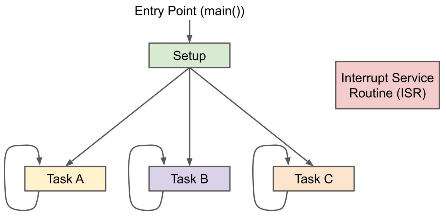
TASK SCHEDULING

- Block diagram of a multi-task program:

What our code looks like



- each tasks run concurrently in its own while loop.

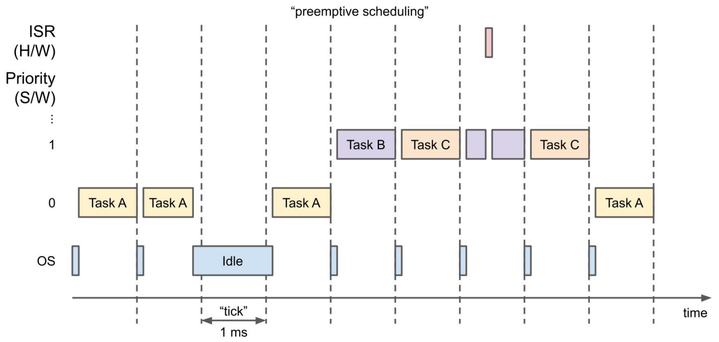
- can also set up ISRs (interrupt service routines) that can preempt any of the tasks, to execute some code.

ISR → • handles hardware timer overflows
• pin state changes
• new communication on a bus

TASK SCHEDULING IN SINGLE CORE SYSTEMS

What actually happens*

*assuming single-core processor



→ CPU divides the tasks into time slices
↳ tasks appear to run concurrently

→ schedulers determine which task to run in each time slice

→ time slice = 1ms = 1 tick
↳ hardware timer creates interrupt every 1 ms
↳ ISR for timer runs the scheduler
↳ chooses the task to run next

→ at each tick interrupt → task w/ highest priority is chosen to run
→ task w/ the same highest priority → uses round-robin execution
→ if a task w/ higher priority than running task is available → immediately run
↓
doesn't wait for next tick

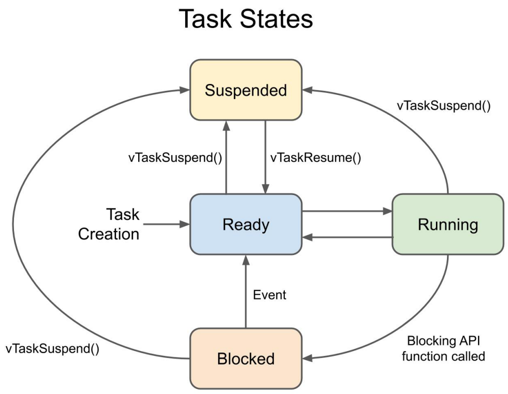
→ hardware interrupt > running software

higher priority

↳ ∴ hardware ISR can interrupt any task

→ can assign task priority using vTaskPrioritySet()

TASK STATES



Ready state → task enters this state when it is created
↳ this means the task is ready to run
→ during each tick → 1 task in ready state is chosen by scheduler to run

Running state → when tasks are running
→ can go to ready state again by the scheduler

Blocked state → when functions cause the task to wait eg. vTaskDelay()
→ occurs when tasks wait for an event to occur eg. timer on vTaskDelay() to expire
→ allows other task to run instead of the Blocked task

Suspended state → like putting task to sleep eg. use vTaskSuspend()
→ any task can put any other task in this mode (including itself)
→ task go back to Ready state by calling vTaskResume()