MicroC/OS-II Chapter 2

中興資科所 盧慶達
學號:79256022
指導教授 張軒彬

CHAPTER 2 Real-time Systems Concepts

Type:

1.soft-real time system:

.Task are performed by the system as fast as possible but the task don’t have to finish by specific times.

2.hard-real time system:

.Task have performed not only correctly but on time.

2.00 foreground/background system

Background (task level):

An application consists of an infinite loop that calls modules to perform the desired operations.

Foreground (interrupt level):

ISR handle asynchronous events

2.01 Critical Sections of Code

1.Critical section of code (critical region) need to be treated indivisibly.

2.interrupt are disabled before the critical section code is executed and enabled when the critical code is finished

2.02 Resource

A resource is an entry used by a task ,such as a printer,a display,a variable,a structure.

2.03&2.04

2.03 Shared Resource

1.Resource can be used by more than one task.

2.Gain exclusive access to the shared resource to prevent data corruption called mutual exclusive

2.04 Multitasking

Process of scheduling and switching the CPU between several tasks.

2.05 Tasks

1.A simple program that thinks it has the CPU all to itself.

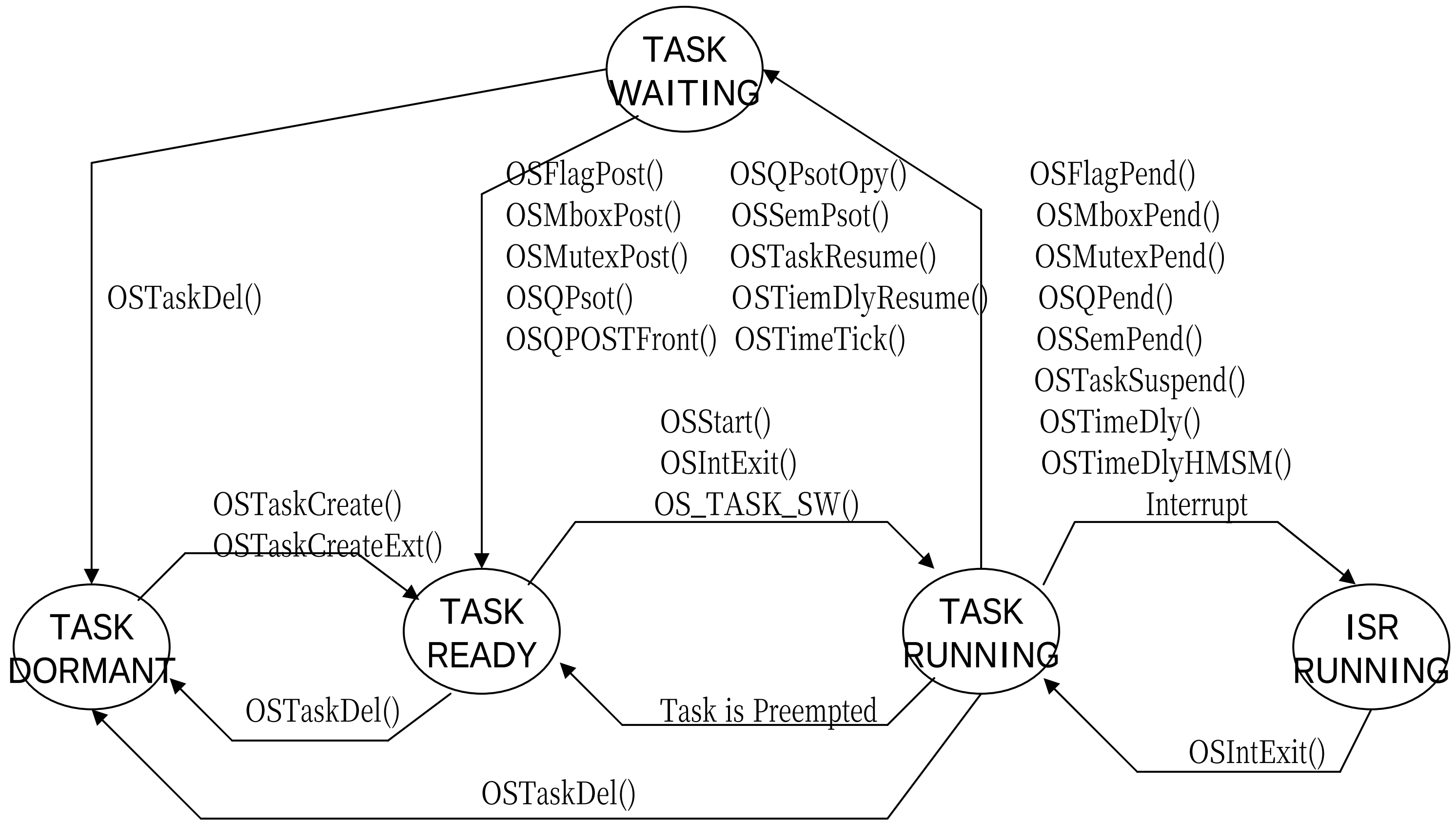
2.Each task is assigned:

a.A priority .

b.its own set of CPU registers.

c.its own stack area.

2.05 Task


TASK WAITING
OSTaskDel()
OSFlagPost()
OSMboxPost()
OSMutexPost()
OSQPsot()
OSQPOSTFront()
OSQPsotOpy()
OSSemPsot()
OSTaskResume()
OSTiemDlyResume()
OSTimeTick()
OSFlagPend()
OSMboxPend()
OSMutexPend()
OSQPend()
OSSemPend()
OSTaskSuspend()
OSTimeDly()
OSTimeDlyHMSM()
OSStart()
OSIntExit()
OS_TASK_SW()
OSTaskCreate()
OSTaskCreateExt()
Interrupt
TASK DORMANT
TASK READY
TASK RUNNING
ISR RUNNING
OSTaskDel()
Task is Preempted
OSIntExit()
OSTaskDel()

2.06 Context Switchesk

kernel save the current task's context in the current task's stack,then the new task's context is restored from its stack and then resumes execution of the new task's code.

1.the more registers a CPU has the higher the overhead.

2.07 Kernels

1. management of tasks.
2. communication between tasks.
3. provide context switch.
4. use between 2 and 5% of CPU time.
5. require extra ROM and RAM for the kernel’s data structure

2.08 schedulers

1.Determine which task run next.

2.Most real-time kernels are priority based.

3.Each task is assigned a priority based on its importance.

4.the highest priority task gets the CPU.

5.two type of priority based kernels exist:

a.non-preemptive kernel

b.preemptive kernel

2.12 Round-Robin Scheduling

- When two or more tasks have the same priority, kernel allows one task to run a quantum.
- Kernel give control to the next task in line if

1.the current of task has no work to do during its time slice.

2.the current of task completes before the end of its time slice.

3.time slice ends.

