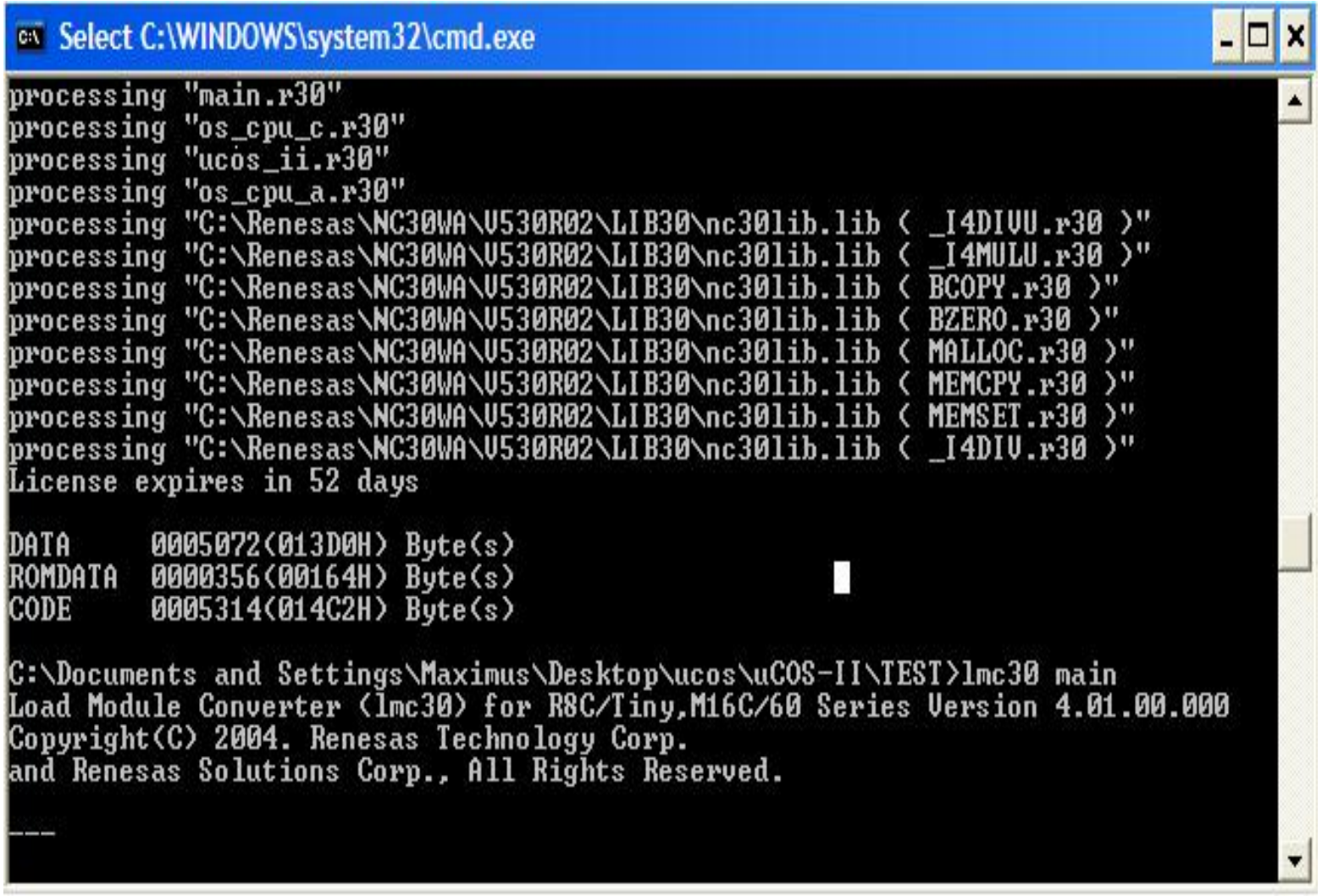
Outlines

- **Introduction**
- **Kernel Structure**
- **Porting**

Introduction

- **Embedded Systems**
  - **Big Picture**
  - **System**
    - **Hardwired**
    - **Real Time**
    - **Development tools**
      - **Compiler**
      - **Linker**
      - **Debugger**
      - **Emulator**
      - **Simulators**


Select C:\WINDOWS\system32\cmd.exe
processing "main.r30"
processing "os_cpu_c.r30"
processing "ucos_ii.r30"
processing "os_cpu_a.r30"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (_I4DIVU.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (_I4MULU.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (BCOPY.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (BZERO.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (MALLOC.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (MEMCPY.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (MEMSET.r30)"
processing "C:\Renesas\NC30WA\V530R02\LIB30\nc30lib.lib (_I4DIV.r30)"
License expires in 52 days
DATA 0005072(013D0H) Byte(s)
ROMDATA 0000356(00164H) Byte(s)
CODE 0005314(014C2H) Byte(s)
C:\Documents and Settings\Maximus\Desktop\ucos\uCOS-II\TEST>lmc30 main
Load Module Converter (lmc30) for R8C/Tiny,M16C/60 Series Version 4.01.00.000
Copyright(C) 2004. Renesas Technology Corp.
and Renesas Solutions Corp., All Rights Reserved.

```
{
    ram : ORIGIN = 0x00000, LENGTH = 512K
    rom : ORIGIN = 0x80000, LENGTH = 512K
}

SECTIONS
{
    data ram :                          /* Initialized data.        */
    {
        _DataStart = . ;
        *(.data)
        _DataEnd   = . ;

    } >rom

    bss :                              /* Uninitialized data.      */
    {
        _BssStart = . ;
        *(.bss)
        _BssEnd   = . ;
    }

    _BottomOfHeap = . ;                /* The heap starts here.     */
    _TopOfStack = 0x80000;             /* The stack ends here.      */

    text rom :                         /* The actual instructions.  */
    {
        *(.text)
```

Operating System

- **History and Purpose**
- **A decent Embedded System**

What is a Real Time Operating System?

