SE350: Operating Systems

Lecture 1: Introduction
Course Mechanics

• All information, lecture notes, and announcements will be posted on the course website
  • https://ece.uwaterloo.ca/~smzahedi/crs/osW19

• Course is added to Piazza and LEARN
  • Links are available on the course website
Teaching Team

• Instructor
  • Seyed Majid Zahedi
  • Office hours Tuesdays from 12:30 to 13:30

• Lab instructor
  • Rollen DSouza

• Teaching assistants
  • Andrew Gapic
  • Guang Zhi Xiao
  • Liuyang Ren
  • Waleed Qadir Khan
Readings

• Textbook
  
  • Operating Systems: Principles and Practice (2nd Edition)

• Optional references
  
  • Operating Systems: Three Easy Pieces (Freely Available Online)
  
  • Operating System Concepts (10th Edition)
Evaluation

• Midterm exam: 15%
  • Feb. 12th from 18:30 to 20:20

• Lab project: 35%
  • Group sign-up deadline: Jan. 18th
  • Lab due dates: Feb. 4th, Mar. 4th, and Mar. 18th
  • Final report: Apr. 5th

• Final exam: 50%
  • To be announced
RTX Project Overview

• Project description
  • What is this project about?
  • What are the deliverables?

• Logistics
  • How to form a group?
  • Help! My group sucks!
  • Lab times

• About the Lab
  • What lab facilities are provided?
  • How to seek help outside lab?
RTX Project

- You will design, implement and test a **real-time kernel**
  - A basic multiprogramming environment
  - Five priority queues and preemption
  - Simple memory management
  - Message-based Inter-Process Communication (IPC)
  - System console Input/Output (UART 0/1)
  - Debugging support (UART 1/0)
- Your RTX operates on Keil MCB1700 (ARM Cortex-M3) Boards
Deliverables

• Four deliverables in total
  • RTX Implementations (P1, P2 and P3)
  • RTX Demonstrations (Weeks 5, 9 and 11)
  • RTX P4: Final project report (30-40 pages) + Code

• Important Policies
  • Three grace days without penalty
  • 10% per day late submission penalty afterwards
  • Zero tolerance policy for plagiarism
    • We use Moss
    • We follow UW Policy 71 for any single incident
Project Groups

• Forming a project group
  • Four members (three is acceptable for special cases)
  • Within the same lab section as much as possible
  • Email your group to Rollen to signup by 23:59 on Jan. 18th

• Leaving a project group
  • One week notice in writing before nearest deadline
  • Only one split-up is allowed
  • All students involved lose one grace day

• Everyone in the same group gets the same grade
Scheduled Labs

• Scheduled lab sessions are in odd weeks
  • Mon., Wed., and Fri. (excluding reading week)
  • Help sessions: Weeks 1, 3 and 7
  • Demo sessions: Weeks 5, 9 and 11
Lab Facilities

- Lab is located in E2-2363
  - Nexus computers
  - Keil MCB1700 LPC1768 (ARM Cortex-M3) Boards
  - MDK-ARM MDK-Lite ed.V4.60 (32KB code size limit)
    - RealView Compilation Tools are included
    - The simulator is good for development work outside the lab*

- SE350 shares the lab with ECE222 students

- ECE222 has scheduled labs during even weeks
  - Tue., Wed., and Thu. at the same time-slot

* It is my understanding that the simulator does not perfectly match behaviour on hardware, so be forewarned: test your code on the hardware well before the deadline!
Seeking Help Outside Lab

• Piazza
  • Looking for group partners
  • Lab/Project administration, Keil IDE, and project Q&A
  • Target response time: one business day
    • Do not wait till the last minute to ask questions

• Please avoid individual emails
  • Use private piazza posts
  • Reserve emails for directed, confidential conversations

• Office hours during even weeks (every week for me)
• By appointment
Your First Task

• Form a group of 4
• Recall how to work with MCB 1700 and Keil
Main Points (for today)

• Operating system definition
  • Software to manage a computer’s resources for its users and applications

• OS challenges
  • Reliability, security, responsiveness, portability, etc.

• OS history
  • How did we get here?

• Device I/O
  • How does I/O work?
What is an Operating System?

• Software to manage a computer’s resources for its users and applications
Operating System Roles

• Referee
  • Resource allocation among users, applications
  • Isolation of different users, applications from each other
  • Communication between users, applications

• Illusionist
  • Each application appears to have entire machine to itself
  • Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport

• Glue
  • Libraries, user interface widgets, etc.
Example: File Systems

- Referee
  - Prevent users from accessing each other’s files without permission
  - Even after a file is deleted and its space re-used

- Illusionist
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save

- Glue
  - Named directories, printf, etc.
OS Challenges: Web Service Example

- How does the server manage many simultaneous client requests?
- How do we keep the client safe from spyware embedded in scripts on a web site?
- How do we make updates to the web site so that clients always see a consistent view?
OS Challenges (cont.)