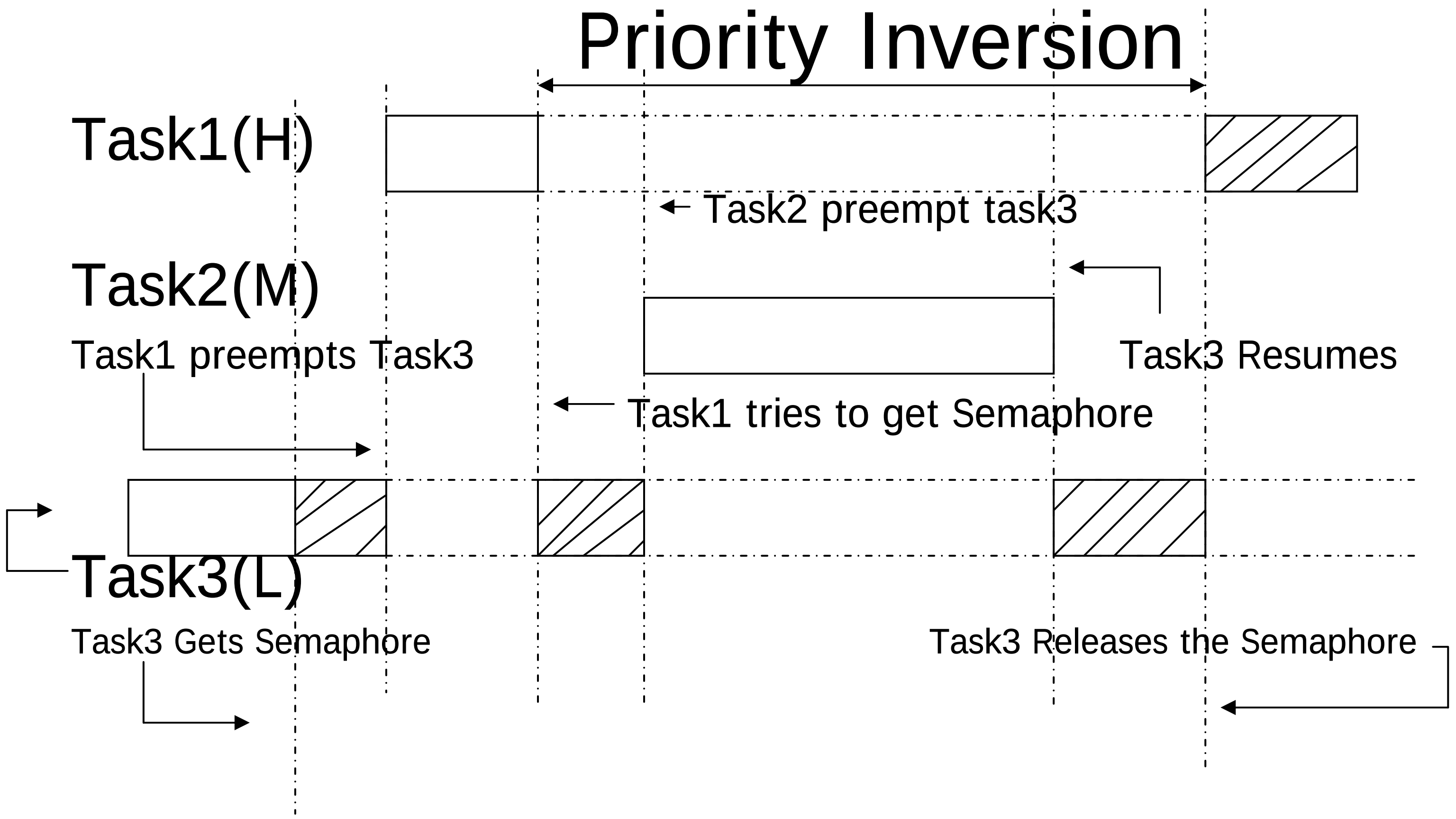
2.13 Task Priority

- Each task is assigned a priority.
- More important the task, the higher the priority given to it.

