Round Robin Non-Preemptive Scheduler

Lecture 17

In These Notes . . .

What is Scheduling?

What is the basis of Round Robin non-preemptive scheduling?

Examples of Round Robin

Round Robin (cooperative) scheduler

What is Scheduling?

We have seen a “reactive” system – activities are processed based on interrupts.

When scheduled activities are needed, you can set up timer interrupts, or you can have the operating system **schedule** these periodic tasks (perhaps triggered by interrupts…).

Scheduling is choosing which task to run and then running it

The rules:

- Define certain functions to be **tasks**
- If there is a task ready to run, then run it
- Finish a task before you start another one
- If there is more than one task to start, run the highest priority task first (0 is highest priority)

A Simple Example

- We will keep track of how long until the task will run ("time") and if it is scheduled now ("run")

	Priority	Length	Frequency
Task 1	2	1	20
Task 2	1	2	10
Task 3	3	1	5

Elapsed time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Task executed						**T3**					**T2**		**T3**			**T3**					**T2**		**T1**	**T3**		**T3**
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
run T1																					**1**	1	1			
run T2											**1**										**1**					
run T3						**1**					**1**	1	1			**1**					**1**	1	1	1		**1**

A More Complex Example

- Note at the end, things "stack up" (one T3 missed)

	Priority		Length		Frequency	
Task 1	2		1		20	
Task 2	1		2		10	
Task 3	3		1		5	
Task 4	0		1		3	

Elapsed time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Task executed				**T4**		**T3**	**T4**			**T4**	**T2**		**T4**	**T3**		**T4**	**T3**		**T4**		**T2**		**T4**	**T1**	**T4**	**T3**
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
time T4	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2
run T1																					**1**	1	1	1		
run T2											**1**										**1**					
run T3						**1**					**1**	1	1	1		**1**	1				**1**	1	1	1	1	**1**
run T4				**1**			**1**			**1**			**1**			**1**			**1**			**1**	1		**1**	

Over-extended Embedded System

This is an “overextended” system because some tasks are missed – several times. There is not enough processor time to complete all of the work. This is covered in more detail in a future lecture.

	Priority	Length	Frequency
Task 1	2	1	20
Task 2	1	2	10
Task 3	3	1	5
Task 4	0	**2**	3

Elapsed ti	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Task executed				**T4**		**T3**	**T4**			**T4**		**T2**		**T4**		**T4**		**T3**	**T4**		**T2**		**T4**		**T1**	**T4**		**T4**		**T3**	**T4**
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15	14	13	12	11	10
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
time T4	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3
run T1																					**1**	1	1	1	1						
run T2											**1**	1									**1**										**1**
run T3						**1**					**1**	1	1	1	1	**1**	1	1			**1**	1	1	1	1	**1**	1	1	1	1	**1**
run T4				**1**			**1**			**1**			**1**	1		**1**			**1**			**1**	1		**1**	1		**1**			**1**

Review of Scheduler Information

Scheduler provided in these slides

Details

- Scheduler uses a software timer per task
- All software timers are decremented using a timer tick based on the Timer B0 hardware overflow interrupt
- Each task runs to completion before yielding control of MCU back to Scheduler (*non-preemptive)*

Round Robin Scheduler API

Init_RR_Scheduler(void)
- Initialize tick timer B0 and task timers

Add Task(task, time period, priority)
- task: address of task (function name without parentheses)
- time period: period at which task will be run (in ticks)
- priority: lower number is higher priority. Also is task number.
- automatically enables task
- return value: 1 – loaded successfully, 0 – unable to load

Remove Task(task)
- removes task from scheduler.

Run Task(task number)
- Signals the scheduler that task should run when possible and enables it

Run RR Scheduler()
- Run the scheduler!
- Never returns
- There must be at least one task scheduled to run before calling this function.

Enable_Task(task_number) and Disable_Task(task_number)
- Set or clear enabled flag, controlling whether task can run or not

Reschedule_Task(task_number, new_period)
- Changes the period at which the task runs. Also resets timer to that value.

Set up Timer B0 in Init_RR_Scheduler

Set up B0 timer to generate an interrupt every 1 millisecond

```
// default tb0 will be = 65536 (timer tick = 5.4613 ms)
// if you load tb0 = 12000, timer tick will = 1.0000ms

init_Task_Timers();          // Initialize all tasks
tb0 = 12000;                 // 1 ms timer tick
DISABLE_INTS
tb0ic = 1;                   // Timer B0 overflow
ENABLE_INTS
tb0st = 1;                   // start timer B0
```

Task List Structure

```
#define USE_ROUND_ROBIN_SCH      1
  // Set to 1 if using Round Robin Task Scheduler
#define MAX_TASKS  5
  // Set maximum number of tasks to be used in system
   // Will affect performance.

typedef struct {
  int initialTimerValue;          // "frequency" of task
  int timer;                      // time to next "run"
  int run;                        // binary - 1 = "run now"
  int enabled;
  void (* task)(void);            // address of function
} task_t;

task_t GBL_task_list[MAX_TASKS];
int    GBL_run_scheduler=0;
```

