Embedded Systems Programming

ES Development Environment
(Module 3)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014
Embedded System Development

- Need a real-time (embedded) operating system (RTOS) or run on bare metal?
- Need a development and test environment?
  - Use the host to edit, compile, and build application programs
  - At the target, use tools to load, execute, debug, and monitor (performance and timing)
Development Environment of Linux

**Host PC workstation**
- gcc cross-compiler
- GDB debugger
- Eclipse IDE
- Linux or Windows

**Target board**
- Applications
- GDB Server
- Linux
- Board support package

Command line or IDE (Eclipse, Kdevelop)
Linux and GNU tools
HelloWorld Example

/* Hello World program */
#include <stdio.h>
#include <time.h>

int main(void) {
    int i;
    time_t curTime;
    for (i=0; i<10; i++) {
        curTime = time(NULL);
        printf("Hello, world. The current time is: \%s\n", ctime (&curTime));
        sleep(i);
    }
}

CC = i586-poky-linux-gcc
all: HelloWorld.c
    $(CC) -o HelloWorld.o HelloWorld.c

scp HelloWorld.o root@10.218.101.25:/home/root/labs/
From Source to Executable

- Compiler, linker, and loader
- In ELF: executable, relocatable, shared library, and core
  - Information for relocation, symbol, debugging
  - Linker resolves symbol reference
- Link script or link command file
  - Assigns absolute memory addresses (program area, static data, bss, stack, vector table, etc.)
- Startup code to disable interrupts, initialize stack, data, zero uninitialized data area, and call main().
ELF -- Executable and Linking Format

- The old one – a.out for UNIX community
- ELF –
  - standard for Linux
  - better support for cross-compilation, dynamic linking, initializer/finalizer
- An ELF file can be one of the 4 types
  - Relocatable
  - Executable
  - Shared object
  - Core file
- Two views
  - Compilers, assemblers, and linkers – a set of logical sections.
  - System loader – a set of segments
A text file that describes how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file.

- ENTRY linker script command to set the entry point.
- SECTIONS command to describe memory layout
- `.' indicates the location counter
- MEMORY command describes the location and size of blocks of memory in the target

```
MEMORY
{
    ROM (rx) : ORIGIN = 0, LENGTH = 256k
    RAM (wx) : org = 0x00100000, len = 1M
}

SECTIONS
{
    .text.start (__KERNEL_BASE__) : {
        startup.o(.text)
    }
    .text : ALIGN(0x1000) {
        __TEXT_START__ = .;
        *(.text)
        __TEXT_END__ = .;
    }
    .data : ALIGN(0x1000) {
        __DATA_START__ = .;
        *(.data)
        __DATA_END__ = .;
    }
    .bss : ALIGN(0x1000) {
        __BSS_START__ = .;
        *(.bss)
        __BSS_END__ = .;
    }
}
```
Debugger for Embedded Systems

- Observability, real-time analysis, and run control
- At host: GUI, source code, symbol table, type information, line number

Communication to the target
- serial port, Ethernet, USB, JTAG, etc.
- read/write memory and registers
- execution control (single step, breakpoint, watchpoint, etc.)

At target:
- debugging stub (software): at breakpoint, replace the instruction with breakpoint inst. or invalid instruction to trigger exception
- (USB) JTAG interface: hardware breakpoint, trace buffer, flash programming
Start-up and Booting Sequence

- Processor, memory, and IO initialization
- Load an operating system (or an application)

Disables interrupts, puts the boot type on the stack, clears caches

The text and data segments are copied from ROM to RAM

Cache initialization, zeroing out system bss segment, initializing interrupt vectors and system hardware to a quiescent state

Initiates the multitasking environment, creates an interrupt stack, creates root stack and TCB

Initializes the I/O system, installs drivers, creates devices, and then sets up the network
Supplementary Slides
## ELF -- Executable and Linking Format (2)

- **Partial list of the ELF sections**
  - `.init` - Startup
  - `.text` - String
  - `.fini` - Shutdown
  - `.rodata` - Read Only
  - `.data` - Initialized Data
  - `.dtdata` - Initialized Thread Data
  - `.tbss` - Uninitialized Thread Data
  - `.ctors` - Constructors
  - `.dtors` - Destructors
  - `.bss` - Uninitialized Data

### ELF Sections Diagram

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.init</td>
<td>Startup</td>
</tr>
<tr>
<td>.text</td>
<td>String</td>
</tr>
<tr>
<td>.fini</td>
<td>Shutdown</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read Only</td>
</tr>
<tr>
<td>.data</td>
<td>Initialized Data</td>
</tr>
<tr>
<td>.tdata</td>
<td>Initialized Thread Data</td>
</tr>
<tr>
<td>.tbss</td>
<td>Uninitialized Thread Data</td>
</tr>
<tr>
<td>.ctors</td>
<td>Constructors</td>
</tr>
<tr>
<td>.dtors</td>
<td>Destructors</td>
</tr>
<tr>
<td>.data</td>
<td>Initialized Data</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read Only</td>
</tr>
<tr>
<td>.text</td>
<td>String</td>
</tr>
<tr>
<td>.init</td>
<td>Startup</td>
</tr>
</tbody>
</table>

### ELF Sections Table

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.init</td>
<td>Startup</td>
</tr>
<tr>
<td>.text</td>
<td>String</td>
</tr>
<tr>
<td>.fini</td>
<td>Shutdown</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read Only</td>
</tr>
<tr>
<td>.data</td>
<td>Initialized Data</td>
</tr>
<tr>
<td>.tdata</td>
<td>Initialized Thread Data</td>
</tr>
<tr>
<td>.tbss</td>
<td>Uninitialized Thread Data</td>
</tr>
<tr>
<td>.ctors</td>
<td>Constructors</td>
</tr>
<tr>
<td>.dtors</td>
<td>Destructors</td>
</tr>
<tr>
<td>.data</td>
<td>Initialized Data</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read Only</td>
</tr>
<tr>
<td>.text</td>
<td>String</td>
</tr>
<tr>
<td>.init</td>
<td>Startup</td>
</tr>
</tbody>
</table>