CSE326

Embedded or Not?
What exactly is an embedded system?

- An embedded system can be defined as a control system or computer system designed to perform a specific task. An embedded system can also be defined as a single purpose computer.
Examples

- Every home has several examples of embedded computers. Microwave, washing machine, television.
- Modern cars have embedded computers onboard that control such things as ignition timing and anti-lock brakes using input from a number of different sensors.
- Network equipment: firewall, router, switch
- Consumer equipment: MP3 players, cell phones, PDAs, digital cameras, camcorders,
- Mission-critical systems: Flight control, satellites, navigation systems on aircraft and intruder alarm systems.
What exactly is an embedded system?

- An embedded system is frequently a computer that is implemented for a particular purpose. In contrast, an average PC computer usually serves a number of purposes: checking email, surfing the internet, listening to music, word processing, etc... However, embedded systems usually only have a single task, or a very small number of related tasks that they are programmed to perform.
Characteristics

- Embedded computers rarely have a generic interface, however. Even if embedded systems have a keypad and an LCD display, they are rarely capable of using many different types of input or output. An example of an embedded system with I/O capability is a security alarm with an LCD status display, and a keypad for entering a password.
Embedded or Not?

- contain a processing engine
- typically designed for a specific purpose
- rarely have a generic interface
- resource limited – small memory and no hard drive
- have power limitations – batteries
- not used as a general-purpose computing platform
- has application software built in, not user selected
- ships with all intended application hardware and software
- intended for applications completely or partially independent of human intervention
- interact with physical elements in our environment, such as controlling and driving a motor, sensing temperature
Example embedded system

- Serial UART
- USB controller
- Ethernet controller
- Ethernet (LAN)
- Wireless Modem
- SDRAM
- FLASH
- Real-time clock
- RS232
- Serial Port
- USB
Typical embedded linux setup

Host Development System

Ethernet Hub

RS232

Embedded Linux Target

Serial Terminal
Trend

- In recent years, hardware such as microprocessors, microcontrollers, and FPGA chips have become much cheaper. So when implementing a new form of control, it's wiser to just buy the generic chip and write your own custom software for it.
What are Some Downfalls of Embedded Computers?

- Embedded computers may be economical, but they are often prone to some very specific problems. A PC computer may ship with a glitch in the software, and once discovered, a software patch can often be shipped out to fix the problem. An embedded system, however, is frequently programmed once, and the software cannot be patched. Even if it is possible to patch faulty software on an embedded system, the process is frequently far too complicated for the user.
Why study embedded systems?

- Embedded systems are playing important roles in our daily lives, even though they might not necessarily be visible. Some of the embedded systems we use every day: a settop box, the timer in a microwave oven, a cellphone, an MP3 player or any other device with some amount of intelligence built-in.

- Embedded systems is a rapidly growing industry where growth opportunities are numerous.
Primary Embedded Operating Systems

- VxWorks (Wind River)
- Windows CE (Microsoft)
- QNX
- Neutrino
- Nucleus (Accelerated Technology)
- eCos (Red Hat)
- ChorusOS (Sun Microsystems)
- LynxOS (LynuxWorks)
- Embedded Linux
- Android
Why Embedded Linux?

- Vendor independence
- Time to market
- Low cost
- Open source
- Standards (POSIX) compliance
- Varied hardware support
- Preferred OS for any hardware or software innovation
- Preferred bring-up platform for hardware manufacturers.
FAQ

- Is Linux too large? - No. Linux is highly modular and has excellent component selection mechanism.

- Small-footprint embedded Linux efforts
  - uClinux, for no-MMU platforms
  - ELKS (Embedded Linux Kernel Subset)
  - ThinLinux
Embedded System Platforms

PC104

eBox

Gumstix

ANATOMY OF A SunSPOT

SUNROOF
SENSOR BOARD
PROCESSOR BOARD
BATTERY
FriendlyARM- Mini2440
CSE326

Processor Basics & Computer Architecture Review
WinCE

Windows CE (also known officially as Windows Embedded Compact or sometimes abbreviated WinCE) is an OS developed by Microsoft for minimalistic computers and embedded systems.
CISC & RISC

- Complete Instruction Set Computer (CISC) System/360 (excluding the 'scientific' Model 44), VAX, PDP-11, Motorola 68000 family, and Intel x86

- Main idea in CISC
  - design instruction sets that directly supported high-level programming constructs such as procedure calls, loop control, and complex addressing modes, allowing data structure and array accesses to be combined into single instructions.

- However, performance is improved by not using a complex instruction (such as a procedure call or enter instruction), but instead using a sequence of simpler instructions. (Reduced Instruction Set Architecture (RISC))
Superscalar CPU Architecture

- A superscalar CPU architecture implements a form of parallelism called instruction-level parallelism within a single processor. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate. A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier.
Simple superscalar pipeline. By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed.
Vector Processor

A vector processor, or array processor, is a CPU design where the instruction set includes operations that can perform mathematical operations on multiple data elements simultaneously. This is in contrast to a scalar processor which handles one element at a time using multiple instructions.
MIPS

- MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC microprocessor architecture developed by MIPS Technologies. (John L. Hennessy)

- Use of MIPS processors on the desktop has now disappeared almost completely. However, MIPS architecture was widely adopted by the embedded market.
Flynn's taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single data</td>
<td>SISD von Neumann architecture</td>
<td>MISD Fault tolerant computers</td>
</tr>
<tr>
<td>Multiple data</td>
<td>SIMD Vector processors</td>
<td>MIMD Multi-core</td>
</tr>
</tbody>
</table>
ARM Architecture

- The ARM architecture (previously, the Advanced RISC Machine, and prior to that Acorn RISC Machine) is the most widely used 32-bit processor architecture in the world.
- The ARM architecture is a 32-bit RISC processor architecture developed by ARM Limited that is widely used in embedded designs. Because of their **power saving features**, ARM CPUs are dominant in the mobile electronics market, where low power consumption is a critical design goal.
- Prominent branches in this family include Marvell's (formerly Intel's) XScale and the Texas Instruments OMAP series.
ARM

