Chapter 2, Section 2.4: Architecture and Characteristics of a RTOS kernel (µC/OS-II)

1. Introduction
2. Interrupt Management
3. Task Management
4. *Time Management*
5. Event Management
6. Memory Management
7. Porting µCOS-II

4. Time Management (Timers)

- µC/OS-II (and VxWorks) needs a periodical interrupt to update the delay of each waiting task (OSTimeDly or taskDelay).
- This is made by an hardware device called Timer
- The Timer is part of the processor peripherals.

4. Time Management (Timers)

- **A Timer is exactly like a counter.**
- **Generally, it consists in a 16-bit or 32-bit counter**
- **Given a 16-bit timer, it can be loaded with a value *x* between 0 and 65535. *x* is determined by the clock period and desired scheduling range.**
- **Then, the *x* value is decremented by 1 at every hardware clock tick of the counter and when 0 is reached, a interrupt signal is raised.**

4. Time Management (Timers)

- **The ISR (Interrupt Service Routine) associated with the timer interrupt is executed :**
  - It increments the OS ticks number
  - It updates the task delay of each waiting task
  - If a task delay is over (i.e. equals to 0), the ISR checks to reschedule the task.

4. Time Management (Timers)

- **A periodical interrupt is called "clock tick" and occurs from 10 to 100 times per second (Hz).**
- **Given counter frequency 2MHz and the scheduling range 10 ms, the initialization value of the timer is:**

$$(10 * 10^{-3}) / (.5 * 10^{-6}) = 20000$$

- **A higher clock frequency involves a higher "overhead" in the system (too much context switches). e.g. it is about μs.**

4. Time Management (Timers)

- **A RTOS provides generic interfaces to handle 2 timers :**
  - System clock
  - Auxiliary clock (if available)

4. Time Management (Timers)
« Watchdog timers »

- A « watchdog timer » represents the software interface of the system clock.
- This interface allows user routines to be executed after a determined delay.
- Once this delay is reached, the routine is exectued as part of the ISR of the system clock.

4. Time Management (Timers)
« Watchdog timers »

- µC/OS-II does not handle special routines for “watchdog timers”.
- Yet, in µC/OS-II, we can easily add these routines by modifying the clock ISR code.
- We can take inspiration from VxWorks…

Example of VxWorks

- **To create a watchdog timer:** ***WDOG_ID wdCreate ()***
- **To (re)start a watchdog timer:**
  STATUS wdStart (wdId, delay, pRoutine, parameters)
  - wdId « watchdog timer » identifier,
  - delay delay count, in ticks
  - pRoutine routine to call on time-out (callback)
  - parameters parameter with which to call the routine

Example of VxWorks

- **To delete a watchdog timer:**
  STATUS wdDelete (wdId);
- **To cancel a currently watchdog timer:**
  STATUS wdCancel (wdId);

Example of VxWorks
- *Periodical Execution* -

```
wdId = wdCreate();                          // creation
wdStart (  wdId, DELAY_PERIOD,              // activation
           myWdIsr, 0);

void myWdISR(int param)                     // periodically executed
{                                           //   routine
   perform (param);
   wdStart (wdId, DELAY_PERIOD,             // re-activation
            myWdIsr, param);
}
```


Example of VxWorks
- *Signaling and Handling Timeouts* -

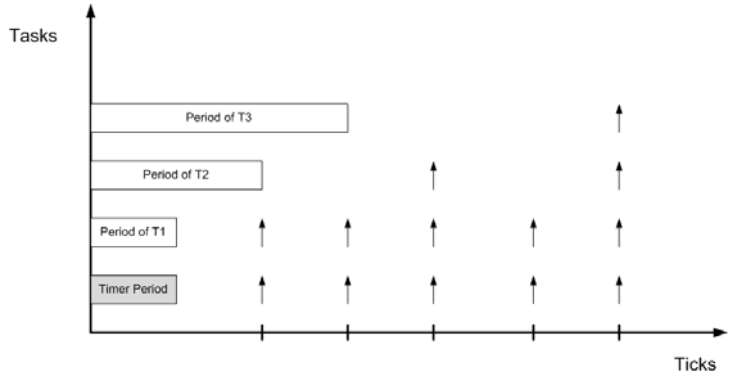
```
• WDOG_ID wdId;
  void foo(void)
  {
       wdId = wdCreate( );
       FOREVER
       {
          /* fooDoWork( ) must be executed in 10 sec */
          wdStart (wdId, DELAY_10_SEC, fooISR, 0);
          fooDoWork( );
       }
  }
  void fooISR (int param)
  { /* Traiter l'échéance dépassée */  }
```

