Demo I

- In class on 3/1 and 3/3

- **10 min** per team
  - 9-min demo + 1-min Q&A

- Must show something **real**

- Submit slides before the class of your demo
Demo II

- In class on **3/29** and **3/31**

- **10 min** per team
  - 9-min demo + 1-min Q&A

- Substantial progress towards final demo

- Submit slides before the class of your demo
Linux Tutorial

- 2/17 (Thursday)

- Lecture by Ruixuan (Corey) Dai

- Use a different Zoom link
RTOS
Real-Time Operating Systems

Chenyang Lu
Since the application interacts with the physical world, its computation must be completed under a time constraint.

CPS are built from, and depend upon, the seamless integration of computational algorithms and physical components. [NSF]
Cyber-Physical Systems (CPS)

Cyber-Physical Boundary

[Images of a robot and a self-driving car]
Interactive Cloud Services (ICS)

Need to respond within 100ms for users to find responsive*. 

* Jeff Dean et al. (Google) "The tail at scale." Communications of the ACM 56.2 (2013)
Interactive Cloud Services (ICS)

Need to respond within 100 ms for users to find responsive*. E.g., web search, online gaming, stock trading etc.

* Jeff Dean et al. (Google) "The tail at scale." Communications of the ACM 56.2 (2013)
General-Purpose vs. Real-Time

General-purpose systems
- Fairness to all tasks (no starvation)
- Optimize throughput
- Optimize average performance

Real-time systems
- Meet deadlines.
- Fairness or throughput is not important
- Hard real-time: worry about worst case performance
Consequence of Deadline Miss

- **Hard deadline**
  - System fails if missed.
  - Goal: guarantee no deadline miss.

- **Soft deadline**
  - User may notice, but system does not fail.
  - Goal: meet most deadlines most of the time.
OS Support for Real-Time

➢ Real-Time OS
  - Required for hard real-time systems

➢ Real-time extensions to general-purpose OS
  - Suitable for soft real-time systems
RTOS: Features for Efficiency

- Small
- Minimal set of functionality
- Fast context switch
- Fast and time bounded response to interrupts
- Fixed or variable partitions of memory
  - May not support paging or virtual memory
  - May support locking code and data in memory
- Sequential file that can accumulate data at a fast rate
  - May be memory-based
<table>
<thead>
<tr>
<th>Name</th>
<th>Code Size</th>
<th>Target CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>pOSEK</td>
<td>2K</td>
<td>Microcontrollers</td>
</tr>
<tr>
<td>pSOSSystem</td>
<td></td>
<td>PII-&gt;ARM Thumb</td>
</tr>
<tr>
<td>VxWorks</td>
<td>286K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>QNX Nutrino</td>
<td>&gt;100K</td>
<td>Pentium II -&gt; NEC</td>
</tr>
<tr>
<td>QNX RealTime</td>
<td>100K</td>
<td>Pentium II -&gt; SH4</td>
</tr>
<tr>
<td>OS-9</td>
<td></td>
<td>Pentium -&gt; SH4</td>
</tr>
<tr>
<td>Chorus OS</td>
<td>10K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>ARIEL</td>
<td>19K</td>
<td>SH2, ARM Thumb</td>
</tr>
<tr>
<td>Creem</td>
<td>560 bytes</td>
<td>ATMEL 8051</td>
</tr>
</tbody>
</table>
RTOS: Features for Real-Time

- **Preemptive priority scheduling**
  - At least 32 priority levels, commonly 128-256 priority levels
  - Priority inheritance/ceiling protocol

- **System calls**
  - Bounded execution times
  - Short non-preemptible code

- **High-resolution system clock**
  - Resolution down to nanoseconds
  - But it takes about a microsecond to process a timer interrupt
Other Important Features

- Conformance to standards
  - Real-Time POSIX API
  - TCP/IP

- Modularity and configurability
  - Small kernel
  - Pluggable modules
Example: VRTX

- **VRTXsa**
  - RT-POSIX compliant
  - Full real-time support

- **VRTXmc**
  - Optimized for power and footprint

- First RTOS certified by FAA
  - 100% code coverage in testing

- Runs the Hubble Space Telescope
Real-Time Extensions to General OS

- Generally slower and less predictable than RTOS
- More functionality and development support
- Standard interfaces
- Suitable for soft real-time applications
How Real-Time Is Linux?

- I believe that Linux is ready to handle applications requiring sub-millisecond process-scheduling and interrupt latencies with 99.99+ percent probabilities of success. No, that does not cover every imaginable real-time application, but it does cover a very large and important subset.
- There are patches out there that are expected to provide latencies in the tens of microseconds. These patches need some work, but are maturing quickly.

Paul McKenney, IBM Linux Technology Center

*Shrinking slices: Looking at real time for Linux, PowerPC, and Cell*
Why isn’t Linux real-time?

- The Linux kernel only allows a process to preempt another under certain circumstances:
  - when the CPU is running **user-mode** code;
  - when kernel code returns from a system call or an interrupt to user space;
  - when kernel code blocks on a mutex, or yields control to another process.

- If kernel code is executing, a high priority thread cannot preempt the kernel code until it yields control.

- In the **worst case**, the latency could potentially be **hundreds milliseconds** or more.

Source: https://rt.wiki.kernel.org/
Linux Scheduling

- Real-time scheduling class
  - Fixed priority
    - `SCHED_FIFO`: First-In-First-Out
    - `SCHED_RR`: Round-Robin
  - Earliest Deadline First
    - `SCHED_DEADLINE`

- Non-real-time scheduling class
  - `SCHED_NORMAL`

- Priorities
  - Real-time: 0 – 99
  - Non-real-time: 100 – 139
Real-time Linux

- **Core kernel modifications:** patches
  - Require modifications to Linux kernel
  - Example: `CONFIG_PREEMPT_RT` patch

- **Compliant kernels:** modified native RTOS
  - Linux binaries can run without modifications
  - Example: LynxOS

- **Dual kernels:** real-time kernel sits below Linux
  - Real-time kernel traps all interrupts and schedules all processes
  - Linux runs as a low-priority process
  - Example: RT-Linux (FSLabs)
CONFIG_PREEMPT_RT Patch

- Convert Linux into a **fully preemptible** kernel.

- **Locks**
  - Make in-kernel locking-primitives preemptible with rt-mutexes.
  - Critical sections protected by spinlock_t and rwlock_t are preemptible.
  - Priority inheritance for in-kernel spinlocks and semaphores.

- Convert **interrupt** handlers into preemptible kernel threads.

- Convert **timer** API for high resolution kernel timers plus one for timeouts, leading to user space POSIX timers with high resolution.

*Source: https://rt.wiki.kernel.org/*
Multi-core Real-Time Scheduler

- Push-pull scheduler schedules tasks across CPUs.

- Every CPU has a runqueue.
  - Avoid contention over a global runqueue
  - Need to enforce global priority order in a decentralized fashion

- Push: consider all the runqueues to find one of a lower priority than the task being pushed.
  - Push a task when it wakes up on a runqueue running a task of higher priority
  - Push a task when a higher-priority task on the same runqueue preempts it

- Pull: whenever a runqueue is about to schedule a task with a lower priority than the previous one, it checks whether it can pull a task of a higher priority from other runqueues.
Example: Push-Pull

Figure 7. An example scenario of real-time tasks queued on runqueues, indicating the current running tasks and the next task to be run on each runqueue, before the push operation.

Figure 10. Layout of real-time on the different runqueues after push-pull operations.

- Real-time kernel traps all interrupts and schedules all processes
- Linux runs as a low-priority process
- Real-time applications cannot take advantage of Linux calls
More Information

