Lesson-10:
VxWorks functions for system time, Watchdog timer, interrupts and Task servicing
1. System Functions
System Clock Enable and Disable

- `sysClkDisable()` – to disable the system clock interrupts
- `sysClkEnable()` – to enable the system clock interrupts
Clock Rate Set and Get

• `sysClkRateSet` (TICK numTicks) sets the number of ticks per second
  • `sysClkRateSet (1000)` will set the RTC tick after every 1/1000 second (1 ms)
• `sysClkRateGet()` returns the system ticks (System RTC interrupts) per second
VxAbsTicks

- Set function should be called in the main () or start up FirstTask or as a starting function.
- vxAbsTicks – 32-bit global integer variable
- A variable, vxAbsTicks. lower that increases after each tick and vxAbsTicks.
- A variable, vxAbsTicks. upper that increases after each $2^{32}$ ticks.
TICK defines as follows.

typedef struct {
    unsigned long lower;
    unsigned long upper;
} TICK

TICK vxAbsTicks;
Connect C functions and System Clock Interrupts

- \texttt{sysClkConnect ( )} connects a 'C' function to the system clock interrupts
Auxiliary Clock

- sysAuxClkDisable () disables the system auxiliary clock interrupts and
- sysAuxClkEnable () enables the system auxiliary clock interrupts
Kernel Time Slice Setting for running tasks of equal priority

- `kernelTimeSlice (int numTicks)` controls the round robin scheduling and time slicing turns on and preemptive priority scheduling turns off
- Refer Example in Section 9.3.3.2
2. Watchdog timer Functions
Watchdog timer function

- Watchdog timer function, which once enabled for a certain timeout period, then after the timeout the system executes a connected watchdog routine with the predefined parameters for that routine.
- If the system is running without getting stuck up somewhere, then WDT is reset or restarted, else the routine will execute to handle the situation.
WDT Create and WDT Start

- WDOG_ID wdCreate ( );
  - to create a watchdog timer control block.
- wdtID = wdCreate ( );
  - to creates a watchdog timer, wdtID
- STATUS wdStart (wdtID, delayNumTicks, wdtRoutine, wdtParameter);
  - to start the created timer
WDT

- **wdtID**
  - defines the identity of the watchdog timer

- **delayNumTicks**
  - to let the timer interrupts after the **delayNumTicks** number of RTC interrupts

- **wdtRoutine**,
  - a function called on each timeout

- **STATUS wdCancel (wdtID)**
  - to cancel timer wdtID
3. Interrupt handling Functions
Interrupt Handling Functions

- An interrupt vector address, ISR_VECTADDR does not auto-generate, that has to be defined by the intVectSet( ) function.
Interrupt Handling Functions

intLock ( ) Disables Interrupts

intVectSet ( ) Set the interrupt vector

intVecGet ( ) Get Interrupt Vector

intVecBaseGet ( ) Get interrupt vector base address

intContext ( ) Returns true when an ISR is calling function.

intUnLock ( ) Enables Interrupts
Interrupt Handling Functions

intCount ( ) Counts number of interrupts nested together

intVecBaseSet ( ) Set the interrupt vector

intLevelSet ( ) Sets the interrupt mask level of the process

intConnect ( ) Connects a 'C' function to the interrupt vector
4. Task Service functions
Task Service Functions

- Each task divides into eight states [Four of these same as in μC/OS-II tasks]
Suspend and Ready

(a) Suspended (Idle state just after creation or state where execution is inhibited) using taskSuspend()

(b) Ready (waiting for running and CPU access in case scheduled by the scheduler but not waiting for a message through IPC) using taskActivate()
(c) Pending (the task is blocked as it waits for a message from the IPC or from a resource; only then will the CPU be able to process further)
Delayed and Delayed + Suspended

(d) Delayed (Sent to sleep for a certain time-interval) using taskDelay
(e) Delayed + Suspended [Delayed and then suspended if is not preempted during the delay period] using taskSuspend ()
(f) Pended + Suspended [Pended and then suspended if the blocked state does not change].
(g) Pended + Delayed [Pended and then preempts after the delayed time-interval].
(h) Pended + Suspended + Delayed.
[Pended and then suspended after the delayed time-interval].
Task Spawn

- unsigned int taskId = taskSpawn (name, priority, options, stacksize, main, arg0, arg1, ........., arg8, arg9); for task creating and activating (task spawning)
- unsigned int taskIdSelf(); returns the identity of the calling task
Task List

- unsigned int [ ] listTasks = taskIdListGet (); will return the tasked list of all existing tasks needed in array, listTasks
Task Verify and Priority Get

- taskIDVerify (taskId) verifies whether task, taskId exists
- taskPriorityGet (taskId, &priority) - the task priority can be changed dynamically
Task Options definable on spawning

- **VX_PRIVATE_ENV**
  - to execute the task in private environment (not public)

- **VX_NO_STACK_FILL**
  - no filling of stack addresses each with the hexadecimal bytes 0xEE at the task sawning
Task Options definable on spawning ...

- **VX_FP_TASK**
  - to execute the task with the floating-point processing units (for greater precession)

- **VX_UNBREAKABLE**
  - to disable the breakpoints at the task during execution. [Breakpoints are inserted for help during debugging]
Summary
We learnt

- VxWorks time-slice scheduling, system clock and service, time delay and task, functions
- Watchdog timer and interrupt handling functions
• VxWorks provides for round robin time sliced scheduling for equal priority tasks as well as preemptive scheduling

• Eight states of a task
End of Lesson-6 on VxWorks functions for system time, Watchdog timer, interrupts and Task servicing