OS Overview

Operating System Lab-in-a-Box

OS Components

- Process management
  - How to run a program?
  - How to allocate resources?
- Memory management
  - Memory allocation
  - Protection
  - Virtual memory
- File systems
  - Secondary storage
- I/O
  - Device Drivers
- Network
- Security
- GUI

Process

- Operating system deals with various kind of activities – process
  - User applications and system applications
  - Abstracted as process - all the execution context
  - An instance of a running program – possible to have multiple processes running the same program at the same time
  - Independent memory space
  - Creation and destruction
  - Schedule
  - Communication
  - Concurrency

Memory

- How to organize processes' memory
  - Programs are stored in memory as well as data
  - Multiprogramming supports
  - Sharing data between processes
  - Pages
  - Virtual memory – use the secondary memory, RAM as a cache

Files and I/O

- File – an abstraction of a bulk of information
  - Secondary storage
  - Standard operations such as create, delete, copy and paste
  - Advanced, searching, backup and so on

- I/O
  - As shown in the previous example
  - Requires device-specific knowledge
  - Device drivers and standard interface

Operating System Structure

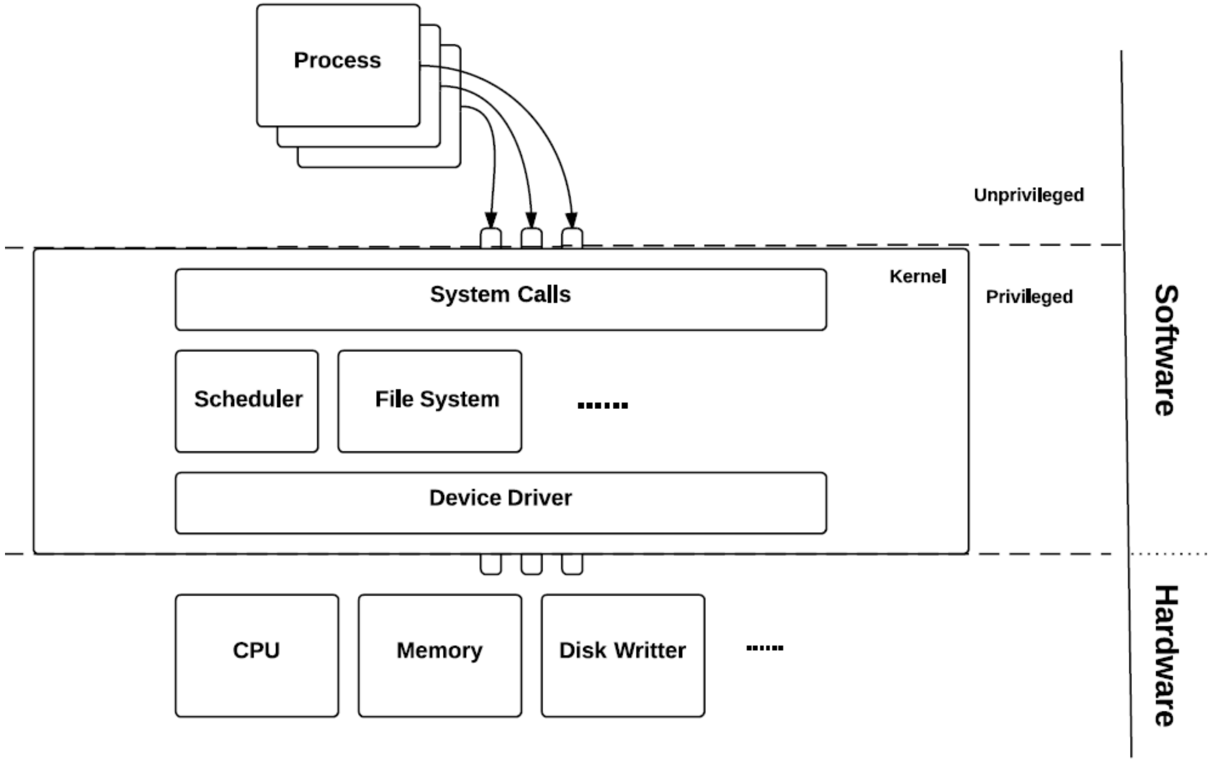
- Different components interact with each other
- Not so straightforward as to how to organize all the components
- A challenging software engineering problem
  - Reliability
  - Backwards compatibility
  - Extensibility
  - Portability

Monolithic

- Traditional OS structure
  - Single program includes all kernel code and offers all OS services
  - System calls
  - Fast and efficient

  - Less portable and difficult to maintain
  - Minor bugs can crash the entire system!