4. Time Management (Timers)

- **Periodical functionalities can be executed at the *task level* or at the *interrupt level***
  - The precision is better at interrupt level (ISRs have higher priority than tasks)
  - The precision using task level has a reduced impact on the whole system. There are 2 differents ways to perform periodicity at the task level:
    - OSTimeDly(): this mechanism is faster, but it can induce « instability »
    - With a semaphore or a *watchdog timer* (if available)

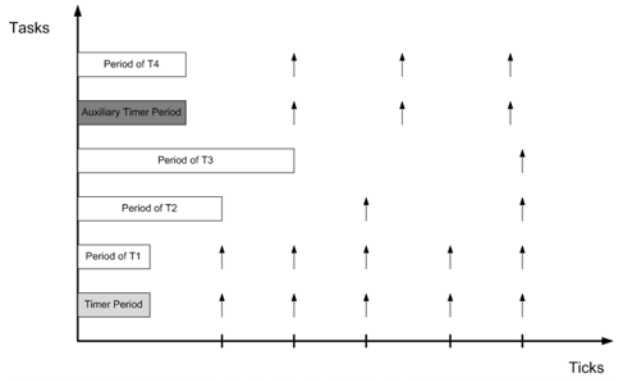
4. Time Management (Timers)

- **If the timer period is optimized, every task period (T1 to T3) fits with timer period:**



4. Time Management (Timers)

- **When a task period is not a multiple of the clock tick (task T4), it can be scheduled independantly by a second clock called auxilliary clock.**



4. Time Management (Timers)
- Auxilliary clock -

- **Here is a typical sequence of auxilliary clock use:**
  - Bind an ISR to the auxilliary clock
  - Initialiaze the frequence of the auxilliary clock
  - Start the auxilliary clock
  - Execute the code periodically
  - Stop the auxilliary clock

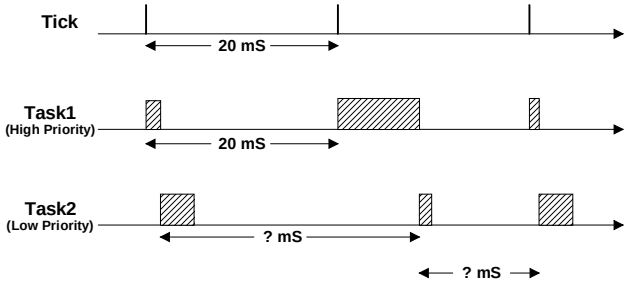
Example of VxWorks
- Auxilliary Clock -

- **Some VxWorks to handle auxilliary clock :**
  - sysAuxClkConnect( ) — Connect a routine to auxiliary clock interrupt
  - sysAuxClkRateSet( ) — Set the auxiliary clock rate
  - sysAuxClkEnable( ) — Turn on auxiliary clock interrupts
  - sysAuxClkDisable( ) — Turn off auxiliary clock interrupts

