

REAL TIME OPERATING SYSTEMS

Lesson-21: Critical Section Handling

1. Disabling and enabling interrupts

Critical Section Service by disabling and enabling interrupts


- Critical section is a section in a system call (OS function), where there is no preemption by tasks.
- A disable interrupts function can be used at beginning of critical section and enable interrupts function executed at exit from the critical section to prevent preemption by tasks as well as ISRs.

Preemption Points in μ COS-II

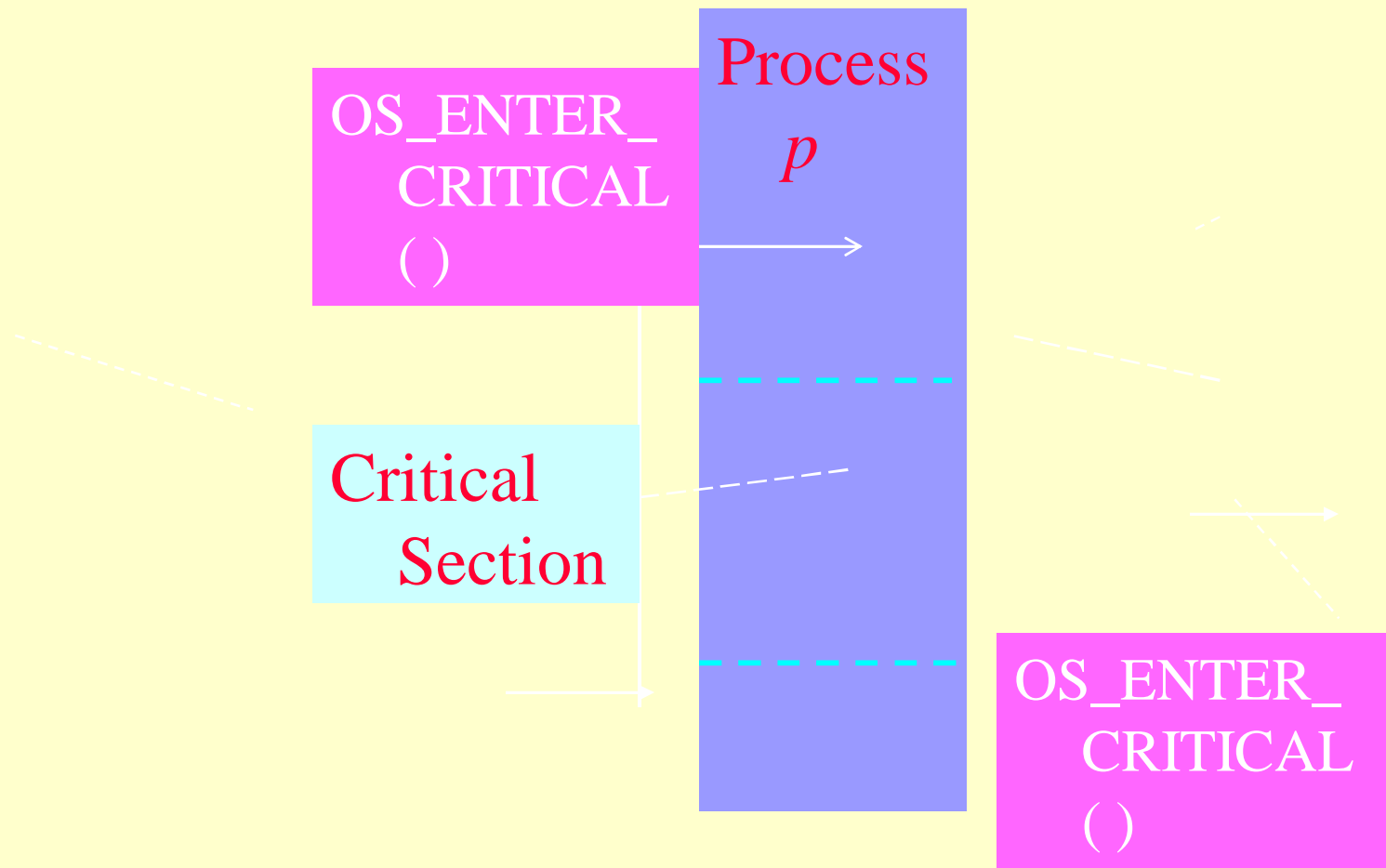
- RTOS μ COS-II provides a function `OS_ENTER_CRITICAL ()` to stop preemption by any task or ISR, as it disable interrupts. The RTOS provides a function `OS_EXIT_CRITICAL ()` to facilitate preemption by high priority task or ISR, as it enable interrupts.

