L4 Development

Ben Leslie

Ben.Leslie@nicta.com.au

Embedded, Real-Time and Operating Systems Program
National ICT Australia

August 2004

©National ICT Australia, 2004

OVERVIEW

- Developments tools
- Configuring and building the L4 kernel
- L4 booting and startup
- The L4 kernel debugger
- L4 "Hello Word"

 ©National ICT Australia, 2004

OVERVIEW

- Developments tools
- Configuring and building the L4 kernel
- L4 booting and startup
- The L4 kernel debugger
- L4 "Hello Word"

Presented as an extended worked example.
Demonstration presented using slides, emacs, terminal and simulator.

©National ICT Australia, 2004

NATIONAL ICT AUSTRALIA LIMITED

IMPLEMENTATION LANGUAGES

L4 Pistachio:

Linux:

Iguana:

IMPLEMENTATION LANGUAGES

L4 Pistachio:

- ➜ C, C++ and assembly.

Linux:

Iguana:

©National ICT Australia, 2004

NATIONAL ICT AUSTRALIA LIMITED

IMPLEMENTATION LANGUAGES

L4 Pistachio:

- ➜ C, C++ and assembly.

Linux:

- ➜ C (with GNU extensions) and assembly.

Iguana:

©National ICT Australia, 2004

IMPLEMENTATION LANGUAGES

L4 Pistachio:

- ➜ C, C++ and assembly.

Linux:

- ➜ C (with GNU extensions) and assembly.

Iguana:

- ➜ C (aims towards ANSI C99 compliance).
- ➜ IDL and other DSLs which produce C.

©National ICT Australia, 2004

TOOLCHAINS

gcc (v3.3.3) — http://gcc.gnu.org

binutils (v2.15) — http://sources.redhat.com/binutils

 ©National ICT Australia, 2004

TOOLCHAINS

gcc (v3.3.3) — http://gcc.gnu.org

- ➜ C compiler (gcc)
- ➜ C++ compiler (g++)

binutils (v2.15) — http://sources.redhat.com/binutils

©National ICT Australia, 2004

TOOLCHAINS

gcc (v3.3.3) — http://gcc.gnu.org

- ➜ C compiler (gcc)
- ➜ C++ compiler (g++)

binutils (v2.15) — http://sources.redhat.com/binutils

- ➜ Assembler (gas)
- ➜ Linker (ld)
- ➜ Disassembler (objdump)
- ➜ Others (addr2line, ar, c++filt, gprof, nlmconv, nlm, nm, objcopy, ranlib, readelf, size, strings, strip, windres)

 ©National ICT Australia, 2004

TOOLCHAINS

gcc (v3.3.3) — http://gcc.gnu.org

- ➜ C compiler (gcc)
- ➜ C++ compiler (g++)

binutils (v2.15) — http://sources.redhat.com/binutils

- ➜ Assembler (gas)
- ➜ Linker (ld)
- ➜ Disassembler (objdump)
- ➜ Others (addr2line, ar, c++filt, gprof, nlmconv, nlm, nm, objcopy, ranlib, readelf, size, strings, strip, windres)

Compiler target: arm-linux

©National ICT Australia, 2004

BUILDING A TOOLCHAIN

Cross-compilers are often difficult to build:

➜ glibc

➜ kernel headers

➜ manual hacking of Makefiles

 ©National ICT Australia, 2004

BUILDING A TOOLCHAIN

Cross-compilers are often difficult to build:

- ➜ glibc
- ➜ kernel headers
- ➜ manual hacking of Makefiles

Crosstool — http://kegel.com/crosstool

An automated tool for creating cross-compiler toolchains

- ➜ Scripts to download and compile toolchain
- ➜ Applies additional patches
- ➜ Takes a long time, *but* manual intervention not required (usually)

©National ICT Australia, 2004

CROSSTOOL

Currently using 0.28-rc32.

① Download crosstool –

`$ wget http://kegel.com/crosstool/crosstool-0.28-rc32.tar.gz`

② Extract – `$ tar zxvf crosstool-0.28-rc32.tar.gz`

③ Edit `demo-arm-softfloat.sh` – Uncomment gcc-3.3.3 line

④ Edit `arm-softfloat.dat` – Set TARGET=arm-linux

⑤ Create build dir – `$ sudo mkdir /opt; sudo chown $USER /opt`

⑥ Run tool – `$ sh demo-arm-softfloat.sh`

CROSSTOOL

Currently using 0.28-rc32.

