Operating Systems

Chenyang Lu
CSE 467S Embedded Computing Systems
In 2001, about a year after I joined IBM, I got involved with the Sony-Toshiba-IBM initiative that's leading up to the Sony PlayStation 3. Of course, game systems have extremely severe real-time requirements. All the gaming systems I have seen recently offer sub-reflex response (and, yes, I do have three teenagers, so I have seen a few video games). Gamers will not put up with jerky response, so one-second response times just will not cut it; milliseconds rather than seconds are required.

So we are in the very interesting state where gaming and entertainment are major factors driving the technology. When was the last time you heard someone say, "Hey, I bought a new PC and Excel really runs a lot faster"? I used to hear that, back in the early '90s.

Paul McKenney, IBM Linux Technology Center

*Shrinking slices: Looking at real time for Linux, PowerPC, and Cell*
Basic Functions

- OS controls resources:
  - who gets the CPU;
  - when I/O takes place;
  - how much memory is allocated;
  - power management.

- Application programs run on top of OS services

- Challenge: manage multiple, concurrent tasks.
Example: Engine Control

Concurrent tasks:
- spark control
- crankshaft sensing
- fuel/air mixture
- oxygen sensor
Example: Sensor

- Sensing
  - Sampling sensors

- Communication
  - Send/receive data
  - Routing

- Storage
  - Read/write Flash

- Computation
  - Data processing
Life without OS

Code turns into a mess:
- interrupt one task for another
- spaghetti code

```c
A_code();
...
B_code();
...
if (C) C_code();
...
A_code();
...
switch (x) {
    case C: C();
    case D: D();
    ...
...```

Chenyang Lu
Operating Systems

- TinyOS
- POSIX
- Real-Time OS
Tasks vs. Interrupts

- **Tasks do intensive computations**
  - FIFO scheduling
  - Bounded number of pending tasks

- **Events handle interrupts**
  - Interrupts trigger lowest level events
  - Events can signal events, call commands, or post tasks

- **Two priorities**
  - Interrupt
  - Tasks
Scheduling

- Interrupt has higher priority than task
- An interrupt preempts a task
  - Respond quickly
  - Ex. get bit out of radio hw before it gets lost
- A task cannot preempt another task
  - Reduce context switch → efficiency
- Scheduler puts processor to sleep when
  - no event/command is running and
  - task queue is empty
Operating Systems

- TinyOS
- POSIX
- Real-Time OS
IEEE standards for application portability between Unix variants.

- IEEE 1003.1 defines a Unix-like OS interface.
- IEEE 1003.2 defines the shell and utilities
- IEEE 1003.4 defines real-time extensions.

Supported by many operating systems

- Variants of UNIX: AIX, HP-UX, Solaris, Linux
- Many commercial RTOS
A process is a **unique execution** of a program.
- Several copies of a program may run simultaneously.

A process has its own **context**:
- Data in registers, PC, status.
- Stored in **Process Control Block (PCB)**

**Thread**: lightweight process
- Threads share memory space (not registers) in a same process.

**OS** manages processes and threads.
Context Switch

process 1

process 2

... 

memory

PC

registers

CPU
Create a process with fork:

- parent process keeps executing old program;
- child process executes new program.

```c
pid = fork();
if (pid == 0 { /* child operations */
} else { /* parent operations*/
}
```
execv()\

Overlay child code:

```
childid = fork();
if (childid == 0) {
    execv("mychild",childargs);
    exit(1);
}
```

file with child code
A process can be in one of three states:
- **executing** on the CPU;
- **ready** to run;
- **waiting** for data.
Process Management

- OS keeps track of:
  - process priorities;
  - scheduling state;
  - process control block.

- Processes may be created:
  - statically before system starts;
  - dynamically during execution.

- OS controls context switches and what process runs.
Priority Scheduling

- Every process has a priority.

- CPU goes to the highest-priority ready process.
  - Fixed vs. dynamic priority
  - Preemptive vs. non-preemptive
Preemptive Priority Scheduling

- Each process has a fixed priority (1 highest);
- $P_1$: priority 1; $P_2$: priority 2; $P_3$: priority 3.
Preemptive Priority Scheduling

- Most common real-time scheduling approach
  - Real-Time POSIX
  - Real-time priorities in Linux, Solaris, and Windows
  - Most RTOS

- Not the only possible way
  - Non-preemptive
  - Clock-driven scheduling
  - Reservation-based scheduling
Race Conditions

- **POSIX**
  - preemptive scheduling $\Rightarrow$ race among processes.

- **TinyOS**
  - non-preemptive scheduling $\Rightarrow$ no race among tasks.
Semaphores

- OS primitive for controlling access to critical regions.
  - Get access to semaphore S with \texttt{wait(S)}.
  - Perform critical region operations.
  - Release semaphore with \texttt{signal(S)}.

- Mutex: only one process can hold a mutex at a time.
Example: Use Semaphore