• Reliability
  • Does the system do what it was designed to do?

• Availability
  • What portion of the time is the system working?
  • Mean Time To Failure (MTTF), Mean Time to Repair

• Security
  • Can the system be compromised by an attacker?

• Privacy
  • Data is accessible only to authorized users
OS Challenges (cont.)

• Performance
  • Latency/response time
    • How long does an operation take to complete?
  • Throughput
    • How many operations can be done per unit of time?
  • Overhead
    • How much extra work is done by the OS?
• Fairness
  • How equal is the performance received by different users?
• Predictability
  • How consistent is the performance over time?
OS Challenges (cont.)

• Portability
  • For programs:
    • Abstract virtual machine (AVM)
    • Application programming interface (API)
  • For the operating system
    • Hardware abstraction layer
Portability

- Compilers
- Web Servers
- Source Code Control
- Databases
- Word Processing
- Web Browsers
- Email
- Portable OS Library
- System Call Interface
- Portable Operating System Kernel
- x86
- ARM
- PowerPC
- 10Mbps/100Mbps/1Gbps Ethernet
- 802.11 a/b/g/n
- SCSI
- IDE
- Graphics Accelerators
- LCD Screens
Microprocessor Trends

[Moore’s Law 1965]

[End of Dennard Scaling 1974]

[Dark Silicon 2011]

[Frequency (MHz)]

[Number of Cores]

End of Dennard Scaling

[R. Dennard et al. 1974]
## Computing Trends

### Approximate Computer Server Performance Over Time

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniprocessor speed (MIPS)</td>
<td>1</td>
<td>200</td>
<td>2500</td>
<td>2.5K</td>
</tr>
<tr>
<td>CPUs per computer</td>
<td>1</td>
<td>1</td>
<td>10+</td>
<td>10+</td>
</tr>
<tr>
<td>Processor MIPS/$</td>
<td>$100K</td>
<td>$25</td>
<td>$0.20</td>
<td>500K</td>
</tr>
<tr>
<td>DRAM Capacity (MiB)/$</td>
<td>0.002</td>
<td>2</td>
<td>1K</td>
<td>500K</td>
</tr>
<tr>
<td>Disk Capacity (GiB)/$</td>
<td>0.003</td>
<td>7</td>
<td>25K</td>
<td>10M</td>
</tr>
<tr>
<td>Home Internet</td>
<td>300 bps</td>
<td>256 Kbps</td>
<td>20 Mbps</td>
<td>100K</td>
</tr>
<tr>
<td>Machine room network</td>
<td>10 Mbps (shared)</td>
<td>100 Mbps (switched)</td>
<td>10 Gbps (switched)</td>
<td>1000</td>
</tr>
<tr>
<td>Ratio of users to computers</td>
<td>100:1</td>
<td>1:1</td>
<td>1:several</td>
<td>100+</td>
</tr>
</tbody>
</table>

---

Figure 1.8: Approximate computer server performance over time, reflecting the most widely used servers of each era: in 1981, a minicomputer; in 1997, a high-end workstation; in 2014, a rack-mounted multicore server. MIPS stands for “millions of instructions per second,” a measure of processor performance. The VAX 11/782 was introduced in 1982; it achieved 1 MIP. DRAM prices are from Hennessey and Patterson, “Computer Architecture: A Quantitative Approach.” Disk drive prices are from John McCallum. The Hayes smartmodem, introduced in 1981, ran at 300bps. The 10 Mbps shared Ethernet standard was also introduced in 1981. One of the authors built his first operating system in 1982, used a VAX at his first job, and owned a Hayes to work from home.

Despite these changes, operating systems still face the same conceptual challenges as they did fifty years ago. To manage computer resources for applications and users, they must allocate resources among applications, provide fault isolation and communication services, abstract hardware limitations, and so forth. Tremendous progress has been made towards improving the reliability, security, efficiency, and portability of operating systems, but much more is needed. Although we do not know precisely how computing technology or application demand will evolve over the next 10-20 years, it is highly likely that these fundamental operating system challenges will persist.