- **An operating system enforcing timing constraints**
  - **VxWorks**
  - **RTLinux**
  - **WinCE**
  - **TinyOS**
  - **Symbian**
  - **uC/OS-II**

Real Time System Concepts

- **Multitasking**
- **Kernel**
- **Scheduling**
- **Mutual Exclusion**
- **Synchronization**
- **Interrupt**

Multitasking

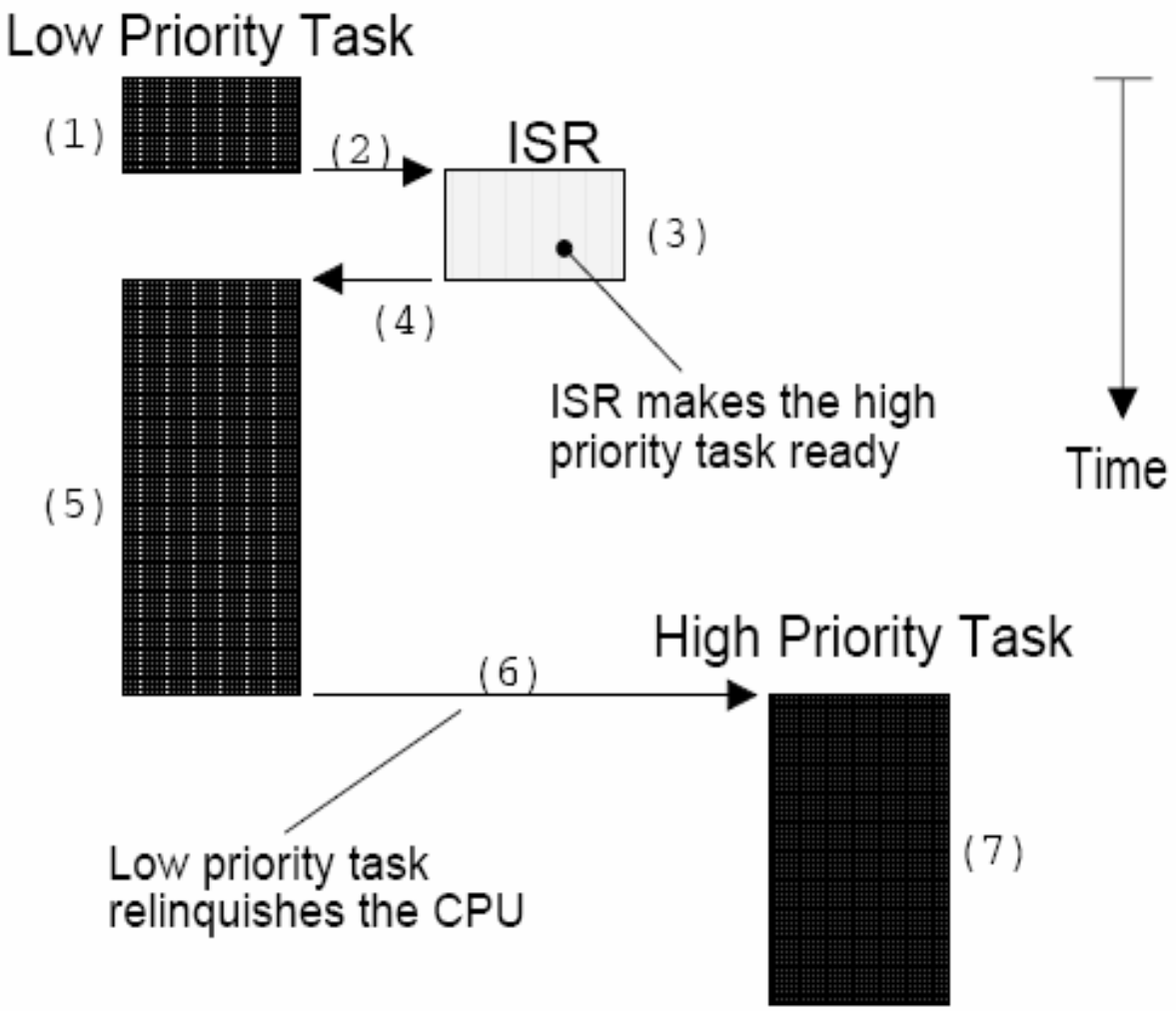
- **Multitasking**
  - **A process of scheduling and switching CPU between several tasks**
  - **Tasks**
    - **Ready, Running, Waiting, ISR, Dormant**
  - **Resource sharing**
  - **Critical section**

