

ECGR-6185

μ C/OS II

Nayana Rao
University of North Carolina at Charlotte

Introduction and Features

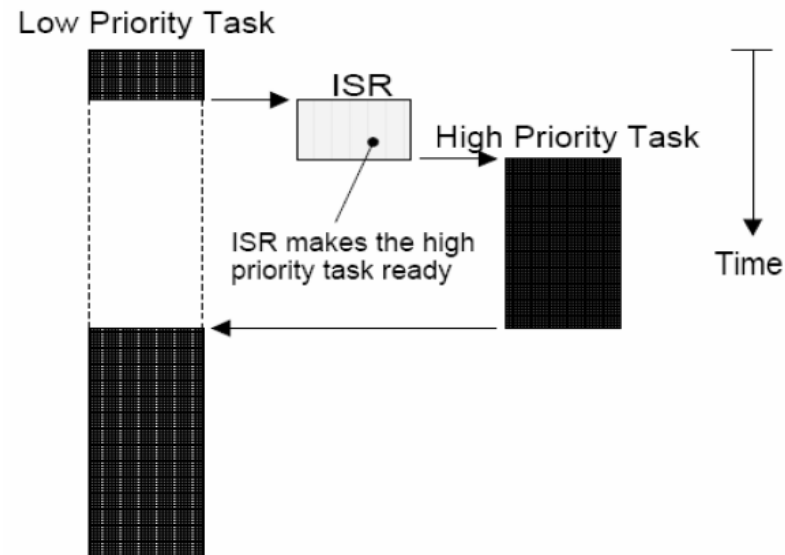
- Very small real time OS kernel: OS that enforces timing constraints
 - Memory footprint is small ~ 20 KB
 - OS source code is written in C-open source but not free for commercial use
-

MULTITASKING

- Process of switching CPU and scheduling between several tasks
 - Resources are shared
 - Each task is an infinite loop
-

KERNEL

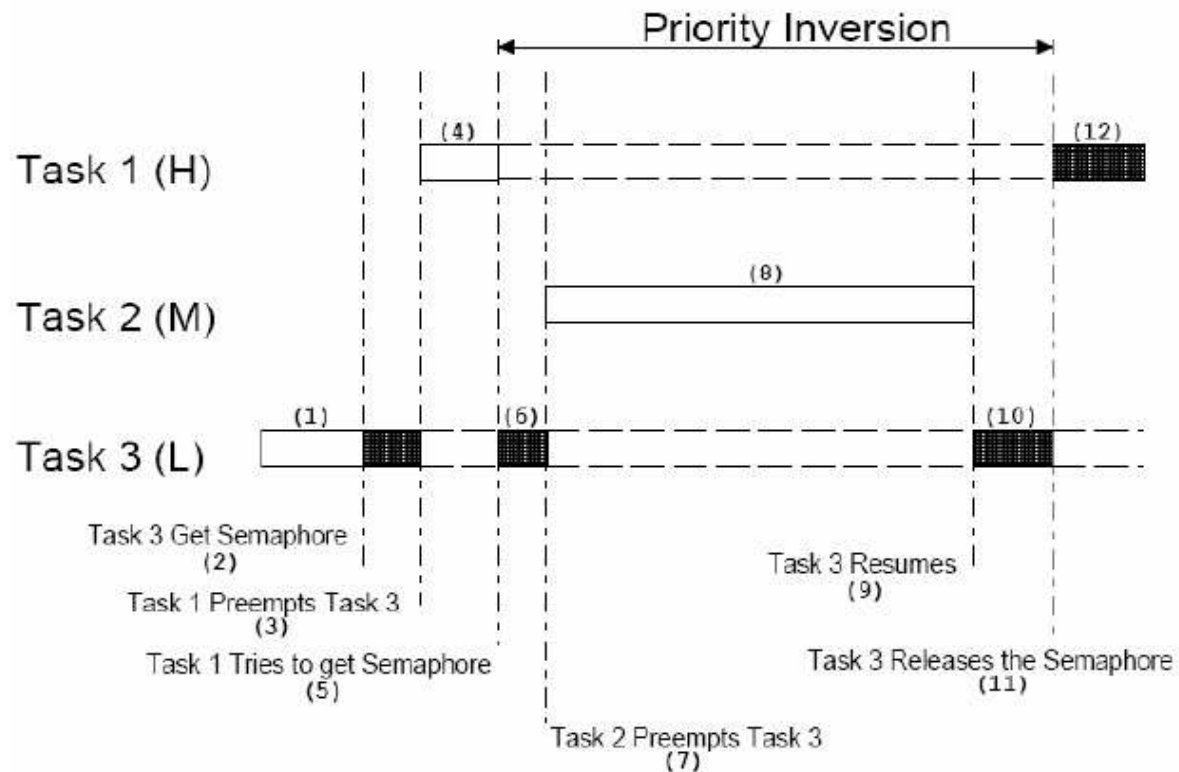
- Responsible for managing tasks and initiate context switching
- Pre-emptive kernel



