Real Time Operating Systems

LY ETRX
Sem VIII

Arati Phadke
AY 2021-22

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

Computer Organization and Architecture

Course Outcomes

At the end of successful completion of the course the student will be able to

CO1: Understand real time operating system concepts.

CO2: Create various instances of a task and send and receive data to and from a queue

CO3: Implement the APIs within an Interrupt Service Routine

CO4: Develop algorithms for real time processes using FreeRTOS

CO5: Understand memory allocation schemes

Module 2

Task Management
2.1 Multi Tasking, Top level task states, Creating tasks, Task Priorities, Time measurement and Tick interrupt, Deleting Tasks, Scheduling algorithms: Preemptive and Non Preemptive
2.2 Queue Management, Characteristics of a queue, Using a queue using APIs to create Mailbox

Task functions in FreeRTOS

- vTaskStartScheduler();
- for(ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) { /* crude delay implementation. There is nothing to do in here. */ }
- **vTaskDelay(TickType_t xTicksToDelay);**The number of tick interrupts that the calling task will remain in the Blocked state before being transitioned back into the Ready state
- vTaskSuspend()
- vTaskResume() or xTaskResumeFromISR()
- vTaskPriorityGet()
- vTaskPrioritySet()
- The FreeRTOS scheduler will always ensure that the highest priority task that is able to run is the task selected to enter the Running state. Where more than one task of the same priority is able to run, the scheduler will transition each task into and out of the Running state, in turn.

Task Utilities in FreeRTOS

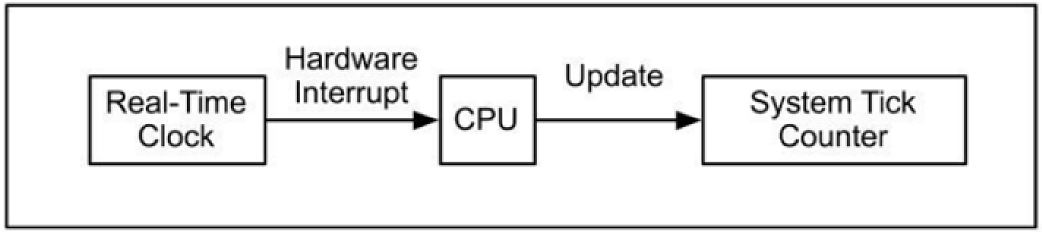
- xTaskGetCurentTaskHandle();
- xTaskGetIdleTaskhandle()
- pcTaskGetName(TaskHandle)
- xTaskGetHandle(“task2”)
- eTaskState mystate;
- mystate = eTaskGetState(mytaskHandle1);
- uxTaskGetNumberOfTasks());
- static char mytasklist[300];
- vTaskList(mytasklist);
- uxTaskGetStackHighWaterMark(mytaskHandle2)
-

Task Control BLock

- **Task Control Block :**
  - Identifies the task.
  - Shows whether the task is ready to run or is suspended.
  - Defines the priority of the task within the system.
  - Gives the identifier of the task which follows (this applies to ready queues, suspended queues, and any other queues which may be used in the system).
  - This field is used only when tasks queues are organised using linked list constructs. Queues ('lists') are formed by linking individual TCBs together using pointers
- Process descriptor (PD). Dynamic information concerning the state of the process (task) is held in the PD likeCPU register continents, Stack Pointer, program counter, general Task memory heap

Timer Tick

- **Tick is the elapsed time counter generated from system Real time clock hardware interrupt**
- It is often implemented as a time-of-day (TOD) counter, supporting four major functions:
    - Scheduling timing. Each time the tick counter generates a signal :ISR that calls a reschedule. Useful in preemptive round-robin and periodic task scheduling algorithms.
    - Polling control (related to scheduling). : event driven status check at regular time intervals : keypad example
    - Time delay generation (related to scheduling). : diverse time demands
    - Calendar time recording.



SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Priorities and system responsiveness

- **System Responsiveness depends on task Priorities**
- High priority tasks of the 'fast service queue' use interrupt signalling. reaction time ('interrupt latency') is small, typically in the range 1-100 microseconds.
- When task execution is tied in with the tick, some time variation or 'jitter' is experienced.
- Only tasks that can tolerate quite slow responses are implemented at the lowest (base) level.

