Linux for ARM® Processors

www.arm.com

Summer/Winter 2013

Robert Boys
bob.boys@arm.com

System Design Division, ARM

V 1.0
We will discuss:

- ARM Roadmap: MMU or ! MMU – which processors to use.
- Linux distributions as pertains to Cortex™-A processors.
- Make your own or Buy/Get ?
- Documentation, resources on Internet.
- Lots of other stuff…
- Legal
We will not discuss:

- Cortex-M and R have no MMU so need a special Linux.
- µClinux: www.uclinux.org/ - no MMU is needed.
- UBUNTU – but can use it. UBUNTU works on ARM.
- UBUNTU is a good place to get GCC tools.
- Not much Debian or Fedora talked about here.
- No ABI (Application Binary Interface) or EABI in ARM….
- note…EABI with Debian owned by RedHat.
- Raspbian based on Debian for Raspberry Pi.
- No shell commands – ls or pwd or cd – get this elsewhere.
- C++.
Versions, cores and architectures?

- What is the difference between ARM7™ and ARMv7?
- Search for ARM architecture on Wikipedia to get the full list.
- ARM doesn’t make chips….well maybe a few test chips.

<table>
<thead>
<tr>
<th>Family</th>
<th>Architecture</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>ARMv4T</td>
<td>ARM7TDMI(S)</td>
</tr>
<tr>
<td>ARM9, ARM9E</td>
<td>ARMv5TE(J)</td>
<td>ARM926EJ-S, ARM966E-S</td>
</tr>
<tr>
<td>ARM11</td>
<td>ARMv6 (T2)</td>
<td>ARM1136(F), 1156T2(F)-S, 1176JZ(F), ARM11 MPCore™</td>
</tr>
<tr>
<td>Cortex-A</td>
<td>ARMv7-A</td>
<td>Cortex-A5, A7, A8, A9, A12, A15</td>
</tr>
<tr>
<td>Cortex-R</td>
<td>ARMv7-R</td>
<td>Cortex-R4(F), Cortex-R5, R7…</td>
</tr>
<tr>
<td>Cortex-M</td>
<td>ARMv7-M, ARMv6-M</td>
<td>Cortex-M3, M4, Cortex-M1, M0, M0+</td>
</tr>
<tr>
<td><strong>NEW !</strong></td>
<td>ARMv8-A</td>
<td>64 Bit: Cortex-A53 and A57</td>
</tr>
</tbody>
</table>
A BIG question:

- Buy Linux or RYO (roll your own)
- Many(most) board makers now provide a Linux distribution.
- Saves you a lot of time….
- RYO is very challenging…
- A BSP supplies instructions and software to compile the kernel.
- Companies like Timesys provide services. Linaro too.
- You can make desired modifications such as new Device Drivers.
- Multi-core processors: not the issue of getting many CPUs on chip : challenge is getting the software working efficiently…
About the Linux Kernel:

- Is the OS itself: the Kernel.
- Written in C. Compiled usually with GCC.
- Kernel is not so useful by itself....
- Needs infrastructure to start threads and processes.
- Applications that these threads and processes execute.
- Needs Device Drivers – to connect to various hardware devices.
- System services: support and communicates with devices and protocols.
- Runs applications: starts processes and threads.
- All stored in the Linux Filesystem. (FS) Use ls to see FS.
- Linux Distribution: a complete set of these components plus a mechanism for booting the system (usually U-Boot)
GCC GNU Compiler Collection.

- Needs binary utilities package “binutils” which provides...
- Gas (GNU assembler), linker/loader ld + other utilities.
- Can use the ARM compiler – but it needs the GCC libraries.
- Code produced with GCC is not considered “derivative work”.
- GNU C Compiler: compiles kernel, bootloader, +++
- For GCC for ARM see: http://elinux.org/ARMCompilers
- See scratchbox for a toolchain: www.scratchbox.org
- GCC needs glibc library. Produces quite large executables.
- For embedded other smaller packages available.
- Newlib (Cygwin), dietlibc, eglibc, uClibc and so on…
- Newlib allows you to remove FP if not needed.
- Has integer only printf(): is much smaller.
Where to get GCC?

