

DESIGNING RTOS ENVIRONMENT FOR TM4C123GH6PM

E3-257 EMBEDDED SYSTEMS DESIGN

Submitted by

BASATI SIVAKRISHNA (22976)

KORNU RAHUL DAS (22742)

03 MAY,2024

Course Instructor: **Prof. HARESH DAGALE RAMJI**



Indian Institute of Science, Bangalore

1. DESIGN FEATURES:

The following features has implemented in the software.

(Board: TIVA TM4C123GH6PM, IDE: CCS)

- APIs Provided:

Task related:

```
void bare_rtos_Addtask(P2FUNC TaskCode,uint8_t ID,uint8_t Priority,uint32_t StackSize, uint32_t task_time);
```

```
void bare_rtos_add_to_head(TCBLinkedList * Queue,TCB *Elem)
```

```
void bare_rtos_stack_init(uint32_t StackSize,StackPtr Stack,TCB *Task,uint8_t Flag)
```

These three APIs will be used to create a task dynamically and add the created task to ready queue. To create a task, we must have a TCB and dedicated handler with some stack memory. The TCB structure is shown below.

```
60 typedef struct TCB
61 {
62     uint8_t ID;
63     uint8_t Priority;
64     uint8_t State;
65     P2FUNC TaskCode;
66     /*-----Stack pointers-----*/
67     StackPtr Sptr;
68     /*-----Memory Pointer-----*/
69     StackPtr TopStack;
70     StackPtr EndStack;
71     /*-----Pointer to Next and Previous TCB----*/
72     struct TCB *Next_Task;
73     struct TCB *Prev_Task;
74     /*-----Pointer to Current Queue---*/
75     pVoid CurrQueue;
76     int delay_value;
77     int c_time;
78 }TCB;
79
```

In this project we are not using any priority related scheduling algorithms. Only Round Robin is considered for all scheduling related things.

- Scheduling Algorithms:

- The design system will provide 3 types of scheduling algorithms.
 - 1) Pre-emptive scheduling: In this, every task will be executed for predefined amount of time and switch to the next available task in ready list. Where ready list will be sorted according to the priorities assigned.
 - 2) Co-operative Scheduling: In this, each task will be executed for whatever time it wants and give up the processor to the next available task in ready list. This process is popularly known as task yielding in RTOS world.
 - 3) Custom Time Scheduling: This scheduling algorithm does not exist in the available RTOSs. Here every task will execute according to the time it got assigned at the initialization and after that, the processor will switch to the next available task.

A brief timing diagram of each scheduling algorithm has been given below.

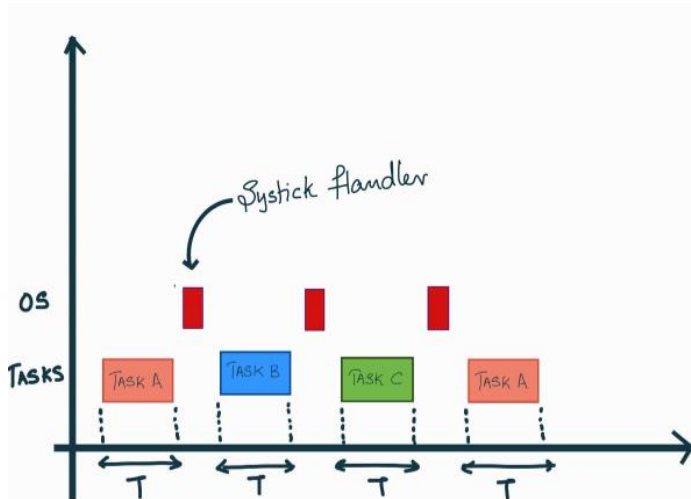


Fig 1. Pre-emptive scheduling

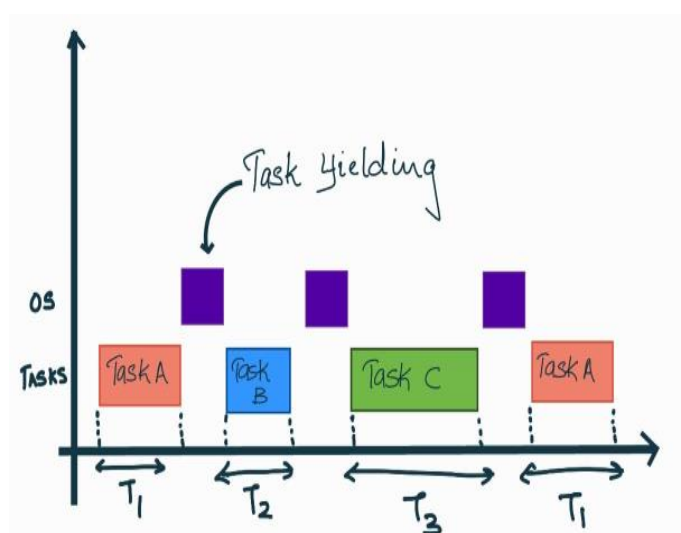


Fig 2. Co-Operative scheduling.