① Download crosstool –

`$ wget http://kegel.com/crosstool/crosstool-0.28-rc32.tar.gz`

② Extract – `$ tar zxvf crosstool-0.28-rc32.tar.gz`

③ Edit `demo-arm-softfloat.sh` – Uncomment gcc-3.3.3 line

④ Edit `arm-softfloat.dat` – Set TARGET=arm-linux

⑤ Create build dir – `$ sudo mkdir /opt; sudo chown $USER /opt`

⑥ Run tool – `$ sh demo-arm-softfloat.sh`

⑦ Get coffee – it takes a couple of hours to build!

©National ICT Australia, 2004

CROSSTOOL

Currently using 0.28-rc32.

① Download crosstool –

`$ wget http://kegel.com/crosstool/crosstool-0.28-rc32.tar.gz`

② Extract – `$ tar zxvf crosstool-0.28-rc32.tar.gz`

③ Edit `demo-arm-softfloat.sh` – Uncomment gcc-3.3.3 line

④ Edit `arm-softfloat.dat` – Set TARGET=arm-linux

⑤ Create build dir – `$ sudo mkdir /opt; sudo chown $USER /opt`

⑥ Run tool – `$ sh demo-arm-softfloat.sh`

⑦ Get coffee – it takes a couple of hours to build!

You should now have an arm-linux C and C++ toolchain.

 ©National ICT Australia, 2004

OTHER TOOLS

Compiling L4 requires common UNIX tools:

- ➜ make
- ➜ perl
- ➜ python

Test building on Linux, Mac OSX, FreeBSD. (Should work under Cygwin).

 ©National ICT Australia, 2004

CONFIGURING AND BUILDING L4

Create a build directory:

```
$ cd pistachio/kernel
$ make BUILDDIR=/full/path/to/build
$ cd /full/path/to/build
$ make menuconfig
```

 ©National ICT Australia, 2004

CONFIGURING AND BUILDING L4

Create a build directory:

```
$ cd pistachio/kernel
$ make BUILDDIR=/full/path/to/build
$ cd /full/path/to/build
$ make menuconfig
```

Now configure for your platform.

©National ICT Australia, 2004

CONFIGURING AND BUILDING L4

Create a build directory:

```
$ cd pistachio/kernel
$ make BUILDDIR=/full/path/to/build
$ cd /full/path/to/build
$ make menuconfig
```

Now configure for your platform.

Build:

```
$ make
```


©National ICT Australia, 2004

Configuring and building L4

Create a build directory:

```
$ cd pistachio/kernel
$ make BUILDDIR=/full/path/to/build
$ cd /full/path/to/build
$ make menuconfig
```

Now configure for your platform.

Build:

```
$ make
```

Should create an `arm-kernel` ELF binary.

 ©National ICT Australia, 2004

BUILDING THE EASY WAY — PRE-CANNED CONFIGS

Building:

```
$ ./tools/autobuild sa1100
```

Autobuild tool uses preconfigured settings. Kernel built in `build/sa1100/kernel`

 ©National ICT Australia, 2004

NATIONAL ICT AUSTRALIA LIMITED

BUILDING THE EASY WAY — PRE-CANNED CONFIGS

Building:

```
$ ./tools/autobuild sa1100
```

Autobuild tool uses preconfigured settings. Kernel built in `build/sa1100/kernel`

Creating a pre-canned config:

① Configure with `make menuconfig`

② `$ ./tools/createautobuild /full/path/to/build config-name`

©National ICT Australia, 2004

RECAP

- Compiler and tool-chain
- Building and configured L4

 ©National ICT Australia, 2004

RECAP

- Compiler and tool-chain
- Building and configured L4

Not bootable yet – need a σ_0 and root task.

 ©National ICT Australia, 2004

BOOT SEQUENCE AND LOADING

©National ICT Australia, 2004

BUILDING THE BOOTIMAGE

Bootloaders generally load a single binary image.