- Cross-compiler and native.
- One runs on x86 and other on ARM processors.
- Quality is the same – compile time is probably an issue.
- CodeSourcery one source: Lite. Search for this.
- UBUNTU another source. Is actually a Linaro provided GCC.
  - `sudo apt-get install g++-arm-linux-gnueabi gcc-arm-linux-gnueabi`
  - `sudo apt-get install g++-4.7-arm-linux-gnueabi gcc-4.7-arm-linux-gnueabi`
  - V 4.6 and 4.7 are current at this time.
What about Cortex-M and Cortex-R?

- These can run a modified version of Linux called uClinux.
- Companies also offer such versions commercially.
- These processors (and ARM7 and some ARM9) have no MMU. Memory Management Unit.
- MMU converts logical addresses into physical addresses.
- See www.uclinux.org for more information, documentation, mailing lists and additional resources in addition to other Linux kernel mailing lists.
- Filesystem images can be produced using Buildroot environment. http://buildroot.uclibc.org/
Bootloaders

- Das U-Boot the most popular. But not for x86.
- Others for Android and others. GRUB desktops & servers.
- Source code available at git.denx.de
- Is first code run by CPU at RESET.
- Initialize hardware – power, clocks, memory controllers etc.
- In multi CPU runs on only 1 core – others in holding pattern.
- Reads kernel into RAM and set kernel parameters (bootargs)
- Starts kernel unless you stop it. “Press any key”.
- U-Boot is bare-metal – needs no OS.
- No MMU or Cache. This turned on by the Linux kernel.
- Has its own Command line interface.
- Is a multi-stage: as in 1st Stage Bootloader.
Debugging:

- Bootloaders: use JTAG or a model of your processor.
- Linux Kernel: use JTAG or a model.
- Linux applications: use a debugger with GDB Server.
- Many open-source and commercial tools available.
- Remember: Free is not Free – there is a cost somewhere.
- DS-5 provides models that boot Linux and can debug kernel.
- Is available to universities for teaching for free.
- Contains Fast Models.
Source Code Debugging:

- You want to debug with assembly correlated with C Source.
- elf format embeds debug info into executable code.
- Done at compile time: matches instructions with source code.
- Elf is usually stripped out and only executable loaded.
- Can sometimes often relate to the sources separately.
- But still need the executable with elf. gcc –g main.c
- Mixed mode. Can get this in trace too.
Heisenbug:

- Named after Heisenberg: he observed that an event can disappear when you look for it.
- Program bugs sometimes/often act just like this!
- People put items in their code to tell them some information.
  - i.e. printf statements.
- Changes code – a Heisenbug could happen and goes away.
- Sometimes new bug appear out of nowhere. Ghosts?
- These are very, very tough problems to solve.
- ARM CoreSight debugging modules try to help.
- No code stubs, not intrusive (almost always) and realtime.
- Hardware breakpoints break before instruction executed.
ETB, ETM and PTM Trace:

- This can also be mixed: plain assembly is hard to figure out.
- A record of where your program has been.
- This is why you need a general understanding of assembly.
Linux Versions

- Older 2.4, 2.6 etc. Might still see these around.
- Current is 3 series. 3.10.5 is current…. 3.11 soon.
- See http://kernel.org/ for the latest data.
- No sense in using older versions. Not supported.
- **Long Term Support Linux:** older, stable version supported. Improvements are backported. 3.10 is the latest to be picked.
- Security issues important. Don’t want to get hacked…do you?
- Everything connected to the Internet is in danger….
- Later versions have the most recent security features/fixes.
- See http://lwn.net/Security/ lwn.net is subscription <cheap>
Other Interesting Versions of Linux…

- Automotive Grade Linux. http://automotive.linuxfoundation.org/
- Real-Time Linux: RTLinux.
- Linux RTOS microkernel is fully pre-emptive process.
  https://rt.wiki.kernel.org/
- There are many other special purpose distributions:
  See http://elinux.org/Embedded_Linux_Distributions
Do Various Versions Cause Troubles?