Running the Scheduler

```
void Run_RR_Scheduler(void) {  // Always running
  int i;
  GBL_run_scheduler = 1;
  while (1) {                    // Loop forever & Check each task
    for (i=0 ; i<MAX_TASKS ; i++) {
      // If this is a scheduled task
      if (GBL_task_list[i].task != NULL) {
        if (GBL_task_list[i].enabled == 1) {
          if (GBL_task_list[i].run == 1) {
            GBL_task_list[i].run=0;  // Reset task timer
            GBL_task_list[i].task(); // Run the task
            break;
          }
        }
      }
    }
  }
}
```

	Priority	Length	Frequency
Task 1	2	1	20
Task 2	1	2	10
Task 3	3	1	5

Elapsed time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Task executed						T3					T2		T3			T3					T2		T1	T3		T3
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
run T1																					1	1	1			
run T2											1										1					
run T3						1					1	1	1			1					1	1	1	1		1

Task List Initialization

```
void init_Task_Timers(void) { // Initialize all tasks
  int i;
  for (i=0 ; i<MAX_TASKS ; i++) {
    GBL_task_list[i].initialTimerValue = 0;
    GBL_task_list[i].run = 0;
    GBL_task_list[i].timer = enabled = 0;
    GBL_task_list[i].task = NULL;
  }
}
```

	Priority	Length	Frequency
Task 1	2	1	20
Task 2	1	2	10
Task 3	3	1	5

Elapsed time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Task executed						**T3**					**T2**		**T3**			**T3**					**T2**		**T1**	**T3**		**T3**
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
run T1																					**1**	1	1			
run T2											**1**										**1**					
run T3						**1**					**1**	1	1			**1**					**1**	1	1	1		**1**

Adding a Task

```
int addTask(void (*task)(void), int time, int priority)
{
  unsigned int t_time;
  /* Check for valid priority */
  if (priority >= MAX_TASKS || priority < 0) return 0;
  /* Check to see if we are overwriting an already scheduled
   task */
  if (GBL_task_list[priority].task != NULL) return 0;
  /* Schedule the task */
   GBL_task_list[priority].task = task;
   GBL_task_list[priority].run  = 0;
   GBL_task_list[priority].timer = time;
   GBL_task_list[priority].enabled = 1;
   GBL_task_list[priority].initialTimerValue = time;
  return 1;
}
```

Task Selection

```
// Make sure to load the vector table with this ISR addr
#pragma INTERRUPT tick_timer_intr
void tick_timer_intr(void) {
static char i;
for (i=0 ; i<MAX_TASKS ; i++) { // If scheduled task
   if (GBL_task_list[i].task != NULL) {
     if (GBL_task_list[i].enabled == 1) {
       if (GBL_task_list[i].timer) {
         if (--GBL_task_list[i].timer == 0){
                GBL_task_list[i].run = 1;
                GBL_task_list[i].timer =
                      GBL_task_list[i].initialTimerValue;
         }
       }
     }
   }
}
```

	Priority	Length	Frequency
Task 1	2	1	20
Task 2	1	2	10
Task 3	3	1	5

Elapsed time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Task executed						T3					T2		T3			T3					T2		T1	T3		T3
time T1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	20	19	18	17	16	15
time T2	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
time T3	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5
run T1																					1	1	1			
run T2											1										1					
run T3						1					1	1	1			1					1	1	1	1		1

Removing a Task

```
void removeTask(void (* task)(void))
{
  int i;

  for (i=0 ; i<MAX_TASKS ; i++) {
    if (GBL_task_list[i].task == task) {
      GBL_task_list[i].task = NULL;
       GBL_task_list[i].timer = 0;
      GBL_task_list[i].initialTimerValue = 0;
       GBL_task_list[i].run = enabled = 0;
      return;
    }
  }
}
```

Enabling or Disabling a Task

```
void Enable_Task(int task_number)
{
  GBL_task_list[task_number].enabled = 1;
}

void Disable_Task(int task_number)
{
  GBL_task_list[task_number].enabled = 0;
}
```

Rescheduling a Task

Changes period of task and resets counter

```
void Reschedule_Task(int task_number, int new_timer_val)
{
  GBL_task_list[task_number].initialTimerValue =
   new_timer_val;
  GBL_task_list[task_number].timer = new_timer_val;
}
```

Start Round Robin System

To run RR scheduler, first add the function (task):

```
addTask(flash_redLED, 25, 3);
addTask(sample_ADC, 500, 4);
```

Then, the last thing you do in the main program is:

```
Run_RR_Scheduler();
```