la misma BeagleBone para modificar la imagen initrd. Podemos hacer estos pasos con el adaptador de la memoria microSD en la computadora pero lo haremos directamente en la BeagleBone.

Hacemos un ssh a la BeagleBone desde una terminal de la computadora.

Dentro de la BeagleBone, la trayectoria del archivo initrd es "/boot/uboot/initrd.img"

Ir al directorio trabajo, que es la partición 3 de la memoria microSD, y extraer el contenido del archivo initrd como sigue:

```
cd /media/trabajo
mkdir -p compila/rootfs
cd compila/rootfs
gzip -d < /boot/uboot/initrd.img | cpio --extract --verbose --make-directories --no-absolute-filenames
```

El directorio donde descomprimimos la imagen "initrd" puede estar localizado en cualquier lugar del sistema de archivos. En el ejemplo lo colocamos en la partición de trabajo para reutilizarlo en caso de futuras modificaciones.

De acuerdo a la documentación del SO Linux, el programa que se ejecuta una vez montado el sistema de archivos es el "init" o "linuxrc".

Para ahorrar tiempo usaremos el archivo "init" que viene en la imagen "initrd" que acabamos de descomprimir.

Sólo cambiaremos la parte que nos interesa, que es justo cuando el programa "init" trata de montar su sistema de archivos de raíz permanente. En ese punto montaremos nuestra partición con acceso de sólo lectura y ejecutaremos el "script" **aplica.sh** que es el nombre de la aplicación

dedicada que iniciará todas las instrucciones que necesitamos que ejecute nuestro controlador dedicado.

Incluimos el listado completo del archivo “init” en el Apéndice para mostrar las partes modificadas.

El siguiente fragmento muestra la parte donde se monta y ejecuta la aplicación dedicada.

```
# Fragmento del programa “init” modificado
mkdir /app
sleep 2
i=0
echo "esperando device ${devapp} 20 segs"
while [ $i -lt 30 ];
do
   if [[ -e ${devapp} ]]; then
      echo "Montando app"
      mount -t ${devapptype} -o ro ${devapp} /app
      sleep 1
      break;
   fi
  sleep 1
  i=$(($i+1))
done
## ejecutamos la aplicación
if [[ -x "/app/aplica.sh" ]]; then
      echo "Iniciando aplicacion"
    sh /app/aplica.sh
fi
## Fin del fragmento
```

Una vez modificado el programa “init”, tenemos que volver a comprimir la imagen con las siguientes instrucciones (se supone que seguimos en el directorio /media/trabajo/compila/rootfs) :

```
find . | cpio -H newc -o > ../initramfs.cpio
cd ..
cat initramfs.cpio | gzip > initrd2.img
```

En este punto ya tenemos la imagen en el archivo initrd2.img.

Ahora podemos editar el programa de aplicación “aplica.sh”. Este archivo debe estar en el directorio de trabajo que se monta en el programa “init” en la parte que dice:

mount -t ${devapptype} -o ro ${devapp} /app

Por omisión asignamos devapp=/dev/mmcblkop3 (ver el listado del programa “init” en el Apéndice).

Por ahora sólo agregaremos al archivo de aplicación un seudo servidor de Telnet utilizando el comando “nc” que está incluido en el paquete “busybox” de la imagen original “initrd”. Y le agregamos una llamada al “script” que ejecutará el servidor de ssh (dropbear).

Editamos el programa “aplica.sh” con la instrucción:

nano /media/trabajo/aplica.sh

para que contenga las siguientes instrucciones:

```
#!/bin/sh
ifconfig etho 192.168.0.123
busybox nc -l -l -p 9090 -e /bin/sh &
sh /app/app/ap_dropbear.sh
## fin del listado de aplica.sh
```

Luego lo hacemos ejecutable con:

chmod +x /media/trabajo/aplica.sh

Creamos el directorio /media/trabajo/app con:

mkdir /media/trabajo/app

Editamos el “script” ap_dropbear.sh que estamos llamando en “aplica.sh” para que tenga el siguiente contenido:

```
#!/bin/sh
mkdir /oldroot
mount -t ext4 -o ro /dev/mmcblkop2 /oldroot
nn=$( cat /oldroot/etc/shadow | grep root|cut -f2 -d':' )
cat /oldroot/etc/passwd| sed "s/root:x/root:${nn}/" > /tmp/passwd
sed -e 's/bash/sh/g' /tmp/passwd > /tmp/passwd1
mv /tmp/passwd1 /etc/passwd
echo "/oldroot/lib" >> /etc/ld.so.conf
echo "/oldroot/lib/arm-linux-gnueabihf/" >> /etc/ld.so.conf
cp -dpa /oldroot/etc/dropbear /etc
/oldroot/sbin/ldconfig
/oldroot/usr/sbin/dropbear -E &
```

En este punto tenemos una imagen "initrd2.img" y los "scripts" aplica.sh y ap_dropbear.sh en el directorio /media/trabajo y /media/trabajo/app.

