Inter-Process Communication and Synchronization of Processes, Threads and Tasks:

Lesson-5: Function, Task and ISR

Function

Function

- Function is an entity used in any program, function, task or thread for performing specific set of actions when called and on finishing the action the control returns to the function calling entity (a calling function or task or process or thread).

Function

- Each function has an ID (name)
- has program counter and
- has its stack, which saves when it calls another function and the stack restores on return to the caller.
- Functions can be nested. One function call another, that can call another, and so on and later the return is in reverse order

Interrupt Service Routine

Interrupt Service routine

- ISR is a function called on an interrupt from an interrupting source.
- Further unlike a function, the ISR can have hardware and software assigned priorities.
- Further unlike a function, the ISR can have mask, which inhibits execution on the event, when mask is set and enables execution when mask reset

Task

Task

- Task defined as an executing computational unit that processes on a CPU and state of which is under the control of kernel of an operating system.

Distinction Between Function, ISR and Task

Uses

- Function— for running specific set of codes for performing a specific set of actions as per the arguments passed to it
- ISR— for running on an event specific set of codes for performing a specific set of actions for servicing the interrupt call
- Task — for running codes on context switching to it by OS and the codes can be in endless loop for the event (s)

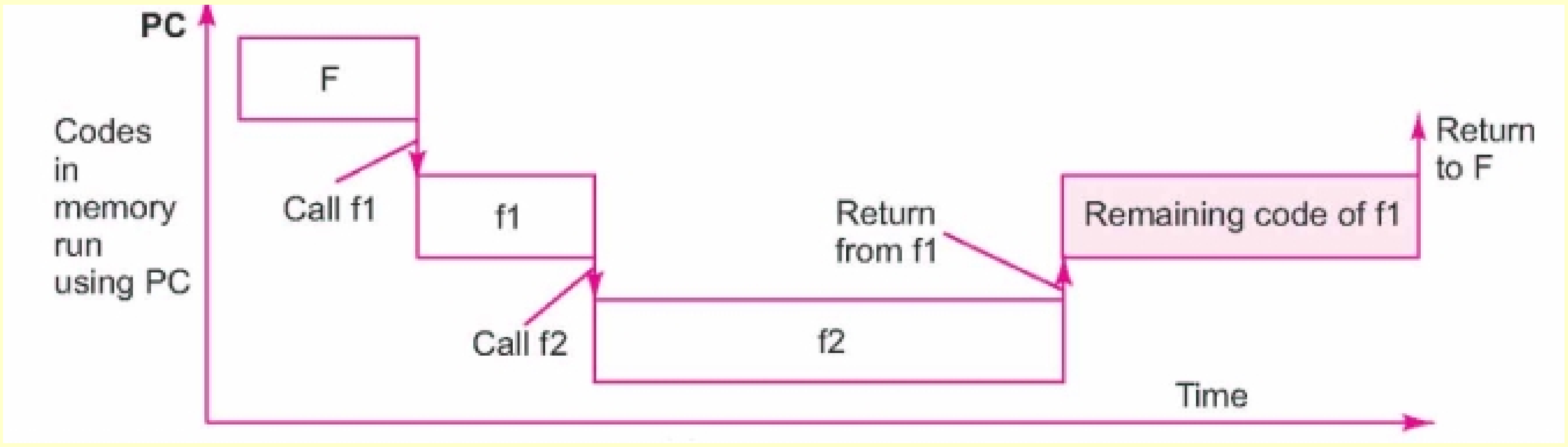
Calling Source

- Function— call from another function or process or thread or task
- ISR— interrupt-call for running an ISR can be from hardware or software at any instance
- Task — A call to run the task is from the system (RTOS). RTOS can let another higher priority task execute after blocking the present one. It is the RTOS (kernel) only that controls the task scheduling

