CENG 336
INT. TO EMBEDDED SYSTEMS DEVELOPMENT
Spring 2015
Recitation08

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:

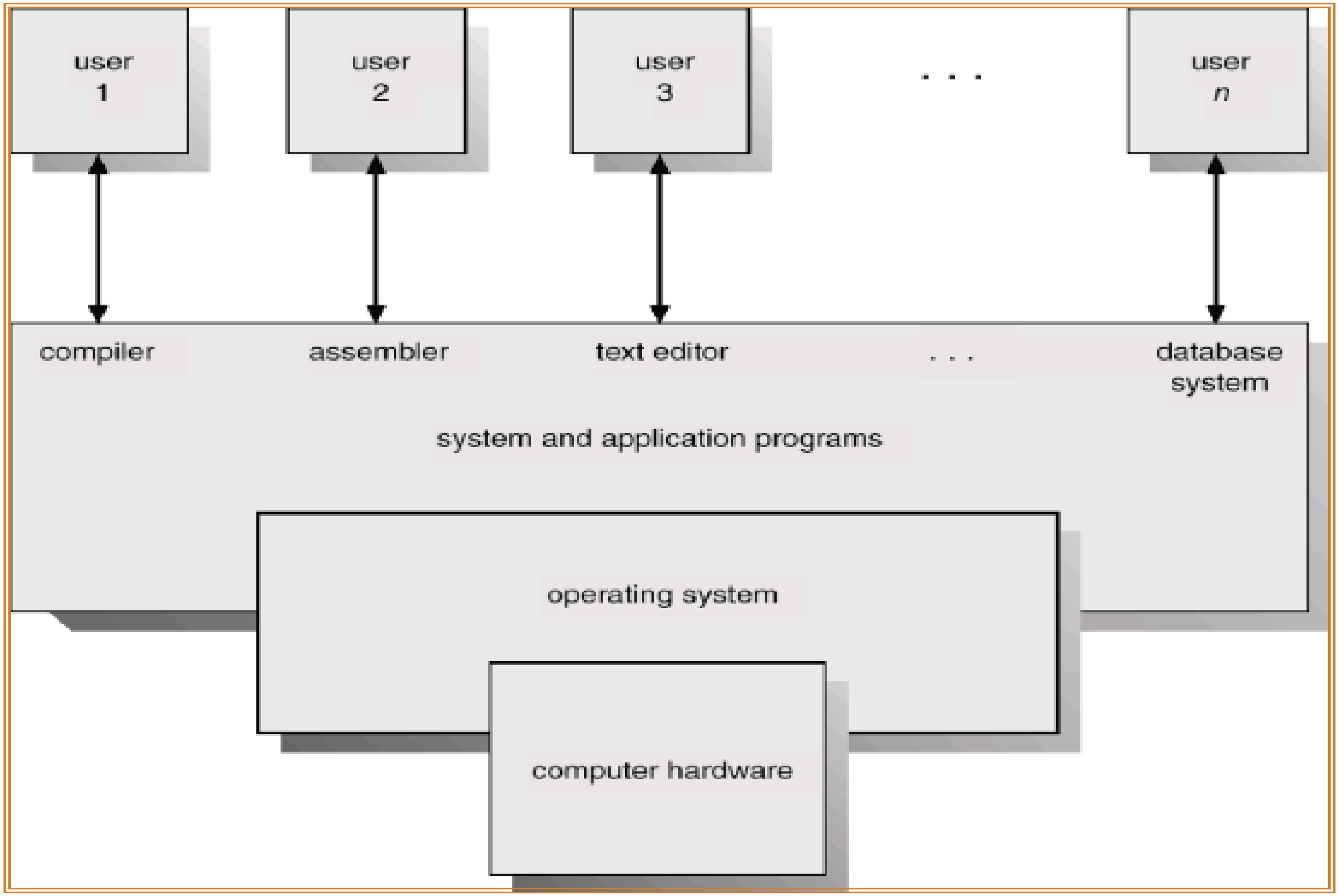
  - Execute user programs and make solving user problems easier.

  - Make the computer system convenient to use
- Use the computer hardware in an efficient manner

Computer System Components

1. **Hardware:** provides basic computing resources (CPU, memory, I/O devices)
2. **Operating system:** controls and coordinates the use of the hardware among the various application programs for the various users
3. **Applications programs:** define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs)
4. **Users** (people, machines, other computers)

Abstract View of System Components



What is an RTOS?