- In the late 1980s Apple Computer and VLSI Technology started working with Acorn on newer versions of the ARM core.
- The new Apple-ARM work would eventually turn into the ARM6, first released in 1991. Apple used the ARM6-based ARM 610 as the basis for their Apple Newton PDA.
- DEC licensed the ARM6 architecture (which caused some confusion because they also produced the DEC Alpha) and produced the StrongARM. At 233 MHz this CPU drew only 1 watt of power (more recent versions draw far less). This work was later passed to Intel as a part of a lawsuit settlement, and Intel took the opportunity to supplement their aging i960 line with the StrongARM.
- Intel later developed its own high performance implementation known as XScale which it has since sold to Marvell.
PowerPC

- PowerPC (short for Power Performance Computing, often abbreviated as PPC) is a RISC instruction set architecture created by the 1991 Apple–IBM–Motorola alliance, known as AIM.
- Originally intended for personal computers, PowerPC CPUs have since become popular embedded and high-performance processors. PowerPC was the cornerstone of AIM's PReP and Common Hardware Reference Platform initiatives in the 1990s and while the architecture is well known for being used by Apple's Macintosh lines from 1994 to 2006 (before Apple's transition to Intel), its use in video game consoles and embedded applications far exceed Apple's use.
Embedded Processors

- **Stand-alone**
  - IBM970FX (superscalar, 64-bit PowerPC architecture)
  - Intel Pentium M (x86)
  - Freescale MPC7448 (PowerPC core G4)

- **SOC**
  - PowerPC (AMCC, Freescale)
  - MIPS (Broadcom, AMD, PMC-Sierra, NEC, Toshiba…)
  - ARM (TI, Freescale, Intel XScale (now Marvell), PMC-Sierra, Altera, …)
DLX Architecture

- Good architectural model for study
- 32 32-bit registers
- Two addressing modes (immediate and displacement)
- I-type, R-type, J-type instructions
### I-type

<table>
<thead>
<tr>
<th></th>
<th>Opcode</th>
<th>rs1</th>
<th>rd</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
# R-type

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>rs1</td>
<td>rs2</td>
<td>rd</td>
<td>func</td>
<td></td>
</tr>
</tbody>
</table>

**Opcodes**

- **6**: Immediate
- **5**: Register
- **5**: Register
- **11**: Function

**Fields**

- **Opcode**: 6 bits
- **rs1**: 5 bits
- **rs2**: 5 bits
- **rd**: 5 bits
- **func**: 11 bits
## J-type

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Offset added to PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>
DLX Datapath

Multiple instructions are overlapped in execution to make fast CPUs.
Instruction fetch cycle (IF)

- IF cycle: Send out the PC and fetch the instruction from memory into instruction register (IR), increment PC. NPC holds the next.

IR<- Mem[PC]
NPC<-PC + 4
Instruction decode/register fetch cycle (ID)

- ID cycle: Decode the instruction and access the register file to read the registers.

A<-Regs[IR_{6..10}];
B<-Regs[IR_{11..15}];
Imm<-((IR_{16})^{16}##IR_{16..31})
Execution/effective address cycle (EX)

- Memory reference
  \[ \text{ALUOutput} \leftarrow A + \text{Imm} \]

- Reg-Reg ALU instruction
  \[ \text{ALUOutput} \leftarrow A \ \text{func} \ B \]

- Reg-Imm ALU instruction
  \[ \text{ALUOutput} \leftarrow A \ \text{op} \ \text{Imm} \]

- Branch
  \[ \text{ALUOutput} \leftarrow \text{NPC} + \text{Imm} \]

  \[ \text{Cond} \leftarrow (A \ \text{op} \ 0) \]
Memory access/Branch completion cycle (MEM)

- Memory reference
  LMD<-Mem[ALUOutput] or Mem[ALUOutput]<- B

- Branch
  if (cond) PC<- ALUOutput else PC<- NPC
Write-Back cycle (WB)

- Reg-Reg ALU instruction
  $\text{Regs}[\text{IR}_{16..20}] \leftarrow \text{ALUOutput}$

- Reg-Imm ALU instruction
  $\text{Regs}[\text{IR}_{11..15}] \leftarrow \text{ALUOutput}$

- Load instruction
  $\text{Regs}[\text{IR}_{11..15}] \leftarrow \text{LMD}$
The Basic Pipeline for DLX
Pipeline Hazards

- Structural hazards
- Data hazards
- Control hazards
Structural Hazard

LOAD
Stall- caused by structural hazard
Data hazard

ADD R1 R2 R3
SUB R4 R1 R5
AND R6 R1 R7
OR R8 R1 R9
XOR R10 R1 R11
Forwarding  —  bypassing, short-circuiting

AND R6 R1 R7
ADD R1 R2 R3
SUB R4 R1 R5
OR R8 R1 R9
XOR R10 R1 R11
Data Hazards

- RAW read after write
- WAW write after write
- WAR write after read
  - We sometimes need to add hardware called a **pipeline interlock** to detect a hazard and stall the pipeline until the hazard is cleared.
  - Compiler scheduling.
Control Hazards

- When a branch is executed, it may or may not change the PC to something other than its current value plus 4.
- Until the end of MEM, PC is not changed.
- A branch causes a 3-cycle stall.
- Stall from branch hazards can be reduced by moving the zero test and branch target calculation into the ID phase.
Typical Memory Hierarchy

- CPU Registers
- Cache
  - Memory bus
- Memory
  - On-chip or off-chip SRAM
  - DRAM
- I/O devices
  - I/O bus
SRAM

- SRAM (Static Random Accessible Memory) - where the word static indicates that it does not need to be periodically refreshed, as SRAM uses bistable latching circuitry (i.e., flip-flops) to store each bit. Each bit is stored as a voltage. Each memory cell requires six transistors, thus giving chip low density but high speed. However, SRAM is still volatile in the (conventional) sense that data is lost when powered down.
- Consumes less power than DRAM.
- More expensive
- In high-speed processors (such as Pentium), SRAM is known as cache memory and is included on the processor chip. However, high-speed cache memory is also included external to the processor to improve total performance.
DRAM

- DRAM(Dynamic Random Accessible Memory)- Its advantage over SRAM is its structural simplicity: only one transistor (MOSFET gates) and a capacitor (to store a bit as a charge) are required per bit, compared to six transistors in SRAM. This allows DRAM to reach very high density. Also it consumes more power and is even cheaper than SRAM (except when the system size is less than 8 K).

