SAFECheckpoints

Verifying Task Scheduling Performance

INTRODUCTION

Within a priority based pre-emptive scheduling scheme, the highest priority Task that is able to run will be the Task selected to run. Hence the priority assigned to each Task needs to be carefully chosen, as it directly effects when the Task will be given processing time. Task scheduling may also be effected by Interrupts, as Interrupts run at a higher priority level than Tasks. Overrunning or rapidly repeating Interrupts and/or higher level Tasks can cause other, lower-priority Tasks to be delayed or blocked in an irregular way. This is called temporal disruption.

AN EXAMPLE OF TEMPORAL DISRUPTION

Sometimes Tasks operating in the background with a low priority still have a time requirement, for example, a CRC check of the entire code may have to be completed every hour. In Figure 1 the CRC Task is implemented in the Low Priority Task, which is pre-empted by a regular Higher Priority Task. Also within this system is an Interrupt that triggers a medium priority Task. If the frequency of this Interrupt is too high it causes irregularity in both the start and duration of a low priority periodic Task implementing the CRC checking. It becomes possible that the low-priority periodic processing may not have completed when the next pass is due to commence.

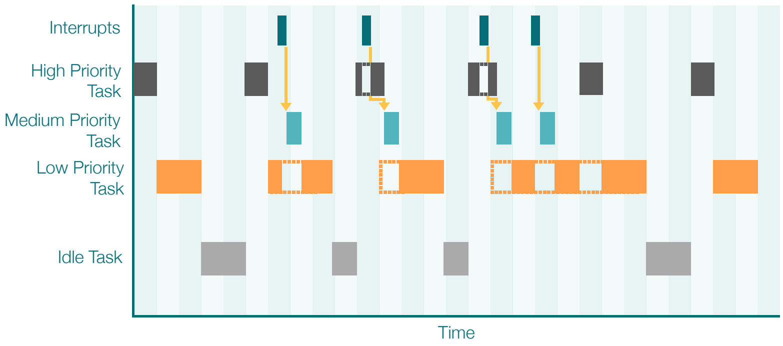


Figure 1. *An Example of Temporal Disruption*

USING SAFECheckpoints TO MONITOR TASK SCHEDULING PERFORMANCE

The SAFE**Checkpoints** module uses a Software Timer to monitor the progress of Tasks, detect scheduling issues and help to improve temporal separation. Note that SAFE**Checkpoints** does not enforce temporal separation; it merely offers a means to detect when breaching of temporal requirements occurs.

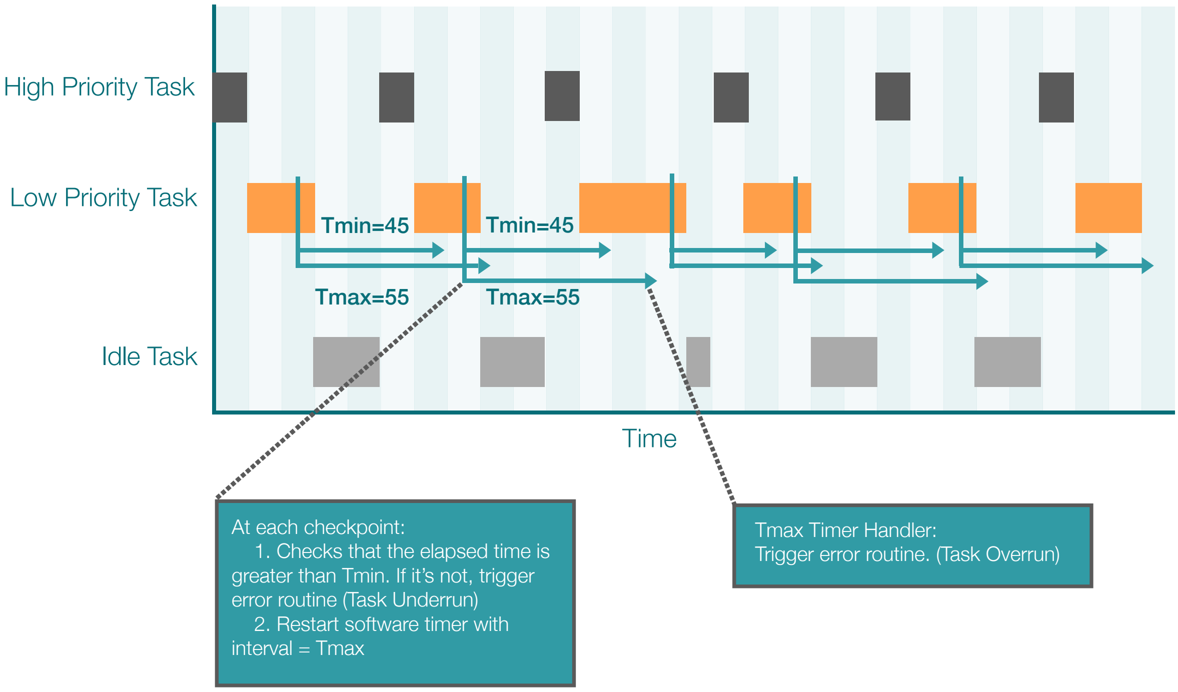
Figure 2 shows a simple system with two periodic tasks running. At a given point in a Tasks' operation, the checkpoint occurs where the temporal performance is measured. The elapsed time is calculated and used to determine whether a scheduler underrun has occurred. The checkpoint software timer used to detect scheduling overrun is reset (restarted). If at any time the checkpoint software timer expires then the task is in breach of its scheduling policy and the error reported. Depending on the nature of the time profiling, it may be more appropriate to implement the time checking at the beginning or end of the tasks run.

