Implementing a Time-Triggered Scheduler

KF6010 – Distributed Real-time Systems

Dr Alun Moon
alun.moon@northumbria.ac.uk

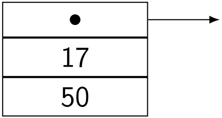
Lecture 4-1

Components of the Scheduler

- Scheduler task table
- Scheduler data declarations
- Initialising the scheduler
- Adding tasks to the table
- Starting the scheduler
- Dispatching tasks
- Sleeping when there is no work to do

Task Table design

Task Control Block




```
void ledtoggle{
    led = !led;
}
```

The Task Control Block (TCB) contains:

- A pointer to a C function
- a delay (in ticks)
- a period (in ticks)

Task Table design

The task table

The task table is an array of Task Control Blocks, indexed from 0 to `MAX-1`

tcb[0]	tcb[1]	tcb[2]	tcb[3]	tcb[4]	tcb[5]	...

Scheduler Data Declarations I

Pointer to Task:

A pointer to a void function (returns no value, takes no parameters)

TTSched/scheduler.h **line:** 11

```
typedef void (*task_t)(void);
```

Scheduler Data Declarations II

Task Control Block

A `struct`

TTSched/scheduler.h **line: 14–18**

```
typedef struct schTCB {
  task_t task;
  uint32_t delay;
  uint32_t period;
} TCB_t;
```

Task Table

The task scheduling table is an array of TCBs

TTSched/scheduler.c **line: 8**

```
static volatile uint32_t tickCount = 0;
```

Scheduler Data Declarations III

Timer Tick

We need a *tick count* that is incremented by the timer interrupt handler and decremented by the dispatcher

Should be declared as `volatile`, since it can be updated by the interrupt handler at any time.

`TTSched/scheduler.c` **line: 8**

```
static volatile uint32_t tickCount = 0;
```

Scheduler Data Declarations IV

Configuration

Configuration parameters to be configured by the application developer
The maximum number of tasks in the scheduling table:

app/ttSchedConfig.h **line: 4**

```
#define TT_SCHED_MAX_TASKS 3
```

The frequency of the timer in Hertz (Hz) *i.e.* number of times-per-second that the timer interrupts

app/ttSchedConfig.h **line: 5**

```
#define TT_SCHED_TICK_HZ    1000
```

Scheduler Function Declarations I

Initialise the task table

TTSched/scheduler.h line: 21

```
void schInit(void);          // initialise the scheduler
```

The job of the initialisation function is very simple: for every TCB in the scheduling table, set each field in the TCB to 0

Initial Task Table

	1	2	3	4	5	6	7	8
task	0	0	0	0	0	0	0	0
delay	0	0	0	0	0	0	0	0
period	0	0	0	0	0	0	0	0

Scheduler Function Declarations II

Add Tasks to Task table

TTSched/scheduler.h line: 22–25

```
void schAddTask(                // add a task to the task set
  task_t,                       // the task to add
  uint32_t,                     // the delay in ms
  uint32_t);                    // the period
```

Example: `schAdTask(ledToggle, 0, 50);`

The job of the `schAddTask` function is to find the first empty slot in the task table array, and set the fields of the TCB there with the parameters passed to the function.

An empty entry in the task table can be identified easily because it's `task` field is a `NULL` pointer (0)

Scheduler Function Declarations III

Scheduler Start

Configure and start a timer to interrupt periodically

TTSched/scheduler.h **line: 26**

```
void schStart(void);          // start ticking
```

- The ARM Cortex-M4 has s simple timer called the `SysTick` timer, that has been introduced to simplify the implementation of a timing source for a scheduler or operating-system.
- There is a CMSIS function `SysTick_Config` that can be used to configure the timer to raise an interrupt periodically after a count
- The CMSIS constant `SystemCoreClock` gives the frequency of the system-core-clock, this can be used with the user defined value for `TT_SCHED_TICK_HZ` to get the value to use with `SysTick_Config`

Scheduler Function Declarations IV

Sys-Tick Interrupt Handler

You also need to declare a handler for the SysTick timer interrupt

TTSched/scheduler.h **line: 20**

```
void SysTick_Handler(void); // install own handler for SysTick
```

The SysTick handler just needs to increment tickCount

Scheduler Dispatch

TTSched/scheduler.h **line: 27**

```
void schDispatch(void);      // run the next task
```

The dispatch function interacts with the SysTick_Handler via the tickCount to decide which tasks to activate.

Scheduler Function Declarations V

Scheduler Sleep

TTSched/scheduler.h **line: 28**

```
void schSleep(void);          // go to sleep to save power
```

It is a good idea to allow the microcontroller to go into a low-power state (sleep) when there is no work for it to do.
Once the tasks have been dispatched the dispatch function can call `schSleep();`.

Wait For Interrupt

The ARM Cortex-M4 has a CMSIS `__WFI()` instruction that is a hint to the microcontroller that it can go to sleep and *Wait for an interrupt* to occur.

Dispatch Function I

`void schDispatch(void);`

The dispatch function interacts with the `SysTick_Handler` via the `tickCount` to decide which tasks to activate.

Protecting data shared with the Interrupt Handler

- the dispatch function shares the `tickCount` variable with the timer interrupt handler
- protect this variable by disabling interrupts while it is used in the dispatch function
- Use `__disable_irq()` and `__enable_irq()` to disable and enable interrupts
- interrupts should be disabled for the shortest possible time to ensure that the system remains responsive

Dispatch Function II

```
void schDispatch(void);
```

The dispatcher acts if the timer has ‘ticked’, *i.e.* `tickCount` is greater than zero.

Update the task table

If `tickCount > 0` then the task table much be updated by decrementing the `delay` field in every non-empty TCB

Dispatch Function III

`void schDispatch(void);`

Running tasks that are ready to run and reinitialising the `delay`

1. decrement the `delay` field for each task
2. If the `delay` field in any TCB has reached 0, the function for the task must be called
   - recall the syntax for calling functions from pointers

     ```
     (*(schTasks[i].task))();
     ```
3. After the task has run (function returns); reinitialise the `delay` field using the `period` value

Update `tickCount`

Now `tickCount` should be decremented to show the tick has been serviced.

Time-triggered scheduler: use in practice

```c
int main () {
    schInit();   /* initialise task table */

    schAddTask(a, 0, 500); /* tick every 500 ticks from now */
    schAddTask(b, 50, 500);/* tick every 500 ticks after 50 ti

    schStart(); /* start timer */

    while (true) {
        schDispatch(); /* run dispatcher ... */
    }
}
```