- Versions not the problem as in the past.
- Troubles during transition from 2.x to 3.x.
- “Out of tree” modules can change at any time.
- Better to use modules in the tree. (GIT)
- Things in User Space – calls and so on – don’t change.
- I/O controls, system calls.
- Things like graphics “atomic page flip” more problematic.
What does ARM do for Linux?

- Supports Linaro. www.linaro.org
- Big 3 company contributors are Red Hat, Linaro and Intel.
- Number 1 is volunteers. See http://lwn.net/Articles/555867/
- www.arm.com/linux and www.linux-arm.org
- Makes sure GCC works with ARM processors.
- ARM does not provide generic Linux support.
The Linux Kernel: the Heart:

- Responsible for memory allocation, process and thread creation, scheduling, filesystem communication, peripherals.
- Kernel is highly configurable.
- It is a good idea to configure the kernel to keep it small.
- Speed vs feature set balance is important.
- Embedded system probably needs no SATA drive.
- Profiling is important.
- ARM DS-5, PERF, gprof and others.
- Cortex-A processors have hardware counters to collect run-time information.
menuconfig

- make ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- menuconfig
- Activate the Profiling Support option in General Setup: CONFIG_PROFILING=y.
- Under Kernel hacking, activate the Tracers and Trace process context switches and events options:
  - CONFIG_FTRACE=y and
  - CONFIG_ENABLE_DEFAULT_TRACERS=y.
Kernel Configuration:

- Debug notifier call chains
- Debug credential management
- Delay each boot printk message by N milliseconds
- Torque tests for RCU
- RCU CPU stall timeout in seconds
- Print additional per-task information for RCU_CPUSTALL_DETECTOR
- Self test for the backtrace code
- Force extended block device numbers and spread them
- Force weak per-cpu definitions
- Debug access to per_cpu maps
- Fault-injection framework
- Sysctl checks
- Debug page memory allocations
- Tracers
  - Enable debugging of DMA-API usage
  - Perform an atomic64_t self-test at boot
  - Sample kernel code
  - KGBDB: kernel debugger
  - Test kstrto*() family of functions at runtime
  - Filter access to /dev/mem
- Enable stack unwinding support (EXPERIMENTAL)
- Verbos user fault messages
- Kernel low-level debugging functions
- Early printk
- Kernel low-level debugging via EmbeddedICE DCC channel
- On-chip ETM and ETB
More Kernel Hacking:

- Debug notifier call chains
- Debug credential management
- Delay each boot printk message by N milliseconds
- torture tests for RCU
- (60) RCU CPU stall timeout in seconds
- Print additional per-task information for RCU_CPUSTALL_DETECTOR
- Self test for the backtrace code
- Force extended block device numbers and spread them
- Force weak per-cpu definitions
- Debug access to per_cpu maps
- Fault-injection framework
- Sysctl checks
- Debug page memory allocations
- Tracers --->

- Enable debugging of DMA-API usage
- Perform an atomic64_t self-test at boot
- Sample kernel code --->
- KGDB: kernel debugger --->
- Test kstrto() family of functions at runtime
- Filter access to /dev/mem
- Enable stack unwinding support (EXPERIMENTAL)
- Verbose user fault messages
- Kernel low-level debugging functions
- Early printk
- Kernel low-level debugging via EmbeddedICE DCC channel
- On-chip ETM and ETB
A bit more:

