Standards of Embedded Computer System

Chapter 2

Why Standards?

- A LCD TV
  - LCD panel made by Samsung
  - TV controller made by RCA
  - TV receiver made by Motorola
  - Cable provided by Time Warner
  - Content provided by HBO
- How many components?
- How many interfaces?
- How many standards are needed?
  - Interactive vs. non-interactive content

Standards

- Market or application specific
  - Most embedded devices have market specific standards : HDTV standards, cell phone standards, ...
  - AT&T vs Verizon
  - iOS vs Android vs Symbian vs WebOS
- General purpose
  - Not for any specific device or application
  - Networking: TCP/IP, HTTP, HTML
  - Security: SSL
  - Quality: ISO9000

Standard Examples

- System development (software)
  - General purpose languages
  - Connection between programmers and devices
  - Common procedure of development
  - Mostly used tools
- Networking
  - The way devices talk to each other
  - Specifications for different devices
  - Specifications for different layers
    - Devices and system components at different layers

Programming Language

- No single language perfect for every system
  - Designed to perform a dedicated function
- Machine code (Atmega128)
  - Binary, hardware dependent
- Assembly (AVR)
  - Maybe hardware independent
- High level (C)
  - C, hardware independent, procedure
- Very high level (NesC)
  - C++, Java, OO

Compilation

- Source code -> executable
  - Preprocess -> source code
  - Compile -> object code
  - Link -> executable code
- Host : where a program is compiled
- Target : where a program is executed
- Host = Target in regular computer
- In embedded system
  - Host is a regular computer
  - Target is the embedded system
  - Cross-compilation : avr-gcc or gcc -b avr

Cross-compilation

- Source to executable : Figure 2-4
  - Pre-process at host
    - nesC -> C
    - Macro
  - Compile at host
    - C -> machine code
    - Source file -> object file
  - Link at host
    - Add library and system codes
    - Object file -> executable file
  - Extract at host
    - Executable -> target machine code (native code)
  - Upload and execute : target

Interpreter

- Compile
  - Java source code (.java) -> Java byte code (.class)
- Generate machine code one line at a time
  - Java byte code -> machine code (native code)
  - Scripting languages : shell, perl
  - Platform dependent (interpreter)
- Benefits
  - Platform independent for applications
  - Flexibility in development
  - Rapid prototyping

JVM

- Application layer
  - Java APIs
  - Class loader
- System layer
  - Java OS : thread, schedule, IO, memory, ...
- Hardware layer
  - Java device driver
  - Code execution
- JVM is in Figure 2-12

(J)VM Architecture

- Windows CE
- Application Layer
  - Applications
  - JVM
- System layer
  - OS
- Hardware layer
  - Driver
  - Hardware

- Sun SPOT
- Application Layer
  - Applications
- System layer
  - JVM
- Hardware layer
  - Driver
  - Hardware

JVM

- JVM classes : APIs
  - Platform independent : J2ME
  - Platform dependent : extension to J2ME
- Class loader
  - Load application classes and JVM classes
- Execution engine
  - Garbage collector
    - Clean deallocated memory
  - Code execution
    - Execute java bytecode

JVM

- Garbage collector
  - Between virtual memory and real memory
  - Copying GC, Figure 2-13
    - Copy objects in use into a new memory block and then remove the old memory block.
    - Copy is slow and cannot be interrupted.
  - Mark and sweep GC, Figure 2-14
    - Mark objects in use and clean objects not in use.
    - Generate memory fragments.
  - Generation GC, Figure 2-15
    - Objects are put into different groups (generations) according to their usages.
    - No need to use copying GC on long-lived objects.

JVM

- Code execution
  - Between virtual code and native code
  - Interpretation, Figure 2-16
    - Each byte code instruction is converted into a native code instruction, regardless how many times it is executed.
  - Just-in-time, Figure 2-17
    - A byte code instruction (a function) is interpreted once and stored in memory for future execution.
  - Way-ahead-of-time, Figure 2-18
    - A whole program is interpreted and stored in memory before execution.

JVM

- C
  - Memory access
  - Code execution
- JVM
  - Memory access
  - Code execution

```
int i;
int c[10]; // in C
// in java
// int[] c=new int[10];
for (i=0;i<20;i++)
  c[i]=i*2;
```

.NET Compact Framework

- Source code
  - C#, VB, ...
- Microsoft intermediate language (MSIL)
  - Microsoft common language specification
- Class loader
- JIT compiler (interpreter)
  - Not compile source code to MSIL
  - But code execution engine
  - Transform/compile MSIL to target native code

Networking

- Distance and network size
  - LAN, 1Km, 100 computers
    - PAN
  - WAN, 10-100Km, 1K-10K computers
    - MAN, CAN
  - Internet, global, all computers
- Physical medium
  - Unshielded twisted pair (UTP), 10M-1Gbps
  - Coaxial, 10M-1Gbps
  - Fiber optic, 1G-1Tbps
  - Microwave, 100K-1Gbps
  - Infrared, 10K-100Mbps
  - Laser, 10M-10Gbps

Network Model

- Architecture
  - Client/server
  - P2P
- OSI, Open System Interconnection
  - Application layer
  - Presentation layer
  - Session layer
  - Transport layer
  - Network layer
  - Data-link layer
  - Physical layer

Embedded System vs OSI

- Application layer
  - OSI's upper three layers
- System layer
  - OSI's middle three layers
- Hardware layer
  - OSI's lower one layer
- Networking
  - Data presentation, specification
  - Data interpretation/processing, protocol

TCP/IP vs OSI

- Application layer
  - Application layer
  - Presentation layer
  - Session layer
- Transport layer (TCP)
  - Transport layer
- Internet layer (IP)
  - Network layer
- Network access layer
  - Data-link layer
  - Physical layer

Bluetooth vs OSI

- Application protocol group
  - Application layer
  - Presentation layer
  - Session layer
- Middleware protocol group
  - Transport layer
  - Network layer
- Transport protocol group
  - Data-link layer
  - Physical layer