Process critical Section

```
static void process_p (void *taskPointer) { ...  
while (1) { ...; ...; ...;  
OS_ENTER_CRITICAL ( ) /*to stop preemption  
or interrupts by any task or ISR, after this  
instruction executes, critical section starts*/  
...; ...; ...;  
OS_ENTER_CRITICAL ( ) /* to start  
preemption or interrupts by any task or ISR */  
...; ...; ...; } }
```



Processes p Critical Section Using enabling and disabling interrupts



2. Disabling and enabling preemption by other processes by using lock () and unlock ()

Critical Section Service by lock () and unlock () in a Preemptive Scheduler

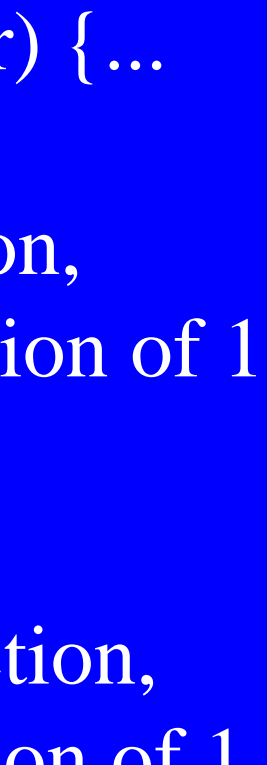
- A lock function can be used at beginning of critical section and an unlock function executed at exit from the critical section.

RTOS μ COS-II lock () and unlock ()

- RTOS μ COS-II provides a function OSSchedLock () to lock scheduling and hence locks preemption by other task waiting to proceed further for taking the lock ()
- OSSchedUnlock () unlocks scheduling and hence unlocks preemption by other task waiting to proceed further after executing lock ()


Process p critical Section

```
static void process_p (void *taskPointer) { ...  
while (1) { ...; ...; ...;  
OSSchedLock ( ); /* after this instruction,  
preemption is disabled and critical section of 1  
starts*/  
...; ...; ...;  
OSSchedUnlock ( ); /* after this instruction,  
preemption is enabled and critical section of 1  
ends*/...; ...; ...; } }
```