  - Unix and Unix-like OSs
  - Linux with loadable modules

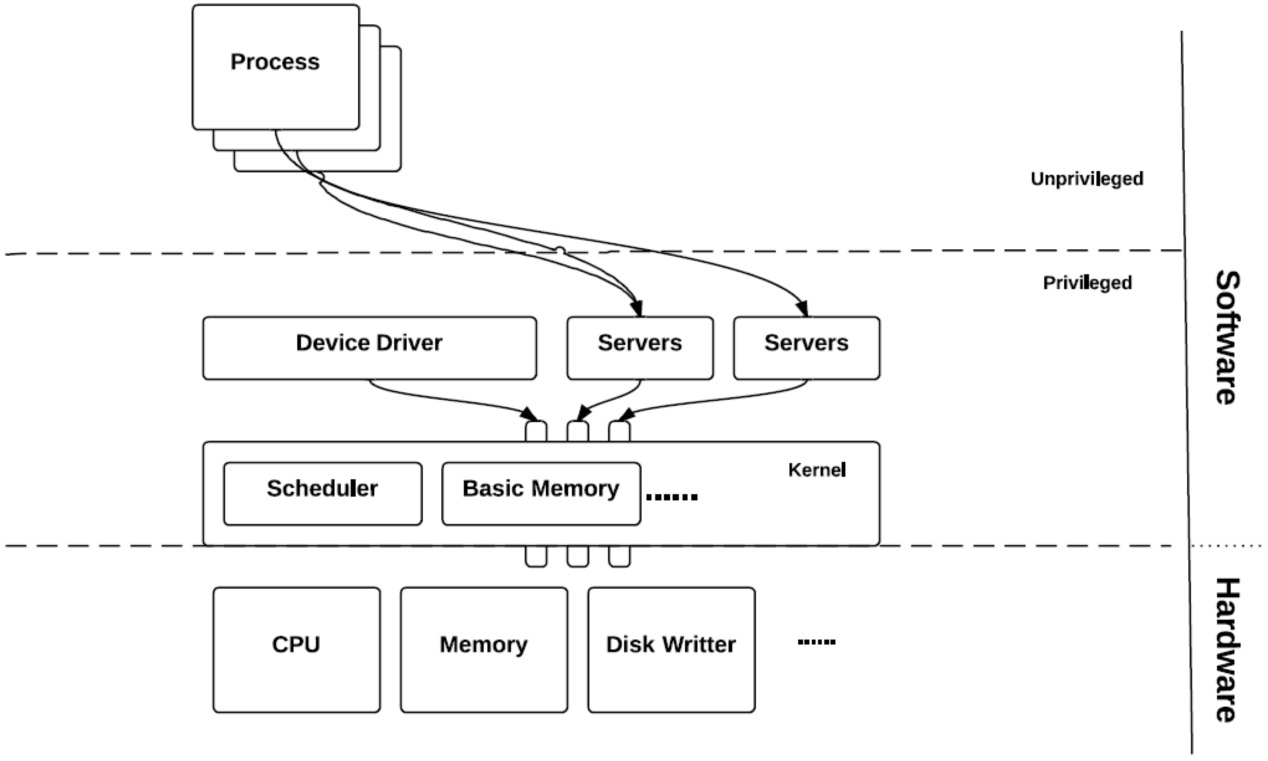


Microkernel

- Minimal Kernel
  - Kernel design is simplified (privilege modes only)
  - User-space servers (may be privileged but usually unprivileged)

  - Rapid development, unit testing, easy maintenance

  - Huge memory footprint
  - Frequent context switching and inter-process communication
  - Not easy to implement



Operating System Structure

- Monolithic has better performance in general
- Microkernels have better modularity and extensibility
  - Price for switching between modes is high

- Modern OSs (most commercial) adopt a hybrid approach
  - The kernel is kept as small as possible
  - But most servers are in the privileged kernel space

  - Windows NT
  - XNU (OS X)

Types of Operating Systems

- Network operating system
  - OS for computer networks
  - Allows and facilitates file sharing and hardware access
  - For local area networks (commonly seen in enterprise environments)
  - More features than the single computer OS: more communication
  - Routers OS (Cisco IOS)

  - Peer-to-peer
    - Manager and subordinate network
    - Structure depends on the topology of the network

Types of Operating Systems

- Distributed Operating Systems
  - Each node carries a "Horcrux " – microkernel plus service components that coordinates with other nodes
  - Work collectively to fulfil all functions of an OS
  - Single node has full access to all system resources
    - Complicated scheduling and parallelism
    - User may not be aware of which specific node is executing the program or where the physical location of the file is – all automatically handled by the OS

