REAL TIME OPERATING SYSTEMS

Lesson-9:

INTERRUPT ROUTINES IN RTOS ENVIRONMENT AND HANDLING OF INTERRUPT SOURCE CALLS

1. INTERRUPT ROUTINES IN RTOS

ISRs in RTOSes

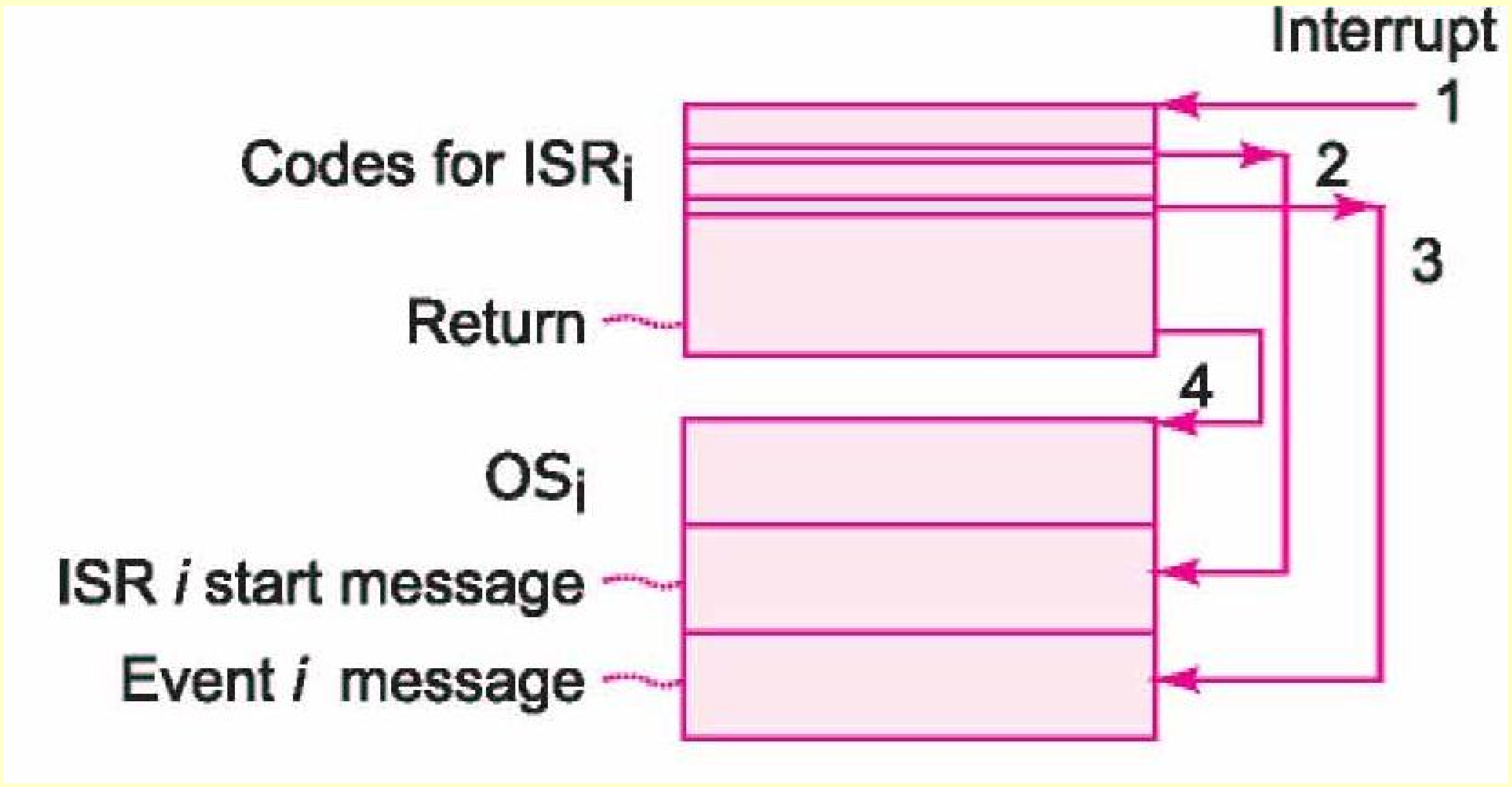
- ISRs have the higher priorities over the RTOS functions and the tasks.
- An ISR should not wait for a semaphore, mailbox message or queue message
- An ISR should not also wait for mutex else it has to wait for other critical section code to finish before the critical codes in the ISR can run.
- Only the IPC accept function for these events (semaphore, mailbox, queue) can be used, not the post function