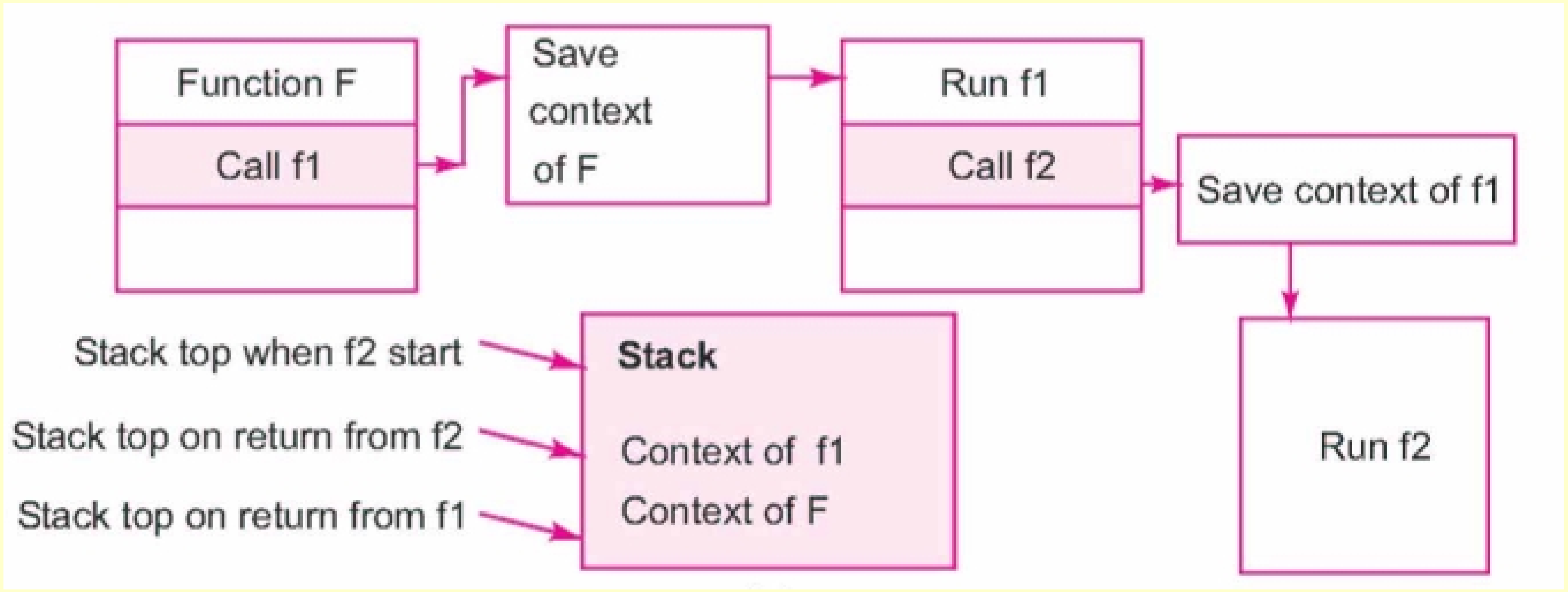
Context Saving

- Function— run by change in program counter instantaneous value. There is a stack. On the top of which the program counter value (for the code left without running) and other values (called functions' context) save
- All function have a common stack in order to support the nesting

