Boot Process Customization

When designing an embedded system, it is often necessary to add or remove applications and tasks from the system initialization sequence. This guide describes the initialization method used for EMAC OE systems and provides information on customizing the boot process.

System V Initialization

EMAC OE uses the System V Initialization method. This is a simple method for system initialization using a set of scripts run in sequential order. When the kernel has finished loading, `/sbin/init` is started to initialize the userspace services in the system. `init` is process id 1 and the parent of all processes in the system.

Runlevels

System V Init uses different runlevels to control the boot process. Each runlevel has a set of scripts that are run sequentially to start various services on the system. The default runlevel on EMAC OE systems is 5. This is set in `/etc/inittab` with the line:

```
# The default runlevel.
id:5:initdefault:
```

There are a total of seven runlevels available in System V Init, labeled 0 through 6. Runlevel 5 is the full user level in EMAC OE systems, regardless of whether a GUI is installed on the board or not. Runlevel 0 halts the system, and runlevel 6 is used for reboot. The other runlevels can be used for other purposes if desired, such as to configure different levels of user functionality in each runlevel.

During boot, the scripts `/etc/init.d/rcS` and `/etc/init.d/rc` are executed to run the scripts in `/etc/rcS.d/` starting with an `S` in lexicographic order followed by the scripts in `/etc/rc5.d/`, assuming that 5 is the default runlevel. All of these startup scripts are passed the argument `start`. During halt or reboot, the scripts in `/etc/rc0.d/` or `/etc/rc6.d/` starting with a `K` are run in lexicographic order with the argument `stop`. To control the order in which the scripts are run, each filename is prefixed with a number from 00-99. For example, the listing below illustrates the files in `/etc/rcS.d/` that will be run in order before entering the default runlevel:

```
som9g45:/etc/rcS.d# ls
S02banner                S20modutils.sh           S40networking
S03sysfs                 S30ramdisk               S41ifplugd
S03udev                  S35mountall.sh           S45mountnfs.sh
S05devices               S37populate-volatile.sh  S55bootmisc.sh
S06alignment             S38devpts.sh             S98ipkg-configure
S10checkroot             S39hostname.sh           S99finish.sh
```

The action to perform at each level is specified in `/etc/inittab`. For example, the following lines are used to trigger the execution of the `/etc/init.d/rcS` and `/etc/init.d/rc` scripts:

```
si::sysinit:/etc/init.d/rcS
....
l5:5:wait:/etc/init.d/rc 5
....
```

Initscripts

The directory `/etc/init.d/` holds initialization scripts that are run by `init` during boot or shutdown. These scripts should be designed to accept at least three arguments: `start`, `stop`, or `restart`. The files in the `/etc/rc*.d/` directories are symbolic links to the scripts in `/etc/init.d/`. This structure allows for easy modification of the boot process and the ability for a script to be run at different places in different runlevels. The detailed listing of the

/etc/rcS.d/ directory is shown below:

```
som9g45:/etc/rcS.d# ls -l
lrwxrwxrwx    1 root     root           16 Dec 31  1969 S02banner -> ../init.d/banner
lrwxrwxrwx    1 root     root           18 Dec 31  1969 S03sysfs -> ../init.d/sysfs.sh
lrwxrwxrwx    1 root     root           14 Dec 31  1969 S03udev -> ../init.d/udev
lrwxrwxrwx    1 root     root           17 Dec 31  1969 S05devices -> ../init.d/devices
lrwxrwxrwx    1 root     root           22 Dec 31  1969 S06alignment -> ../init.d/alignment.sh
lrwxrwxrwx    1 root     root           19 Dec 31  1969 S10checkroot -> ../init.d/checkroot
lrwxrwxrwx    1 root     root           21 Dec 31  1969 S20modutils.sh -> ../init.d/modutils.sh
lrwxrwxrwx    1 root     root           17 Dec 31  1969 S30ramdisk -> ../init.d/ramdisk
lrwxrwxrwx    1 root     root           21 Dec 31  1969 S35mountall.sh -> ../init.d/mountall.sh
lrwxrwxrwx    1 root     root           30 Dec 31  1969 S37populate-volatile.sh -> ../init.d/populate
lrwxrwxrwx    1 root     root           19 Dec 31  1969 S38devpts.sh -> ../init.d/devpts.sh
lrwxrwxrwx    1 root     root           21 Dec 31  1969 S39hostname.sh -> ../init.d/hostname.sh
lrwxrwxrwx    1 root     root           20 Dec 31  1969 S40networking -> ../init.d/networking
lrwxrwxrwx    1 root     root           17 Dec 31  1969 S41ifplugd -> ../init.d/ifplugd
lrwxrwxrwx    1 root     root           21 Dec 31  1969 S45mountnfs.sh -> ../init.d/mountnfs.sh
lrwxrwxrwx    1 root     root           21 Dec 31  1969 S55bootmisc.sh -> ../init.d/bootmisc.sh
lrwxrwxrwx    1 root     root           24 Dec 31  1969 S98ipkg-configure -> ../init.d/ipkg-configure
lrwxrwxrwx    1 root     root           19 Dec 31  1969 S99finish.sh -> ../init.d/finish.sh
```