---

Early Operating Systems

Computers were expensive; users would wait. The first operating systems were runtime libraries intended to simplify the programming of early computer systems. Rather than the tiny, inexpensive yet massively complex hardware and software systems of today, the first computers often took up an entire floor of a warehouse, cost millions of
Early Operating Systems: Very Expensive Computers

- One user/application at a time
  - Had complete control of hardware
  - OS was runtime library
  - Users would stand in line to use the computer

- Batch systems
  - Keep CPU busy by having a queue of jobs
  - OS would load next job while current one runs
  - Users would submit jobs, and wait, and wait, and wait …
Time-Sharing Operating Systems

• Multiple users on computer at the same time
  • Multiprogramming: run multiple programs at same time
  • Interactive performance: try to complete everyone’s tasks quickly
• As computers became cheaper, more important to optimize for user time, not computer time
Today’s Operating Systems

• Smartphones
• Embedded systems
• Laptops
• Tablets
• Virtual machines
• Data center servers
 Tomorrow’s Operating Systems

• Giant-scale data centers
• Increasing numbers of processors per computer
• Increasing numbers of computers per user
• Very large scale storage
Device I/O

- OS needs to communicate with devices
  - Network, disk, video, keyboard, mouse, etc.
- There are 2 methods to access I/O devices
  - Memory mapped I/O (MMIO)
  - Port mapped I/O (PMIO)
Memory Mapped vs Port Mapped I/O

- **Address space**
  - MMIO: Physical address space is shared between DRAM and I/O
  - PMIO: Address space for I/O is distinct from main memory

- **I/O-access Instructions**
  - MMIO: Same as memory-access
    - MOV register, memAddr // To read
    - MOV memAddr, register // To write
  - PMIO: Distinct access instructions
    - IN register, portAddr // To read
    - OUT portAddr, register // To write
Programmed I/O

• I/O operations take time (physical limits)
• OS pokes I/O memory on device to issue request
• OS **polls** I/O memory to wait until I/O is done
• Device completes, stores data in its buffers
• Kernel copies data from device into memory
Interrupt-Driven I/O

- OS pokes I/O memory on device to issue request
- CPU goes back to work on some other task
- Device completes, stores data in its buffers
- Device triggers **interrupt** to signal I/O completion
- Device specific handler code runs
- When done, resume previous work
Direct Memory Access (DMA) I/O

- OS sets up DMA controller with # words to transfer
- OS commands I/O device to initiate data transfer
- DMA controller provides address and r/w control
- Device reads/writes data directly to main memory
- DMA controller interrupts CPU on I/O completion
Programmed I/O vs DMA

• Programmed (and interrupt-driven) I/O
  • I/O results stored in the device
  • CPU reads and writes to device memory
  • Each CPU instr. is an uncached read/write (over I/O bus)

• Direct memory access (DMA)
  • I/O device reads/writes the computer’s memory
  • After I/O interrupt, CPU can access results in memory
Faster DMA I/O: Buffer Descriptors

• OS sets up a queue of I/O requests

• Each entry in the queue is a buffer descriptor
  • It contains I/O operation and buffer to contain data
  • Buffer descriptor itself is DMA’ed!

• CPU and device I/O share the queue
  • I/O device reads from front
  • CPU fills at tail

• Interrupt only if buffer empties/fills
Device Interrupts

• How do device interrupts work?
  • Where does the CPU run after an interrupt?
  • What is the interrupt handler written in? C? Java?
  • What stack does it use?
  • Is the work the CPU had been doing before the interrupt lost forever?
  • If not, how does CPU know how to resume that work?
Interrupt Vector

- Table set up by OS kernel; pointers to code to run on different events

```
handleTimerInterrupt() {
    ... 
}

handleDivideByZero() {
    ... 
}

handleSystemCall() {
    ... 
}
```
Interrupt Masking

- Interrupt handler runs with interrupts off
  - Re-enabled when interrupt completes
- OS kernel can also turn interrupts off
  - E.g., when determining the next process/thread to run
- On x86
  - CLI: disable interrupts
  - STI: enable interrupts
- Only applies to the current CPU (on a multicore)
- We’ll need this to implement synchronization
Challenge: Saving/Restoring State

• We need to be able to interrupt and transparently resume execution
  • I/O device signals I/O completion
  • Periodic hardware timer to check if app is hung
  • Multiplexing multiple apps on a single CPU
  • Code unaware it has been interrupted!

• Not just the program counter
  • Condition codes, registers used by interrupt handler, etc.
Question

• What (hardware, software) do you need to be able to run an untrustworthy application?
Another Question

• How should an operating system allocate processing time between competing uses?
  • Give the CPU to the first to arrive?
  • To the one that needs the least resources to complete?
  • To the one that needs the most resources?
Acknowledgment

• This lecture is a slightly modified version of the one prepared by Tom Anderson
• RTX Project slides are adapted from Irene Huang’s slide deck on the course