RTOSes

- Three alternative ways systems to respond to hardware source calls from the interrupts

2. Direct Call to an ISR by an Interrupting Source and ISR sending an ISR enter message to OS

Direct Call to an ISR



Direct Call to an ISR and ISR Enter message

- **On an interrupt, the process running at the CPU is interrupted**
- **ISR corresponding to that source starts executing.**
- **A hardware source calls an ISR directly.**
- **The ISR just sends an ISR enter message to the RTOS.** ISR enter message is to inform the RTOS that an ISR has taken control of the CPU

ISR IPC messages and Exit message

- ISR code can send into a mailbox or message queue but the task waiting for a mailbox or message queue does not start before the return from the ISR
- When ISR finishes, it send s Exit message to OS
- On return from ISR by retrieving saved context, The RTOS later on returns to the interrupted process (task) or reschedules the processes (tasks).
- RTOS action depends on the event-messages, whether the task waiting for the event message from the ISR is a task of higher priority than the interrupted task on the interrupt

Event Message (IPC) from the ISR

- On certain RTOSes, there may be a function OSISRSemPost ( ).

Example in RTOS μCOS-II

- Assume that a microcontroller timer programmed to interrupt every 10 ms.
- Timer interrupt calls and program counter changes to an ISR vector address, ISR_Timer_Addr. At ISR_Timer_Addr, there is a routine ISR_Timer for the servicing the timer interrupt.
- ISR_Timer first executes OSIntEnter ( ) just after the start of ISR_Timer

Example in RTOS µCOS-II

- ISR_Timer then executes OSIntExit ( ) before the return code.
- OSIntEnter ( ) sends the message to RTOS that there should be return to the ISR only after any system call is made by the ISR until the OSIntExit ( ) executes in the ISR code.
- Any task waiting for the post of semaphore or mailbox message or queue message should not start on execution of the post function within the ISR or in any other task or ISR

