Requirements for embedded system

- Advanced embedded system
  - Expansion of application field
    - Complication GUI, Network
  - More complicated, grow up code sizes
    - Car navigation system, cellular telephone, digital TV

- The requirements of embedded system
  - Real-time
    - Multimedia applications
    - In order to process video, audio streams, it needs soft real-time control.
    - Even in overload condition, predictable control is necessary for them.
  - Responsiveness
    - Key inputs
    - Even if the multimedia applications are running in the foreground, the
      responsiveness is required.
  - Avoid CPU occupation
    - Real-time applications
      - In case of the download, the programs would be buggy or malicious,
        they will use up the whole system resources.
      - Resource protection mechanism is needed especially for real-time
        applications

- Appropriate CPU resource management system is needed
**Linux Scheduling policies**

<table>
<thead>
<tr>
<th>policy</th>
<th>contents</th>
<th>priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time</td>
<td>SCHED_FIFO</td>
<td>First in first out</td>
</tr>
<tr>
<td></td>
<td>SCHED_RR</td>
<td>Round robin</td>
</tr>
<tr>
<td>Time-sharing</td>
<td>SCHED_OTHER</td>
<td>Time-sharing</td>
</tr>
</tbody>
</table>

- **Real-time (static priority)**
  - Static Priority (POSIX 1003.1b)

- **Time-sharing (dynamic priority)**
  - At regular time interval (time slice)
  - Switch processes compared with their priorities
  - Priority is decided by the execution + sleep time
  - Set higher priority to the interactive process

Default policy is timesharing
RT scheduling policies are used with system call.

**CABI (CPU Accounting and Blocking Interfaces)**

- **Purpose**
  - Provide a framework for the CPU resource management

- **Approach**
  - Control the consumptions of the CPU resources quantitatively
    - CPU should be limited for each application or application groups
    - e.g.) The audio video application $\rightarrow$ 60%
      The downloaded applications $\rightarrow$ 40%

- **Design policy**
  - **Fine-grained**
    - With High Resolution Timer
  - **Simple**
    - Easy to use the interfaces and services
  - Independent from the scheduler
    - Not change the Linux scheduler

- **Graph**
  - Applications
    - Downloaded Applications
      - 40%
    - Audio & Video Application
      - 60%
Accounting model

- Two parameters: T (period), C (computation time)
  - needed to control the execution time of the application

\[
\text{The CPU usage (\%)} = \frac{\text{Computation time}}{\text{Period}} \times 100
\]

- Process or process group can not excessively use the CPU resources than their proportion

![Diagram showing accounting model](image)

System Architecture

- Interfaces
  - Accounting API Library
  - System Call Interfaces

- Functions
  - Timer Management
  - AO Management

- It independent from the kernel
  - Only a few hooks
    - Schedule, fork, exit
    - ISR

![Diagram showing system architecture](image)
Guaranteed responsiveness is required for TS applications

- **CABI provides a framework for building real-time system**
  - Set appropriate priorities for real-time applications
    - By using algorithms like Rate Monotonic, EDF, with admission control
    - Developer can give guarantees for real-time applications

- **There are no guarantees for time-sharing applications**
  - Recently, embedded developers are tend to use the time-sharing (normal) to develop a new embedded application
    - Because no need to adjust their priorities
  - Time-sharing applications can’t get responsiveness
    - If real-time application is running in foreground
    - The priorities of time-sharing are always lower than the real-time applications
      - Linux scheduler constraints
    - Need a function to set higher priorities and guarantee them to use some resource

---

CPU Reservation

- **Priority Boost approach**
  - Time-sharing processes are boosted temporarily to real-time processes, and minimum resource is reserved
  - CPU resource to a particular process which takes care of GUI
  - Responsibility of time-sharing process is increased
Conclusion

- Background
  - Overview of the embedded system resource requirements

- Proposal
  - CABI (CPU Accounting and Blocking Interfaces)
    - This can effectively control the CPU consumption of a process or processes
  - Priority boost approach
    - It provides CPU Reservation especially for time-sharing application

- Demonstration
  - Penguin will show you how our system control the applications

Thank you!

Source and Documentations

- Source and example applications
  - Sourceforge
    - US: http://sourceforge.jp/
  - Emblix
    - http://www.emblix.org

- Documentations
  - Specification
  - TEST specification
    - Sourceforge

- Papers

- Patch release
  - Kernel 2.4
    - ppc, sh, mips, arm, x86
  - Kernel 2.6:
    - sh, mips, arm