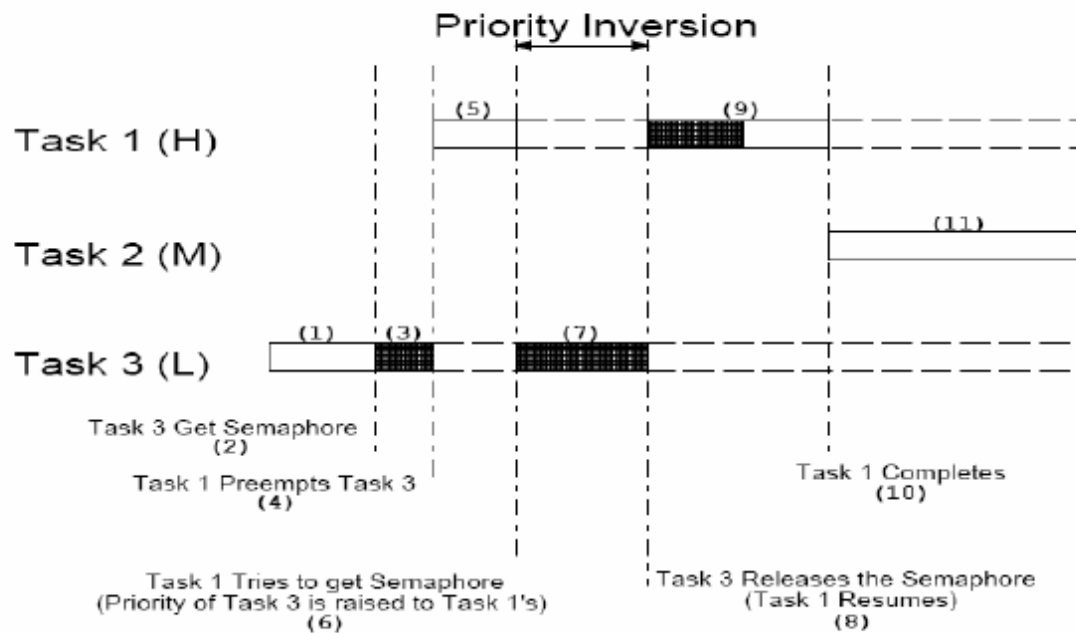
SCHEDULING

- Preemptive priority driven real time scheduling
 - 64 priority levels which means maximum allowed tasks = 64
 - Priority inversion problem: When a task with lower priority holds a resource required by high priority task.
-

Priority Inversion problem:



Solution: Priority Inheritance



MUTUAL EXCLUSION

Avoidance of simultaneous use of resources

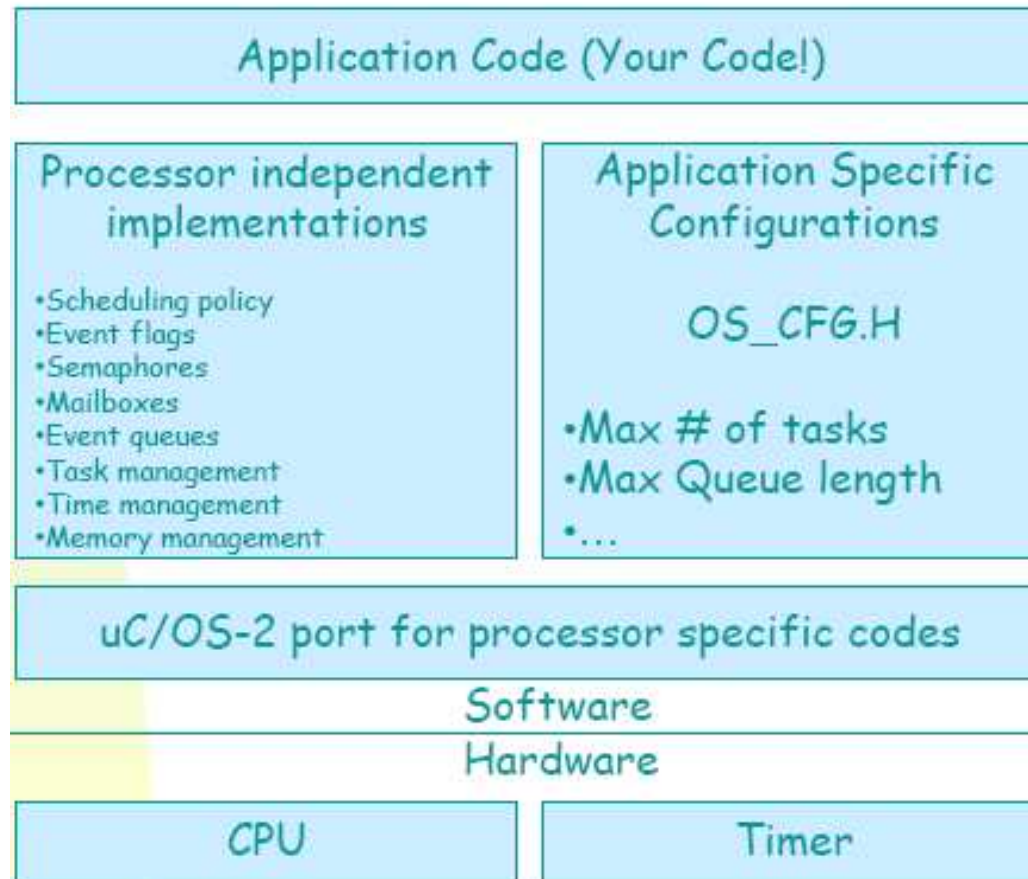
- Interrupt Handlers: Enabling and disabling interrupts to manipulate and protect critical sections so that no other ISR or context switch occurs
 - Semaphores: binary or counting
 - If Deadlock-> Set timeout
-

SYNCHRONIZATION AND INTERRUPTS

- Between two tasks or between task and ISR
- Interrupt informs the CPU of asynchronous events
- Interrupt latency is important and should be kept low to improve responsiveness

Interrupt latency = max duration for which interrupts are disabled+ time to start executing first instruction in ISR

