TT Scheduler

Context

You already have at least a design or prototype for your system based on some form of ET/P architecture.

You and / or your development team are using pattern EVENTS TO TIME

You are in the process of creating or upgrading an embedded system, based on a single processor.

Because predictable and highly-reliable system operation is a key design requirement, you have opted to employ a “time-triggered” system architecture in your system.

Problem

How will you decide which form of time-triggered scheduler should you use for your application?

Background

TT schedulers that we can use can take two forms: Co-operative and Pre-emptive. Both of these types of schedulers provide various options, some of these options are given below:

1. Co-operative Schedulers
   a. Super loop
   b. TTC Dispatch

2. Pre-emptive Schedulers
   a. Full Pre-emption
      i. TTRM Scheduler
   b. Limited Pre-emption
      i. TTH Scheduler

In co-operative scheduling, tasks co-operate with each other and wait for their turn to execute until the currently running task finishes execution.

In pre-emptive scheduling a task of higher priority which is ready to execute can pre-empt a currently running task of lower priority.

Overview of TT Schedulers

TTC-SL Scheduler

The simplest way of implementing a TTC scheduler is by means of a “Super Loop” or “Endless loop” (e.g. Pont, 2001; Kurian and Pont, 2007). A possible implementation of such a scheduler is illustrated in Listing 1.
int main(void)
{
...
while(1)
{
    TaskA();
    Delay_6ms();
    TaskB();
    Delay_6ms();
    TaskC();
    Delay_6ms();
}
    // Should never reach here
    return 1
}

Listing 1: Illustrating a TTC Super Loop Scheduler

Applications based on a TTC-SL SCHEDULER have extremely small resource requirements. Systems based on such a pattern (if used appropriately) can be both reliable and safe, because the overall architecture is extremely simple and easy to understand, and no aspect of the underlying hardware is hidden from the original developer, or from the person who subsequently has to maintain the system.

TTC Dispatch Scheduler

The TTC scheduler implementation referred to here as a “TTC-Dispatch” scheduler provides a more flexible alternative see Listing 2.

The type of TTC scheduler implementation discussed in this pattern is usually implemented using a hardware timer, which is set to generate interrupts on a periodic basis (with “tick intervals” of around 1 ms being typical). In most cases, the tasks will be executed from a “dispatcher” (function), invoked after every scheduler tick. The dispatcher examines each task in its list and executes (in priority order) any tasks which are due to run in this tick interval (see Figure 1). The scheduler then places the processor into an “idle” (power saving) mode, where it will remain until the next tick.

![Figure 1: Illustrating TTC design](image-url)
Provided that an appropriate implementation is used, a time-triggered, co-operative (TTC) architecture is a good match for a wide range of low-cost, resource-constrained applications. TTC architectures also demonstrate very low levels of task jitter (Locke, 1992) and can maintain their low-jitter characteristics even when techniques such as dynamic voltage scaling (DVS) are employed to reduce system power consumption.

```c
void main(void)
{
  // Set up the scheduler
  SCH_Init_T2();

  // Init tasks
  TaskA_Init();
  TaskB_Init();

  // Add tasks (10 ms ticks)
  // Parameters are filename, offset (ticks), period (ticks)
  SCH_Add_Task(TaskA, 0, 3);
  SCH_Add_Task(TaskB, 1, 3);
  SCH_Add_Task(TaskC, 2, 3);

  // Start the scheduler
  SCH_Start();

  while(1)
  {
    SCH_Dispatch_Tasks();
    SCH_Go_To_Sleep();
  }
}
```

**Listing 2: TTC Implementation**

**TTRM architectures**

Where a TTC architecture is not found to be suitable for use in a particular resource constrained embedded systems, fixed-priority scheduling has been proposed as the most attractive alternative (Audsley, Burns et al., 1991; Bate, 1998).

“Time-triggered rate monotonic” (TTRM) is a well-known fixed-priority scheduling algorithm that was introduced by (Liu and Layland, 1973) in 1973. Technically, TTRM is a pre-emptive scheduling algorithm which is based on a fixed priority assignment (Kopetz, 1997). In particular, the priorities are assigned to periodic tasks accord to their occurrence rate or, in other words, priorities are inversely proportional to their period, and they do not change through out of the operation (because their periods are constant).
To illustrate the use of TTRM scheduling, Figure 2 above shows how a set of periodic tasks can be scheduled by this algorithm. Task T1 is executed periodically at the fastest rate, every 10 ms, and is determined to be the highest priority in this scheduling policy, while task T2 and T3, which are run every 20 and 40 ms respectively, have lower priority levels according to their rates. A task scheduled by the TTRM algorithm can be pre-empted by a higher priority task. As illustrated in Figure 5, task T3 - which is running - is pre-empted by task T1 is at time 10: it carries on after the completion of task T1.