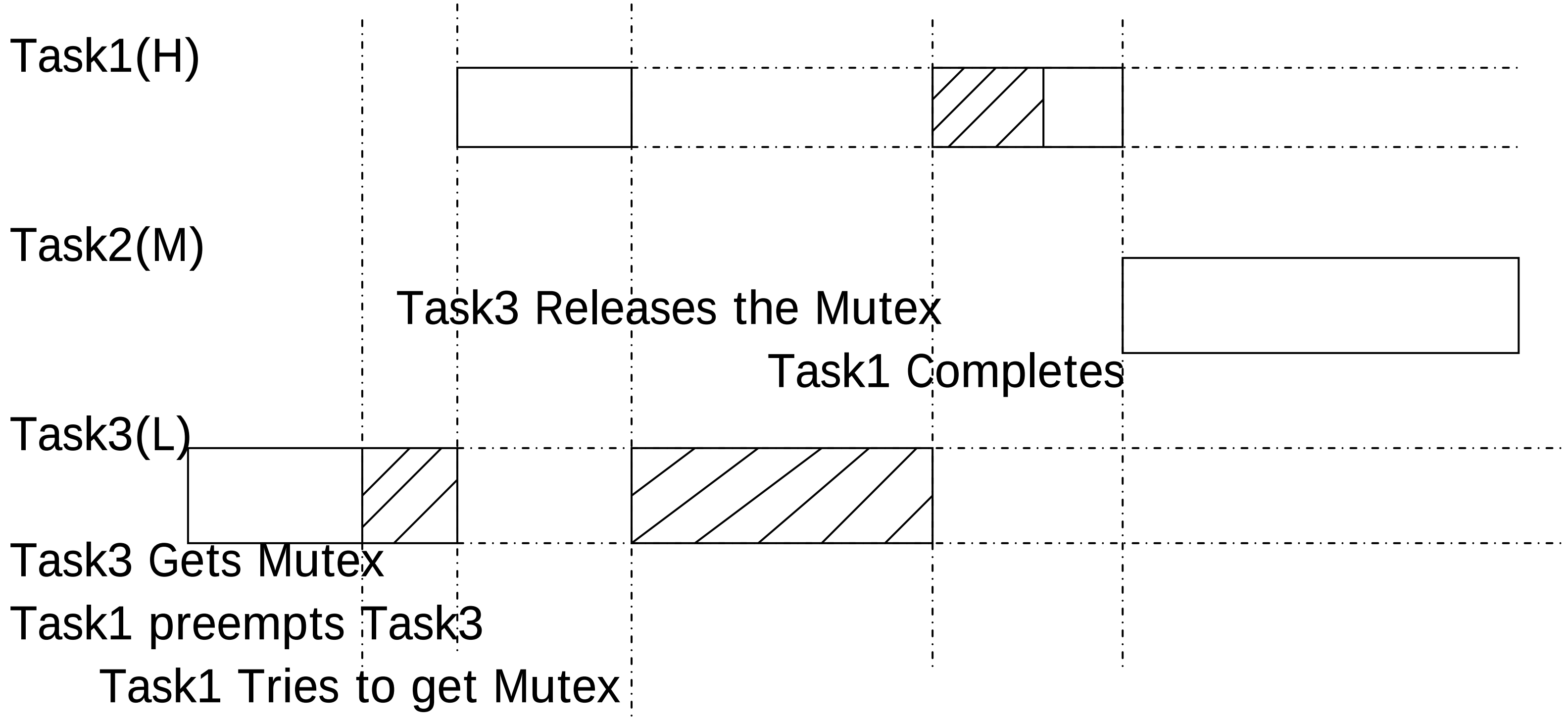
Static Priorities & Dynamic Priorities

- Task‘s priority can’t changed at the application’s execution is the static priorities.
- Task’s priority can changed at the application’s execution is the dynamic priorities.(task can change its priority at run time)

2.16 Priority Inversions



2.16 Priority Inversions



2.17 Assigning Task Priorities

- Rate monotonic scheduling(RMS) has been established to assign task priorities based on how often tasks execute.

RMS make a number of assumptions:

- All tasks are periodic.
- Tasks do not synchronize with one another,share resource,or exchange data.
- The CPU must always execute the highest priority task that is ready to run.

2.17 Assigning Task Priorities

- The basic RMS theorem states that all task hard real-time deadlines are always met if the inequality in Equation is verified.

$$\Sigma_i E_i/T_i \leqq n(2^{1/n}-1)$$

- E_i corresponds to maximum execution time of task i.
- T_i corresponds to execution period of task i.
- n:n numbers tasks have ready to run.

