

OS Overview

Operating System

Ravi Suppiah
Lecturer, NUS SoC

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

- 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

Real-Time Operating System

■ 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**
 - Responsiveness and user control: do the right thing fast enough and priorities can be dynamically adjusted by users
 - Is (Real-Time == Fastest Response) ?
 - 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 constraints, 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
- Fast Enough
- What might not be important?
 - GUI?
 - Security? Depends on the application

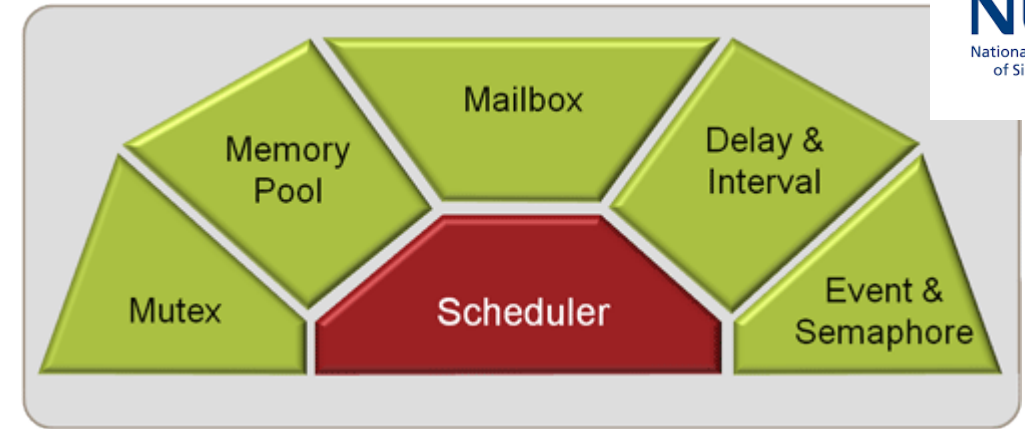
RTOS on Embedded System vs. Super Loop

- 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

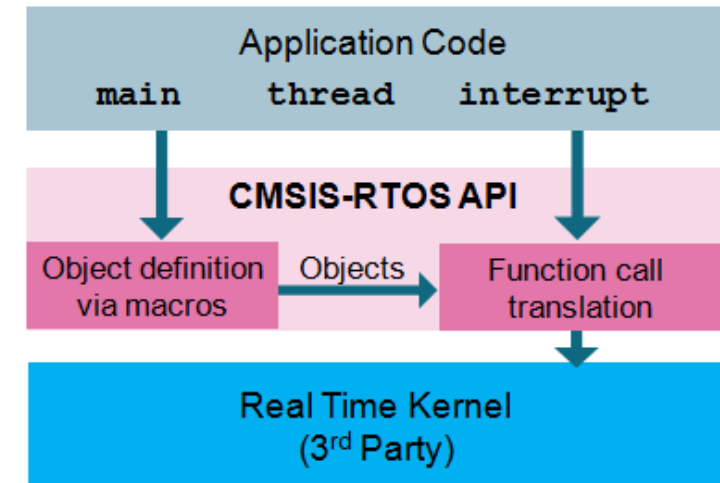
- Support ARM Cortex-M cores
- Well-rounded RTOS for ES
- Scheduler/ Mutex/ Event/ Semaphore/ Mailbox...



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