```c
wait(mutex_busy);
localBusy = busy;
busy = TRUE;
signal(mutex_busy);

if (!localBusy)
    call ADC.getData();
```
Test and Set Instruction

```c
event result_t Timer.fired() {
    localBusy = test_set(busy);
    if (!localBusy)
        call ADC.getData();
    return SUCCESS;
}
```

- Supported by many processors
- OS uses test_set to implement semaphores
Supervisor Mode

- The mode in which the OS usually runs.
- Provide protective barriers between applications and OS.
  - Prevent applications from corrupting OS data.
- Can do the following only in the supervisor mode
  - Access the kernel address space
  - Execute privileged instructions
    - Example: Set real-time priority
  - Access special hardware
- Careful with memory access when
  - programs run in supervisor mode
  - processor has no supervisor mode
Trap (Software Interrupt)

- Enter supervisor mode.
- Make system calls.
  - Open file, read from network…
- Example: ARM
  - Use SWI instruction to enter supervisor mode:
    ```
    SWI CODE_1
    ```
  - Sets PC to 0x08.
  - Argument to SWI is passed to supervisor mode code.
Exception

- Internally detected error.
  - Example: div by 0

- Caused by instruction execution.
  - Unpredicted

- Build on top of interrupt mechanism.

- Usually prioritized and vectorized.
The Differences

- **Interrupt**: generated by external devices
- **Exception**: generated by CPU due to software errors
- **Trap**: enter supervisor mode
Operating Systems

- TinyOS
- POSIX
- Real-Time OS
OS Support for Real-Time

- Real-Time OS
- Real-time extensions to general-purpose OS
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 fast rate
  - May be memory-based
# Code Size

<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>pSOSystem</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
Development Environment

- **Self-hosted: applications are developed on target platforms**
  - OS must support compilers, debuggers, performance profilers
  - Large memory demand
  - Example: LynxOS

- **Cross-platform development**
  - Example: Tornado environment for VxWorks
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.

- The Linux 2.6 kernel, if configured carefully and run on fast hardware, can provide sub-millisecond interrupt and process scheduling latencies with extremely high probabilities of success. 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 will only allow one 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 explicitly yields control to another process.

- If kernel code is executing, a high priority thread cannot preempt the kernel until the kernel code explicitly 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
  - Scheduling threads with a same priority
    - SCHED_FIFO: First-In-First-Out
    - SCHED_RR: Round-Robin

- Non-real-time scheduling class (SCHED_NORMAL)
  - Priority is adjusted dynamically to favor I/O bound threads

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

- **Core kernel modifications:** patches
  - Require modifications to Linux kernel
  - Example: RT-Preempt

- **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
  - Real-time applications cannot take advantage of Linux calls
  - Example: RT-Linux (FSLabs)
RT-Preempt Patch

- Officially called CONFIG_PREEMPT_RT patch.

- Convert Linux into a fully preemptible kernel.

- Locks
  - Making in-kernel locking-primitives preemptible through reimplemented with rtmutexes.
  - Critical sections protected by spinlock_t and rwlock_t are preemptible.
  - Priority inheritance for in-kernel spinlocks and semaphores.

- Converting interrupt handlers into preemptible kernel threads.

- Converting the timer API into separate infrastructures 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.

- Push: considers all the run queues within its root domain to find the one that is of a lower priority than the task being pushed.
  - Push a lower-priority task when it wakes up but is on a runqueue running a task of higher priority
  - Push a low-priority task when a higher-priority task on the same runqueue wakes up and preempts it

- Pull: whenever a run queue is about to schedule a task that is lower in priority than the previous one, it checks to see whether it can pull tasks of higher priority from other run queues.
• 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
Reading

- Textbook 6.1-6.4