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
- Well-defined fixed-time constraints

Terminology

- **Critical section**, or critical region, is code that needs to be treated indivisibly.
  - No interrupts
  - No context switch
- **Resource** is an entity used by a task.
  - Printer, keyboard, CAN bus, serial port
- **Shared resource** is a resource that can be used by more than one task.
  - mutual exclusion
- **Multitasking** is the process of scheduling and switching the CPU between several tasks.

KERNEL

- **Kernel** is a set of functionalities, regroup under the term SERVICES for most of them.

  - e.g. In the case of Linux, the kernel is composed of the task manager, the hardware access manager, the file system manager

  - One of the most famous functionality of the kernel (without being a service) is the **SCHEDULER** in charge of the task processing in parallel.

KERNEL

Non-preemptive kernels, also cooperative multitasking:

- The task needs to explicitly give up control of the CPU.
- Allows low interrupt latency, because they may be never disabled.
- Allows non-reentrant functions at the task level.
- Response time is determined by the longest task.
- No overhead for protecting shared data.
- Responsiveness may be low, because of low priority task requiring a lot of time until it releases the CPU.

KERNEL

Preemptive kernel:

- Responsiveness is good, because tasks get preempted.
- A higher-priority task can preempt a lower priority task that still requires more time to compute.
- Response time becomes deterministic, because at the next tick, the OS switches to the other new task.
- Non-reentrant functions require careful programming.
- Periodic execution of the ‘tick’ adds to the overhead.

PICos18

PICOS18 is a preemptive RTOS for the PIC18 series.

PICOS18 provides:

- Core services: initialization, scheduling
- Alarm and counter manager
- Hook routines
- Task manager
- Event manager
- Interrupt manager

