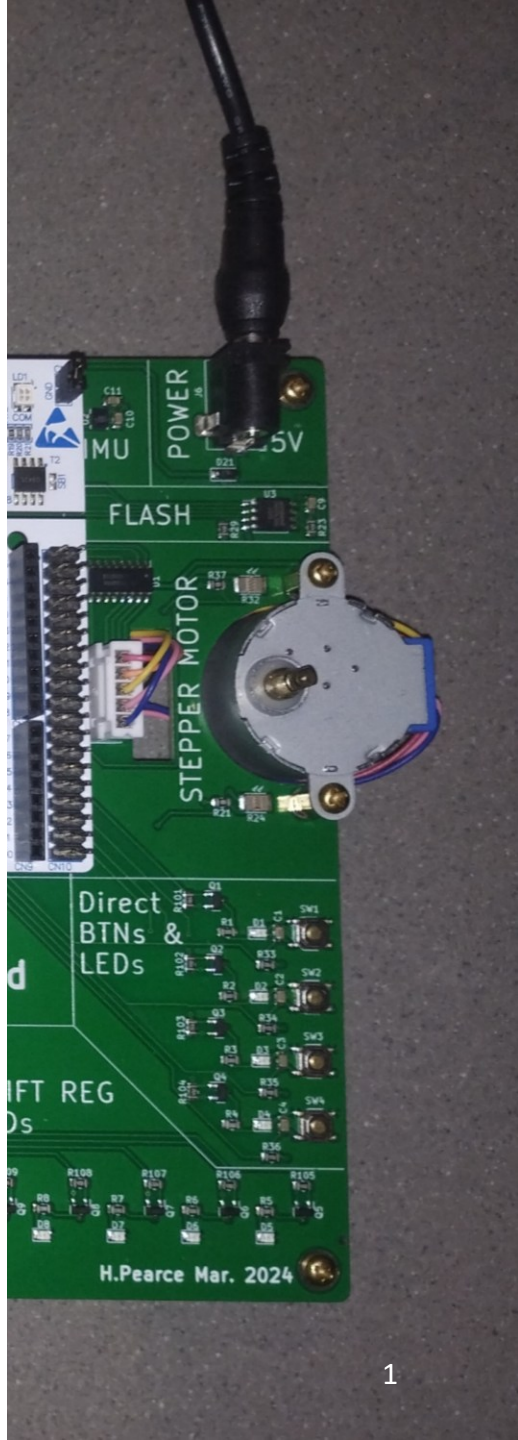
DESN2000 (Computer Engineering) 2026 T2

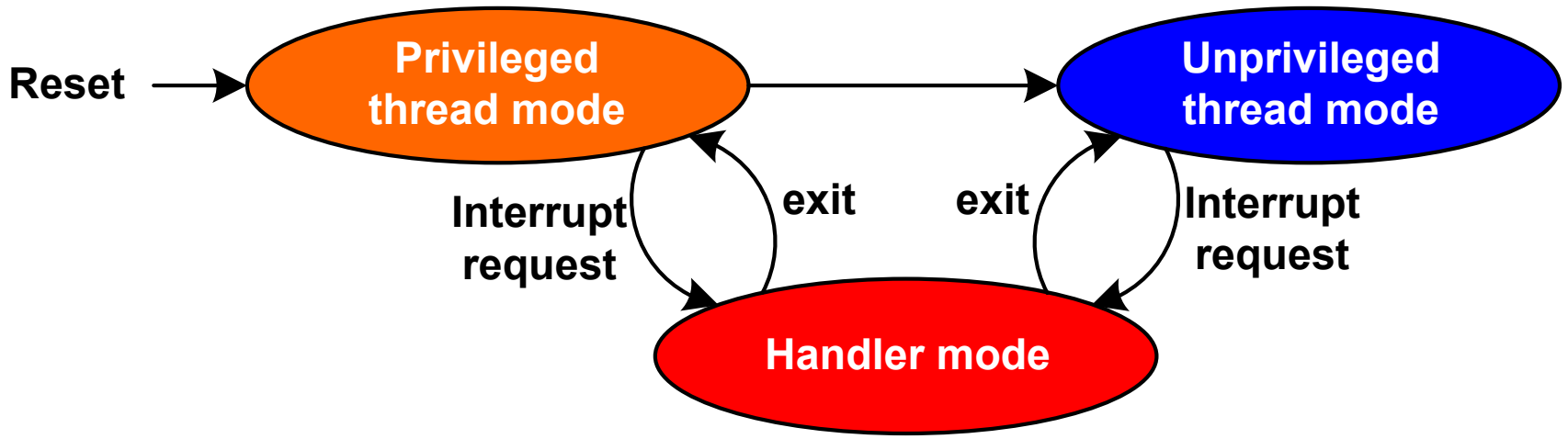
Concurrent Software and RTOS

Hasindu Gamaarachchi



Processor Modes

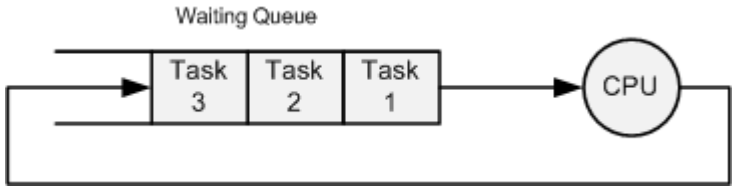
Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu



- Handler mode is always privileged.