Nos resta modificar el ambiente de inicio del programa U-boot ("boot loader") de la BeagleBone para que cargue la imagen "initrd2.img" en lugar de "initrd.img".

Primero copiamos la imagen al directorio adecuado con:

cp /media/trabajo/compila/initrd2.img /boot/uboot/

El archivo donde se carga el ambiente del U-boot se llama "**uEnv.txt**" y está ubicado en el directorio */boot/uboot* de la BeagleBone.

Sólo hay que modificar en "uEnvt.txt" la línea del nombre del archivo initrd, donde dice:

```
initrd_file=initrd.img
```

Hay que cambiar la línea para que se lea:

```
initrd_file=initrd2.img
```

que es el nombre de nuestra imagen.

Salvamos el archivo *uEnv.txt* y ya estamos listos para que al siguiente reinicio de la BeagleBone, cargue la imagen en un disco de memoria RAM.

Reiniciamos la BeagleBone con un "reboot" o por medio del botón de restablecimiento. Después de unos segundos la BeagleBone debe estar en línea.

Para probar si todo salió bien podemos escribir en una terminal de la PC la siguiente instrucción:

PC$ nc 192.168.0.123 9090

El "nc" no nos contesta con un mensaje de inicio como el "telnetd" pero podemos darle instrucciones, como por ejemplo "ls -x" que debe responder:

```
ls -x
app     bin     conf    dev   etc    init   lib   proc   root
run     sbin    scripts sys   tmp    uboot  var
```

También podemos entrar en la BeagleBone por medio del ssh (dropbear) si todo salió bien con:

ssh root@192.168.0.123

En este punto podemos agregar al programa "aplica.sh" nuestras aplicaciones o scripts.

En caso de que no inicie la Beaglebone se puede usar una terminal serie para monitorear el proceso de "bootstrap" del "U-boot" y diagnosticar dónde está la causa de que no inicie. Con la BeagleBone blanca se puede acceder a la terminal serie con el programa "minicom" o el programa "screen".

Para el caso de la BeagleBone negra es necesario usar un cable adaptador como el "TTL-232R-3V3" para poder usar los programas de terminal serie.

3. NOTAS Y RECOMENDACIONES

Se pueden incluir muchas bibliotecas o programas dentro de la imagen initrd modificada, pero es mejor utilizar el sistema de archivos de Debian que ya tenemos instalado. Montar este sistema de archivos con acceso de sólo lectura y agregar las bibliotecas y las trayectorias adecuadas a las variables de ambiente en el "script" aplica.sh o en otros "scripts" ejecutados a partir de éste. De esta manera es posible usar toda la potencia del Linux, como Python, editores de texto como el nano, etc.

Una posible secuencia de instrucciones para una aplicación en Python es:

```
#!/bin/sh
## listado script ap1.sh
mkdir /oldroot
mount -t ext4 -o ro /dev/mmcblkop2 /oldroot
echo "/oldroot/lib/" >> /etc/ld.so.conf
echo "/oldroot/lib/arm-linux-gnueabihf/" >> /etc/ld.so.conf
/oldroot/sbin/ldconfig
export PATH=$PATH:/oldroot/bin:/oldroot/usr/bin
python /app/app/mi_aplicacion.py
```

Este puede ser otro "script" de aplicación ("ap1.sh"o que se ejecute desde el programa de inicio "aplica.sh".

Para modificar el archivo de aplicación "aplica.sh", o hacer cambios a los programas, se puede hacer de dos maneras:

Una: Montando la memoria en el adaptador y editando los programas en nuestra PC.

Dos: Podemos editar en la misma BeagleBone. Esta sería la manera de poder hacerlo mediante la red en un sistema de control en funcionamiento.