---

**Tickless System (Dynamic Ticks)**

* High Resolution Timer Support
* Symmetric Multi-Processing
  * Allow booting SMP kernel on uniprocessor systems (EXPERIMENTAL)
  * Memory split (3G/1G user/kernel split) --->
* Maximum number of CPUs (2-32)
* Support for hot-pluggable CPUs (EXPERIMENTAL)
* Use local timer interrupts
  * Preemption Model (Preemptible Kernel (Low-Latency Desktop)) --->
* Compile the kernel in Thumb-2 mode (EXPERIMENTAL)
* Use the ARM EABI to compile the kernel
* Allow old ABI binaries to run with this kernel (EXPERIMENTAL)
* High Memory Support
  * Allocate 2nd-level pagetables from highmem
* Enable hardware performance counter support for perf events
  * Memory model (Flat Memory) --->
* Allow for memory compaction
* Enable KSM for page merging
* Low address space to protect from user allocation
* Enable cleancache driver to cache clean pages if tmem is present
* Use kernel mem{cpy,set}() for {copy_to,clear} user() (EXPERIMENTAL)
* Enable seccomp to safely compute untrusted bytecode
* Enable -fstack-protector buffer overflow detection (EXPERIMENTAL)
* Provide old way to pass kernel parameters

---
...and the last one I have...
Profiling:

- Shows what functions used the most time. Work on these first.
- Can also show which CPU ran which threads.
Trace Profiling:

- A fast Profiler:
- Note mixed C and Assembly code:
- This is ETB. Needs a JTAG adapter.
- ETM and PTM also available with DSTREAM.
- Does the same thing but many more frames.
User and Kernel Space:

- A process or threads is in Kernel or User space.
- User: has normal privilege – can’t do everything.
- Kernel: can do anything.
- Switch with system calls.
- There are exceptions and mixed cases.
Processes and Threads:

- These are about the same thing to the kernel.
- Often both are called Tasks.
- Threads share virtual memory space. Processes do not.
- Threads share file descriptors: Processes do not.
- Threads share filesystem context: Processes do not.
- Threads share signal handling: Processes do not.
- Both created with clone() system call.
- Clone() in practice is called with Pthread_create() .
- Pthread_create() creates a new thread at a higher level.
- Context switches for processes are fast. Threads are faster.
- Another says use fork and exec.
Process ID (PID)

- Process is assigned a Process ID (PID).
- Can find with the ps command. Process Status.
- `ps -e | grep gatord`
- Can use PID to KILL a process. `kill PID`

```bash
# ps USER TIME COMMAND
PID   USER  TIME    COMMAND
 1 root 0:01 init
 2 root 0:00 [kthreadd]
 3 root 0:00 [ksoftirqd/0]
 4 root 0:00 [watchdog/0]
 5 root 0:00 [events/0]
 6 root 0:00 [khelper]
 9 root 0:00 [async/mgr]
10 root 0:00 [pm]
3065 root 0:00 [usb_wakeup thre]
3069 root 0:00 [usb_wakeup thre]
3072 root 0:00 [sync_supers]
3091 matrix 0:00 [kbd]
3097 matrix 0:00 [events/0]
3099 matrix 0:00 [khelper]
3065 root 0:00 [async/mgr]
3069 root 0:00 [usb_wakeup thre]
3106 matrix 0:00 [matchbox-desktop]
3115 matrix 0:00 [matchbox-window]
3116 matrix 0:00 [matchbox-panel]
3117 matrix 0:00 [matchbox-window]
3120 matrix 0:00 [matchbox-desktop]
3121 matrix 0:00 [matchbox-window]
3122 matrix 0:00 [matchbox-panel]
3130 root 0:00 [atd]
3133 root 0:00 [khubd]
3204 root 0:00 [ps -e]
```
Device Drivers:

- They can be:
  - compiled into the Kernel… or
  - Dynamically loaded by demand with a loadable kernel module (LKM)
- Many consider device drivers as part of the kernel.
- LKMs are stored outside the kernel in the filesystem.
- Have a .ko extensions (kernel object) i.e. gator.ko
- The gator daemon used by ARM DS-5 for Profiling.
- Search Arm DS-5 gator for example how to compile this.
- Instructions on building gator and kernel are in the DS-5 docs.
- Configure the kernel with kconfig. Search for it for more info.
More on Devices:

- Linux accesses device files as device nodes in /dev
- Devices can be static or dynamic.
- Embedded can use pre-created static devices – then no udev or associated resources are needed.
- If plug-in devices used i.e. USB drives, udev is useful.
- udev manages devices in /dev.
- udev runs as a daemon and listens for uevents from kernel.
- uevents tells if a new device is initialized or removed.
- udev has persistent naming.
- Order the devices plugged in then doesn’t matter.
Kernel Boot Options

- Instead of continuously rebuilding the kernel:
- Change options during boot time.
- Override default option.
- Supply arguments to the bootloader or kernel when loaded into memory.
Filesystem (FS): Introduction

- Used to store data, kernel, many other things.
- FS must be **mounted** in order to be accessed.
- Mount and umount (not unmount)
- Mounted at mount point – a directory associated with FS.
- Is where your FS is grafted onto the Linux tree. /dev/sda9
- You can create a different directory for this.
- In embedded systems, FS is usually a SD card or serial flash.
- Is memory volatile or non-volatile?
- Embedded systems need both.
- Linux FS often needs to be written to.
- Usually NAND Flash (not NOR) is used.
- Wear Leveling is necessary to prevent early flash failure.
More on FS:

- Usually, FS from NAND Flash is copied into processor RAM.

- initrd: Initial RAMdisk
  - Pre-created fixed size FS image. Can be compressed
  - Copied into RAM to be executed.
  - initrd is not persistent – changes are not written back. Lost at pwr off.

- initramfs: Initial RAM filesystem
  - Created by extracting a compressed file in RAM used by the kernel.
  - initramfs is most common as can easily grow in size as needed.
  - Is very fast.
  - Provided as an archive file and put into a temporary RAM FS.
  - Then the boot process starts.
FS Types and Formats:

- Most common are ext2, ext3 and ext4.
- Most embedded designs use NAND flash SD cards or serial Flash (SPI and QSPI).
- ext2: early for Linux
- ext4: adds features for higher efficiency at larger sizes.
- FAT: Microsoft. FAT12, 16, 32. No permissions possible.
- FAT has advantage can make SD cards with Windows.
- There are others: cramfs, jffs2, ubifs, etc.
- For others a Linux computer (UBUNTU etc) is easiest.
- Copy Linux using dd command.
  - `dd if=bb.image of=/dev/sdb bs=1M` (is much faster)
/proc

- Information about internal Linux processes.
- CPUINFO
- modules, version
- Are text files you can easily read. `cat filename`
- See also /dev/kmem for access to the kernel memory.
Linux Initialization and Startup Process

- At RESET, a boot program (such as U-Boot) must start.
- Bootloader is Bare Metal – has no OS.
- Initializes hardware of the processor:
  - Initial Stack Pointer, MMU off, caches on or off, etc.
  - Only one CPU does this: the others will be stopped.
  - Loads/uncompress the Linux kernel into target RAM.
  - Bootloader starts Linux kernel.
  - Page Table (MMU mapping) are built by the kernel soon after it starts.
  - Kernel finishes hardware init and mounts/loads any RAM filesystem.
  - Uses initrd or initd file to execute some scripts.
Here is an example of what the kernel does:
- Is a file `rcS` inside `init.d`
- From iMx53 Linux.
- There are other filenames.
/init and /linuxrc

- IF RAM disk or FS is found or specified in initrd:
- 1) Mount it and execute a ini file provided in RAM disk or FS.
- IF Initial RAM FS: executes shell script /init
- IF Initial RAM disk: executes /linuxrc
- These can also be a version of /sbin/init on next slide.
Linux Initialization Steps (if no initial RAM found)