- But the disadvantage is that since it stores bit information as charge which leaks; therefore information needs to be read and written again every few milliseconds. This is known as refreshing the memory and it requires extra circuitry, adding to the cost of system.
Principle of locality

- The data most recently used is very likely to be accessed again in the near future.
- We should try to keep recently accessed items in the fastest memory.
- Smaller memories are faster.
cache & memory, memory & disk

- The cache and main memory have the same relationship as the main memory and disk.
- Cache hit & miss.
- Page fault.
Categories of cache organization

- Direct Mapped
  - Block # 0 1 2 3 4 5 6 7
  - Data
  - Tag
  - Search

- Set Associative
  - Set # 0 1 2 3
  - Data
  - Tag
  - Search

- Fully Associative
  - Tag
  - Search
Categories of cache organization

Suppose memory has 32 blocks. Block 12 can be placed in

- Direct mapped
  (memory) block\# % #sets (12 % 8 = 4)
- Set associative
  (memory) block\# % #sets (12 % 4 = 0)
- Fully associative
  can be placed anywhere.
Determining the Set# and Tag.

- The Set# = (memory) block# mod #sets.
- The Tag = (memory) block# / #sets.
How do we find a memory block?

- How do we find a memory block in an associative cache (with block size 1 word)?
- Divide the memory block number by the number of sets to get the index into the cache.
- Mod the memory block number by the number of sets to get the tag.
- Check all the tags in the set against the tag of the memory block.
- If any tag matches, a hit has occurred and the corresponding data entry contains the memory block.
- If no tag matches, a miss has occurred.
How big is the cache?

- This means, what is the capacity or how big is the blue?
  - \[256 \times 4 \times 4B = 4KB.\]

- How many bits are in the cache?
  - The 32 address bits contain 8 bits of index and 2 bits giving the byte offset.
  - So the tag is 22 bits.
  - Each block contains 1 valid bit, 22 tag bits and 32 data bits, for a total of 55 bits.
  - There are 1K blocks.
  - So the total size is 55Kb (kilobits).

- What fraction of the bits are user data?
  - \[4KB / 53Kb = 32Kb / 53Kb = 32/53.\]
Virtual Memory
Translation Lookaside Buffer (TLB)

- A TLB is a cache of the page table.
- TLB is a CPU cache that is used by memory management hardware to improve the speed of virtual address translation.
- A TLB has a fixed number of slots containing page table entries, which map virtual addresses onto physical addresses.
- It is typically a content-addressable memory (CAM), in which the search key is the virtual address and the search result is a physical address.
- Needed because otherwise every memory reference in the program would require two memory references, one to read the page table and one to read the requested memory word.
TLB & PTE

- Modern MMUs typically divide the virtual address space (the range of addresses used by the processor) into pages, each having a size which is a power of 2, usually a few kilobytes.
- The bottom n bits of the address (the offset within a page) are left unchanged. The upper address bits are the (virtual) page number.
- The MMU normally translates virtual page numbers to physical page numbers via an associative cache called a TLB. When the TLB lacks a translation, a slower mechanism involving hardware-specific data structures or software assistance is used. The data found in such data structures are typically called page table entries (PTEs), and the data structure itself is typically called a page table. The physical page number is combined with the page offset to give the complete physical address.
Operating Systems

Based on how OS makes use of hardware for protection OS' can be classified as:

- Real-time executive (RTOS)
- Monolithic kernel
- Microkernel
Real-Time Executive (RTOS)

- Traditional real-time executives are meant for MMU-less processors. (uClinux)
- On these OS', the entire address space is flat or linear with no memory protection between the kernel and applications.
- These OS' have small memory and size footprint as both OS and applications are bundled into a single image.
Memory Management Unit (MMU)

- A key benefit of an MMU is memory protection: an OS can use it to protect against errant programs, by disallowing access to memory that a particular program should not have access to. Typically, an OS assigns each program its own virtual address space.
# Architecture of RTOS

<table>
<thead>
<tr>
<th>App 1</th>
<th>App 2</th>
<th>App 3</th>
<th>App N</th>
</tr>
</thead>
<tbody>
<tr>
<td>File System</td>
<td>Network Stack</td>
<td>Device Drivers</td>
<td></td>
</tr>
<tr>
<td>Kernel (scheduler, memory management, IPC)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RTOS

- No overhead of system calls, message passing, or copying of data (real-time in nature)
- Difficult to add new applications or kernel modules dynamically as the system has to be brought down.
Today

- Prices of memory and flash go down.
- Embedded systems started having more and more software on their systems.
- Software also started becoming differentiating factor in the market.
- RT executives were not suited for large-scale software integration.
Monolithic Kernels

- Monolithic kernels have a distinction between the user and the kernel space.
- When software runs in user space normally it cannot access the system hardware nor can it execute privileged instructions.
- Using special entry points (provided by hardware) an application can enter the kernel mode from user space.
Monolithic Kernels

- The user space programs operate on a **virtual address** so that they cannot corrupt another application's or the kernel's memory.

- However, the kernel components share the same address space; so a badly written driver or module **can cause the system to crash**.
# Architecture of monolithic kernel

<table>
<thead>
<tr>
<th>Kernel Space</th>
<th>System Call Layer</th>
<th>User Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>File System</td>
<td>IPC</td>
<td>App 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>App 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>App 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>App N</td>
</tr>
<tr>
<td>Network Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device Drivers</td>
<td>Scheduler</td>
<td></td>
</tr>
<tr>
<td>Memory Mgmt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Abstraction Layer (HAL)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Monolithic kernel

- Any fault in application will cause only that application to misbehave without causing any system crash.
- Applications can be added to a live system without bringing down the system.
- Most of the UNIX OS' are monolithic.
The microkernel makes use of a small OS that provides the very basic service (scheduling, interrupt handling, message passing).