- The privileged state allows software to access all resources
- The unprivileged state prevents software from configuring or controlling some protected resources directly.

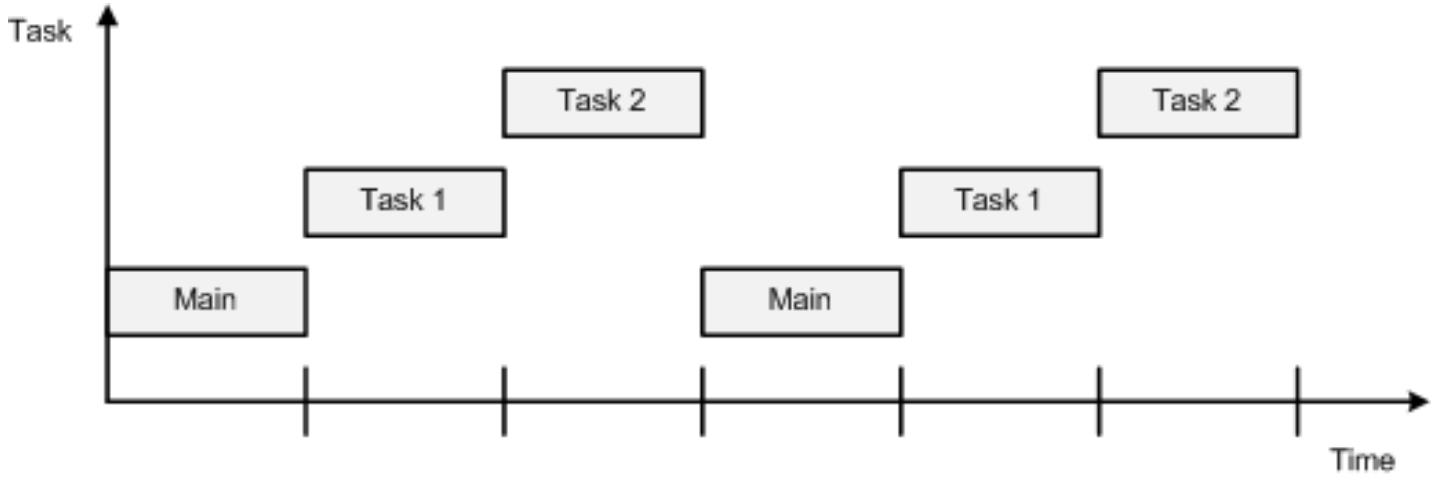
CPU Scheduling

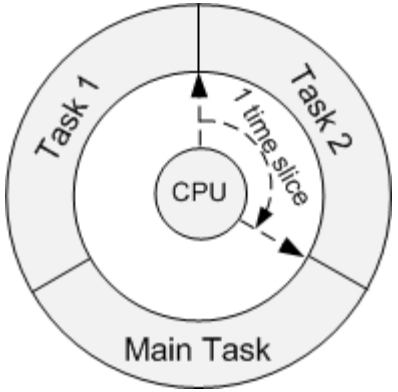
Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu



Time Sharing

Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu

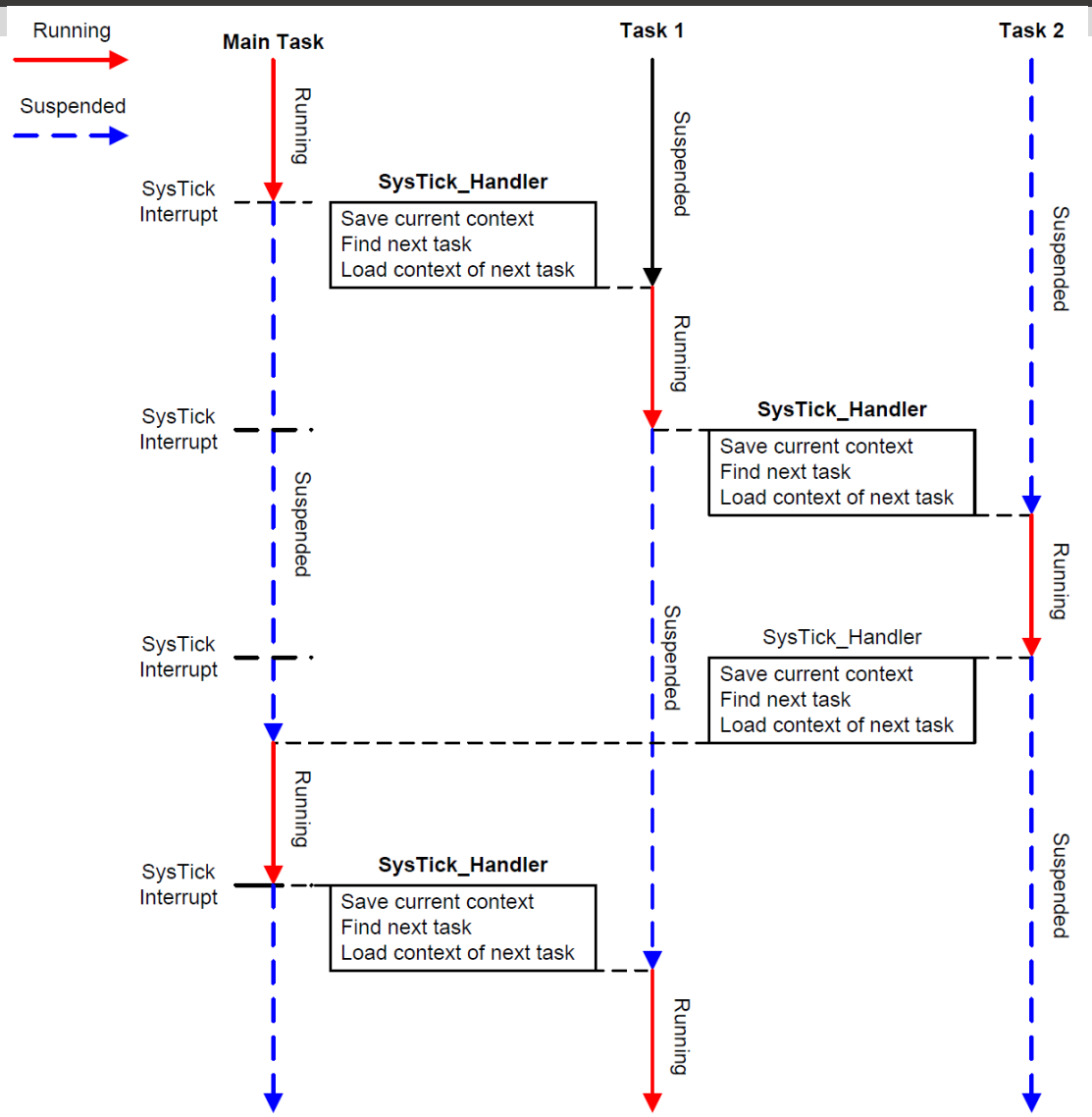




Round Robin Scheduling

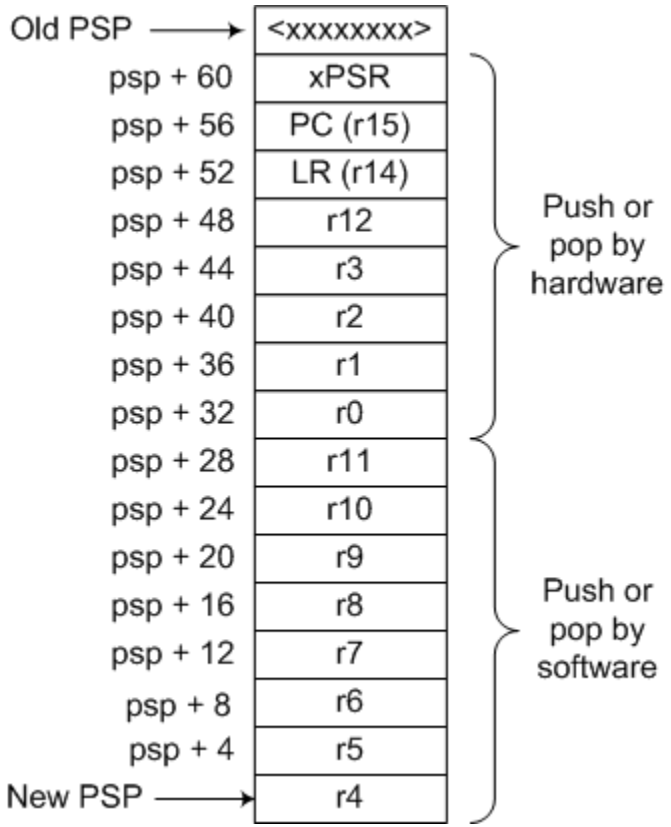
Context Switch

Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu



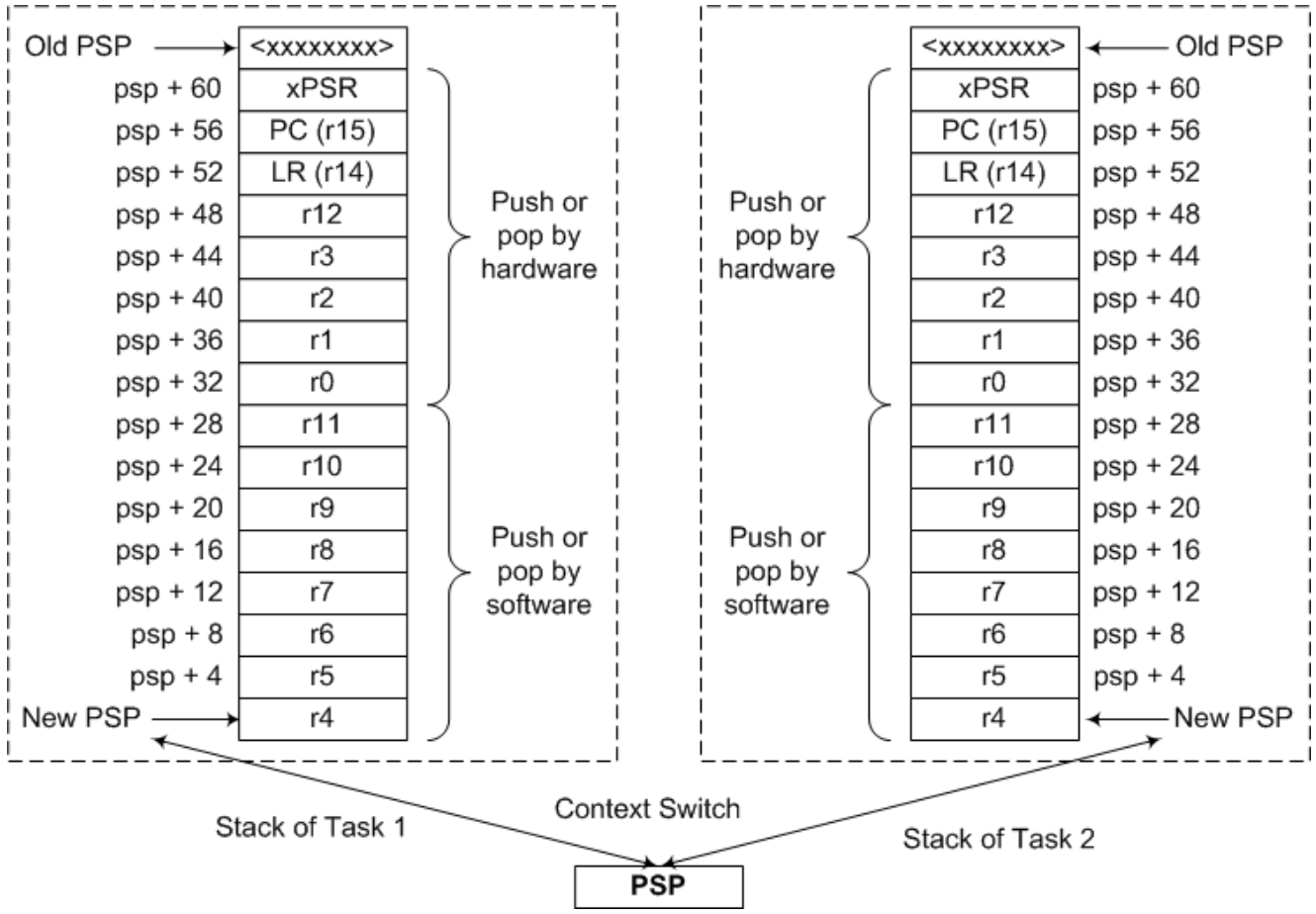
Frame

Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu



Context Switch

Slide from “Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition)” – Yifeng Zhu



Real-Time Operating System (RTOS)

Slide adapted from https://freertos.org/Why-FreeRTOS/What-is-FreeRTOS

- The scheduler in an RTOS provides a predictable (deterministic) execution pattern.
  - Especially relevant for embedded systems, which often have real-time requirements
    - Real-time requirement = system must respond to an event within a strictly defined time (the deadline)
  - Meeting real-time guarantees requires a scheduler whose behaviour is predictable (deterministic)
- An RTOS is typically smaller and lighter weight than a general-purpose operating system.

Why RTOS?

https://freertos.org/Why-FreeRTOS/FAQs/What-is-this-all-about#why-use-an-rtos

- You do not need to use an RTOS to write good embedded software.
- At some point though, as your application grows in size or complexity, the services of an RTOS might become beneficial:
  - Abstract out timing information
  - Maintainability/Extensibility
  - Modularity

RTOS

- There are several RTOS available:
  - FreeRTOS
  - Zephyr
  - Keil RTX5
  - Azure RTOS
  - and many more

CMSIS

- Different RTOS have different APIs
  - So, changing the RTOS means you have to rewrite
- CMSIS developed by ARM is an abstraction layer for Arm Cortex-M processors
  - CMSIS can call the relevant native RTOS function
  - CMSIS stands for Common Microcontroller Software Interface Standard

We will use CMSIS-RTOS

Components of an RTOS

from https://www.st.com/content/st_com/en/support/learning/stm32-moocs/freertos-common-microcontroller-software-interface-standard-osv2.html

- **Tasks or threads** – mini programs run in an endless loop
- **Scheduler** – to manage tasks and switch between them
- **Queues** – to pass information among tasks or among tasks and interrupts
- **Semaphores and mutexes** – locking mechanisms used to synchronise access to a resource
- **Software timers** – to manage task execution at particular moments

Task Control Block

from https://controllerstech.com/

- Information about each task is stored inside a data structure called Task Control Block (TCB):
  - Stack pointer
  - Task priority
  - Current state (Running, Ready, etc.)
  - Timing information
  - Task name

Task Priority

from https://controllerstech.com/

- Preemption
  - allows a higher priority task to interrupt a lower priority one and take control of the CPU, without waiting for the lower priority task to finish.

- Priority levels:
  - Below Normal
  - Low
  - Normal
  - Above Normal
  - High

Task States

from https://controllerstech.com/

- Running
  - The task is using the CPU. If single core, only one task can be in running state.
- Ready
  - The task is ready to run, but waiting till its turn.
- Blocked
  - The task is not ready to run as it is waiting for something like a delay, semaphore or an event.
- Terminated
  - Task is deleted. No longer exists.

Example 1

- Blink LED D1 and D2 using two concurrent tasks

Example 2

- Use separate tasks to do the following:
  - Blink LED D1 each 200ms
  - Blink LED D2 each 500ms
  - Blink LED D3 each 1s
  - Blink LED D4 each 2s

- Without an RTOS, what would a single super-loop implementation look like?

More Resources

- Chapter 20: Multitasking in Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Fourth Edition) – Yifeng Zhu

- https://www.st.com/content/st_com/en/support/learning/stm32-moocs/freertos-common-microcontroller-software-interface-standard-osv2.html

- https://controllerstech.com/category/stm32/stm32-freertos/