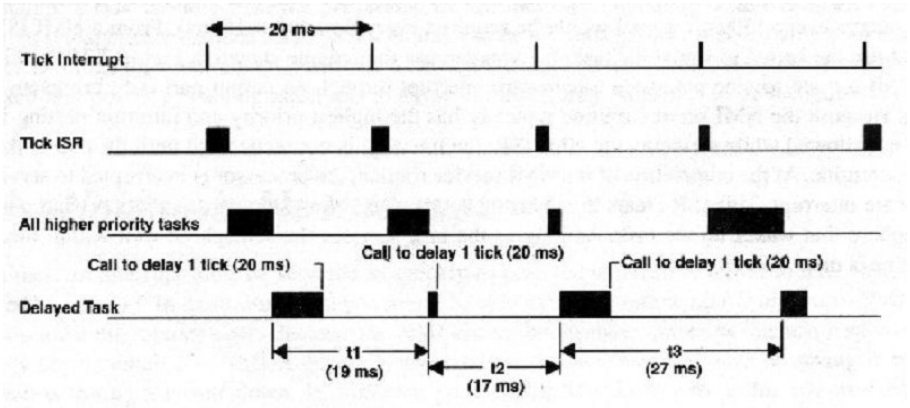
4. Time Management (Timers)
- Design issues : delay instability -

```
Task1 (void)
{
    for (;;) {
        OSTimeDly(1);
            .
            .
    }
}
```

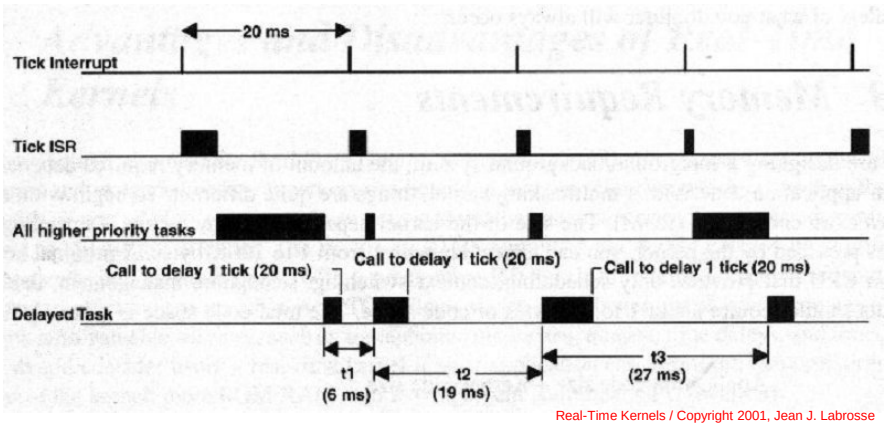
```
Task2 (void)
{
    for (;;) {
        OSTimeDly(1);
            .
            .
    }
}
```



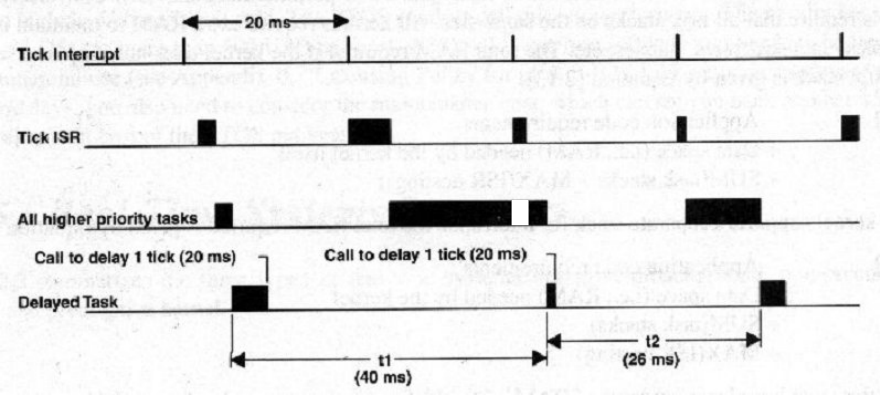
4. Time Management (Timers)
- Design issues : delay instability -



4. Time Management (Timers)
- Design issues : delay instability -



4. Time Management (Timers)
- Design issues : delay instability -



Real-Time Kernels / Copyright 2001, Jean J. Labrosse

4. Time Management (Timers)
- Design issues : delay instability -

- **As shown on the previous example, the delay instability happens when multiple waiting tasks ended their delay at the same time.**
- **To avoid this situation:**
  - **Use a faster clock**
  - **Reduce the context switch time**
  - **Reduce the interrupt latency**
  - **Change task priorities**