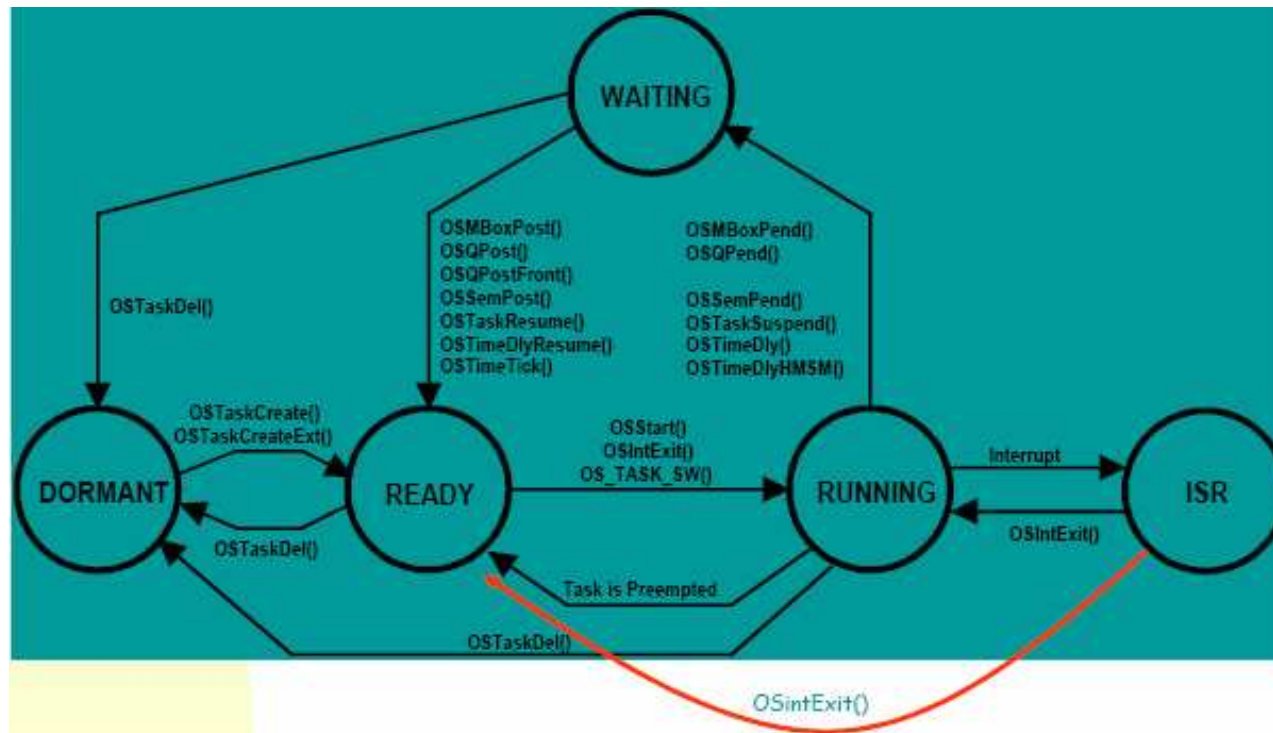
μC/OS II File Structure



Tasks

- Small piece of code that performs some function:
infinite loop
 - Several tasks controlled by a Task Control Block. Also maintains state of task
 - Task information maintained by OS-Task's Context
 - Each task has a unique priority
-