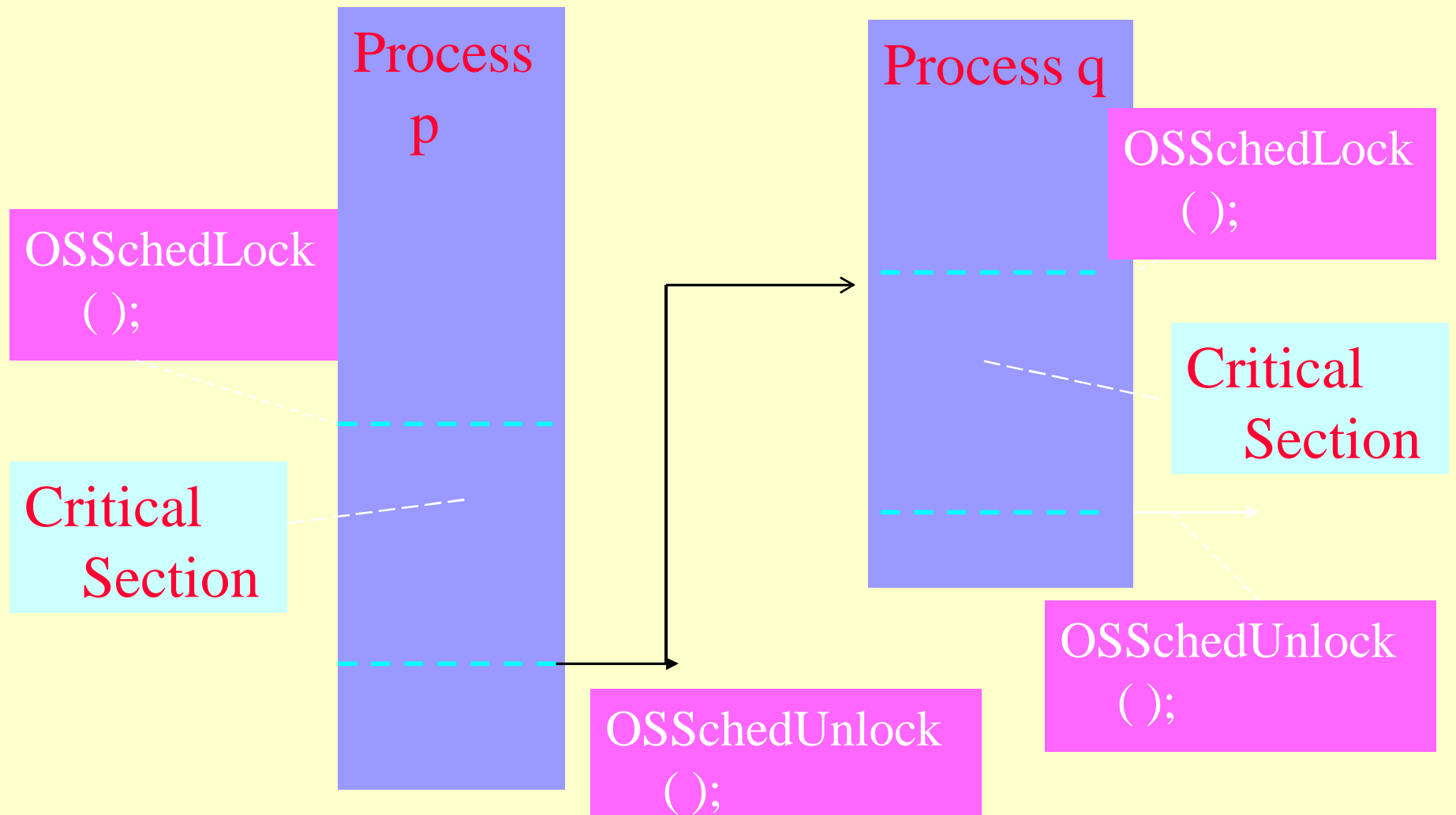


Process q critical Section

```
static void process_q (void *taskPointer) {..  
while (1) {...; ...; ...;  
OSSchedLock ( ); /* after this instruction,  
preemption is disabled and critical section of 1  
starts*/  
...; ...; ...;  
OSSchedUnlock ( ); /* after this instruction,  
preemption is enabled and critical section of 1  
ends*/...; ...; ...; } }
```



Processes Critical Sections [using lock and unlock () to prevent and permit preemption by any other process]



Preemption Points in Windows CE

- (a) RTOS kernels, for example, Windows CE, provide for preemption points. These are the OS function codes in between the critical sections.


3. Taking and releasing semaphore (mutex) by processes by using semTake() and semGive ()

Critical Section Service by semTake () and semGive () in a Preemptive Scheduler

- A mutex semaphore can be used.
- Take the mutex at critical section 1 start
- Release the mutex at critical section end
- Same mutex is then taken and released by another critical section 2 when the access to the section is permitted after taking semaphore at an instance

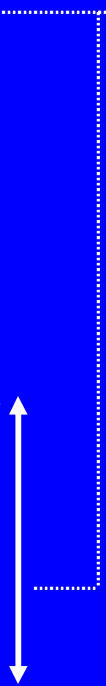
Process 1 critical Section

```
static void process_1 (void *taskPointer) { ...  
while (1) { ...; ...; ...;  
OSSemtake (Sem_m); /* after this instruction  
executes, critical section of 1 starts and the  
shared variable can be operated)*/  
...; ...; ...;  
OSSemPost (Sem_m); /* after this instruction  
executes the next process critical section can  
operate on the shared variable */  
...; ...; ...; } }
```