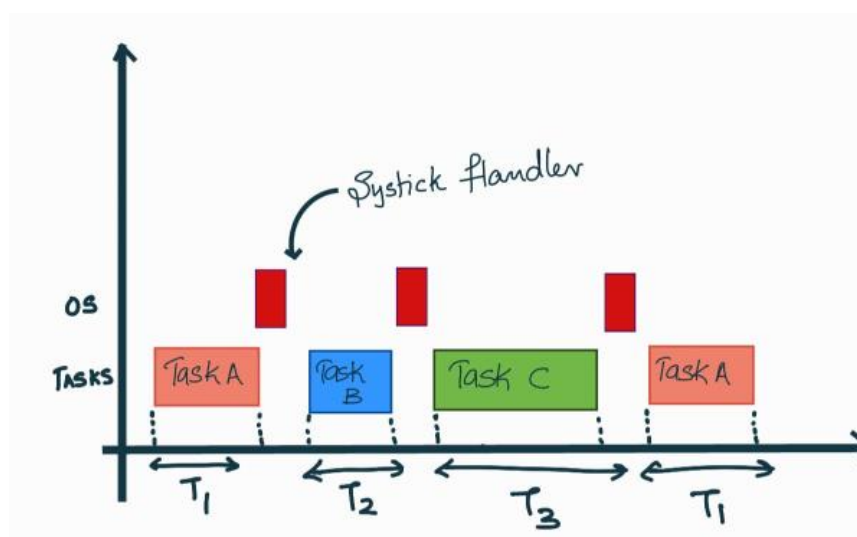


Fig 3. Custom time scheduling.

QUEUE RELATED API:

Queues are actually used to provide inter task communication among the processes by keeping a variable into a linked list structure.

Functions:

```
--  
16 struct queue* bare_rtos_queue_create(int max_size);  
17 void bare_rtos_queue_insert(int value, struct queue *root);  
18 int bare_rtos_queue_get(int index, struct queue *root);  
19 void bare_rtos_queue_insert_indx(int value, int index, struct queue *root);  
20 void bare_rtos_queue_delete(void); // to delete entire queue  
21
```

Usage:

```
#if(check_program == 2)  
  
    head_node = bare_rtos_queue_create(10);  
    head_node->element = 10;  
    bare_rtos_queue_insert(30, head_node);  
    bare_rtos_queue_insert(40, head_node);  
    bare_rtos_queue_insert(60, head_node);  
    bare_rtos_queue_insert(70, head_node);  
    bare_rtos_queue_insert(80, head_node);  
  
    //  
    int get_val = 0;  
    get_val = bare_rtos_queue_get(2, head_node);  
    bare_rtos_queue_insert_indx(23, 2, head_node);  
#endif
```

- The above API can be called to post or get data from the queues.

DEBUGGING THE DESIGNED FEATURES

- A macro named `check_program` is used to check the functionalities of the provided interface.
- By changing the check program variable to respective value, user can see output in debugging window or serial monitor provided by TIVA.

```
17
18 * check_program variable is used to check the features developed
19 * check_program = 1 -> basic task switching (with preemptive scheduling algorithm) //debug
20 * check_program = 2 -> Intra task communication (Queue) //debug
21 * check_program = 3 -> power saving delay function(ps_delay) //serial monitor
22 * check_program = 4 -> Co-operative scheduling algorithm //debug
23 * check_program = 5 -> Custom time period algorithm //debug
24 */
25
```

❖ Check_program = 1

- This will check the basic preemptive scheduling algorithm which switch between tasks (task_1, task_2, task_3)
- Each task has its own variable, and in that task handler the respective variable will be incremented and a delay 10mSec is given.
- And SysTick handler is configured to interrupt for every 1mSec.
- And the handler will pend the pendSV interrupt.
- So immediately after coming out of the SysTick handler, the pendSV handler will be executed. Where the system will do context switching(next task is decided by round robin scheduler).

Expression	Type	Value	Address
> ♦ pCurrentTask	struct TCB *	0x20001488 (ID=2 '\x02',Priority=5 '\x05',State=1...	0x200010FC
task_1_var	long long	23	0x20000E90
task_2_var	long long	23	0x20000E98
task_3_var	long long	23	0x20000EA0
queue_var_3	int	0	0x20000EA8
queue_var_2	int	0	0x20000EAC
queue_var_1	int	0	0x20000EB0
Add new expression			

❖ Check_program = 2

- This will be used to examine the Queues developed in the environment. Queue are actually developed using linked list structure.
- So each task can access queue that created in the main function. The task should be able to read, write and modify data at each index.
- In this project we are checking by accessing a variable from queue and after 2 seconds each task will write a random data into another task accessing queue element.