The rest of the kernel (file system, device drivers, networking stack) runs as applications.
## Architecture of microkernel

<table>
<thead>
<tr>
<th>App 1</th>
<th>App 2</th>
<th>...</th>
<th>App N</th>
<th>File System</th>
<th>IPC</th>
<th>Network Stack</th>
<th>Device Drivers</th>
<th>Scheduler</th>
<th>Memory Mgmt</th>
</tr>
</thead>
</table>

Kernel (message passing)
Microkernel

- Translating theory into practice caused too many bottlenecks.
- They require robust message-passing schemes.
Summary of OS types

- RTOS provides no memory protection to make more real-time but at the cost of reliability.
- On the other hand, microkernel provides memory protection to individual kernel subsystems at the cost of complexity.
- Linux takes the middle path of monolithic kernels where the entire kernel operates on a single memory space.
- Linux kernel is one of the most employed software in a varied range of systems; because of its stability during the examination process to make sure introducing new kernel software does not cause any reliability issues.
“tools”: Unix-like operating systems don't put so much weight on application programs. Instead, they come with many small programs called tools. Each tool is generally capable of performing a very simple, specific task, and performing it well -- one tool does nothing but output the file(s) or data passed to it, one tool spools its input to the print queue, one tool sorts the lines of its input, and so on.
Linux (Unix) philosophy

- “pipes”: a way to pass the output of one tool to the input of another. By knowing what the individual tools do and how they are combined, a user could now build powerful "strings" of commands.

Example: who is on the system
who | wc -l

- tool directories (`/bin' and `/usr/bin'
Accounts

- Linux is a multi-user system, meaning that many users can use one Linux system simultaneously, from different terminals.
- You need an account,
  Special account: root, su (superuser)
Console Basics

- A Linux *terminal* is a place to put input and get output from the system, and usually has at least a keyboard and monitor.

- Linux systems can be accessed in other ways, such as through a network or via another terminal connected to a serial line.
Running a command

- A tool or application may take any number of **options** (sometimes called "flags"), which specify a change in its default behavior. It may also take **arguments**, which specify a file or some other text to operate on. Arguments are usually specified after any options.
Shell Basics

- Shell is a **command-line interpreter**.
- It provides a traditional user interface for the Unix operating system and for Unix-like systems.
- Users direct the operation of the computer by entering command input as text for a command line interpreter to execute or by creating text **scripts** of one or more such commands.
Shell Basics

- Similarly, graphical user interfaces for Unix, such as GNOME, KDE, and Xfce can be called visual shells or graphical shells.

- On MS-DOS, OS/2, and Windows, equivalents to Unix system scripts are called **batch** files, and have either a ".bat" or "cmd" extension.

- A newer CLI - Windows PowerShell
Shells

- Bourne shell (sh)
- Bourne again shell (bash)
- Almquist shell (ash)
- Debian Almquist shell (dash)
- Korn shell
- C shell (csh)
- TENEX C shell (tcsh)
Commands

- cd, ls, rm, mkdir, who, ps, top, whoami, apropos, man, less, more, tar, chmod, ...

Version Control

- Subversion: svn
- Git: git
- Cvs
- Rcs
Make

- Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.
- Make gets its knowledge of how to build your program from a file called the makefile, which lists each of the non-source files and how to compute it from other files. When you write a program, you should write a makefile for it, so that it is possible to use Make to build and install the program.
Make Rules and Targets

- A rule in the makefile tells Make how to execute a series of commands in order to build a target file from source files. It also specifies a list of dependencies of the target file. This list should include all files (whether source files or other targets) which are used as inputs to the commands in the rule.

  target: dependencies ...
  commands
  ...
...
Make Rules and Targets

- When you run Make, you can specify particular targets to update; otherwise, Make updates the first target listed in the makefile. Of course, any other target files needed as input for generating these targets must be updated first.

- Make uses the makefile to figure out which target files ought to be brought up to date, and then determines which of them actually need to be updated. If a target file is newer than all of its dependencies, then it is already up to date, and it does not need to be regenerated. The other target files do need to be updated, but in the right order: each target file must be regenerated before it is used in regenerating other targets.
Integrated Development Environments for Linux

- Anjuta
- Eclipse
- Glimmer
- Kdevelop
- SourceNavigator
CSE326

RS232 Serial Interface & Terminal Emulators
Serial Port

- Embedded system developer's best friend.
- Many embedded systems are developed and debugged using an RS232 serial link between the host and the target.
- Simplicity of RS232 interface has encouraged wide-spread use & adoption (even though its bandwidth is limited).
Universal Asynchronous Receiver/Transmitter (UART)
RS232

- RS-232 (Recommended Standard 232) is a standard for serial binary data signals connecting between a DTE (Data Terminal Equipment) and a DCE (Data Circuit-terminating Equipment).

- A minimal "3-wire" RS-232 connection consisting only of transmit data, receive data, and ground, is commonly used when the full facilities of RS-232 are not required.
EIA232 (Electronic Industries Association 232) standard
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.
RS-232 Frame  (1 start bit, 7 data bits, 1 parity bits, and 2 stop bits)
RS232 Serial Protocol

- **Data Words:**
  - 5, 6, 7, or 8
  - Least significant bit first

- **Markers:**
  - 1 start bit before the word
  - 1 parity bit and 1 (1.5, 2) stop bit after the word

- **Possible Parity Bits:**
  - Even - make the number of 1s even
  - Odd - make the number of 1s odd
  - None
Parity

- For even parity checking, the number of 1's in the data plus the parity bit must equal an even number.
- For odd parity, this sum must be an odd number.
- Parity bits are used to detect errors in transmitted data.
Bit time & Baud Rate

- The **bit time** is the basic unit of time used in serial communication.
- The **baud rate** is the total number of bits (information, overhead, and idle) per time that is transmitted over the serial link. So we can compute the baud rate as the reciprocal of the bit time.
TTY Drivers

- A tty device gets its name from teletypewriter.
- Originally associated with physical or virtual terminal connection.
- Over time, the name also came to mean any serial port style device, as terminal connections could also be created.
TTY Drivers

- Some examples
  - Serial ports
  - USB-to-serial-port converters
  - Some types of modems

- To determine what kind of tty drivers currently loaded in the kernel and what tty devices are present look into
  
  /proc/tty/drivers
Serial devices in Linux are uniformly accessed as terminal devices.
You need read and write access /dev/ttyS* and /var/lock directory.
If you are using USB-to-serial converter then /dev/ttyUSB.
Terminal Emulators

- Kermit
- Minicom
- PuTTY
- Hyperterminal
- cu
Opening a Terminal Device File

- When a terminal device file is opened, it normally causes the thread to wait until a connection is established.
- As described in `open()`, opening a terminal device file with the O_NONBLOCK flag clear causes the thread to block until the terminal device is ready and available.
open()

- If CLOCAL mode is not set, this means blocking until a connection is established. If CLOCAL mode is set in the terminal, or the O_NONBLOCK flag is specified in the open(), the open() function returns a file descriptor without waiting for a connection to be established.
Open- opens a file

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ... );

- O_RDWR, O_NOCTTY, O_NONBLOCK
close, read, write

- close
  - Closes the file

- read
  - The read() function attempts to read nbyte bytes from the file associated with the open file descriptor

- Write
  - The write() function attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor.
The `fcntl()` function provides for control over open files.

```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```
TTY Line Settings

- When a user wants to change the line settings of a tty device or retrieve the current line settings, he makes one of the many different `termios` user-space library function calls or directly makes an `ioctl` call on the tty device node. The tty core converts both of these interfaces into a number of different tty driver function callbacks and ioctl calls.
The termios functions describe a general terminal interface that is provided to control asynchronous communications ports.