Kernel

- **A part of the multitasking system responsible for management of tasks.**
- **Context switching is the fundamental service of a kernel**
  - **Can be Preemptive and Non-Preemptive**

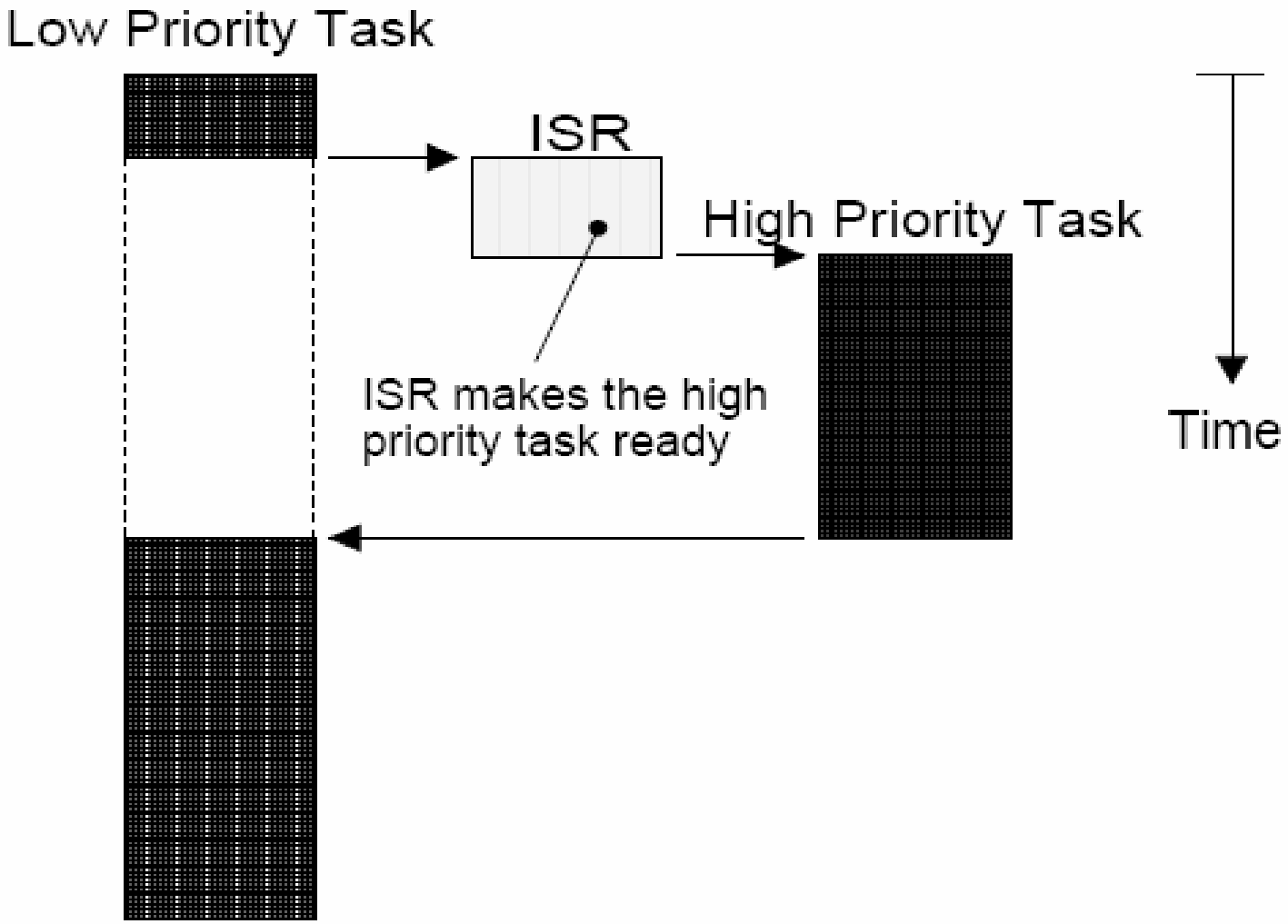
Non-Preemptive Kernel

- **The new higher priority task gains control of the CPU only when current task gives up CPU.**
- **An ISR can make a higher priority task ready to run, but ISR will eventually return to the interrupted task.**
  - **Interrupt latency is low**
  - **Low responsiveness.**


Low Priority Task
(1)
(2)
ISR
(3)
(4)
ISR makes the high
priority task ready
Time
(5)
High Priority Task
(6)
Low priority task
relinquishes the CPU
(7)

Preemptive Kernel

- **System responsiveness is important.**
- **Most real time kernels are preemptive in nature.**
  - **How about uC/OS-II??**


Low Priority Task
ISR
High Priority Task
ISR makes the high priority task ready
Time

Function Reentrancy

Non-Reentrant Function

```
static int Temp;

void swap(int *x, int *y)
{
    Temp = *x;
    *x   = *y;
    *y   = Temp;
}
```

