# Building Root filesystem

The organization of sub-directories and files in **linux root filesystem (/)** is well-defined by the Filesystem Hierarchy Standard, and is as follows:

| | |
|---|---|
| **/bin** | Basic programs |
| **/boot** | Kernel image |
| **/dev** | Device files |
| **/etc** | System-wide configuration |
| **/home** | Directory for the users home directories |
| **/lib** | Basic libraries |
| **/media** | Mount points for removable media |
| **/mnt** | Mount points for static media |
| **/proc** | Mount point for the proc virtual filesystem |
| **/root** | Home directory of the root user |
| **/sbin** | Basic system programs |
| **/sys** | Mount point of the sysfs virtual filesystem |
| **/tmp** | Temporary files |
| **/usr** | |
| **/usr/bin** | Non-basic programs |
| **/usr/lib** | Non-basic libraries |
| **/usr/sbin** | Non-basic system programs |
| **/var** | Variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files |

Creating root filesystem for a target machine is, creating required directories and populating them with appropriate files.

## Creating minimal Root filesystem

*Create a work space for root filesystem to be built:*

**# mkdir roofs**

**# cd rootfs**

*Now create the root filesystem structure for target machine:*

**# mkdir dev proc sys etc bin sbin lib mnt usr usr/bin usr/sbin**

## Populate bin, sbin, usr/bin, usr/sbin

- Binaries for target machine is created using busybox.

- A Linux system needs a basic set of programs to work
    - An init program
    - A shell
    - Various basic utilities for file manipulation and system configuration

- **Busybox** is an alternative solution, extremely common on embedded systems
    - Rewrite of many useful Unix command line utilities
    - Integrated into a single project, which makes it easy to work with
    - Designed with embedded systems in mind: highly configurable, no unnecessary features

- All the utilities are compiled into a single executable, /bin/busybox, symbolic links to /bin/busybox are created for each application integrated into Busybox





Configure Busybox

*Download busybox source from http://busybox.net and extract it:*

# tar -xvf <archived-busybox-source-file>

*Configure busybox,*

# make menuconfig

as follows:

Busybox Settings --->

Build Options --->

[*] Build BusyBox as a static binary (no shared libs)

Coreutils-->

[*] sync

[ ] syncfs

Linux System Utilities --->

[*] mdev

[ ] nsenter

*Compile busybox using cross-compiler:*

# make CROSS_COMPILE=$(PREFIX-CROSSCOMPILER)

*Example:*

# make CROSS_COMPILE=arm-linux-

*Install the commands in target root filesystem:*

# make CROSS_COMPILE=arm-linux- CONFIG_PREFIX=<path-to-rootfs> install

*Example:*

# make CROSS_COMPILE=arm-linux- CONFIG_PREFIX=/root/elinux/workspace/rootfs install





Populate etc

init:

- Init is the first process started during booting, and is typically assigned PID number 1.
- Its primary role is to create processes from a scriptstored in the file /etc/inittab file.
- It is started by the kernel, and if the kernel is unable to start it, a kernel panic will result.
- All System V init scripts are stored in /etc/rc.d/init.d/ or /etc/init.d directory. These scripts are used to control system startup and shutdown.

- **BusyBox** can handle the system's startup. BusyBox **/sbin/init** is particularly well adapted to embedded systems, because it provides most of the init functionality an embedded system typically needs without dragging the weight of the extra features found in System V init.

- BusyBox init does not provide runlevel support.
- The **init** routine of BusyBox carries out the following main tasks in order:

1. Sets up signal handlers for init.
2. Initializes the console(s).
3. Parses the inittab file, **/etc/inittab**.
4. Runs the system initialization script. **/etc/init.d/rcS** is the default for BusyBox.
5. Runs all the inittab commands that block (action type: wait).
6. Runs all the inittab commands that run only once (action type: once).

- Once it has done this, the init routine loops forever carrying out the following tasks:



1. Runs all the inittab commands that have to be respawned (action type: respawn).
2. Runs all the inittab commands that have to be asked for first (action type: askfirst).