- Kernel gets loaded as described and run-time FS is mounted.
- Executes /sbin/init script which executes /etc/inittab.
- This determines how the kernel should boot.
  - `id:<run level>:<action>:<process>`
  - `Id` = TTY process should run on – if nothing then /dev/console
  - `runlevel` = Linux runlevel to be used. Often not used.
  - `action`: e.g. respawn – starts the process and restarts if it exits.
  - `Process` = run this.
- 3 examples shown here:
  - `rcS` = START.
  - `rcK` = KILL.
Linux FileSystem Contents:

- The FS contains programs, libraries, scripts and data files.
- In Embedded apps – must keep FS small to conserve space.
- Do you really need the man pages? All the sources?
- The gcc compiler, assembler and linker.
- Simplification is also a desirable target.
- This makes it easier to debug and maintain.
- Linux sources in /usr/src/linux – often left out to save space.
BusyBox:

- A binary program that performs functions of most GNU system.
- If not using udev, BusyBox provides standard device nodes.
- Very popular.
- Examples are bz2, compress, uuencode, fdisk, gz, chmod, fdisk, gzip, vi and many more.
- www.busybox.net/
Linux Libraries:

- These are collections of precompiled code.
- Usually no source code is provided – no debug with symbols.
- Different types available.

1) STATIC: simplest. Linked into applications at compile time.
   - .a (archive) or .sa (static archive) extension
   - `gcc program.o -llib2 -wl -Bstatic -lpost`  <********!!!>
   - Easy to implement but result in larger code sizes. Legal issues.

2) Shared Libraries: ARM uses these in our examples.
   - `libgames-support.so` for DS-5 gnometris example. (shared object)
   - Application links at runtime on demand. Pre-loaded in RAM.
   - You must point to their location so app can find it.
   - Apps smaller because so outside. Easier to maintain.
   - Memory requirements are smaller for embedded systems.
   - A C library is usually a shared library.
Shared Libraries

2) Shared Libraries: ARM uses these in our examples.
   - libgames-support.so for DS-5 gnometris example. (shared object)
   - Application links at runtime on demand. Pre-loaded in RAM.
   - You must point to their location so app can find it.
   - Apps smaller because so outside. Easier to maintain.
   - Memory requirements are smaller for embedded systems.
   - A C library is usually a shared library.

The point of shared libraries is that two processes can use the same shared library and only one copy (at most) of the library code needs to be in memory.
More on Libraries:

3) Dynamically Linked Libraries: App load this when needed.

- Can reference this at any time.
- No pre-loading is necessary: is deferred until actually needed.
- Use a standard API to open one.
- This API is defined in system header dlfcn.h
- Use functions `dlopen()` and `dlclose()`.
- `dlsym()`, looks up the value of a symbol in an opened library.
- Store DLLs in directory listed in LD_LIBRARY_PATH environ variable
- Or in `/lib` or `/usr/lib`.
- DLLs are easier to develop, use and maintain.
- minimizing both application size and system overhead are important.
Licensing

- I am not a lawyer and this is not legal advice of any kind.
- Some things you should look into for commercial products:
  - **GPL**: complicated license. Must return your changes and improvements to the community. Must provide source code. Statically linked libraries not good dynamically linked OK.
  - **Lesser GPL**: permits use of a library in proprietary programs.
  - **BSD**: Berkeley Software Distribution: essentially free.
  - **MIT**: [http://opensource.org/licenses/mit-license.php](http://opensource.org/licenses/mit-license.php)
  - **Apache**: somewhat like BSD – For details see: [www.apache.org/licenses/LICENSE-2.0.html](http://www.apache.org/licenses/LICENSE-2.0.html)
  - **Public Domain**: no owner. Do with it as you like.
  - Many companies have or hire a lawyer(s) to advise them.
  - Lots of info on Internet about this…but be careful!
If you change/improve a GPL licensed source code – you have to provide it **in source form** back to the community.