Multiple ISRs Nesting

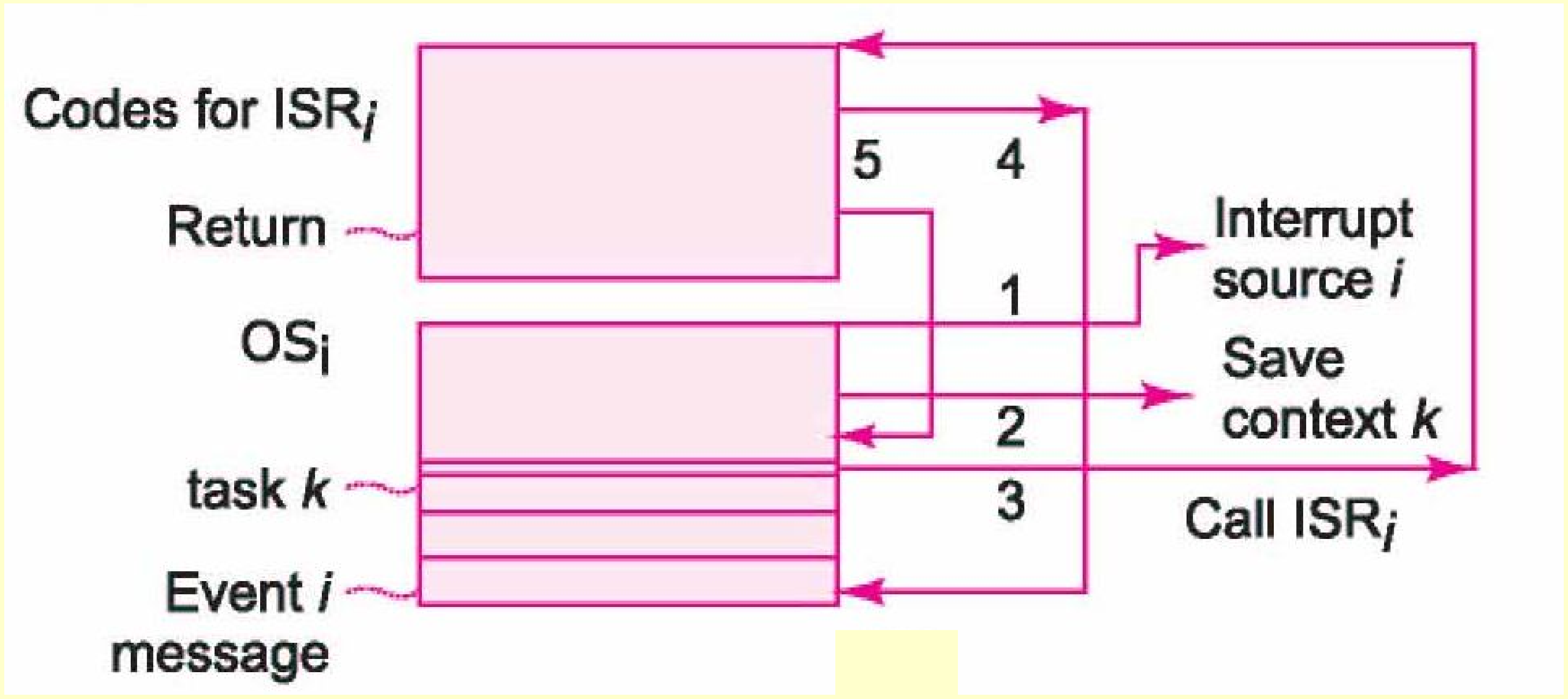
- Each ISR low priority sends on high priority interrupt the ISR interrupt message (ISM) to the OS to facilitate return to it on the finishing and return from the higher priority interrupt.
- Nesting means when an interrupt source call of higher priority, for example, SysClkIntr occurs, then the control is passed to higher priority and on return from the higher priority the lower priority ISR starts executing.

Multiple ISRs Nesting…

- Each ISR on letting a higher priority interrupt call sends the ISM to the RTOS.
- Common stack for the ISR nested calls, similar to the nested function calls.

3. RTOS first interrupting on an interrupt, then RTOS calling the corresponding ISR

RTOS first interrupting on an interrupt



RTOS interrupting on an interrupt

- On interrupt of a task, say, k-th task, the RTOS first gets itself the hardware source call and initiates the corresponding ISR after saving the present processor status (or context)
- Then the ISR during execution then can post one or more outputs for the events and messages into the mailboxes or queues.

RTOS first interrupting on an interrupt

- The ISR must be short and it must simply puts post the messages for another task.
- This task runs the remaining codes whenever it is scheduled.
- RTOS schedules only the tasks (processes) and switches the contexts between the tasks only.
- ISR executes only during a temporary suspension of a task.

Mobile PDA device example

- Each device event has the codes for an ISR, which executes only on scheduling it by the RTOS and provided an interrupt is pending for its service.
- Assume that using an RTOS, the touch screen ISR, ISR_TouchScreen has been created using a function OS_ISR_Create ( ).
- The ISR can share the memory heap with other ISRs.
- A function, IntConnect connects the touch screen event with the event identifier in an interrupt handler, ISR_handler.