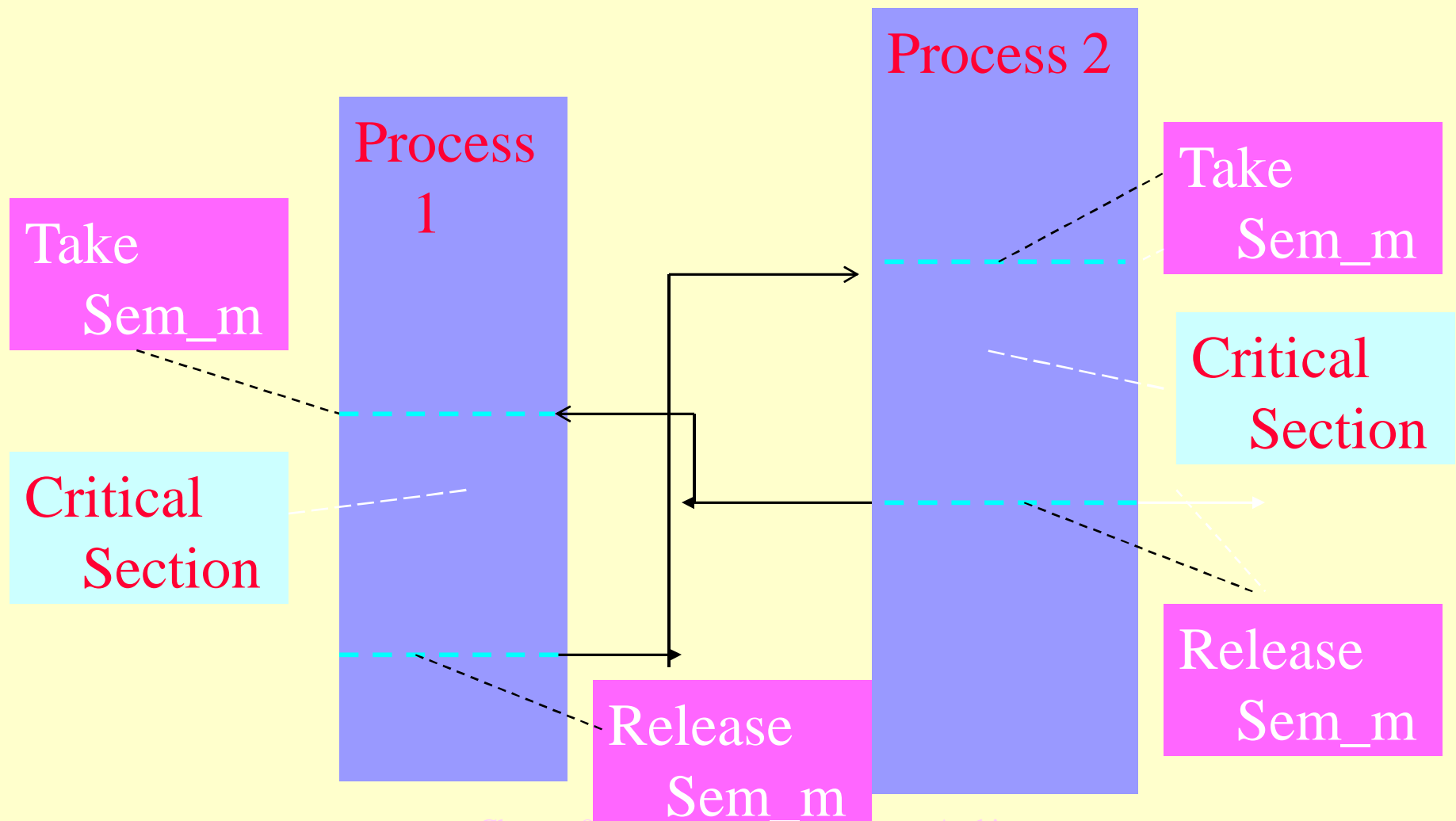


Process 2 critical Section

```
static void process_2 (void *taskPointer) {..  
while (1) {...; ...; ...;  
OSSemTake (Sem_m); /* after the instruction  
executes Sem_m = 0 (taken state)*/  
...; ...; ...;  
OSSemPost(Sem_m); /* Release mutex to let  
process 1 section proceed further if of higher  
priority */ ...; ...; ...; } }
```



Processes 1 and 2 Critical Sections semaphore (used as mutex); preemption and interrupts permitted



Summary

We learnt

Critical sections can be handled by:

- disabling interrupts,
- disabling process preemption by lock () function and
- taking and releasing semaphore (mutex)

We learnt

- Preemption of a low priority tasks takes place when higher priority task becomes ready due to wait-over from timeout or getting necessary input or IPC

End of Lesson 21 of Chapter 8