```
#include <termios.h>
#include <unistd.h>
```
Termios struct members

Member Description

- c_cflag  Control options
- c_lflag  Line options
- c_iflag  Input options
- c_oflag  Output options
- c_cc    Control characters
- c_ispeed Input baud (new interface)
- c_ospeed Output baud (new interface)
Some useful functions

```c
int tcgetattr(int fd, struct termios *termios_p);
int tcsetattr(int fd, int optional_actions, struct termios *termios_p);
int tcsendbreak(int fd, int duration);
int tcdrain(int fd);
int tcflush(int fd, int queue_selector);
int tcflow(int fd, int action);
int cfmakeraw(struct termios *termios_p);
speed_t cfgetispeed(struct termios *termios_p);
speed_t cfgetospeed(struct termios *termios_p);
int cfsetispeed(struct termios *termios_p, speed_t speed);
int cfsetospeed(struct termios *termios_p, speed_t speed);
pid_t tcgetpgrp(int fd);
int tcsetpgrp(int fd, pid_t pgrp);
```
Curses Library

- The Curses library provides a set of functions that enable you to manipulate a terminal's display regardless of the terminal type.

- The basis of curses programming is the window data structure. Using this structure, you can manipulate data on a terminal's display.

- You can instruct curses to treat the entire terminal display as one large window or you can create multiple windows on the display.

- The windows can be different sizes and can overlap one another.
Curses Library

- **endwin**
  - Terminates the curses function libraries and their data structures.

- **initscr**
  - Initializes the curses function library and its data structures.

- **cbreak**
  - Sets the input mode for the current terminal to cbreak mode.
Curses library

- Compile
  - -lcurses
CSE326

BOA Webserver
Web Server

- A computer program that is responsible for accepting HTTP requests from clients (user agents such as web browsers), and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.).
BOA Webserver

- Boa is a single-tasking HTTP server.
- GNU/Linux is the development platform at the moment, other OS's are known to work.

Files used by BOA

- **boa.conf**: This file is the sole configuration file for Boa. The directives in this file are defined in the DIRECTIVES section.
- `/var/www/index.html`
HTTP client & server

- embedded platforms usually have an HTTP server called Boa and an http client called wget.

- To post
  - Uncomment ServerName in /etc/boa/boa.conf and save it.
  - Put whatever you want to post in /var/www/index.html
  - Start boa if it has not already.
CSE326

Bootloaders
Das U-Boot
Steps in generation of executable (ELF)

- Generate preprocessed output
- Generate assembly language output
- Generate object output
- linking
Executable Linkage Format (ELF)

- The Executable and Linking Format was originally developed and published by UNIX System Laboratories (USL) as part of the Application Binary Interface (ABI).
Types of object files

- A *relocatable* file holds code and data suitable for linking with other object files to create an executable or a shared object file.
Types of object files

- An **executable** file holds a program suitable for execution; the file specifies how exec(BA_OS) creates a program’s process image.
Types of object files

- A *shared object* file holds code and data suitable for linking in two contexts. First, the link editor [see ld ] may process it with other relocatable and shared object files to create another object file. Second, the dynamic linker combines it with an executable file and other shared objects to create a process image.
Linux Start-up Sequence

- Boot loader phase:
  - hardware initialization and testing,
  - loading kernel image
  - transferring control to linux kernel.
Linux Start-up Sequence

- Kernel initialization phase
  - Platform-specific initialization
  - Bring up kernel subsystems
  - Turn on multitasking
  - Mount the root file system
  - And then jump to user space
Linux Start-up Sequence

- User space initialization phase
  - Bring up the services
  - Network initialization
  - Issue a log-in prompt
Boot Loader Phase (Hardware Init.)

- Configure the CPU speed
- Memory initialization, such as setting up the registers, clearing memory and determining the size of onboard memory
- Turn on caches
- Set up serial port for the boot console
- Do hardware diagnostics
Boot Loader Phase (Downloading Kernel Image)

- Boot loader needs to locate the kernel image
  - May be on the system flash
  - May be on the network
- Image needs to be loaded into the memory
  - If image is compressed, it needs to be uncompressed first
Boot Loader Phase
(Downloading Kernel Image)

- Memory address to where the kernel image is downloaded is decided by the boot loader by reading the ELF header of the kernel image.
Boot Loader Phase (Setting up arguments)

- Argument passing is a very powerful option supported by Linux kernel
- Linux provides a generic way to pass arguments to the kernel across all platforms
Boot Loader Phase
(Jumping to Kernel Entry Point)

- The kernel entry point is decided by the linker script when building the kernel.
- Once the boot loader jumps to the kernel entry point, its job is done.
Boot loader/monitor

- A boot loader, sometimes referred to as a boot monitor, is a small piece of software that executes soon after powering up a computer.

- In addition to booting capabilities, a monitor provides a command-line interface that can be used for debugging, reading/writing memory, flash reprogramming, configuring, etc.
On your desktop Linux PC you may be familiar with **lilo** or **grub**, which resides on the master boot record (MBR) of your hard drive. After the PC BIOS performs various system initializations it executes the boot loader located in the MBR.
The boot loader then passes system information to the kernel and then executes the kernel. For instance, the boot loader tells the kernel which hard drive partition to mount as root.
Boot Loaders Galore

- LILO
- GRUB
- ROLO
- Loadlin
- Etherboot
- LinuxBIOS
- Compaq's bootldr
- Blob
- PMON
- Sh-boot
- U-boot
- Redboot
Das U-Boot

- U-Boot provides out-of-the-box support for hundreds of embedded boards and a wide variety of CPUs including PowerPC, ARM, XScale, MIPS, Coldfire, NIOS, Microblaze, and x86.
Learn more about Das U-Boot

http://www.denx.de/wiki/view/DULG/UBoot
Creating Boot Scripts

- U-Boot environment variables can be used to create boot scripts.
- Such boot scripts are actually environment variables containing a set of U-Boot command sequences.
- `bootcmd`: this variable is recognized by U-Boot as the script to run automatically.
Creating Boot Scripts