Use the other boot scripts on the system for examples when creating custom initscripts. The application that is being started should be stored in the system PATH, such as /usr/bin/, and started from the script. For example, the busybox-httpd initscript is shown below.

```
#!/bin/sh
DAEMON=/usr/sbin/httpd
NAME=httpd
DESC="Busybox HTTP Daemon"
#HTTPROOT="/srv/www"
HTTPROOT="/home/www"
ARGS="-h $HTTPROOT"

test -f $DAEMON || exit 0

set -e

case "$1" in
    start)
        echo -n "starting $DESC: $NAME... "
        if [ ! -d $HTTPROOT ]; then
                echo "$HTTPROOT is missing."
                exit 1
        fi
        start-stop-daemon -S -b -n $NAME -a $DAEMON -- $ARGS
        echo "done."
        ;;
    stop)
        echo -n "stopping $DESC: $NAME... "
        start-stop-daemon -K -n $NAME
        echo "done."
        ;;
    restart)
        echo "restarting $DESC: $NAME... "
        $0 stop
        $0 start
        echo "done."
        ;;
    reload)
        echo -n "reloading $DESC: $NAME... "
        killall -HUP $(basename ${DAEMON})
        echo "done."
        ;;
    *)
        echo "Usage: $0 {start|stop|restart|reload}"
        exit 1
        ;;
esac

exit 0
```

The `start-stop-daemon` can be used to control the creation and termination of the application as illustrated in the `busybox-httpd` initscript above. The available options for the Busybox `start-stop-daemon` included with EMAC OE are shown below.

```
BusyBox v1.13.2 (2009-06-24 18:00:48 CDT) multi-call binary

Usage: start-stop-daemon [OPTIONS] [-S|-K] ... [-- arguments...]

Search for matching processes, and then
-K: stop all matching processes.
-S: start a process unless a matching process is found.

Process matching:
        -u,--user USERNAME|UID  Match only this user's processes
        -n,--name NAME          Match processes with NAME
                                in comm field in /proc/PID/stat
        -x,--exec EXECUTABLE    Match processes with this command
                                in /proc/PID/cmdline
        -p,--pidfile FILE       Match a process with PID from the file
        All specified conditions must match
-S only:
        -x,--exec EXECUTABLE    Program to run
        -a,--startas NAME       Zeroth argument
        -b,--background         Background
        -N,--nicelevel N        Change nice level
        -c,--chuid USER[:[GRP]] Change to user/group
        -m,--make-pidfile       Write PID to the pidfile specified by -p
-K only:
        -s,--signal SIG         Signal to send
        -t,--test               Match only, exit with 0 if a process is found
Other:
        -o,--oknodo             Exit with status 0 if nothing is done
        -v,--verbose            Verbose
        -q,--quiet              Quiet
```

Adding and Removing Scripts

A new or existing initscript can be added or removed from the start process by simply creating or destroying the symbolic links from the `/etc/rc*.d/` directories. A utility application, `update-rc.d` is provided to automate this process. The usage of `update-rc.d` is shown below:

```
usage: update-rc.d [-n] [-f] [-r <root>] <basename> remove
       update-rc.d [-n] [-r <root>] [-s] <basename> defaults [NN | sNN kNN]
       update-rc.d [-n] [-r <root>] [-s] <basename> start|stop NN runlvl [runlvl] [...] .
                -n: not really
                -f: force
                -r: alternate root path (default is /)
                -s: invoke start methods if appropriate to current runlevel
```

`update-rc.d` can be used to add or remove the startup links for any initscript in the system. For example, the `busybox-httpd` script has the following links to it in the boot process.

```
/etc/rc0.d/K20busybox-httpd
/etc/rc1.d/K20busybox-httpd
/etc/rc2.d/S20busybox-httpd
/etc/rc3.d/S20busybox-httpd
/etc/rc4.d/S20busybox-httpd
/etc/rc5.d/S20busybox-httpd
/etc/rc6.d/K20busybox-httpd
```