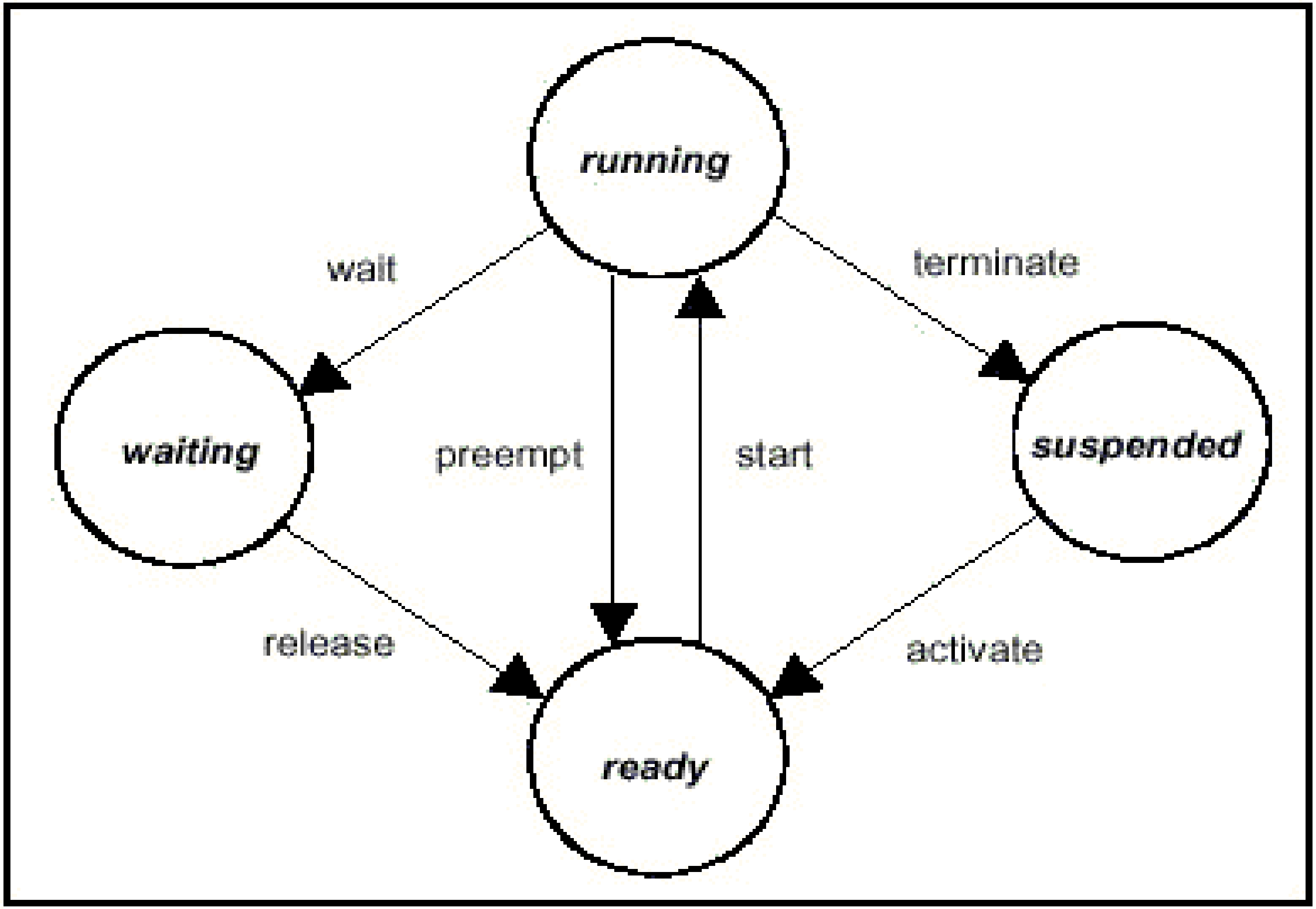
TASK

- Task, also called thread, is a user application.

  - Shares the CPU and resources with other tasks

  - Follows a defined life cycle

TASK



There are 4 possible task states in PICos18

TASK

There are 4 task states in PICos18:

- **SUSPENDED:** The task is present in the application but is not taken into account by the kernel.
- **READY:** The task is available to be executed by the kernel then is taken into account by the scheduler.
- **WAITING:** The task is sleeping and so is temporarily SUSPENDED and will be READY as soon as an event occurs.
- **RUNNING:** There is only one task running at a certain time => this is the task in a READY state with the highest priority.

TASK

Context Switches

- A context switch occurs whenever the multitasking kernel decides to run a different task.

  - Save the current task's context in the storage area.

  - Restores the new task's context from the storage area.

  - Resumes the new task
- Context switching adds overhead.
- The more registers a processor has, the higher the overhead => irrelevant for RTOS as long as its known.
- Context swithching is done by kernel.

TASK

Software Stack

- PICos18 is compliant with the C18 software stack management. Each task has its own software stack so that each task of the application can work as alone in its own private space without disturbing the other processing. Only the 3 below values are possible:
  - A minimum software stack is 64.
  - A maximum software stack is 256.
  - A typical stack size is 128.

TASK

Task definition:

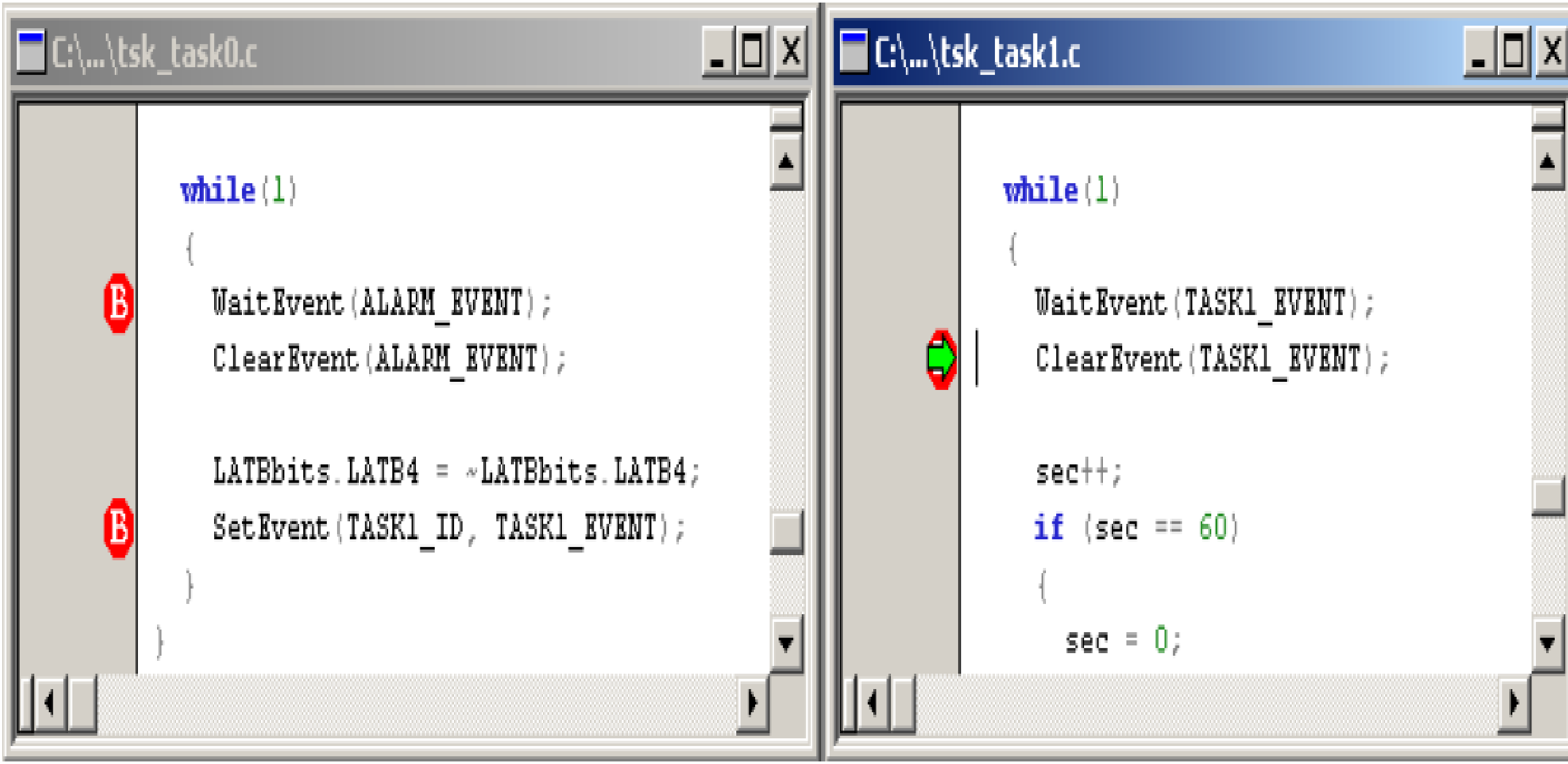
```
/**********************************************************************
 * ----------------------------  task 0 -----------------------------
 **********************************************************************/
rom_desc_tsk rom_desc_task0 = {
   TASK0_PRIO,                      /* prioinit from 0 to 15      */
   stack0,                          /* stack address (16 bits)    */
   TASK0,                           /* start address (16 bits)    */
   READY,                           /* state at init phase        */
   TASK0_ID,                        /* id_tsk from 1 to 15        */
   sizeof(stack0)                   /* stack size    (16 bits)    */
};
```

ALARMS

```
AlarmObject Alarm_list[] =
  {
   /*******************************************************************
    * ------------------------- First task ---------------------------
    *******************************************************************/
   {
     OFF,                                    /* State                */
     0,                                      /* AlarmValue           */
     0,                                      /* Cycle                */
     &Counter_kernel,                        /* ptrCounter           */
     TASK0_ID,                               /* TaskID2Activate      */
     ALARM_EVENT,                            /* EventToPost          */
     0                                       /* CallBack             */
   },
 };
```