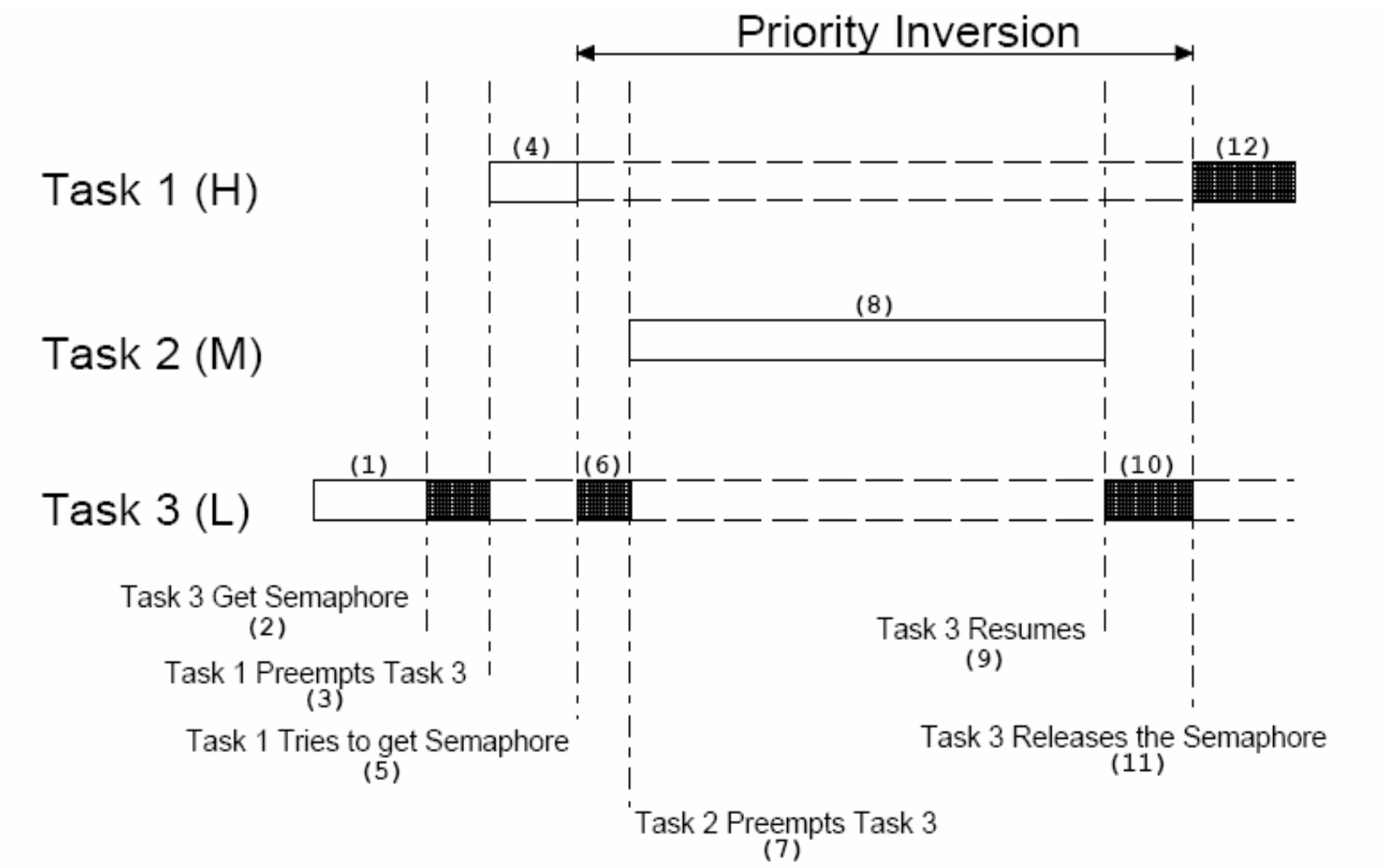
Reentrant Function

```
void strcpy(char *dest, char *src)
{
    while (*dest++ = *src++) {
        ;
    }
    *dest = NULL;
}
```