Removing the links from the system startup without removing the `/etc/init.d/busybox-httpd` file will effectively disable the application from running at system startup or shutdown. To do this, use the `remove` argument to `update-rc.d` as illustrated below.

```
root@som9g45:~# update-rc.d -f busybox-httpd remove
update-rc.d: /etc/init.d/busybox-httpd exists during rc.d purge (continuing)
 Removing any system startup links for busybox-httpd ...
  /etc/rc0.d/K20busybox-httpd
  /etc/rc1.d/K20busybox-httpd
  /etc/rc2.d/S20busybox-httpd
  /etc/rc3.d/S20busybox-httpd
  /etc/rc4.d/S20busybox-httpd
  /etc/rc5.d/S20busybox-httpd
  /etc/rc6.d/K20busybox-httpd
```

The `-f` argument is required in this case because `/etc/init.d/busybox-httpd` exists.

To add the links for the `busybox-httpd` init script back into the system, use the `defaults` argument to `update-rc.d`, or specify the start and stop number for each runlevel explicitly as show below. Both methods will yield the exact same results in this case, as the default action is to add `S20` and `K20` links for the startup and shutdown runlevels respectively.

```
root@som9g45:~# update-rc.d busybox-httpd start 20 2 3 4 5 . stop 20 0 1 6 .
 Adding system startup for /etc/init.d/busybox-httpd ...
  /etc/rc2.d/S20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc3.d/S20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc4.d/S20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc5.d/S20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc0.d/K20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc1.d/K20busybox-httpd -> ../init.d/busybox-httpd
  /etc/rc6.d/K20busybox-httpd -> ../init.d/busybox-httpd
```

Custom Initialization

In some designs, the target application is so specific that only one main task will be performed. After a custom application has been developed and tested, it may be possible to simplify the boot process significantly by replacing System V Init altogether. The Linux kernel accepts a boot argument named `init` to specify the application to execute as `init`. This section provides a simple example of how to create a custom initialization method.

Using this method will disable all of the standard system services unless they are explicitly started. It is possible to lock out all access to the system if you have no way of accessing the bootloader.

System Configuration

Generally, some basic system configuration and initialization will be necessary prior to running a custom application. This can easily be done using a shell script as the `init` process. The example provided below mounts procfs and sysfs and then configures the networking interface before starting the custom application. The `exec` shell builtin is used to start the custom application as this replaces the process image of the shell script with the custom application. The script is stored in `/sbin/` and is made executable by root.

custom-init.sh

```
#!/bin/sh

# mount procfs and sysfs
if [ -e /proc ] && ! [ -e /proc/mounts ]; then
  mount -t proc proc /proc
fi

if [ -e /sys ] && grep -q sysfs /proc/filesystems; then
  mount sysfs /sys -t sysfs
fi

ifconfig lo 127.0.0.1 up
ifconfig eth0 10.0.2.41 netmask 255.255.255.0 up

exec /usr/bin/custom-application
```

More system configuration may be required in certain systems, such as loading required kernel modules.

Custom Application

The custom application that is executed by the script can be simple or complex depending on the purpose. Note that the application should not exit, as this would leave the system in an invalid state and result in a kernel panic. In this simple example, the custom application is a shell script that prints “Hello world!” every second in an infinite loop. The file is given executable permissions by root and stored at `/usr/bin/` on the system.

custom-application

```
#!/bin/sh

while [ 1 ]
do
    echo "Hello world!"
    sleep 1
done
```

Changing the Kernel Arguments

In order to get the kernel to use the custom init design, the arguments passed to the kernel need to be changed to add the value `init=/sbin/custom-init.sh`. This must be accomplished by changing the bootloader configuration. For U-Boot, the `bootargs` variable will need to be changed to add the `init` value. RedBoot stores this configuration in the `boot_script_data` configuration. The kernel arguments can be modified for LILO by entering them directly at the LILO prompt or in the LILO configuration file. Refer to the documentation for the bootloader on your system for more detailed information.

Be very careful when making any changes to settings in the bootloader. It is possible to render a system unusable from incorrect configuration.

Testing

After booting the system with the new initialization settings, the custom application should be executed. Using the example above, the “Hello world!” message will be printed to the console repeatedly. Note that boot time will be significantly reduced because the system is performing a minimal amount of work during the boot process.

» import » qt » install » getting_started » eclipse » uboot_image_loading » emac_oe_fact » emac_oe_getting_started » linux_start » boot_process

---