You might have to provide source to your proprietary program.

Oops…..

If you use GPL source code in your sources…maybe a problem?

This is why Dynamically Linked is better than Static Linked.

Static: source is linked/compiled into your executable.

Dynamic: aka Shared Libraries (.so)

Loaded only when needed. Only a reference to .so is used.

Your executable does not contain any GPL code.
Statically Linked Libraries

- Archives: a collection of object files stored as 1 file.
- .a extension. libc_nomalloc.a
- Object file is extracted by linker and inserted into executable.
- You can create them:
  - ar cr libtest.a example1.0 example2.o
- You probably have to have source available to everybody.
Shared Libraries:

- aka Dynamically Linked Library.
- Is not a collection of files as such – is a single object file.
- Has a .so extension. i.e. libdbus.so
- The executable does not contain any code from .so
- Executable only includes a reference to the .so.
- ldd command displays .so linked to executable.
- You can make these too:
  - First compile it:
    - gcc –c –fPIC example1.c
  - Then make it:
    - gcc –shared –fPIC -0 libexample.so example1.o example2.o
- PIC = Position independent Code. Can run at any address.
LD_LIBRARY_PATH

- This is seen in embedded systems.
- Linker does not provide the full path to .so, only the filename.
- Searches /lib and then /usr/lib
- Can use –rpath, /usr/local/lib OR BETTER:
- set LD_LIBRARY_PATH /usr/bob; /tmp; /ARM/lib
Samsung Galaxy S2

- Samsung has included a huge amount of copyright notices.
- `/system/lib/libwwwscriptcore.so` from Apple
- `/system/lib/libsonivox.so` Apache licence
- `/system/lib/libca.a, lib.c.so, libc_common.a` Android
- Arc4 random number generator OpenBSD
- `/system/bin/dbus-daemon.so` D-Bus Academic Free License
- Appears to have the entire GPL license listed. All of it…
- `/modem` Has nine copyright holders.
- `/system/lib/libft2.a` FreeType Project
Starting Applications at Linux Startup:

- Useful to start an application when Linux finishes booting.
- Put a command line or two at the end of a script file.
- RAM-based systems: /linuxrc or /init
- BusyBox or SysVinit:
  - /etc/rc.local – is run after all scripts in /etc/init.d have executed.
  - Existing init script: just add to it.
  - A custom init script located in: /etc/init.d/
- If you modify a script that comes with your root filesystem, you will need to make this change in the Linux distribution. Else it will be lost when you reboot.
- Use “&” to allow app to run in background and allow the command to complete. Else you won’t get the prompt back.
- An example is: this starts gatord: /usr/sbin/gatord &
Some things that need Attention:

- **GIT**: a program created by Linus Torvalds.
  - A repository for Linux sources. Used for other programs now.
  - Version control and source code management (SCM) system.
  - [http://gitscm.com/](http://gitscm.com/)  
    See also [http://subversion.apache.org/](http://subversion.apache.org/)
  - Free open source software with a GPL license.
QEMU and other models:

- QEMU: http://wiki.qemu.org/
- A model of a system.
- Real chips are available early and low cost: not so much need for models as in the past.
- Still good for before chips are available. i.e. ARM V8 64 bit
- Usually use sophisticated ($$$) models like Fast Models.
- Xilinx had QEMU but ZedBoard is low cost and available.
- It all depends on where on the “curve” you work at.
- ARM has Fast Models: provided in DS-5.
- Work fast but are not cycle accurate.
Vmlinux and zImage

- vmlinux is the executable kernel. Vmlinuz is kernel zipped.
- Must be made bootable – add files plus setup routines.
- This bootable image is called vmlinuz or zImage.
- Might contain debugging info to allow source debugging.
- Usually in / or /boot. See below: This ZedBoard SD contents.
- readelf –h vmlinux
- Note ramdisk8m: is FS to be copied to target RAM.
- devicetree.dtb? Next slide…
**devicetree.dtb**