- By using a combination of `run` command and the ; (semicolon) operator, you can make U-Boot run boot scripts.
Creating Boot Scripts

setenv kernel_addr 40100000
setenv mycmd setenv bootargs root=/dev/nfs
   ... ; bootm \$(kernel_address)
setenv bootcmd run mycmd
Creating shell scripts for U-Boot

- The `autoscr` command allows “shell” scripts to run under U-Boot. To create a U-Boot script image, commands are written to a text file. Then the `mkimage` tool is used to convert this text file into a U-Boot image using the image type script. This image can be loaded like any other image file. Autoscr runs the commands in this image.
CSE326

Embedded Storage
Flash

- Storing the boot loader
- Storing the OS image
- Storing the apps and app. library images
- Storing the read-write files
Memory Technology Device

- MTD is the subsystem used to handle onboard storage devices.
- MTD is not a separate class of driver set.
  - Char device – sequential access i.e. serial driver
  - Block device – random access i.e. disk driver
  - Network device – interact network protocol
Flash vs Hard Disk

- Hard disks have a sector that divides a page size (generally 4K, standard 512 bytes). Flash chips have larger sector sizes (standard 64K)
- Flash sectors normally have to be erased before writing.
- Flash chips have a limited lifetime (# of times a sector is erased)
Flash vs Hard Disk

- Normal disk I/O is slow; to speed up a cache in memory called the buffer cache stores the IO data to the disk.
- This is the reason why you need to shut down your OS on the PC before switching it off.
- Embedded systems can be powered off without proper shut down.
Flash chips

- Flash chips
  - NAND by Toshiba (usually for media storage)
  - NOR by Intel (on gumstix we have NOR flash)

- Access to data:
  - NOR: random like SRAM
  - NAND: reads/writes are done in terms of pages
Flash chips

- Execution of code:
  - NOR: code can be executed directly because it is directly connected to address/data bus
  - NAND: code needs to be copied to memory for execution.

- Performance:
  - NOR: slow erase, slow write, and fast read
  - NAND: fast erase, fast write, and fast read

- Bad blocks:
  - NOR: not expected to have bad blocks
  - NAND: expect bad blocks. (ECC)
Flash chips

Usage:

- NOR: used for code execution. Boot loaders can exist on NOR flashes. Pretty expensive, lesser memory densities, shorter life span.

- NAND: used mainly for storage devices for embedded systems such as set-top boxes, MP3 players. If you plan to use a board with only NAND, you may have to put in an additional boot ROM. High density, lower prices, and longer life span.
Embedded File Systems

- Ramdisk- linux emulates a hard disk using memory.
- RAMFS: RAM file system
- CRAMFS: Compressed ROM file system
- JFFS, JFFS2: Journaling flash file systems
- NFS: Network file system
- PROC: used by kernel to export its info to external world (monitor, debug...)
CSE326

OpenEmbedded & Bitbake
OpenEmbedded
OpenEmbedded & Bitbake

- OpenEmbedded is a collection of recipes (like makefiles), suitable for embedded devices.
- BitBake is a tool to generate packages out of this source collection.
- Together they form a cross-compilation system.
Adding a Graphical Framework
Adding GUI

- A lot of embedded systems need a graphical user interface to interact with the user.
- But an embedded system is often not powerful enough to use the full blown graphical system used by a Linux desktop (X server, window manager, desktop manager).
Adding GUI

- This is why special implementations exist for embedded system, for example stripped X servers and window managers. A desktop manager is in most cases not needed. Beside these, other solutions that don't follow the default Linux way exist. One of these is Qt/Embedded, specially created for embedded devices.
The growing demand for portable personal digital media and information environments is driving the convergence of computing and communications technologies and influencing the emergence of a new class of devices.
The convergence of devices will continue to blur the distinction between mobile phones and personal digital assistants making it increasingly important for manufacturers to be using device-independent platforms such as Qt/Embedded.
Qt “cute”

- a cross-platform application development framework
- a widget toolkit
- produced by Nokia's Qt Development Frameworks division, which came into being after Nokia's acquisition of the Norwegian company Trolltech, the original producer of Qt, on June 17, 2008.
Qt

- Qt uses standard C++, but makes extensive use of the C pre-processor to enrich the language.
- Qt can also be used in several other programming languages via language bindings.
- It runs on all major platforms, and has extensive internationalization support.
Qt

- Non-GUI features include SQL database access, XML parsing, thread management, network support and a unified cross-platform API for file handling.
Platforms

- Linux/X11 – Qt for X Window System (Unix / Linux)
- Mac OS X – Qt for Apple Mac OS X. Support for applications on top of Cocoa APIs
- Windows – Qt for Microsoft Windows
- Embedded Linux – Qt for embedded platforms (PDA, Smartphone, etc.)
- Windows CE – Qt for Windows CE
- Symbian – Qt for the Symbian platform.
- Maemo – Qt for Maemo
Uses

- KDE - desktop environment
- Xilinx ISE WebPack
- Google Earth
- Skype
- Opera - web browser
Qtopia (now Qt/Embedded)

- Development environment
- Games and multimedia
- Input methods
- Internet applications
- Java integration
- Localization support
- Personalization options
- PIM applications
- Productivity applications
- Synchronization framework
- Windowing system
- Wireless support
GUI Designer
GUI Designer

- Qt Designer (now Qt Creator)
Generating C++ Source Code From The Qt Designer File

- uic – user interface compiler
- Qt Designer saves in its .ui files. These files are a textual description of the dialog you just created. The format used is the markup language XML.
- You need to call uic twice to generate both the header file and the implementation file.
XML

<!DOCTYPE UI><UI>
<class>PEntry</class>
<widget>
  <class>QDialog</class>
  <property>
    <name>name</name>
    <cstring>PEntry</cstring>
  </property>
</widget>
…………………………
MOC- Meta Object Compiler

- a tool that is run on the sources of a Qt program. It interprets certain macros from the C++ code as annotations, and uses them to generate additional C++ code with "Meta Information" about the classes used in the program. This meta information is used by Qt to provide programming features not available natively in C++. 
Adding Functionality To A Dialog By Subclassing