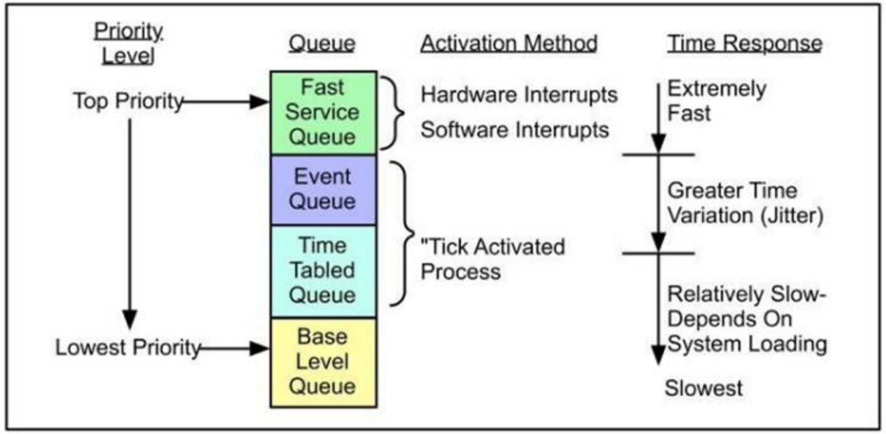


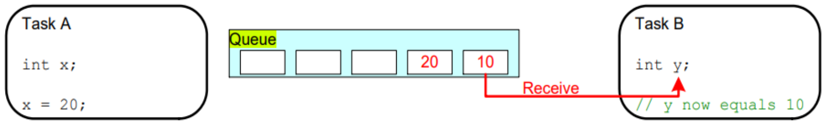
Figure 2.18 System responsiveness

Priorities and system responsiveness

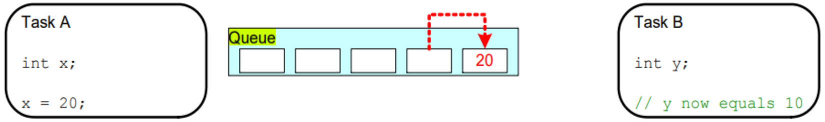
- The tick period has a significant effect on system responsiveness. The issue is interlinked with scheduling strategies.
- CPU Performance
- Preemption right or wrong?
- Issues of shorter tick period... context switch overhead..available task processing time drops
- The tick timing rate is determined both by response requirements and scheduling techniques. For fast real-time applications it usually lies in the range 1-100 milliseconds.
- A good balance between responsiveness, throughput and functional behaviour.
- mix of scheduling policies

Queue Management in RTOS

- ‘Queues’ provide a task-to-task, task-to-interrupt, and interrupt-to-task communication mechanism.
- A queue can hold a finite number of fixed size data items. The maximum number of items a queue can hold is called its ‘length’.
- Queues are normally used as First In First Out (FIFO) buffers,
- where data is written to the end (tail) of the queue and
- removed from the front (head) of the queue
- Queue by copy
- Queue by Reference
- FreeRTOS uses Queue by Co



Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).



Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Queue Management in RTOS

- **Access by Multiple Tasks** Queues are objects in their own right that can be accessed by any task or ISR that knows of their existence. In practice , common to have multiple writers, but much less common for a queue to have multiple readers.
- **Blocking on Queue Reads** When a task attempts to read from a queue, it can optionally specify a 'block' time. To wait for data to be available from the queue, should the queue already be empty.
- Blocked to ready : either data arrives in queue or time elapses
- Multiple readers, single queue to have more than one task blocked ....
- On data arrival : only one task will be unblocked : either higher priority or long awaited in case of equal priority.
- **Blocking on Queue Writes** a task can specify a block time when writing to a queue. In case of queue full, the block time is the maximum time the task should be held in the Blocked state to wait for space to become available on the queue
- Queues can have multiple writers, multiple blocked tasks.....only one task will be unblocked when space on the queue becomes available.
- On space availability : only one task will be unblocked : either higher priority or long awaited in case of equal priority.

Queue Management in FreeRTOS

- xQueueCreate
- QueueHandle_t xQueueCreate(UBaseType_t Queue_Length, UBaseType_t Item_Size);
- This function use reference by handles with a return type variable QueueHandle_t. This handler decides if enough memory is available on Arduino to create queue not.
- xQueueHandle long_queue;
- long_queue = xQueueCreate(5, sizeof(long));
- if(long_queue != NULL) { //create read and write message routines }

Argument	Description
Return Value	Handler returns NULL value, if enough memory is not available on heap to create queue otherwise it returns Non-Null value
Queue_Length	The length of queue that is the total number of elements it can hold
Item_Size	Size of each item in bytes

Queue Management in FreeRTOS

- BaseType_t xQueueSendToFront(QueueHandle_t xQueue, const void * pvItemToQueue, TickType_t xTicksToWait);
- xQueueSend(msg_queue, &lValueToSend, portMAX_DELAY);

Arguments	Details
xQueue	This provides the reference of the queue to which we want to write data. It is the same handle variable that we used while creating queue with xQueueCreate()
pvItemToQueue	A pointer variable to the message that we want to send to queue
TickType_t xTicksToWait	In case, if queue is full, it specifies the maximum blocking time of a task until space to become available. Passing value zero will force the task to return without writing, If queue is full. Similarly, passing portMAX_DELAY to function will cause the task to remain in blocking state indefinitely
Return value	It returns two values such as pdPASS (if data is successfully written) and errQUEUE_FUL(if data could not be written)

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
Somaiya TRUST

Queue Management in FreeRTOS

- BaseType_t xQueueReceive(QueueHandle_t xQueue, void * const pvBuffer, TickType_t xTicksToWait);
- const pvBuffer: It is a pointer to a variable to which we want to store received data.
- xQueueReceive(msg_queue, &lReceivedValue, portMAX_DELAY)
- **queue Read/Write Data Example**
- The queue is created to hold data items of type long. (queue size can be varied)
- The tasks that send data do not specify a block time, while the task that receives from the queue does.
- The priority of the tasks that send to the queue is lower than the priority of the task that receives from the queue. (try keeping same , and changing)
- Observe queue and task behaviors : queue values, retrieval of values , blocking / preemption of tasks etc.
- taskYIELD() informs the scheduler that a switch to another task should occur now rather than keeping the executing task in the running state until the end of the current time slice. A task that calls taskYIELD() is in effect volunteering to be removed from the Running state.