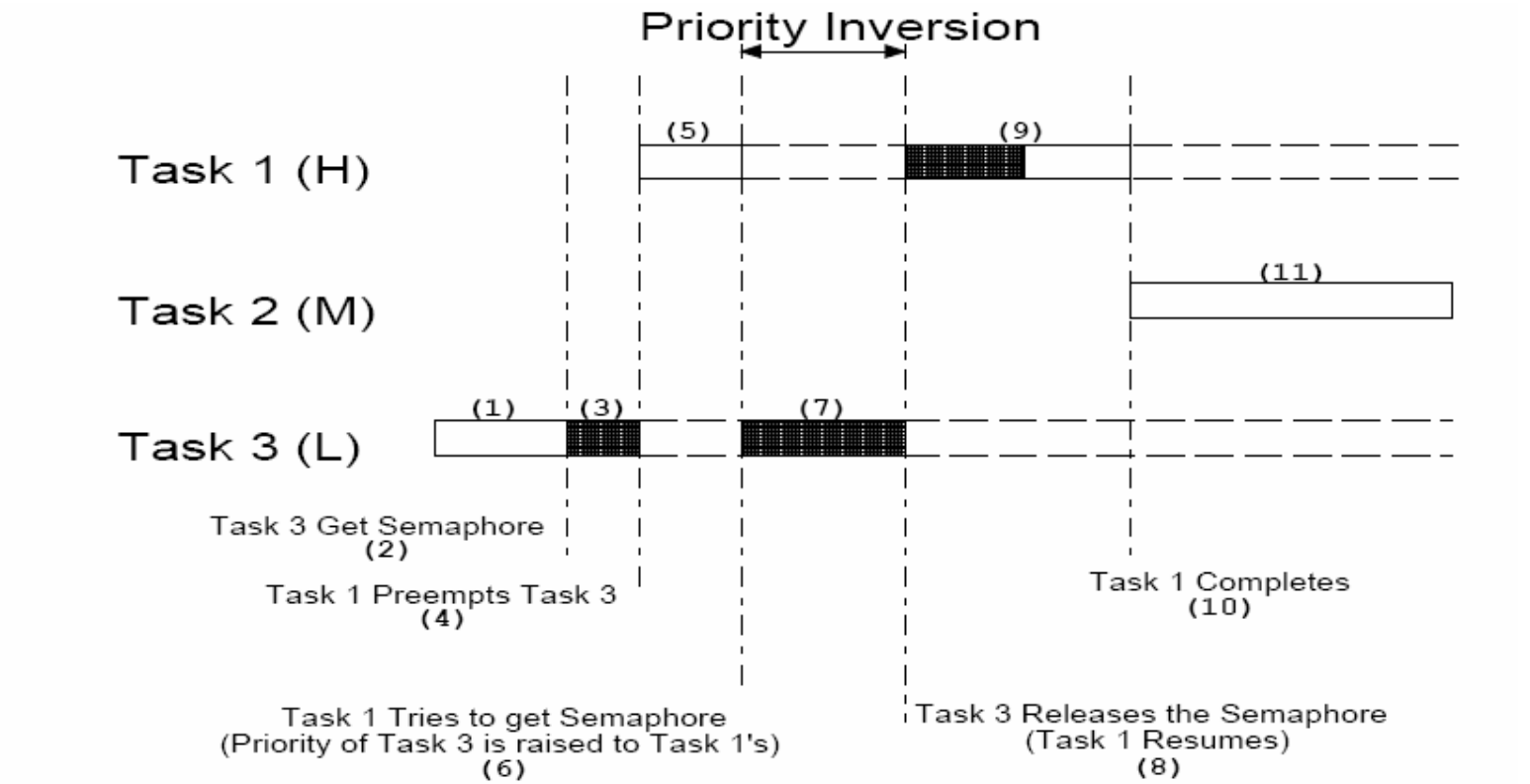
Scheduling

- **Priority based scheduling**
- **Round Robin Scheduling**
  - **How about uC/os-II**
- **Issues:**
  - **Priority Inversion**
    - **Priority inheritance is needed**

Priority Inversion



Priority Inheritance



Mutual Exclusion

- **Protecting shared data of processes**
  - **Disabling and enabling Interrupts**
  - **Semaphores**
    - **Binary**
    - **Counting**
- **Deadlock- Set timeout**

```
OS EVENT *SharedDataSem;


void Function (void)
{
    INT8U err;


    OSSemPend(SharedDataSem, 0, &err);
    .
    .     /* You can access shared data in here (interrupts are recognized) */
    .
    OSSemPost(SharedDataSem);
}
```

Synchronization

- **Synchronization mechanism is used between tasks or task to ISR.**
- **Unilateral rendezvous**
- **Bilateral rendezvous**

Interrupts

- **An interrupt is a hardware mechanism to inform CPU that an asynchronous event has happen.**
- **Interrupt response**
- **Interrupt Recovery**
- **Interrupt Latency**
- **NMI**

Interrupt Latency

- **To manipulate critical sections Interrupts are disabled.**
  - **longer Interrupts are disabled, higher is Interrupt Latency**
  - **Interrupt Latency is given by:**

Maximum amount of time interrupts are disabled +
Time to start executing the first instruction in the ISR