This poses a problem for us, since we need to load multiple binaries:

- ➜ L4 kernel
- ➜ σ_0
- ➜ Root task
- ➜ Other applications

Also need to initialise the kernel configuration page (KCP).

Solution:

 ©National ICT Australia, 2004

BUILDING THE BOOTIMAGE

Bootloaders generally load a single binary image.

This poses a problem for us, since we need to load multiple binaries:

- ➜ L4 kernel
- ➜ σ_0
- ➜ Root task
- ➜ Other applications

Also need to initialise the kernel configuration page (KCP).

Solution: dite

- ➜ Combine multiple binaries into one binary
- ➜ Setup KCP and bootinfo

©National ICT Australia, 2004

DITE EXAMPLE

```
$ dite -n -B --binfo-sexec \
       -k -K 0xf0010000 -p arm-kernel \
       -s sigma0 \
       -i iguana \
       -o bootimage
```

- -n — Don't generate a dit header (historical)
- -B — Generate bootinfo structure
- –binfo-sexec — Generate simple exec headers
- -k — Next file listed is the kernel
- -K — The address of the Kernel Configuration Page
- -p — Rewrite physical address
- -s — Next file listed is sigma0
- -i — Next file listed is roottask
- -o — Output file

 ©National ICT Australia, 2004

What does it do?

Create a new ELF file containing all the program headers and data, but just one ELF header.

Adds a new program section containing the bootinfo data.

©National ICT Australia, 2004

What does it do?

Create a new ELF file containing all the program headers and data, but just one ELF header.

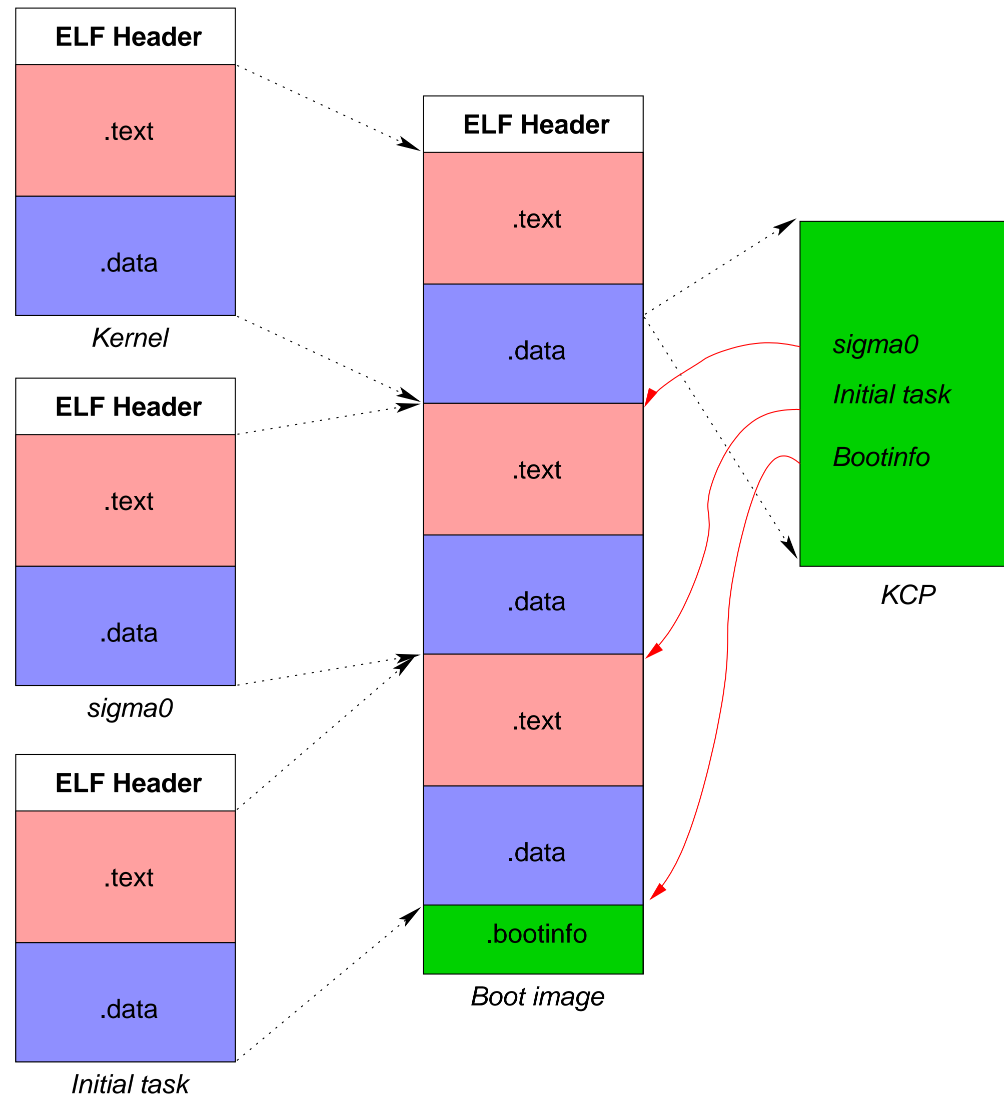
Adds a new program section containing the bootinfo data.

```
$ dite -d bootimage
```