Hacemos un ssh a la BeagleBone

PC$ ssh root@192.168.0.123

La BeagleBone debe contestar algo parecido al siguiente mensaje

```
[190] Jan 01 01:37:39 lastlog_perform_login: Couldn't stat /var/log/lastlog: No such file or directory
[190] Jan 01 01:37:39 lastlog_openseek: /var/log/lastlog is not a file or directory!
BusyBox v1.20.2 (Debian 1:1.20.0-7) built-in shell (ash)
Enter 'help' for a list of built-in commands.
~#
```

Montamos la partición de trabajo con acceso de lectura/escritura

```
~# mount -o remount,rw /app
```

Y usamos el "vi" que está incluido en el "busybox".

```
~# vi /app/app/ap1.sh
```

Al concluir los cambios volvemos a montar la partición como de sólo lectura y listo.

```
~# mount -o remount,ro /app
```

4. PARA REGRESAR LA BEAGLEBONE A SU ESTADO ORIGINAL

Para regresar la BeagleBone a su estado original basta regresar la línea del archivo "uEnv.txt" a su valor inicial de

```
initrd_file=initrd.img
```

Es posible hacerlo en la PC con el adaptador de la memoria microSD, o montando la partición tipo vfat y editando el archivo en la misma BeagleBone.

5. CONCLUSIONES

Se presentó un procedimiento para generar un sistema de control dedicado con operativo Linux basado en una BeagleBone que ejecuta sus programas desde memoria RAM.

El objetivo es maximizar la vida de la memoria flash de tarjeta microSD montando los sistemas de archivos con acceso de sólo lectura. También se trata de minimizar la probabilidad de falla al momento de iniciar.

Es importante mencionar que el procedimiento presentado es aplicable a otras plataformas con operativo Linux. Por ejemplo, si requerimos usar una computadora como controlador dedicado, podemos usar el initrd original de Debian o el Ubuntu y modificar la configuración del Grub en lugar del programa Uboot.

6. REFERENCIAS

[1] Manual de usuario de la BeagleBone:
http://beagleboard.org/Products/BeagleBone

[2] Documentación del núcleo de Linux:
https://www.kernel.org/doc/Documentation/initrd.txt