**TTH architectures**

Where a TTC architecture is not found to be suitable for a particular system, use of a TTRM design may not be necessary. For example, a single, time-triggered, pre-empting task can be added to a TTC architecture, to give what we have called a “time-triggered hybrid” (TTH) scheduler (Pont, 2001; Maaita and Pont 2005) and others have called a “multi-rate executive with interrupts” (Kalinsky, 2001)

Use of a TTH SCHEDULER allows the system designer to create a static schedule made up of (i) a collection of tasks which operate co-operatively and (ii) a single – short - pre-empting task (see Figure 3). In many of the systems employing a TTH architecture, the pre-empting task will be used for periodic data acquisition, typically through an analogue-to-digital converter or similar device.

Such requirements are common in, for example, control systems (Buttazzo, 2005) and applications which involve data sampling and Fast-Fourier transforms (FFTs) or similar techniques:
Solution

Here are the guidelines about choosing appropriate TT architecture.

**When to use TTC**

Use TTC architecture where ever possible as first choice because of its simple and efficient design. Of course, this architecture is not always appropriate. The main problem is that long tasks will have an impact on the responsiveness of the system. This concern is succinctly summarised by Allworth: “[The] main drawback with this [co-operative] approach is that while the current process is running, the system is not responsive to changes in the environment. Therefore, system processes must be extremely brief if the real-time response [of the] system is not to be impaired.” (Allworth, 1981).

We can express this concern slightly more formally by noting that if the system must execute one of more tasks of duration X and also respond within an interval T to external events (where T < X), a pure co-operative scheduler will not generally be suitable. In more simple words duration of a task (execution time) must be less than the tick interval of the system.

In practice, it is sometimes assumed that a TTC architecture is inappropriate because some simple design options have been overlooked, see pattern BUFFERED OUTPUT

**When to use TTH**

For systems where TTC is not an appropriate choice, avoid jumping to fully pre-emptive architectures as they incur higher overheads because of context switching involved during task pre-emption. Check for TTH solution which provides a limited level of pre-emption. For example, consider a wireless electrocardiogram (ECG) system. An ECG is an electrical recording of the heart that is used for investigating heart disease. In a hospital environment, ECGs normally have 12 leads (standard leads, augmented limb leads and precordial leads) and can plot 250 sample-points per second (at minimum). In the portable ECG system considered here, three standard leads (Lead I, Lead II, and Lead III) were recorded at 500 Hz. The electrical signal were sampled using a (12-bit) ADC and – after compression – the data...
were passed to a “Bluetooth” module for transmission to a notebook PC, for analysis by a clinician see (Phatrapornnant and Pont, 2006).

In one version of this system, we are required to perform the following tasks:
- Sample the data continuously at a rate of 500 Hz. Sampling takes less than 0.1 ms.
- When we have 10 samples (that is, every 20 ms), compress and transmit the data, a process which takes a total of 6.7 ms.

In this case, we will assume that the compression task cannot be neatly decomposed into a sequence of shorter tasks, and we therefore cannot employ a pure TTC architecture. However, even if you cannot – cleanly - solve the long task / short response time problem, then you can maintain the core co-operative scheduler, and add only the limited degree of pre-emption that is required to meet the needs of your application.

For example, in the case of our ECG system, we can use time-triggered hybrid architecture.

In this case, we allow a single pre-empting task to operate: in our ECG system, this task will be used for data acquisition. This is a time-triggered task, and such tasks will generally be implemented as a function call from the timer ISR which is used to drive the core TTC scheduler. As we have discussed in detail elsewhere (Pont, 2001: Chapter 17) this architecture is extremely easy to implement, and can operate with very high reliability. As such it is one of a number of architectures, based on a TTC scheduler, which are cooperatively based, but also provide a controlled degree of pre-emption.

**When to use TTRM**

If both TTC and TTH architectures are not appropriate for your application and full pre-emption is a necessary, then TTRM architecture may match your requirements.

Overall, it has been claimed that the main advantage of TTRM scheduling is flexibility during design or maintenance phases, and that such flexibility can reduce the total life cost of the system (Locke, 1992; Bate, 1998). The schedulability of the system can be determined based on the total CPU utilisation of the task set: as a result - when new functionalities are added to the system – it is only necessary to recalculate the new utilisation values. In addition, unlike a TTC design, there is no need to break up long individual tasks in order to meet the length limitations of the minor cycle. The need to employ harmonic frequency relationships among periodic tasks is also avoided. Finally, the scheduling behaviour can be predicted and analysed using a task model proposed by Liu and Layland (1973).

However, the scheduling overheads of TTRM schedulers tend to be larger than those of TTC schedulers because of the additional complexity associated with the context switches when
saving and restoring task state (Locke, 1992). This is a concern in embedded systems with limited resources.

**Locking mechanisms**

If you use any architecture which involves pre-emption (TTH or TTRM), you need to consider ways of preventing more than one task from accessing critical resources at the same time. See patterns CRITICAL SECTION and related patterns for more details at link [http://hdl.handle.net/2381/9679](http://hdl.handle.net/2381/9679).

**Overall strengths and weaknesses**

😊 Use of a TT scheduler tends to result in a system with highly predictable patterns of behaviour.

😢 Inappropriate system design using this approach can result in applications which have a comparatively slow response to external events.

**References**


Gergeleit, M. and E. Nett (2002). Scheduling Transient Overload with the TAFT scheduler. Fall meeting of GI/ITG specialized group of operating systems.