Create inittab file in etc

Each line in the inittab file follows this format:

id:runlevel:action:process

| | | |
|---|---|---|
| id | - | specify the tty for the process to be started. |
| runleven | - | completely ignores the runlevel field |
| action | - | can be any of the following: |

| Action | Effect |
|---|---|
| 1. sysinit | Provide init with the path to the initialization script. |
| 2. respawn | Restart the process every time it terminates. |
| 3. askfirst | Similar to respawn, but It prompts init to display "Please press Enter to activate this console." |
| 4. wait | Tell init that it has to wait for the process to complete before continuing. |
| 5. once | Run process only once without waiting for them. |
| 6. ctrlaltdel | Run process when the Ctrl-Alt-Delete key combination is pressed. |
| 7. shutdown | Run process when the system is shutting down. |
| 8. restart | Run process when init restarts. Usually, the process to be run here is init itself. |

*Sample inittab file for beaglebone:*

# cd <path-to-rootfs>/etc





# vim inittab

*Copy following commands into inittab file and save it:*

```
# Startup the system
null::sysinit:/etc/init.d/rcS
# Start getty on serial for login
ttyO0::respawn:/sbin/getty -L ttyO0 115200 vt100
# Stuff to do before rebooting
null::shutdown:/bin/umount -a -r
```

This inittab file does the following:

1. Sets **/etc/init.d/rcS** as the system initialization file.
2. Starts getty on serial port.
3. Tells init to run the umount command to unmount all filesystems it can at system shutdown

## Create profile file in etc

*profile* file has environment variables.

# vim profile

*Copy the following into profile file and save it:*

```
# Used for prompt format
PS1='[\u@\h:\W]\# '
PATH=$PATH
HOSTNAME=`/bin/hostname`
export HOSTNAME PS1 PATH
```

## Create passwd file

*passwd* file has user's password information.





# vim passwd

*Copy following command for veda user and save it:*

veda::0:0:root:/root:/bin/sh

- Create rcS file under /etc/init.d
    - This script can be quite elaborate and can actually call other scripts.
    - Use this script to set all the basic settings and initialize the various components of the system like Mount additional filesystems, Initialize and start networking interfaces.
    - Start system daemons.

*Sample rcS file for beaglebone:*

# mkdir init.d

# vim init.d/rcS

*Copy follwing commands into the rcS file and save it:*

```
#!/bin/sh
# ---------------------------------------
# Mounting procfs
# ---------------------------------------
mount -n -t proc null /proc
# ---------------------------------------
# Mounting sysfs
# ---------------------------------------
mount -n -t sysfs null /sys
# ---------------------------------------
# Mounting ramfs to /dev
# ---------------------------------------
mount -n -t ramfs null /dev
# ---------------------------------------
# Create /dev/shm directory and mount tmpfs to /dev/shm
```





```
# -----------------------------------------
mkdir -p /dev/shm
/bin/mount -n -t tmpfs tmpfs /dev/shm
# -----------------------------------------
# Enabling hot-plug
# -----------------------------------------
echo "/sbin/mdev" > /proc/sys/kernel/hotplug
# -----------------------------------------
# Start mdev
# -----------------------------------------
mdev -s
# -----------------------------------------
# Set PATH
# -----------------------------------------
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin
# -----------------------------------------
# Set ip address
# -----------------------------------------
/sbin/ifconfig lo 127.0.0.1 up
/sbin/ifconfig eth0 10.0.0.111 up
# -----------------------------------------
sleep 3
# -----------------------------------------
# Assigning hostname
# -----------------------------------------
/bin/hostname boneblack
# -----------------------------------------
```

*Give executable permissions to rcS script file:*

```
# chmod +x init.d/rcS
```





Populate dev

As part of inittab & rcs files as we are using "**null & ttyO0**" device nodes we need to create them manually.

*Create device nodes for beagle bone:*

```
# cd <path-to-rootfs>/dev
# mknod console c 5 1
# mknod null c 1 3
# mknod ttyO0 c 204 64
```

Populate lib

As we used buildroot for building cross-compiler copy all libs created by buildroot for target machine.

```
# cp -Rfp <path-to-buildroot-source>/output/host/usr/arm-buildroot-linux-uclibcgnueabi/lib/* <path-to-rootfs>/lib

# cp -Rfp <path-to-buildroot-source>/output/host/usr/arm-buildroot-linux-uclibcgnueabi/sysroot/lib/* <path-to-rootfs>/lib

# cp -Rfp <path-to-buildroot-source>/output/host/usr/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib/* <path-to-rootfs>/lib

# cp -Rfp <path-to-buildroot-source>/output/host/usr/lib/* <path-to-rootfs>/lib
```

- And copy other libraries cross-compiled for target machine.

Install kernel modules





- At the time of building kernel if ant service is selected as modules, than we neet to install those modules as part of target root file system.

- The following command will install kernel modules in target root file system

- Change directory to cross-compiled linux kernel and give following command:

```
# cd <path-to-linux-source>

# make ARCH=arm CROSS_COMPILE=arm-linux- modules

# make ARCH=arm CROSS_COMPILE=arm-linux-
INSTALL_MOD_PATH=<path-to-rootfs> modules_install
```