Reminders

- Send your team members to Corey **today**
  - Corey will help match up remaining students
  - Request permission for a two-member team

- Proposal presentation: in class **next Tuesday**

- Proposal submission: by 11:59pm **next Tuesday**

- See **Project Guidelines** for details
Operating Systems

Chenyang Lu
Diverse Platforms

**TelosB**
- TI MSP430 microcontroller, 4/8 MHz, 8 bit
- Memory: 10KB data, 48 KB program
- IEEE 802.15.4 radio: max 250 Kbps

**Raspberry Pi 4**
- Quad core Cortex-A72 64-bit SoC, 1.5GHz
- 2GB-8GB SDRAM
- IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
**IoT OS**

- **Contiki**: open-source, multi-threaded OS, plain C.
- **Amazon FreeRTOS**: open-source FreeRTOS kernel + libraries to securely connect devices to AWS cloud services.
- **Arm Mbed**: open-source OS based on an Arm Cortex-M microcontroller.
- **Windows 10 IoT Core**: Windows 10 optimized for both ARM and x86/x64 devices.
- **Linux**
Amazon FreeRTOS

IoT operating system for microcontrollers that extends the FreeRTOS kernel with libraries for security, connectivity, and updateability.

Microcontroller-based smart lighting

Amazon FreeRTOS Bluetooth Low Energy
Secure direct connection to mobile devices via Bluetooth Low Energy

AWS IoT Core
AWS IoT Device Management
AWS IoT Device Defender
AWS IoT Analytics

Direct connection to cloud services like AWS IoT Core devices

https://aws.amazon.com/freertos
Linux

A Brief History: https://youtu.be/aurDHyL7bTA
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"?

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
POSIX (Portable Operating System Interface)

- 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: Linux, MacOS, AIX, HP-UX, Solaris.
  - Many commercial RTOS.
Process

- 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, Program Counter (PC), status.
  - Stored in **Process Control Block (PCB)**

- **Thread**: lightweight process
  - Threads share memory space in a same process.

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

CPU

process 1

process 2

memory

PC

registers

CPU
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 ready process with the highest priority.
  - 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
  - Most RTOS

- *Not* the only possible way
  - Non-preemptive
  - Clock-driven scheduling
  - Reservation-based scheduling
Semaphores

- **OS primitive for controlling access to critical regions.**
  - Get access to semaphore S with `sem_wait(S)`.
  - Perform critical region operations.
  - Release semaphore with `sem_post(S)`.

- **Mutex:** only one process can hold a mutex at a time.

```c
sem_wait(mutex_info_bus);
Write data to info bus;
sem_post(mutex_info_bus);
```
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.