Third element: kernel

- Version numbers
- About “BSPs”
- Configuring and cross compiling
- Booting
Kernel vs user space

Your program

- Application code
- C library
- System interface
- Generic services
- Device drivers

User space

Linux kernel

Kernel space

Hardware
Kernel version numbers

Example: 2.6.35.1

2: very unlikely to change

6: unlikely to change
2.6.0 released in December 2003

35: changes with each release, every 12 weeks or so

1: bug fix number: changes every time a bug is fixed, sometimes several times per week
Bug fix releases

• Maintained by Greg Kroah-Hartman
• Serious bugs are fixed in the current stable version immediately
• Sometimes older versions are fixed as well
• Special note: the 2.6.27 and 2.6.32 stable kernels maintained by Adrian Bunk
  • Current releases (October 2010)
    - 2.6.27.54
    - 2.6.32.24
Board Support Packages

• Mainline kernel works out-of-the-box for a number of development boards
  • e.g. Beagleboard
• But in most cases you will need a BSP from the board or chip vendor
  • Lags mainline by a few versions
  • Levels of support vary between vendors
• For custom boards you will have to write your own BSP
Levels of board support

• Architecture
  • arm, mips, powerpc, x86, ...

• Chip (also known as System on Chip, SoC)
  • Atmel 91sam9, Freescale i.MX, TI OMAP, ...

• Board
  • SoC manufacturer evaluation boards
    - Freescale Babbage, TI EVM, ...
  • COTS boards
    - Digi, Eurotech, ...
Levels of board support (cont.)

• Chip level support mostly done by manufacturer
  • often in own kernel tree: e.g. Freescale
• Board level support done by board manufacturer
  • based on SoC kernel
Board support

• Usually a kernel patch and a configuration file
• Typical procedure is

    tar xjf linux-2.6.34.tar.bz2
    cd linux-2.6.34
    patch -p 1 < ../linux-2.6.34-some_bsp.patch
    cp ../some_bsp-kernel.config .config
    make oldconfig
Kernel modules

• Kernel code that is loaded after the kernel has booted

• Advantages
  • Load drivers on demand (e.g. for USB devices)
  • Load drivers later – speed up initial boot

• Disadvantages
  • Adds kernel version dependency to root file system
  • More files to manage
Kernel configuration

• Typical kernel has >> 1000 configuration options
• Default configuration part of the BSP
• Tweak configuration using
  • make menuconfig (ncurses text menu)
  • make xconfig (graphical menus using Qt)
  • make gconfig (graphical menus using Gtk+)
• Files generated
  • .config
  • include/linux/autoconf.h
Building the kernel

- Set CROSS_COMPILE and ARCH
  
  ```
  export ARCH=arm
  export CROSS_COMPILE=arm-angstrom-linux-gnueabi-
  ```

- Make targets
  - zImage - compressed kernel image
  - uImage - zImage plus U-Boot header

- Files generated
  - vmlinux
  - arch/arm/boot/zImage
  - arch/arm/boot/uImage
Kernel command line

- Kernel behaviour set by “command line”
  - see Documentation/kernel-parameters.txt
- Some examples

  console: device to send kernel messages to, e.g.
  \[ \text{console=/ttyS0,115200} \]

  root: set device to load root file system from, e.g.
  \[ \text{root=/dev/sda1} \]

  quiet: output fewer console messages

  debug: output all console messages
Fourth element: user space

- What is user space?
- Obtaining a root file system
- Busybox
- Two types of init: Busybox and System V
- Managing device nodes: udev
- Mounting a root file system over the network and from flash memory
What is user space?

• A sane (POSIX) environment for applications (unlike the kernel)

• The main components are
  • Programs – e.g. init and a shell
  • Libraries - e.g. libc
  • Configuration files in /etc
  • Device nodes in /dev
  • User data in /home
The root file system

- Mounted by the kernel during boot
  - requires a `root=`... kernel command line

- Loaded from:
  - ram disk (initramfs)
  - storage device: flash, SD, hard disk
  - network: nfs
“I got a rootfs with my board”

• As with the toolchain, this is usually a trap!
• Board vendors usually over-configure to show off the board
  • bloated root file system
  • slow boot
• ... yet, they only offer a limited set of packages
• and limited or no update service
Other options for a root file system

• Roll-Your-Own (RYO)
• Use an integrated build tool
  • Buildroot
  • OpenEmbedded
• Use a binary distro
  • Ångström
  • Ubuntu or Debian
Busybox