The same mechanism can also be used to measure whole safety functions, where a number of events occur and a number of Tasks interact to produce the single final output.

AVOIDING TIMING PROBLEMS

There are many established ways of handling the effect of scheduling overrun. Methods of reducing the effects of Temporal Disruption include:

- Minimising the processing that occurs within Interrupt handlers;
- Analysing the worst-case Interrupt processing within periodic frames and if necessary taking steps to limit the maximum Interrupt processing;
- Analysing the worst-case processing within Tasks and using this to determine average and peak CPU load;
- Wherever possible, making effective use of priority-based scheduling so that essential Tasks are performed in as deterministic manner as possible;
- Correct selection of processor.



At each checkpoint:
1. Checks that the elapsed time is greater than Tmin. If it's not, trigger error routine (Task Underrun)
2. Restart software timer with interval = Tmax

Tmax Timer Handler:
Trigger error routine. (Task Overrun)

Figure 2. *Software Timer Checks*

SAFE**Checkpoints** FOR AUTOMOTIVE

There is an expectation within ISO 26262 that runtime verification monitors will be used to detect, indicate and handle systematic faults within software rated ASIL C and D. SAFE**Checkpoints** has been designed to meet this Automotive ISO 26262 ASILC/D requirement, as it provides a sophisticated Task monitoring capability, ensuring the scheduling of Tasks is occurring as intended.

Both SAFE**Checkpoints** and SAFE**RTOS** are supplied with a Design Assurance Pack supporting certification to ISO 26262 ASIL D and IEC 1508 SIL 3.

Visit: www.highintegritysystems.com/white-papers for more information on this topic and the SAFE**Checkpoints** component in more detail.

USING SAFE**Checkpoints**

Using SAFE**Checkpoints** allows the RTOS to verify:

- Periodic Tasks run within tolerances;
- Sections of processing within Tasks complete;
- Processing of Interrupt events by handler Tasks completes within allowable tolerances;
- Complex functionality involving multiple Tasks completes within allowable tolerances.

In order to use SAFE**Checkpoints** effectively, the timing requirements of the system must first be specified. For periodic Tasks:

- What frequency must the code run at?
- What is the allowable tolerance?
- Is timing relative to the last execution or is it fixed frame?

For event driven Tasks:

- Is there a frequency or limit on event generation?
- What is the maximum response time to an event?

WITTENSTEIN high **integrity** systems

Worldwide Sales and Support
Americas: +1 408 625 4712
ROTW: +44 1275 395 600
Email: sales@high**integrity**systems.com
Web: www.high**integrity**systems.com

WITTENSTEIN