Interrupt Response

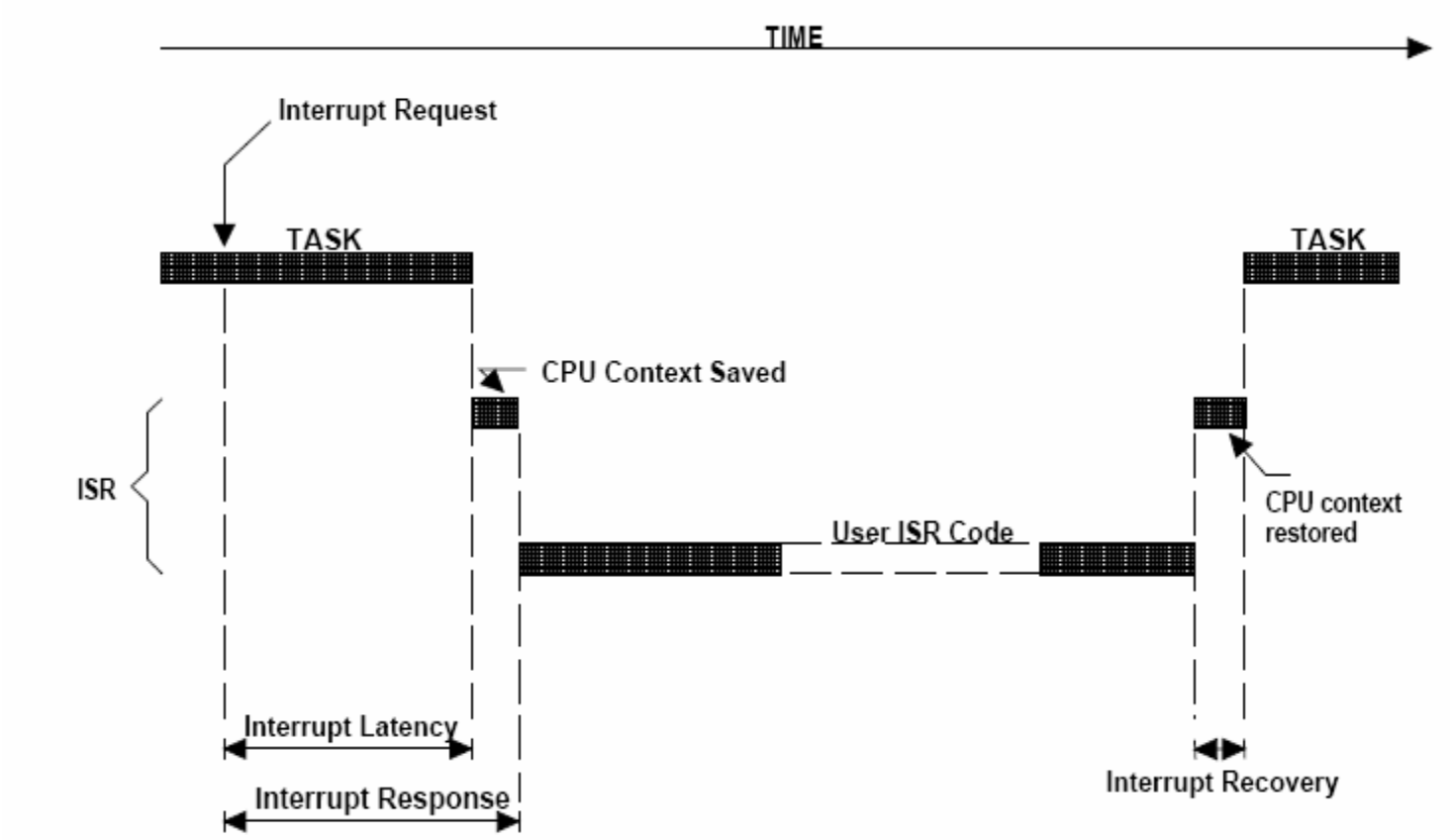
- **It is the time between the reception of the interrupt and start of the user code that handles the interrupt.**
  - **It is given by:**

Interrupt latency +
Time to save the CPU's context

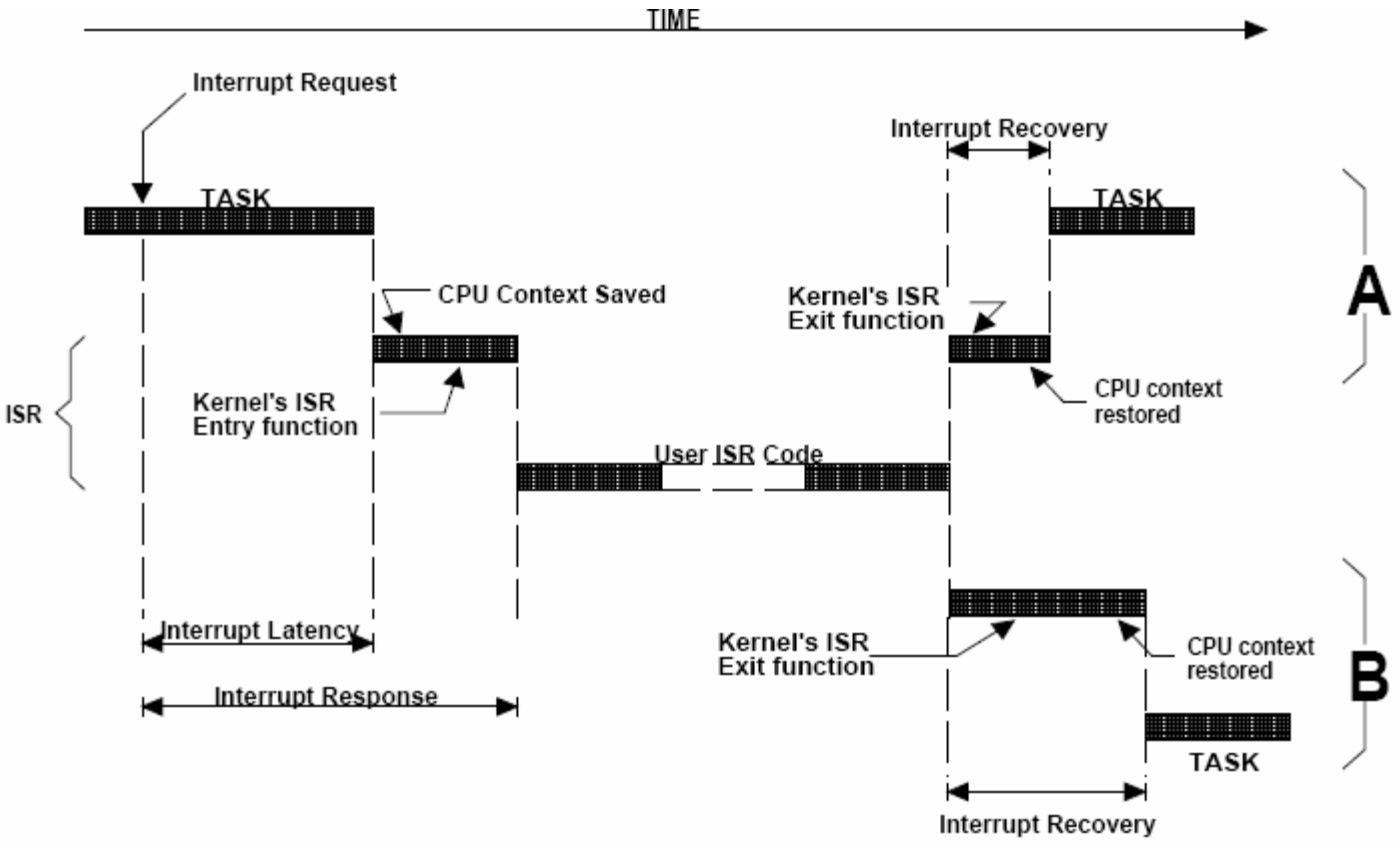
Interrupt Response(contd)