[3] Documentación del U-boot:
http://www.denx.de/wiki/U-Boot

[4] Documentación del BusyBox:
http://www.busybox.net/downloads/BusyBox.html

APÉNDICE A. LISTADO DEL PROGRAMA "INIT"

```
#!/bin/sh

echo "Loading, please wait..."

[ -d /dev ] || mkdir -m 0755 /dev
[ -d /root ] || mkdir -m 0700 /root
[ -d /sys ] || mkdir /sys
[ -d /proc ] || mkdir /proc
[ -d /tmp ] || mkdir /tmp
mkdir -p /var/lock
mount -t sysfs -o nodev,noexec,nosuid sysfs /sys
mount -t proc -o nodev,noexec,nosuid proc /proc

# Note that this only becomes /dev on the real filesystem if udev's scripts
# are used; which they will be, but it's worth pointing out
tmpfs_size="10M"
if [ -e /etc/udev/udev.conf ]; then
       . /etc/udev/udev.conf
fi
if ! mount -t devtmpfs -o size=$tmpfs_size,mode=0755 udev /dev; then
       echo "W: devtmpfs not available, falling back to tmpfs for /dev"
       mount -t tmpfs -o size=$tmpfs_size,mode=0755 udev /dev
       [ -e /dev/console ] || mknod -m 0600 /dev/console c 5 1
       [ -e /dev/null ] || mknod /dev/null c 1 3
fi
mkdir /dev/pts
mount -t devpts -o noexec,nosuid,gid=5,mode=0620 devpts /dev/pts || true
mount -t tmpfs -o "nosuid,size=20%,mode=0755" tmpfs /run
mkdir -m 0755 /run/initramfs

# Export the dpkg architecture
export DPKG_ARCH=
. /conf/arch.conf

# Set modprobe env
export MODPROBE_OPTIONS="-qb"

# Export relevant variables
export ROOT=
export ROOTDELAY=
export ROOTFLAGS=
export ROOTFSTYPE=
export IP=
export BOOT=
export BOOTIF=
export UBIMTD=
export break=
export init=/sbin/init
export quiet=n
export readonly=y
export rootmnt=/root
export debug=
export panic=
```

```
export blacklist=
export resume=
export resume_offset=

################################## modificado app dedicada ######################
export devapp=/dev/mmcblk0p3
export devapptype=ext4

################################## fin modificado          ######################

# Bring in the main config
. /conf/initramfs.conf
for conf in conf/conf.d/*; do
       [ -f ${conf} ] && . ${conf}
done
. /scripts/functions

# Parse command line options
for x in $(cat /proc/cmdline); do
       case $x in
       init=*)
              init=${x#init=}
              ;;
       root=*)
              ROOT=${x#root=}
              case $ROOT in
              LABEL=*)
                     ROOT="${ROOT#LABEL=}"

                     # support any / in LABEL= path (escape to \x2f)
                     case "${ROOT}" in
                     */*)
                     if command -v sed >/dev/null 2>&1; then
                            ROOT="$(echo ${ROOT} | sed 's,/,\\x2f,g')"
                     else
                            if [ "${ROOT}" != "${ROOT#/}" ]; then
                                   ROOT="\x2f${ROOT#/}"
                            fi
                            if [ "${ROOT}" != "${ROOT%/}" ]; then
                                   ROOT="${ROOT%/}\x2f"
                            fi
                            IFS='/'
                            newroot=
                            for s in $ROOT; do
                                   newroot="${newroot:+${newroot}\\x2f}${s}"
                            done
                            unset IFS
                            ROOT="${newroot}"
                     fi
                     esac
                     ROOT="/dev/disk/by-label/${ROOT}"
                     ;;
              UUID=*)
                     ROOT="/dev/disk/by-uuid/${ROOT#UUID=}"
                     ;;
              /dev/nfs)
```

```
                [ -z "${BOOT}" ] && BOOT=nfs
                ;;
        esac
        ;;
    rootflags=*)
        ROOTFLAGS="-o ${x#rootflags=}"
        ;;
    rootfstype=*)
        ROOTFSTYPE="${x#rootfstype=}"
        ;;
    rootdelay=*)
        ROOTDELAY="${x#rootdelay=}"
        case ${ROOTDELAY} in
        *[![:digit:].]*)
                ROOTDELAY=
                ;;
        esac
        ;;
    nfsroot=*)
        NFSROOT="${x#nfsroot=}"
        ;;
    ip=*)
        IP="${x#ip=}"
        ;;
    boot=*)
        BOOT=${x#boot=}
        ;;
    ubi.mtd=*)
        UBIMTD=${x#ubi.mtd=}
        ;;
    resume=*)
        RESUME="${x#resume=}"
        ;;
    resume_offset=*)
        resume_offset="${x#resume_offset=}"
        ;;
    noresume)
        noresume=y
        ;;
    panic=*)
        panic="${x#panic=}"
        case ${panic} in
        *[![:digit:].]*)
                panic=
                ;;
        esac
        ;;
    quiet)
        quiet=y
        ;;
    ro)
        readonly=y
        ;;
    rw)
        readonly=n
        ;;
```

```
        debug)
                debug=y
                quiet=n
                exec >/run/initramfs/initramfs.debug 2>&1
                set -x
                ;;
        debug=*)
                debug=y
                quiet=n
                set -x
                ;;
        break=*)
                break=${x#break=}
                ;;
        break)
                break=premount
                ;;
        blacklist=*)
                blacklist=${x#blacklist=}
                ;;
        netconsole=*)
                netconsole=${x#netconsole=}
                ;;
        ## modificado app dedicada
        devapp=*)
          devapp=${x#devapp=}
                  ;;
          devapptype=*)
          devapptype=${x#devapptype=}
                  ;;

        BOOTIF=*)
                BOOTIF=${x#BOOTIF=}
                ;;
        esac
done

# Default to BOOT=local if no boot script defined.
if [ -z "${BOOT}" ]; then
        BOOT=local
fi

if [ -n "${noresume}" ]; then
        export noresume
        unset resume
else
        resume=${RESUME:-}
fi

maybe_break top

# Don't do log messages here to avoid confusing graphical boots
run_scripts /scripts/init-top

maybe_break modules
[ "$quiet" != "y" ] && log_begin_msg "Loading essential drivers"
```

```
load_modules
[ "$quiet" != "y" ] && log_end_msg

[ -n "${netconsole}" ] && modprobe netconsole netconsole="${netconsole}"

maybe_break premount
[ "$quiet" != "y" ] && log_begin_msg "Running /scripts/init-premount"
run_scripts /scripts/init-premount
[ "$quiet" != "y" ] && log_end_msg


## modificado app dedicada
#maybe_break mount
log_begin_msg "Mounting root file system"
#. /scripts/${BOOT}
#parse_numeric ${ROOT}
#maybe_break mountroot
#mountroot
log_end_msg

################################### modificado app dedicada ########################
mkdir /app
sleep 2
i=0
echo "esperando device ${devapp} 20 segs"
while [ $i -lt 30 ];
do
    if [[ -e ${devapp} ]]; then
        echo "Montando app"
        ## mount -t ${devapptype} -o ro ${devapp}  /app
        mount -t ${devapptype} -o ro ${devapp}  /app
        sleep 1
        break;
    fi
    sleep 1
    i=$(($i+1))
done

if [[ -x "/app/aplica.sh" ]]; then
        echo "Iniciando aplicacion"
   sh /app/aplica.sh
fi

BB=busybox
ln -s /bin/busybox /bin/login
echo "Ok"

exec sh
panic "Please file a bug on initramfs-tools."

################################ modificado app dedicada ############################
```