➜ -d — Dump the dite file.

©National ICT Australia, 2004

BOOTIMAGE



©National ICT Australia, 2004

LOADING

We have a valid binary with the kernel as the entry point.

Loading the binary is very platform specific, but basically requires:

- ➜ Copying sections into memory at correct locations. Note that code needs to be in executable memory, data needs to be in read/write memory).
- ➜ Jump to the entry point.

©National ICT Australia, 2004

KERNEL STARTUP

- `kernel/src/arch/arm/head.S:_start()` – Starts in privileged mode with translation disabled
- `kernel/src/glue/v4-arm/init.cc:startup_system()` – Initialise hardware
  - CPU `init_cpu()`
  - Memory `init_memory()`
  - Platform specific `init_platform()`, `init_console()`
- Say hello! `init_hello()`
- Setup kernel data structures `init_kernel_space()`, `init_kip()`, `init_mdb()`
- Initialise interrupts and timers
- Initialise scheduler
  - Create initial servers
  - Create idle thread
- Start! — Switch to the idle thread

 ©National ICT Australia, 2004

KERNEL DEBUGGER

The kernel debugger provides:

- ➜ Very simple console driver
- ➜ Inspection of kernel state
- ➜ Tracing of kernel/user interaction

Kernel debugger enabled in configuration

Useful options:

- ➜ Enter kernel debugger on startup
- ➜ Kernel debugger break in
- ➜ Verbose initialisation
- ➜ Enable tracepoints

©National ICT Australia, 2004

USING THE KERNEL DEBUGGER (KDB)

Use ? command to display help:

```
 BS - back up to previous menu
 ?  - this help message
ESC - back to previous menu
 a  - architecure specifics
 c  - KDB configuration
 B  - generic bootinfo
SPC - show current user exception frame
 F  - show exception frame
 K  - dump kernel interface page
 s  - search for an exception frame
 g  - continue execution
 d  - dump memory
 D  - dump memory in other space
 6  - Reset system
 q  - show scheduling queue
 t  - show thread control block
 T  - shows thread control block (extended)
 0  - sigma0 interaction
 #  - statistics
 r  - enable/disable/list tracepoints
```


©National ICT Australia, 2004

STARTUP STATE

Currently have a bootimage loaded, and kernel initialisation completed. Waiting to start sigma0 and root task.

Dump KIP: K command
Prints a formatted version of the KIP. Including the location of sigma0 and root server.

```
Root servers:
  sigma0          ip: 0xc0530000, sp: 0x00000000, 0xc0530000:0xc053a0e0
  sigma1          ip: 0x00000000, sp: 0x00000000, 0x00000000:0x00000000
  root server     ip: 0xc0540000, sp: 0x00000000, 0xc0540000:0xc05528a4
```

 ©National ICT Australia, 2004

Scheduling Queue: q command

Display the scheduling queue, gives you a list of all threads in the system:

```
> q
```


©National ICT Australia, 2004

Scheduling Queue: q command

Display the scheduling queue, gives you a list of all threads in the system:

```
> q

[255]: 00100001 00108001
[  0]: (00080001)
idle : f0021000
```

 ©National ICT Australia, 2004

Scheduling Queue: q command

Display the scheduling queue, gives you a list of all threads in the system:

```
> q

[255]: 00100001 00108001
[  0]: (00080001)
idle : f0021000
```