- Each task could be in 1 of 5 states: Dormant, Ready, Running, Waiting, ISR



Dormant: Procedure residing on RAM/ROM. Not a task till it is called to execute

Ready: Not delayed or waiting. Contained in a ready list

Running: Task ready or schedule to run on the CPU

Waiting: Task is waiting for an event to occur eg: timeout or for semaphore

ISR: A task may be pre-empted by an ISR

Example of Task

```
class Task
{
    public:

        Task(void (*function) (), Priority p, int stackSize);

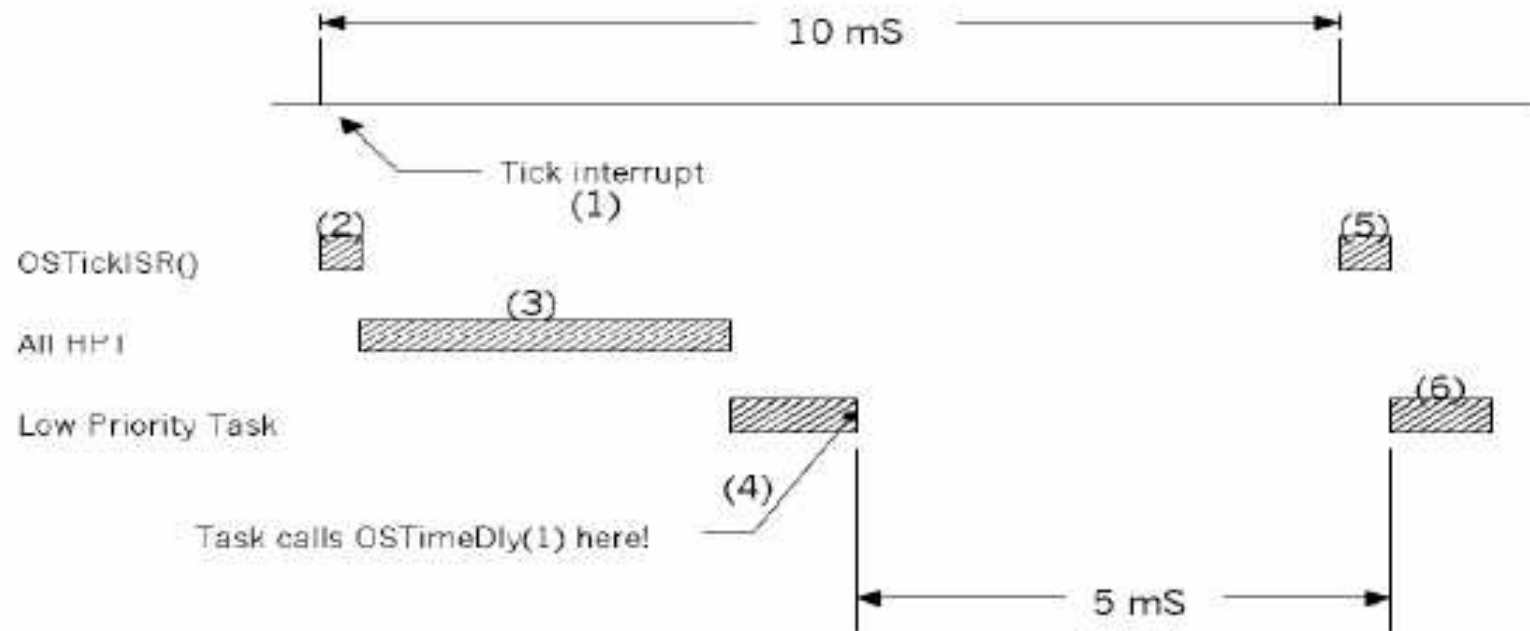
        TaskId      id;
        Context     context;
        TaskState   state;
        Priority     priority;
        int *       pStack;
        Task *      pNext;

        void (*entryPoint) ();

    private:

        static TaskId nextId;
};
```

A task may Delay itself for some clock cycles



Clock Ticks

- Keeps track of timing and delays
- It is an interrupts that is triggered by a timer interrupt

```
void OSTickISR(void)
{
    Save processor registers;
    Call OSIntEnter() or increment OSIntNesting;
    If(OSIntNesting == 1)
        OSTCBCur->OSTCBS+kPtr = SP;
    Call OSTimeTick();
    Clear interrupting device;
    Re-enable interrupts (optional);
    Call OSIntExit();
    Restore processor registers;
    Execute a return from interrupt instruction;
}
```

Task level Scheduling and Context Switching

Scheduler schedules highest priority task to run-

Interrupts are disabled during scheduling so that new tasks are not added by other interrupts to the ready list

Context Switching (handled at interrupt level) must save CPU registers of the pre-empted task to its stack and be able to restore highest priority task from its task

μ C/OS II and the Renesas μ C

- μ C/OS II can be loaded
 - Initialize and start μ C/OS II
 - Build required application
-