Mobile PDA device example…

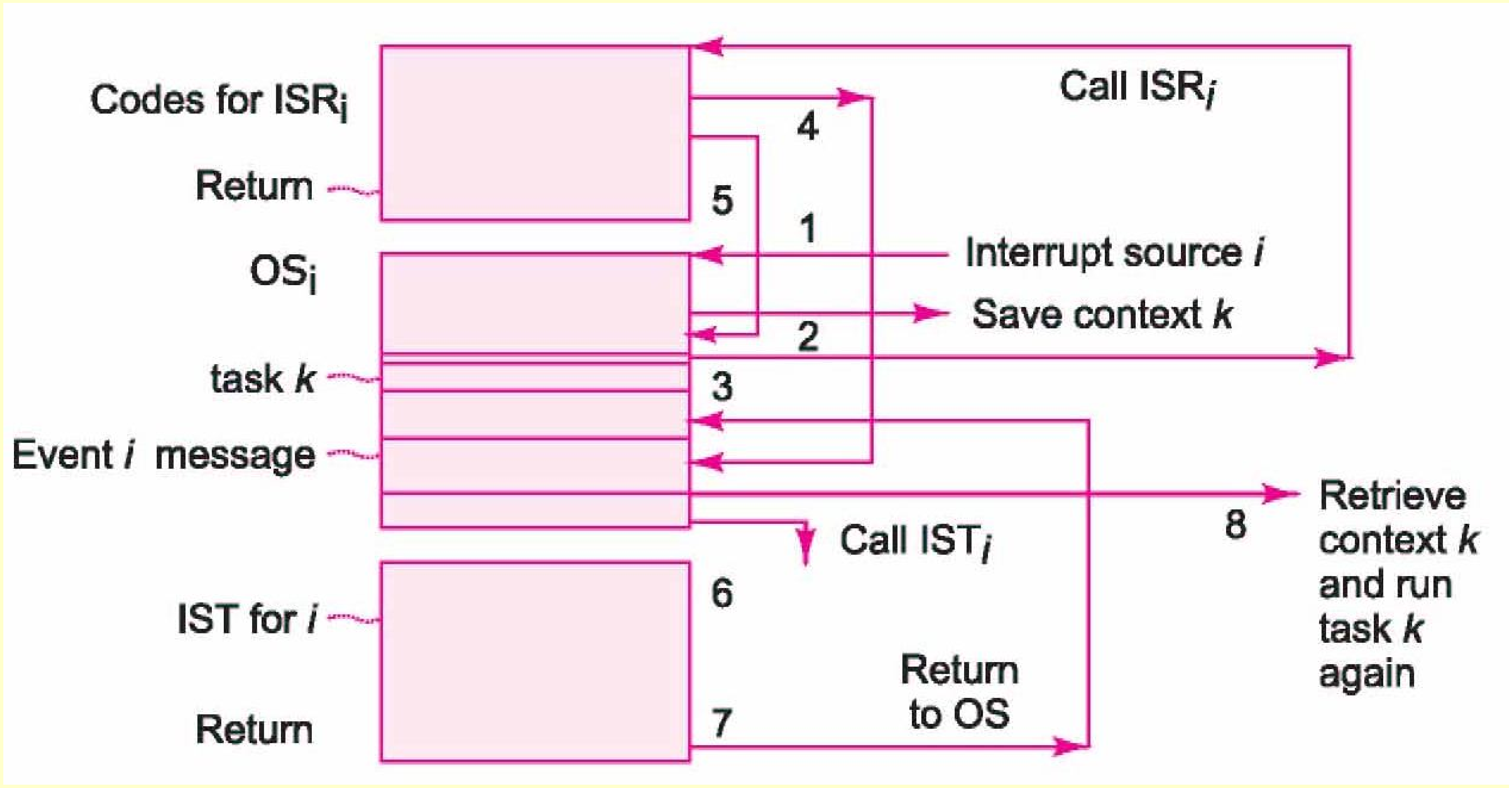
- When a touch screen event occur on tap at the screen to select icon or menu the OS sends signal on behalf of the ISR_handler to the waiting ISR_TouchScreen.
- ISR_TouchScreen runs on an interrupt call message
- ISR_TouchScreen executes as per its priority, IST_TouchScreenPriority among the other pending ISRs before it starts executing