- Responding to user interaction with Qt widgets is mostly done by connecting signals that these widgets emit to slots.
Signals & Slots

- The connection between user interaction and program functionality in Qt is done via so-called signals and slots.
A widget emits a signal when something interesting happens. What this means depends on the widget; for a push button, one of the signals is the clicked() signal, which is emitted when the user clicks the push button. Other widgets have other signals.
- Slots are just ordinary C++ methods that are marked as slots in the class declaration. Any C++ method can be a slot. Methods that are declared as slots can still be called the ordinary way.
Connections

Edit Connections

Signals (PushButton2):
- stateChanged(int)
- toggled(bool)
- clicked()
- released()
- pressed()

Slots (PizzaEntry):
- reject()
- accept()

Connections:
- Sender: PushButton2
  - Signal: clicked()
  - Receiver: PizzaEntry
  - Slot: reject()
CSE326

Embedded Graphics
Graphics System

- Managing the display hardware
- Managing one or more human input interfaces
- Providing an abstraction to the underlying display hardware
- Managing different applications so that they co-exist and share the display and input hardware effectively
Graphics System Architecture

- Layer 5: Graphics Application
- Layer 4: Toolkit
- Layer 3: Windowing Environment
- Layer 2: Device Drivers
- Layer 1: Display Hardware
- Layer 1: Input Hardware
Layer 1:
- A touchscreen as both input interface and display hardware
- A front panel LCD as display hardware and remote controller as input interface

Layer 2:
- Driver layer interfacing OS

Layer 3:
- Drawing engine responsible for graphics rendering and a font engine responsible for font rendering
Layer 4:
- Toolkit layer built over a particular windowing environment providing APIs (buttons, edit & list boxes)

Layer 5:
- Graphics application (does not always need a windowing environment and toolkit- directly interacts with hardware via driver interface)
Graphics Application on different OS

**Toolkit Layer**

- **WINCE MFC/SDK**
- **SYMBIAN Graphics API**
- **FLNX/Qt-E/GTK-fb**

**Graphics Engine Layer**

- **WINDOWS GDI Layer**
- **SYMBIAN Graphics Engine**
- **Nano-X/Qt-E/GDK-fb**

**OS Specific Driver Layer**

- **WINDOWS Video Miniport Driver**
- **SYMBIAN Graphics Driver**
- **LINUX Frame-buffer Driver**

**GRAPHICS HARDWARE**
Linux Desktop Graphics- The X

- Written for desktop computer
- Desktop graphic cards follow VGA/SVGA standards
- Keyboard and mouse as input drivers have standards
- Implements a driver interface to interact with PC display hardware (isolating from hardware specific details.)
X Toolkit architecture

- X APPLICATIONS
- X TOOLKIT
- X CLIENT USING Xlib
- X PROTOCOL
- X SERVER
Windowing Environment in X

- client/server model
- X app.s are clients
- X server controls the display and requests
- Based on sockets, can run over a network
- X-lib
Embedded Systems and X

- X is highly network oriented and **does not directly apply over** an embedded system
  - X can export a display over a network (not required in an embedded system). Client-server model is not aimed at single-user environments.
  - X has many dependencies (X font server, X resource manager ....) ----> size&memory
  - Running X as-is on power/cycle savvy embedded microprocessors is not possible.
Requirements for a Graphics Framework on an embedded system

- Quick/near real time response
- Low on memory usage
- Small toolkit library size

Tiny-X and Nano-X are popular embedded versions based on X graphics system.
Display Hardware

- Display System
- Input interface
Display System

- Video/display controller
- Video controller has an area of memory known as the **FRAME BUFFER**.
- Image is available on the frame buffer memory of the controller before it is presented on the screen.
- Digital image is divided into discrete memory regions called pixels.
Display System

- The number of pixels in the horizontal and vertical direction is expressed as screen resolution.
  - 1024X768 is a pixel matrix of 1024 columns and 768 rows.

- The frame buffer memory area
  - Frame Buffer-Memory = Display width X Display Height X Bytes-per-pixel
  - 160X120 with 16 colors (4-bits)
Whole image is composed of horizontal lines scanned from top to bottom and each scan cycle is called a **refresh**.

Number of screen refreshes that happen in a second is expressed as the **refresh rate**.
An embedded system's input hardware generally uses:
- Buttons
- IR (infra-red) remote (over serial)
- Touchscreen
- Standard interfaces such as keyboards and mice
Embedded Linux Graphics

- **Frame Buffer Driver**
  - Generic device interface for graphics applications
  - Simple single-user, direct display
  - No dependency on legacy display architecture, no network, no client-server model
  - Access video memory directly

- **Frame Buffer Interface**
  - Character device interface
Linux Frame Buffer Interface

- System calls such as `open()`, `read()`, `write()`, and `ioctl()`
- The frame buffer device in user space is available as
  - `/dev/fb[0-31]`
Frame Buffer Interface

- Normal I/O: open, read, write over /dev/fb
- ioctl: commands for setting the video mode, query chipset information, etc.
- Mmap: map the video buffer area into program memory
Repeated seek/write

Costly and time consuming
Mmapped write

open → mmap → Mmapped user space buffer

recommended
- Call open() to access /dev/fb
- ioctl() to set the resolution, pixel width, refresh rate
- Call mmap()
Windowing Environments, Toolkits, and Applications

A generic windowing environment consists of:

- An interface layer for low-level drivers
- A graphics engine for drawing objects
- A font engine that decodes one or more font file formats and renders
- APIs
Popular windowing environments

- Nano-X: Win32, X-11 like APIs
- FLNX
- MiniGUI
- DirectFB
- PicoGUI
- Qt/Embedded
- GTK+/FB
Package Management Systems (PMS)

- A package management system is a collection of tools to automate the process of
  - installing,
  - upgrading,
  - configuring,
  - removing software packages from a computer.
PMS

- Packages are distributions of software and metadata such as
  - the software's full name,
  - description of its purpose,
  - version number, vendor, checksum, and
  - a list of dependencies necessary for the software to run properly.

- Upon installation, metadata is stored in a local package database.
PMS

- Typically part of the operating system.
- Uses a single installation database.
- Can verify and manage all packages on the system.
- Single package management system vendor.
- Single package format.