- Web - http://www.busybox.net
- Very common in embedded systems
- Single binary that masquerades as many Linux utilities, including
  - init
  - ash (a Bourne shell)
  - file system utilities: mount, umount,…
  - network utilities: ifconfig, route,…
  - and of course, the vi editor
Busybox example

```bash
# ls -l /bin
lrwxrwxrwx 1 root  root       7 2008-08-06 11:44 addgroup -> busybox
lrwxrwxrwx 1 root  root       7 2008-08-06 11:44 adduser -> busybox
lrwxrwxrwx 1 root  root       7 2008-08-06 11:44 ash -> busybox
-rwxr-xr-x 1 root  root  744480 2008-05-16 15:46 busybox
lrwxrwxrwx 1 root  root       7 2008-08-06 11:44 cat -> busybox
...
```

So when you type (for example)
```
cat /etc/inittab
```

... launches /bin/busybox with `argv [0] = "/bin/cat"`

Busybox main() parses `argv[0]` and jumps to cat applet
init

• /sbin/init is the first program to be run
  • change by setting kernel parameter “init=...”

• Two common versions of init
  • Busybox init
    – e.g. by buildroot
  • System V init
    – e.g. by Angstrom
Busybox init

- Begins by reading `/etc/inittab`, for example:

```
/etc/inittab

::sysinit:/etc/init.d/rcS
::respawn:-/sbin/getty -L ttyS0 115200 vt100
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::restart:/sbin/init
```

```
/etc/init.d/rcS

#!/bin/sh

echo "Starting rcS"

mount -t proc proc /proc

mount -t sysfs sysfs /sys

ifconfig lo 127.0.0.1

ifconfig eth0 192.168.1.101
```
System V init

• Also begins by reading `/etc/inittab`
  • More complex format than Busybox
• System V runlevels
  • A runlevel defines a system state
    – 0 is halt
    – 1 is single user
    – 2-5 are multi-user
    – 6 is reboot
System V inittab

Format:
id:runlevels:action:process

id:5:initdefault:

si::sysinit:/etc/init.d/rcS

~~:S:wait:/sbin/sulogin

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

z6:6:respawn:/sbin/sulogin
S:2345:respawn:/sbin/getty 38400 ttyS1

Default runlevel = 5
Boot script = /etc/init.d/rcS
Single-user mode: add 'S' to kernel command line
Scripts for each runlevel
Launch a login on the console
Initialisation scripts

- Each service is controlled by a script in /etc/init.d:

```
# ls /etc/init.d
alignment.sh       modutils.sh       sendsigs
banner              mountall.sh       single
bootmisc.sh         mountnfs.sh       syslog
checkroot           networking        syslog.busybox
devpts.sh           populate-volatile.sh  udev
dropbear            ramdisk           udev-cache
finish.sh           rc                 umountfs
functions           rcS                umountnfs.sh
halt                reboot             urandom
hostname.sh         rmnologin          
hwclock.sh          save-rtc.sh
```

- Most take parameters *start* and *stop*, e.g.

```
/etc/init.d/syslog stop
```
/dev: device nodes

• Most hardware appears as nodes in /dev

• Create by hand:
  
  mknod /dev/ttyS0 c 4 64

• Or, use a dynamic device manager (udev)

• udev pros
  
  • less hassle; handles removable devices (e.g. USB)

• udev cons
  
  • slow
The rootfs during development

• Advantages of mounting rootfs over NFS
  • easy to access and modify the rootfs
  • No limit on size

Step 1. Export a directory on the development host with a line like this in /etc/exports

  /home/chris/rootdir *(rw,sync,no_subtree_check,no_root_squash)

Step 2. Set kernel parameters

  root=/dev/nfs rw nfsroot=192.168.1.1:/home/chris/rootdir ip=192.168.1.101
# The rootfs in production

- Usually stored in a partition of flash memory

<table>
<thead>
<tr>
<th>Flash file system: jffs2, yaffs2, ubifs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTD (Memory Technology Devices)</td>
</tr>
<tr>
<td>Flash driver</td>
</tr>
<tr>
<td>Flash chip(s)</td>
</tr>
</tbody>
</table>

**Typical kernel parameters:**

```
root=/dev/mtdblock1 rootfstype=jffs2
```
Flash file systems

- jffs2 (Journalling Flash File System 2)
  - This is the most common Linux flash fs
  - Robust, but slow (especially mount time)
- yaffs2 (Yet Another Flash File System 2)
  - Optimised for NAND flash memory
  - Not part of main-line kernel
- ubifs (Unsorted Block Image File System)
  - Fast and robust
Summary

• Kernel
  • Your choice of kernel is limited by BSP
  • Many build-time kernel configuration options
  • Boot-time configuration via command line

• User space
  • Starts when kernel mounts rootfs
  • First program to run is (default) /sbin/init
  • Both Busybox init and System V init are common