Expression	Type	Value	Address
> ♦ pCurrentTask	struct TCB *	0x20001878 (ID=4 '\x04',Priority=7 '\x07',State=0 ...	0x200010FC
task_1_var	long long	0	0x20000E90
task_2_var	long long	0	0x20000E98
task_3_var	long long	0	0x20000EA0
queue_var_3	int	60	0x20000EA8
queue_var_2	int	23	0x20000EAC
queue_var_1	int	30	0x20000EB0
Add new expression			

❖ Check_program = 3

- Here we are demonstrating a delay function called `ps_delay()` which will move the particular task into block state according to time mentioned in `ps_delay()` function.
- This feature has been implemented by declaring a delay variable in Task TCB.
- Initially it will be 0 and task will get executed by the processor. If we call `ps_delay (int n)` then the task's TCB delay variable will be loaded with n.
- Then for every tick the delay variable will be decremented, and the task will get executed for the next context switch.
- This output can be seen in serial monitor, where task_2 has `ps_delay` of 4 times the task_1 and task_3.

```
Terminal x
COM4 x
Initialization done
task_3
***task_2***
task_1
task_3
task_1
task_3
task_1
task_3
task_1
***task_2***
task_3
task_1
task_3
task_1
```

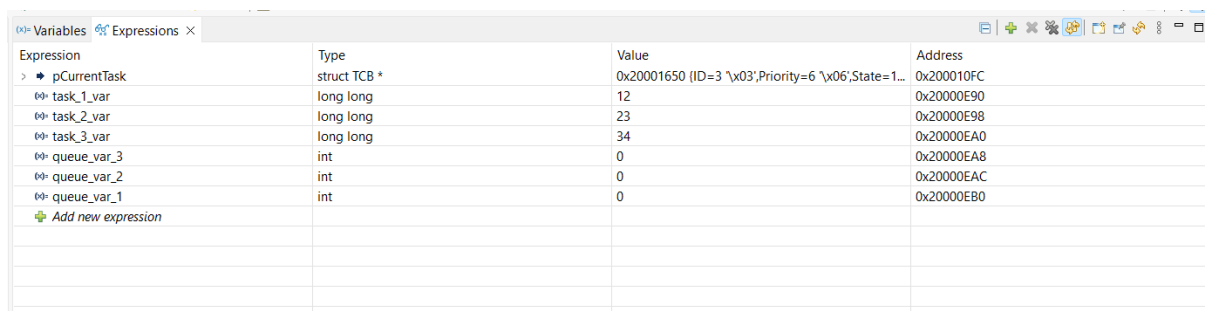
❖ Check_program = 4

- This will show the working of Co-operative scheduling algorithm. Where each task will execute till it comes to task_yield function and give up the processor to the next eligible task in queue.
- This can be done by disabling SysTick interrupts completely and pend the pendSv so the context switching will happen immediately after coming out of task_yield().

Expression	Type	Value	Address
> pCurrentTask	struct TCB *	0x20001818 (ID=4 '\x04';Priority=7 '\x07';State=0 ...	0x200010FC
task_1_var	long long	48609	0x20000E90
task_2_var	long long	48609	0x20000E98
task_3_var	long long	48610	0x20000EA0
queue_var_3	int	0	0x20000EA8
queue_var_2	int	0	0x20000EAC
queue_var_1	int	0	0x20000EB0
Add new expression			

❖ Check_program = 5

- Here we will be demonstrating a custom time scheduling algorithm. Where we will be declaring a c_time variable in each task's TCB.
- While creating a task we need to pass this parameter also.
- This parameter will make the task to execute n ticks where n is the c_time value.
- In this program we have declared task_1's c_time as 2 and task_2's c_time as 4 and task_3's as 6.
- So, task_3 variable will be incremented by 3 times the task_1 variable.



The screenshot shows a debugger's 'Variables' window. It contains a table with four columns: 'Expression', 'Type', 'Value', and 'Address'. The table lists several variables, including 'pCurrentTask' (a struct TCB pointer), and task-specific variables 'task_1_var', 'task_2_var', 'task_3_var' (all of type 'long long'), and 'queue_var_1', 'queue_var_2', 'queue_var_3' (all of type 'int'). The values for the task variables are 12, 23, and 34 respectively, while the queue variables are all 0. The addresses are in hexadecimal format, starting from 0x200010FC for pCurrentTask and 0x20000E90 for task_1_var.

Expression	Type	Value	Address
> pCurrentTask	struct TCB *	0x20001650 (ID=3 '\x03',Priority=6 '\x06',State=1...	0x200010FC
task_1_var	long long	12	0x20000E90
task_2_var	long long	23	0x20000E98
task_3_var	long long	34	0x20000EA0
queue_var_3	int	0	0x20000EA8
queue_var_2	int	0	0x20000EAC
queue_var_1	int	0	0x20000EB0
Add new expression			

Future scope and Conclusion:

- Memory management is not included in the project.
- Interrupt handling is not done while context switching.
-
- More task communication techniques can be introduced like semaphore, mutex and task notification.
- By modifying little bit code we can make this code compatible with all available arm cortex microcontrollers