Installer

- Each product comes bundled with its own installer.
- Performs its own installation, sometimes recording information about that installation in a registry.
- Only works with its bundled product.
- Multiple installer vendors.
- Multiple installation formats.
Functions?

- Verifying file checksums to ensure correct and complete packages.
- Verifying digital signatures to authenticate the origin of packages.
- Applying file archivers to manage encapsulated files.
Functions?

- Upgrading software with latest versions, typically from a software repository.
- Grouping of packages by function to help eliminate user confusion.
- Managing dependencies to ensure a package is installed with all packages it requires.
- Challenges with shared libraries
- Maintenance of configuration
- Upgrade suppression
  - version pinning
  - blacklist a package
  - hold
- Cascading package removal
Package Formats

- deb (debian)
- RPM (red hat)
- PISI (pardus)
- tgz
- ipk
- dpkg, used originally by Debian and now by other systems, uses the .deb format and was the first to have a widely known dependency resolution tool (APT).
- The RPM Package Manager was created by Red Hat, and is now used by a number of other Linux distributions.
A simple tgz package system combines the standard tar and gzip. (Slackware linux)

Pacman for Arch Linux, Frugalware and Lunar Linux uses pre-compiled binaries distributed in a compressed Tar archive.

Smart Package Manager, used by CCux Linux

pisi, used by pardus
- ipkg, a dpkg-inspired, very lightweight system targeted at storage-constrained Linux systems such as **embedded devices and handheld computers**
- opkg, fork of ipkg,
Ipkg (the Itsy Package Management System)

- Ipkg is available from Handhelds.org.
- To create your own ipkg, you need to:
  - Create a directory structure as your files will be installed:
    - `mkdir example-ipkg`
  - Most files get installed to `opt/Qtopia`, so create these directories.
- mkdir -p example-ipkg/opt/Qtopia/apps/Applications
- mkdir -p example-ipkg/opt/Qtopia/bin
- mkdir -p example-ipkg/opt/Qtopia/pics/example
- mkdir -p example-ipkg/opt/Qtopia/help/html
- cp example.desktop example-ipkg/opt/Qtopia/apps/Applications/example.desktop
- cp example example-ipkg/opt/Qtopia/bin/example
- cp Example.png example-ipkg/opt/Qtopia/pics/example/Example.png
- cp example.html example-ipkg/opt/Qtopia/help/html/example.html
Create a file named control in a directory named CONTROL (all capitals!) The control file is simply a text file with a number of parameters that ipk will use:

- `mkdir example-ipkg/CONTROL`
- `emacs example-ipkg/CONTROL/control`

The control file should contain lines like these:
CONTROL/control

Package: example

Files: bin/example apps/Applications/example.desktop
      pics/example/Example.png help/html/example.html

Priority: optional

Section: qtopia/applications

Maintainer: Your Name

Architecture: arm

Arch: arm

Version: 1.0.0

License: Public Domain

Description: Example program

An example program for the Qtopia environment.

Does nothing interesting.
Run the ipkg-build command that comes with ipkg

- ipkg-build example-ipkg.
Power Management

- Many types of embedded devices have different power requirements depending on the usage.
  - Network routers need to have minimal energy consumption to avoid heating.
  - PDAs and cell phones need to consume less energy so that battery life is not cut short.
Power Management Schemes in Linux

- Spans across the
  - BSP (board support package)
  - Drivers
  - Application layers
Hardware & Power Management

- Embedded system consists of
  - A processor
  - Memory
  - A network card such as a wireless
  - A sound card
  - An LCD-based display unit (biggest power consumer followed by CPU, sound card, memory, and the network card)
Power Management of the CPU

- Power consumed by a processor is directly proportional to the clock frequency.
- Power consumed by a processor is directly proportional to the square of the voltage.
New Embedded Processors offer

- Two schemes
  - Dynamic frequency scaling (SA1110 –Intel StrongARM)
  - Dynamic voltage scaling (Transmeta’s Crusoe processor)
- Modes offered by CPUs are controlled typically by the OS which can deduce the mode depending on the system load.
- Embedded systems are event-driven.
- Processor spends a lot of time waiting for events from the user or from the external world.
- OS can tune the processor’s power consumption depending on the system load.
- **Idle mode**: where the processor clocks are stopped.
- **Sleep mode**: wherein power to the CPU and most of the peripherals is turned off.
Power Management Standards by Linux

- APM
- ACPI

- Both have their roots in the x86 architecture.
APM

- More dependent on the BIOS for doing power management.
- Introduced by Microsoft and Intel allocated most of the power management control to the BIOS.
- BIOS monitors the list of the devices.
Disadvantages of APM

- BIOS may choose to put a system into low-power mode when the system is actually involved in a computationally intensive task.
- This is because the BIOS assumes the system state by just looking at the activity on the IO ports such as the keyboard.
Disadvantages of APM

- BIOS detects activity only on devices that are residing on the motherboard. Devices not on the motherboard such as those plugged into the USB bus cannot participate in power management.
- Each BIOS had its own set of limitations and interfaces (and bugs!).
ACPI

- Power management should be handled by the OS because it can make the best judgment with respect to the system load and hence it can manage the power of the CPU and the peripherals.

- ACPI makes system still dependent on BIOS but to a lesser degree.
ACPI Machine Language (AML)

- Interpreted language
- An OS can operate on the devices without knowing much about the devices.
Unified Driver Framework for Power Management

- Device drivers are a central piece in the power management software.
- Kernel separates the device drivers from the actual power management software in the kernel by allowing the device drivers to register themselves before they participate in the power management.
- `pm_register()` call
- `pm_dev_idle` interface to identify idle devices so that they can be put to sleep.
Power Management Applications

- Linux kernel provides mechanisms to implement power management but leaves the decision making to user land.
- Both APM and ACPI come with applications that are used to initiate system suspend/standby transition.
  - apmd daemon uses /proc/apm interface
  - acpid daemon listens on a file /proc/acpi/event