- After Linux kernel uncompressed and copied to RAM:
  - It is started and is a bare-metal program at this point.
  - It must setup the processor and a lot of other stuff.
- `devicetree.dtb` tells it what to do. “d” = binary file
- `devicetree.dts` is the source code. “s” = source code
- Here is a sample (lots of non-printables cut out):

```
Xilinx Zynq ZC770 - V&C dt1 - minimum) xlnx,zynq-zc770- xm010
memory=memorychosenh
Mconsole=ttyPS0,115200
root=/dev/ram rw initrd=0x800000,8M earlyprintk maxcpus=2 debug
ip=:eth0:dhcpV/amba@0/uart@E0001000amba@0
simple-bushintc@f8f01000arm,gic
uart@e0001000xlnx,ps7-uart-1.00.a timer@0xf8001000
xlnx,ps7-ttc- 1.00.a eth@e000b000xlnx,ps7-ethernet-1.00.a
phy@0marvell,88e1510=ethernet-phyI
sdhci@e0100000
```
Secure Shell (SSH)

- Usually used to access shell account in Linux.
- Uses a server-client format. Both client and servers available.
- Replaces not very secure telnet. SSH uses cryptography.
- **OpenSSH (OpenBSD Secure Shell)**
  - **Dropbear**: MIT license (is free)
  - SFTP is the SSH file transfer and is very common.
  - **sftp** = Secure File Transfer Protocol
  - Many debuggers (including ARM DS-5) want this.
  - To connect to the processor(s) for communications.
Poweroff?

- Most has been about starting up Linux.
- BUT! Powering a system down is important:
- If just shut power off – some things might not be saved…
- Like on a SD card … Must unmount a device like SD card.
- So – use poweroff command to start a proper shutdown.
- Also: see shutdown command.
- If your system seems to be missing something or not working the same and all you do is pull out the power plug: try shutting it down in a more orderly fashion.
Operating Systems

- Linux
- WinCE
- Android
- Micrium, ExpressLogic, Quadros, QNX and so on.
- Big differentiator is the MMU.
- Linaro – an open source community supported by ARM.
- FreeRTOS is widely used.
- Keil RTX has BSD license now – means this it is free.
- CMSIS – header files and startup files
- CMSIS-DSP – DSP libraries
- CMSIS-RTOS – for the RTOS market.
Linux Support for the ARM Architecture

Linux is an open source operating system running on all major processor architectures, including ARM processors. It enjoys support by a large group of engineers contributing back into the open source (similar process to the FSF’s GNU tools). This makes Linux a very dynamic and fast moving operating system.

Overview

The 3.x upstream Linux kernel include support for ARM Development Boards, including support for the following ARM processors:

- ARM720T*, ARM920T*, ARM922T*, ARM926EJ-S
- ARM1136JF-S, ARM1176JZF-S, ARM11 MPCore
- Cortex-A8, Cortex-A9, Cortex-A5, Cortex-A7, Cortex-A15

ARMv8 support for Linux

Linux kernel support for AArch64 has now been pulled into the Linux 3.7 kernel version. Instructions on how to run Linux on ARMv8 model are available from Linaro ARMv8 engineering wiki page.
Linaro is a not-for-profit engineering organization consolidating and optimizing open source Linux software and tools for the ARM architecture. Find out more...

Linaro Networking Group (LNG)
Accelerating Linux ARM networking ecosystem development

Linaro 13.07 Released
The Linaro 13.07 release is now available for download! The 13.07 Linaro release highlights the collaborative efforts of all the Linaro Teams Working Groups, Landing... See More
What is there to do?

- Converting GCC to ARM Compiler. Many people ask for this.
- Benchmark development (and debunking)
- Speed and codesize. Lots of ways to “game” a benchmark.
- Security improvements.
END