Thread information: t command

Display a formated thread control block

```
> showtcb
tcb/tid/name [current]: 00108001
```

 ©National ICT Australia, 2004

Scheduling Queue: q command

Display the scheduling queue, gives you a list of all threads in the system:

```
> q

[255]: 00100001 00108001
[  0]: (00080001)
idle : f0021000
```

Thread information: t command

Display a formated thread control block

```
> showtcb
tcb/tid/name [current]: 00108001

=== TCB: e0042000 === ID: 00108001 = dff00000/f070e000 === PRIO: 0xff ========
UIP: c0540000   queues: Rswl      wait : 00000000:00000000   space: f0718000
USP: 00000000   tstate: RUNNING   ready: 00100001:00100001   pdir : 00000000
KSP: e0042f98   sndhd : 00000000  send : 00000000:00000000   pager: 00100001
total quant:           0us, ts length  :          10000us, curr ts:    10000us
abs timeout:           0us, rel timeout:              0us
sens prio: 255, delay: max=0us, curr=0us
resources: 00000000 []
partner: 00000000, saved partner: 00000000, saved state: ABORTED, scheduler: 00108001
```


©National ICT Australia, 2004

Currently both sigma0 and the root task are runnable threads.

Tracepoint menu: r command
Allows us to view and manipulate tracepoints

Enable all tracepoints: E command

Break into KDB on tracepoints: e command
This command lets you enable a specific tracepoint, and gives you the option of breaking in to the kernel debugger when the tracepoint is executed.

 ©National ICT Australia, 2004

Currently both sigma0 and the root task are runnable threads.

Tracepoint menu: r command
Allows us to view and manipulate tracepoints

Enable all tracepoints: E command

Break into KDB on tracepoints: e command
This command lets you enable a specific tracepoint, and gives you the option of breaking in to the kernel debugger when the tracepoint is executed.

Start executing: g command
This command will switch back to the kernel and continue execution.

©National ICT Australia, 2004

DEMONSTRATION

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

 ©National ICT Australia, 2004

DEMONSTRATION

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

Root task page fault more complicated:

① Root task faults

 ©National ICT Australia, 2004

DEMONSTRATION

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

Root task page fault more complicated:

① Root task faults

② Kernel manufactures pagefault IPC from root task $\Longrightarrow$ sigma0

©National ICT Australia, 2004

Demonstration

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

Root task page fault more complicated:

① Root task faults

② Kernel manufactures pagefault IPC from root task $\Longrightarrow$ sigma0

③ Sigma0 sends page mapping IPC $\Longrightarrow$ root task

©National ICT Australia, 2004

DEMONSTRATION

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

Root task page fault more complicated:

① Root task faults
② Kernel manufactures pagefault IPC from root task $\Longrightarrow$ sigma0
③ Sigma0 sends page mapping IPC $\Longrightarrow$ root task
④ Kernel creates mapping database entry, updates page tables

©National ICT Australia, 2004

DEMONSTRATION

Sigma0 page fault handled automatically by the kernel – easy since just direct mapped.

Root task page fault more complicated:

① Root task faults

② Kernel manufactures pagefault IPC from root task $\Longrightarrow$ sigma0

③ Sigma0 sends page mapping IPC $\Longrightarrow$ root task

④ Kernel creates mapping database entry, updates page tables

⑤ Root task continues

©National ICT Australia, 2004

SUMMARY

Covered the basics of getting a simple L4 application running:

- ➜ Tools required
- ➜ Configuring and building a kernel
- ➜ Creating a boot image using dite
- ➜ Loading and kernel initialisation
- ➜ Basic use of the L4 kernel debugger
- ➜ Interaction between sigma0 and root task during start up

©National ICT Australia, 2004

SUMMARY

Covered the basics of getting a simple L4 application running:

- ➜ Tools required
- ➜ Configuring and building a kernel
- ➜ Creating a boot image using dite
- ➜ Loading and kernel initialisation
- ➜ Basic use of the L4 kernel debugger
- ➜ Interaction between sigma0 and root task during start up

Next time: Building and programming in Iguana

- ➜ SCons build tool and the Iguana build system
- ➜ Boot image generation and Wombat booting
- ➜ Writing device drivers

©National ICT Australia, 2004