PC Changes on nested function calls



Common Stack and Nested Function calls



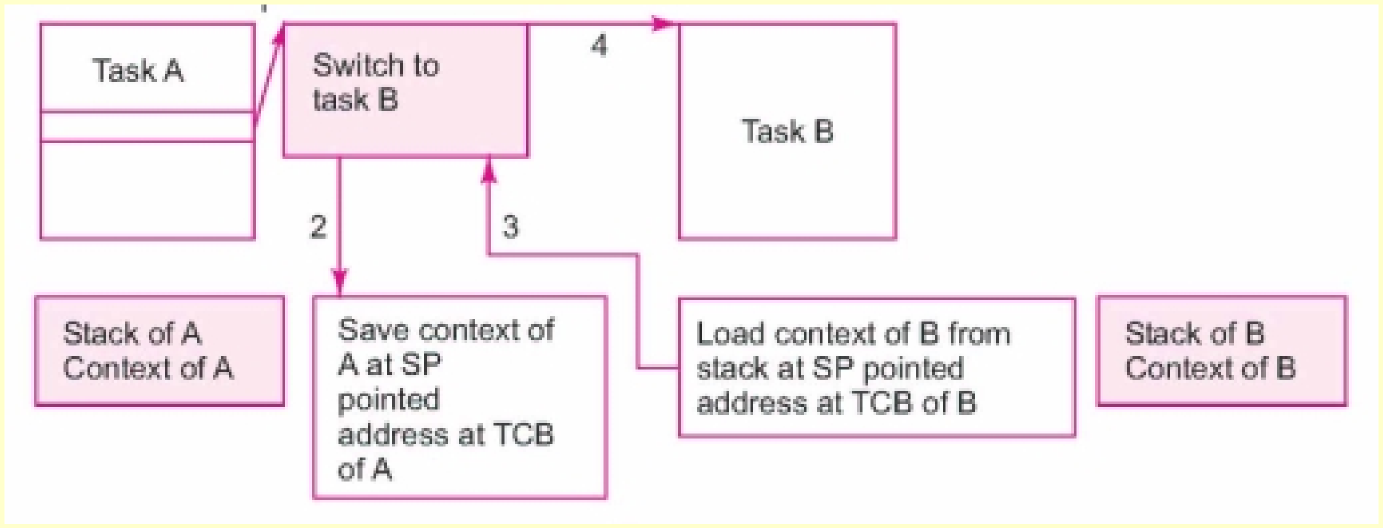
Context Saving

- ISR— Each ISR is an event-driven function code. The code run by change in program counter instantaneous value. ISR has a stack for the program counter instantaneous value and other values that must save.
- All ISRs can have common stack in case the OS supports nesting

Context Saving

- Task ─ Each task has a distinct task stack at distinct memory block for the context (program counter instantaneous value and other CPU register values in task control block) that must save
- Each task has a distinct process structure (TCB) for it at distinct memory block

Tasks and their separate contexts


Task A
Switch to task B
4
Task B
2
3
Stack of A Context of A
Save context of A at SP pointed address at TCB of A
Load context of B from stack at SP pointed address at TCB of B
Stack of B Context of B

Response and Synchronization

- Function─ nesting of one another, a hardware mechanism for sequential nested mode synchronization between the functions directly without control of scheduler or OS

Response and Synchronization

- ISR— a hardware mechanism for responding to an interrupt for the interrupt source calls, according to the given OS kernel feature a synchronizing mechanism for the ISRs, and that can be nesting support by the OS

Response and Synchronization

- Task ─ According to the given OS kernel feature, there is a task responding and synchronizing mechanism. The kernel functions are used for task synchronisation because only the OS kernel calls a task to run at a time. When a task runs and when it blocks is fully under the control of the OS

Structure

- Function— can be the subunit of a process or thread or task or ISR or subunit of another function

Structure

- ISR— Can be considered as a function, which runs on an event at the interrupting source.
- A pending interrupt is scheduled to run using an interrupt handling mechanism in the OS, the mechanism can be priority based scheduling.
- The system, during running of an ISR, can let another higher priority ISR run.

Structure

- Task ─ is independent and can be considered as a function, which is called to run by the OS scheduler using a context switching and task scheduling mechanism of the OS.
- The system, during running of a task, can let another higher priority task run. The kernel manages the tasks scheduling

Global Variables Use

- Function— can change the global variables. The interrupts must be disabled and after finishing use of global variable the interrupts are enabled

Global Variables Use

- ISR─ When using a global variable in it, the interrupts must be disabled and after finishing use of global variable the interrupts are enabled (analogous to case of a function)

Global Variables Use

- Task ─ When using a global variable, either the interrupts are disabled and after finishing use of global variable the interrupts are enabled or use of the semaphores or lock functions in critical sections, which can use global variables and memory buffers

Posting and Sending Parameters

- Function— can get the parameters and messages through the arguments passed to it or global variables the references to which are made by it. Function returns the results of the operations

Posting and Sending Parameters

- ISR— using IPC functions can send (post) the signals, tokens or messages. ISR can't use the mutex protection of the critical sections by wait for the signals, tokens or messages

Posting and Sending Parameters

- Task ─ can send (post) the signals and messages
- can wait for the signals and messages using the IPC functions,
- can use the mutex or lock protection of the code section by wait for the token or lock at the section beginning and messages and post the token or unlock at the section end

Summary

We learnt

- **Function** is used in any routine for performing specific set of actions as per the arguments passed to it and which runs when called by a process or task or thread or from another function.
- Functions run by nesting and thus have a common stack.
- Function runs after the previous context saving and after retrieving the context from a common stack

We learnt

- **An ISR** is a function, which executes on interrupts (events).
- Executes on an event and pending ISRs run as per priority based scheduling.
- Can post the events or signals or messages ISRs generally run by nesting.
- Runs after the context saving and after retrieving the context
- Generally OS provides for a common stack for ISRs.
- Runs as per the hardware based interrupt-handling mechanism

We learnt

- **A task** is a function, which executes on scheduling and has independent stack and context, can wait as well as post the events or signals or messages.
- Runs after saving of the previous context at the task TCB and the context switching to new context at the new task TCB.
- The tasks run as per the task scheduling and IPC management mechanism of the OS

End of Lesson 5 of Chapter 7