- **For Preemptive Kernel, an extra execution time of kernel ISR entry Function.**

Non-Preemptive Kernel



Pre-emptive Kernel



Tasks

- **Small piece of independent code or ‘Threads’.**
- **OS maintains information about each task---called as Task’s context.**
- **Uses a data structure for keeping track of tasks...called as Task control Block.**

Example-code

```
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;
};
```

Creating a Task

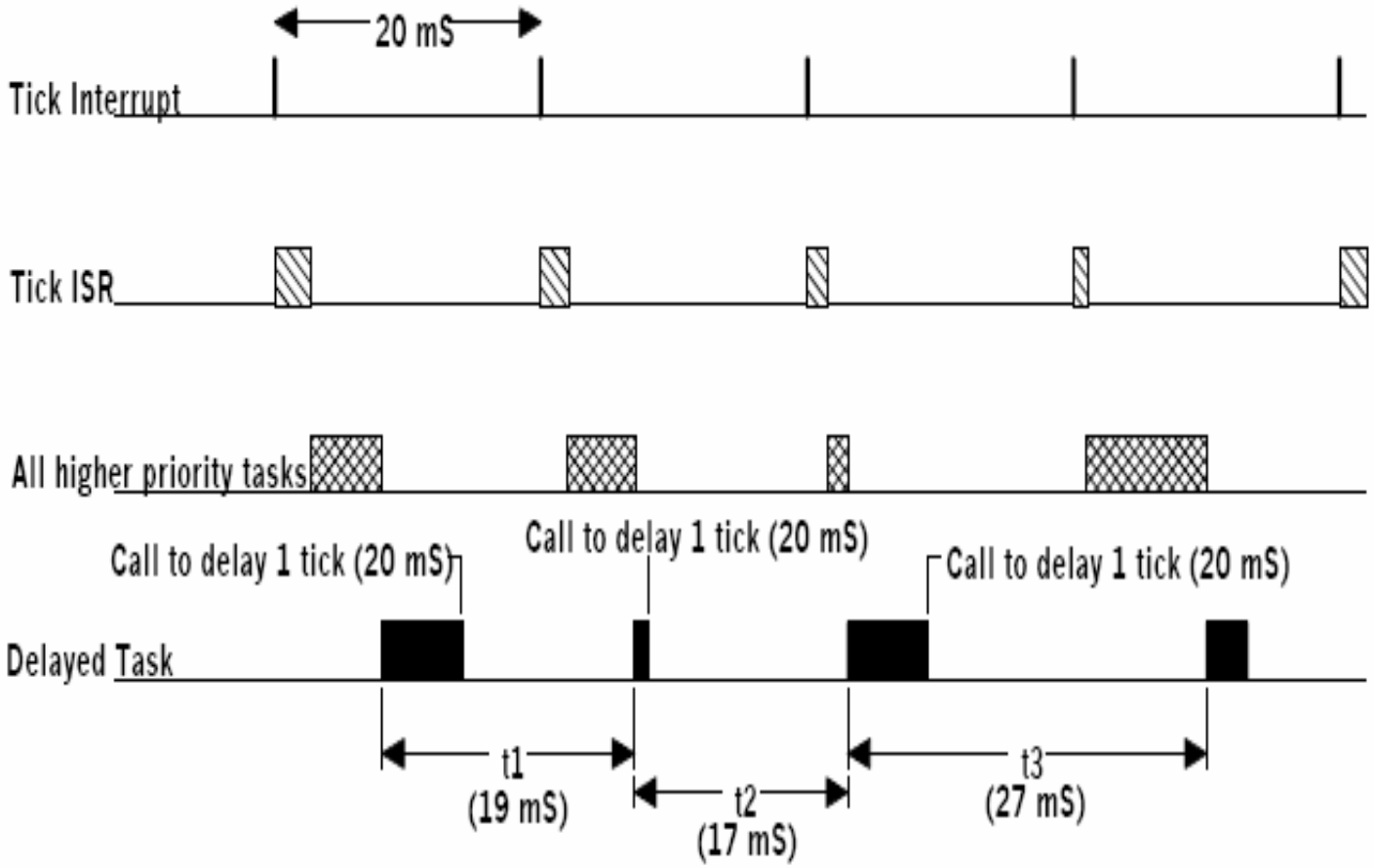
```
INT8U OSTaskCreate (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U prio)
{
    void   *psp;
    INT8U   err;


    if (prio > OS_LOWEST_PRIO) {                                          (1)
        return (OS_PRIO_INVALID);
    }
    OS_ENTER_CRITICAL();
    if (OSTCBPrioTbl[prio] == (OS_TCB *)0) {                              (2)
        OSTCBPrioTbl[prio] = (OS_TCB *)1;                                 (3)
        OS_EXIT_CRITICAL();                                               (4)
        psp = (void *)OSTaskStkInit(task, pdata, ptos, 0);                (5)
        err = OSTCBInit(prio, psp, (void *)0, 0, 0, (void *)0, 0);        (6)
        if (err == OS_NO_ERR) {                                           (7)
            OS_ENTER_CRITICAL();
            OSTaskCtr++;                                                  (8)
            OSTaskCreateHook(OSTCBPrioTbl[prio]);                         (9)
            OS_EXIT_CRITICAL();
            if (OSRunning) {                                             (10)
                OSSched();                                               (11)
            }
        } else {
            OSTCBPrioTbl[prio] = (OS_TCB *)0;                            (12)
        }
        return (err);
    } else {
        OS_EXIT_CRITICAL();
```