2.17 Assigning Task Priorities

Number of Tasks	$n(2^{1/n}-1)$
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
.	.
.	.
.	.
-	0.693

2.17 Assigning Task Priorities

- CPU use of all time-critical tasks should be less than 70 percent.
- highest rate task has the highest priority.
- The highest rate task might not be the most important task.

Mutual Exclusion

1.to prevent multiple tasks used global ,pointers , buffers , linked list and ring buffer to at the same time.

2.use mutual exclusion access to avoid contention,data corruption.

3.most common method to obtain exclusive access:

a.disabling interrupt: μC/OSⅡuse to access internal variable and data structure

OS_ENTER_CRITICAL()

OS_EXIT_CRITICAL()

Mutual Exclusion

b.perform test-and-set operations.

Check a global variable if ‘0’ access the resource else wait.

Must disable interrupt before operate test-and-set.

c.disabling scheduling.

Semaphore

d. Use semaphore.

1.Control access to a shared resource.

2.Signal the occurrence of an event.

3.Allow two tasks to synchronize their activities.

4.two types of semaphore

a.binary semaphore(0,1)

b.counting semaphore(depend on Kernel used)

Semaphore

5.two operation

a.wait:task need semaphore

b.signal:release semaphore

6.timeout occur:

Task is made ready to run,error code returned to caller.

7.task receives the semaphore is either

a.the highest priority task waiting for the semaphore

b.the first task that requested the semaphore

8.semaphore encapsulate.

Deadlock

Definition:Two tasks are each unknowingly waiting for resources held by the other.

Solution:

1.acquire all resources before proceeding.

2.acquire the resources in the same order.

3.release the resources in the reverse order.

Synchronizations

Definition:a task can be synchronized with an ISR by using a semaphore.

Two type of synchronization

a.unilateral rendezvous

b.bilateral rendezvous

Event flages

Task needs to synchronize with the occurrence of multiple events.

a.disjunctive synchronization

b.conjunctive synchronization

Intertask Communication

Task or ISR communicate information to another task through

a.global variable(exclusive access)

b.send messages

Mailbox

1.Mailbox is a pointer size variable.

2.When deposited into mailbox

a.priority based

b. FIFO

Message Oueue

1.to send one or more message to a task.

2.an array of mailboxs

3.message extracted from the queue in FIFO

But μC/OSⅡ allow task to get message LIFO

Interrupt

1. Inform CPU that an asynchronous event has occurred.
2. recognized interrupt CPU save of its context switch and jumps to special subroutine caled ISR.
3. ISR back to

a. the background for a foreground/background System

b. the interrupted task for a Non-preemptive kernel.

c. the highest priority task ready to run for a Preemptive kernel.

Disable interrupt

1.as little as possible

2.affect interrupt latency,cause interrupt missed

Interrupt latency

The Longer interrupts are disabled,the higher the interrupt latency.

Interrupt latency=Mid+Tst

Mid=Maximum amount of time interrupt are disabled

Tst=Time to start executing the first instruction in the ISR

Interrupt response

Time between the reception of interrupt and start of the user code is executed.

Foreground/background system interrupt response=

Interrupt latency+Time to save the CPU'scontext

Non-preemptive kernel interrupt response=

Interrupt latency+Time to save the CPU'scontext

Interrupt response

Preemptive kernel a function needs to be called to notify the kernel that an interrupt starting.

This function allow kernel to keep track of interrupt nesting.

Preemptive kernel interrupt response=

Interrupt latency+Time to save the CPU'scontext+

Execution time of the kernel ISR entry function

Interrupt Recovery

Definition:time require for the processor to return to the interrupted code or to a higher priority task.

Foreground/background system interrupt recovery=

Time to restore the CPU's context +time to execute the return from interrupt instruction

Non-preemptive kernel interrupt recovery=

Time to restore the CPU's context +time to execute the return from interrupt instruction

Interrupt Recovery

Preemptive kernel interrupt recovery=

Time to determine if higher task is ready+

Time to restore the cPU's context of highest priority code+

Time to execute the return from interrupt instruction.