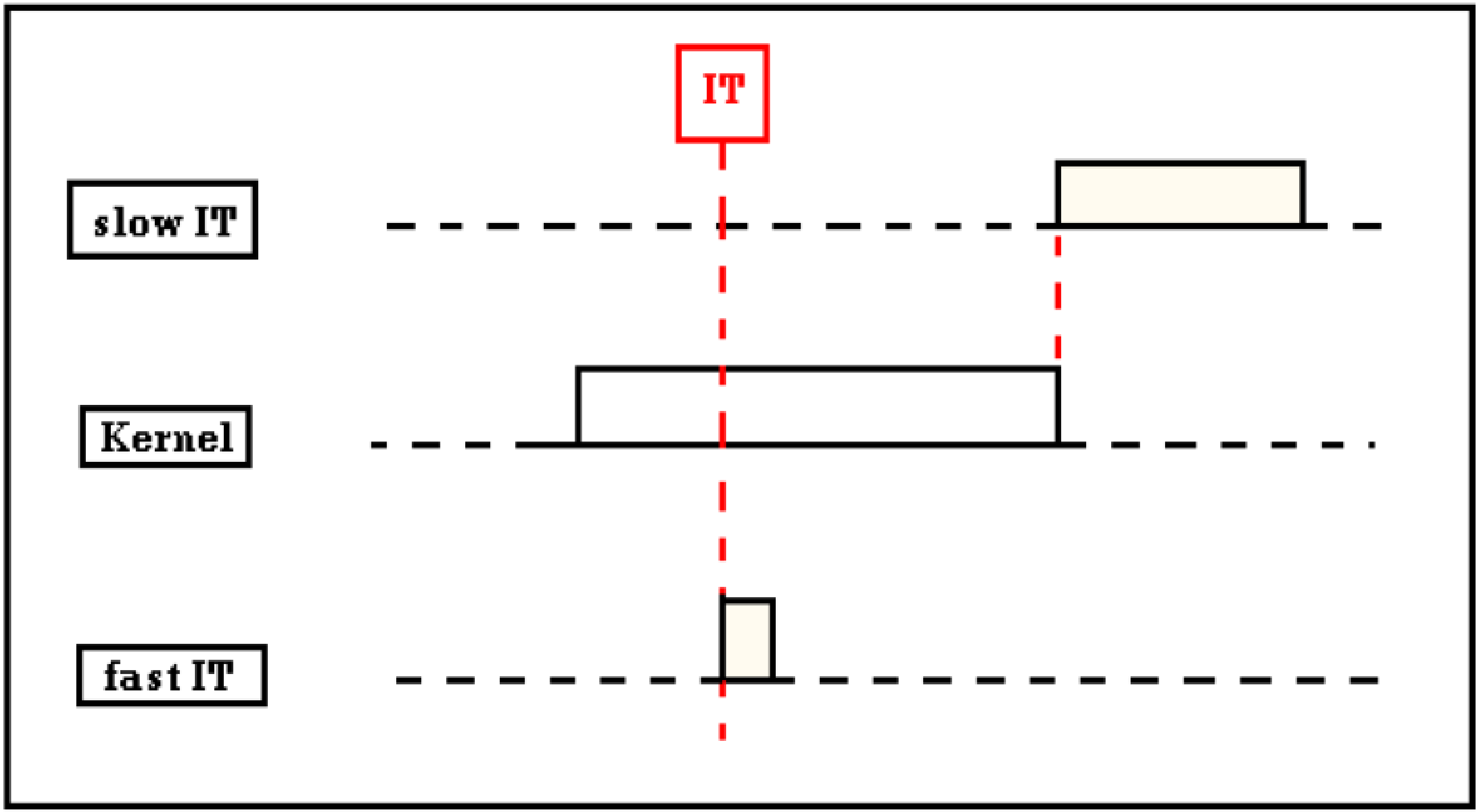
- Each tick the alarm counters get incremented by one.
- If the alarm value equals the counter value, then the alarm will cause an event.

PREEMPTION



Assume **task0_priority > task1_priority**. How does it run through?
Assume **task1_priority > task0_priority**. How does it run through?

INTERRUPTS



IMPORTANT THINGS

- ***Note1:*** You can put to sleep your task on any event but keep in mind the **event** must be defined as a power of 2 in the "define.h" file. The allowed values are : 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02 et 0x01.
- ***Note2:*** The preemption is not a sufficient reason to say that a kernel is a real-time kernel or not. To do it we have to talk about the **determinism,** it means the fact a kernel is able to warranty the time necessary to switch from one task to another one is a constant. With PICos18 the time **(latency time)** is 50 µs then each time the task 0 sends an event to the task 1 the time to switch from task 1 to task 0 is 50 µs.
- ***Note3:*** With ‘Round Robin’ algorithm the idle tasks (tasks with the lowest priority) can share the PIC18 processor during a time slice of 1ms if they have the equal priority.

REFERENCES

- **http://www.cis.upenn.edu/~lee/06cse480/lec-rtos.pdf**
- PICos18 Tutorial
- PICos18 API