Types of Operating Systems

- RTOS
  - Real-time operating system dedicated to meeting specific timing constraints
  - Two types: hard real-time (ensures the critical tasks are to be completed on time) and soft real-time (if the deadline is not met, it is still worth finishing the task)
  - Industrial applications: robots, aircraft control …
  - Key design requirements:
      - Predictability and determinism
      - Speed is practically important. Usually achieved via a simplified OS design and sometimes traded off for predictability and determinism
      - Responsiveness and user control: do the right thing fast enough and priorities can be dynamically adjusted by users
      - Fail-safety: sometimes simply shutting down everything may not be a good option
    - Demands advanced scheduling and memory allocation

Embedded Operating Systems

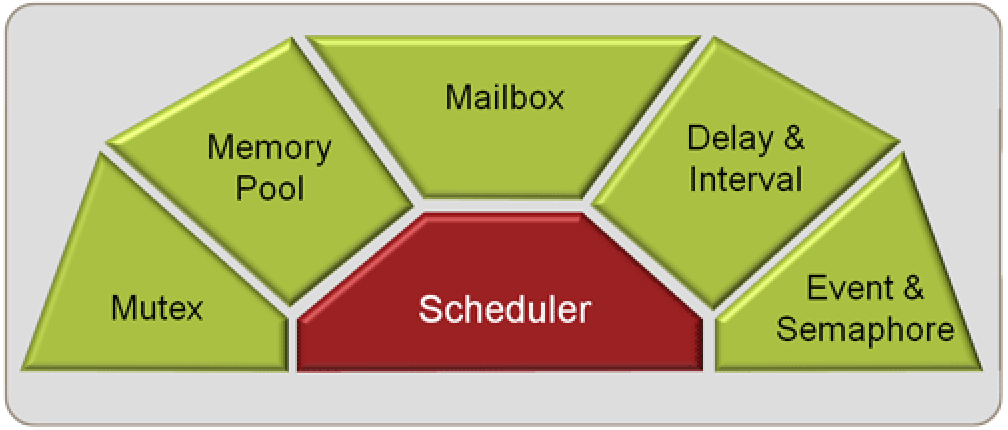
- RTOS and EOS are not exactly the same thing, but most EOSs are RTOSs, and aim at meeting the same timing constrains, therefore interchangeable in this course
- Predictability and determinism again
  - Major scheduling algorithms based on predicting the upper bound of the execution time
  - Interrupts
- “Real-Time”
  - Unified understanding of the deadline
  - Precise time services: TAI, UTC
- Fast, everything sits on the real-time kernel, even device driver
- What might not be important?
  - GUI?
  - Security? Depends on the application

RTOS on Embedded System vs. Super Loop

- Super-Loop is straightforward to implement and fits the computational model of ES
  - Depends on lengthy interrupt service routine (ISR)
  - Needs to keep the synchronization between ISRs
  - Poor predictability (nested ISRs) and extensibility
  - Change of the ISR or the Super-Loop ripples through entire system
- RTOS: all computation requests are encapsulated into tasks and scheduled based on the demand
  - Better program flow and event response
  - (Illusionary) multitasking
  - Concise ISRs thus deterministic
  - Better communication
  - Better resource management

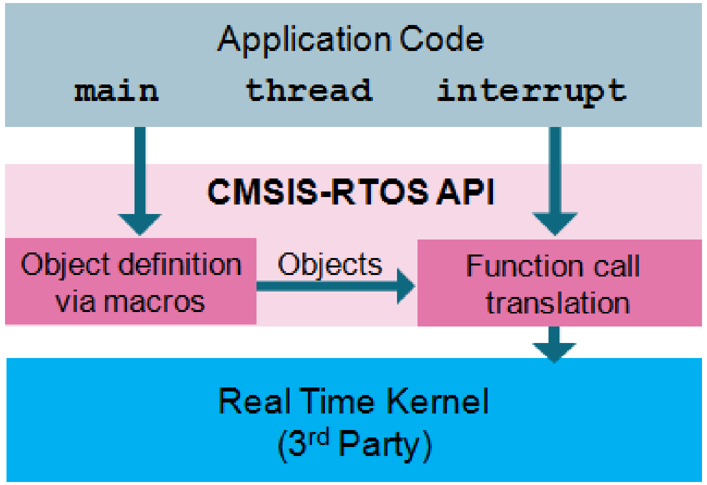
RTOS for this Course

- Keil RTX
  - Support ARM Cortex-M cores
  - Well-rounded RTOS for ES
  - Scheduler/ Mutex/ Event/ Semaphore/ Mailbox…



RTX Structure

- CMSIS-RTOS API
  - Generic RTOS interface
  - CMSIS RTOS for ST is based on Keil RTX
  - Utilise some Cortex-M instructions



CMSIS-RTOS API Structure

Next

- Processes