Clock Tick

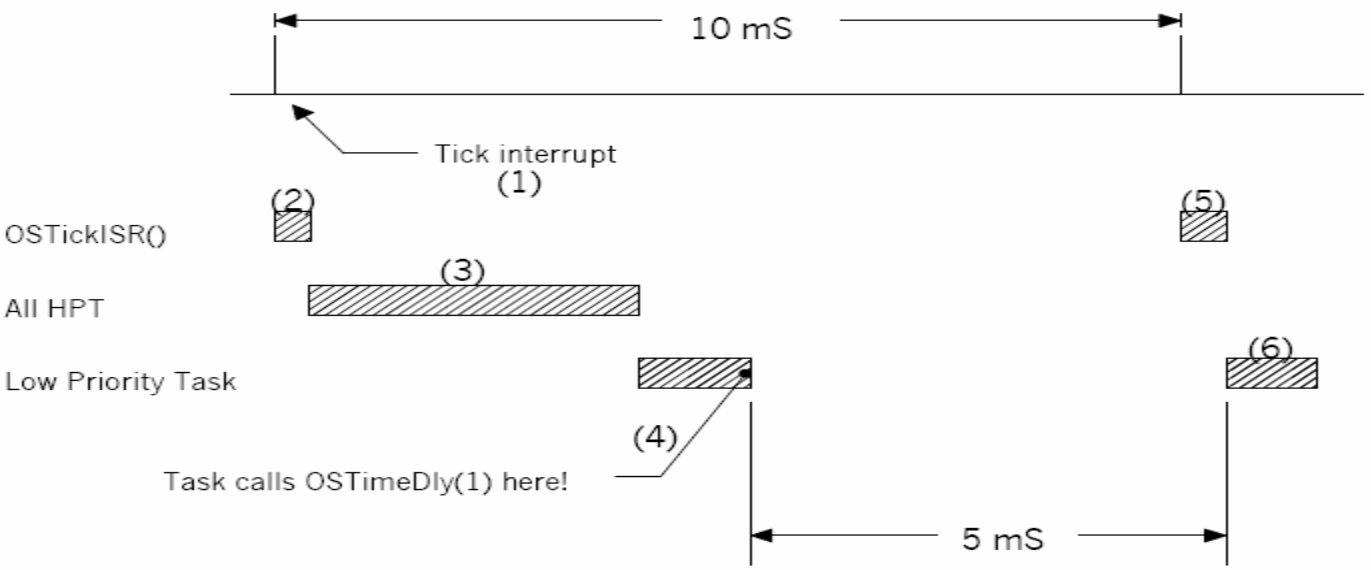
- **It's a special interrupt that occurs periodically.**
- **It is triggered by timer Interrupt.**
- **It keeps track of time delays and Timeouts.**
  - **Should occur between 10-100 times per second.**
  - **Defined by function OSTimeTick().**

Case 1 (Figure 2-24) shows a situation where higher priority tasks and ISRs execute prior to the task, which needs to delay for 1 tick. As you can see, the task attempts to delay for 20 mS but because of its priority, actually executes at varying intervals. This will thus cause the execution of the task to *jitter*.



Delaying a task, OSTimeDly()

- **It allows the calling task to delay itself for a user specified number of clock ticks.**



OSTimeDlyHMSM()

- **Specifying time in Hours, minutes, seconds and miliseconds.**
- **Resuming a delayed task: OSTimeDlyResume().**

Boot Strap Loader

- **A small program that loads the operating system into the computer's memory when the system is booted and also starts the operating system**

Porting UC/OS-II on Renesas

- **Loading uC/OS-II**
- **Initializing the hardware**
- **Building the application**

UCOS-II Hardware/Software Arch.