Mobile PDA device example…

- Before return from the ISR_TouchScreen, it sends a message to kernel using a function OS_eventPost ( ) or OS_ISR_Exit ( ) just before the end of the codes in the ISR_TouchScreen

4. RTOS first interrupting on interrupt, then RTOS calling the corresponding ISR, the ISR sending messages to priority queue of Interrupt Service threads by Temporary Suspension of a scheduled Task

RTOS calling the corresponding ISR, the ISR sending messages to priority queue of Interrupt Service threads



RTOS calling the corresponding ISR, the ISR sending messages to IST

- An RTOS can provide for two levels of interrupt service routines, a fast level ISR, FLISR and a slow level ISR (SLISR).
- The FLISR can also be called hardware interrupt ISR and the SLISR as software interrupt ISR.
- FLISR is called just the ISR in RTOS Windows CE The SLISR is called interrupt service thread (IST) in Windows CE.

ISR sending messages to priority queue of Interrupt Service threads

- The use of FLISR reduces the interrupt latency (waiting period) for an interrupt service and jitter (worst case and best case latencies difference) for an interrupt service.
- An IST functions as deferred procedure call (DPC) of the ISR. An i-th interrupt service thread (IST) is a thread to service an i-th interrupt source call.

Windows NT

- Priority level of interrupts always higher than the user-thread priorities. [Threads are also assigned priorities.]
- Only very critical operations of ISR are performed and the remaining processing passed to a DPC or IST.
- DPCs executes ass priority FIFOs.

Window NT

- Even very low priority task ISR and DPC can preempt a real time process.
- DPCs run in FIFO mode. therefore, a DPC of the very low priority ISR may not release a resource. [For example, DPC of network IO or keypad interrupt]

Windows CE

- In Window CE provides for priority inheritance by the IST.
- Interrupt service threads (equivalent of DPCs) of high priority ISR will inherit the priority of the ISR.
- Window CE provides hard real time support.

Mac OS X– RTOS for iPod

- An interrupt handler first receives the primary interrupt and then it then generates a software interrupt known as a secondary interrupt. The secondary software interrupt using SWI is sent to initiate an IST
- There is three level strategy

Mac OS X

- OS does not receive the actual interrupt but the low level *LISR* intercepts the interrupt directly.
- When *LISR* is called, it resets the pending interrupt bit in the device interrupt controller and calls a device-specific ISR, say, DISRi.
- The DISRi posts a message to an ISTi specific to the device.

Mac OS X

- The message notifies to ISTi that an interrupt has occurred, and then the DISRi returns to LISR.
- *LISR* resets another pending interrupt bit in the device interrupt controller and calls the another device-specific ISR, say, DISRj.

Mac OS X…

- When no further interrupts are pending, the OS control returns to the currently executing thread, which was interrupted and when the OS passed control to the LISR.
- The IST i are scheduled by the OS, the IST i finds that the SWI interrupt has occurred, it starts and run the codes.
- ISTs run as if a thread is running

Summary

We learnt

- RTOS uses one of three strategies on interrupt source calls
- (i) An ISR directly servicing directly , after and ISR just merely informing the RTOS on enter and sends exit information before return.

We learnt

- (ii) Kernel intercepting the call and calling the corresponding ISR and tasks.
- RTOS kernel schedules only the tasks (processes) and ISR executes only during a temporary suspension of the task by the RTOS

We learnt

- (iii) Kernel intercepting the call and calling the ISR, which initiates and queues the ISR call into a priority FIFO. The RTOS kernel schedules the ISTs and tasks (processes) as per priorities
- In MAC OS X, there is hardware level LISR, which calls device specific DISR, which posts messages to ISTs

End of Lesson 9 of Chapter 8