SCHEDULING AND ADMISSION TESTING FOR JITTER CONSTRAINED PERIODIC THREADS

Andreas Mauthe and Geoff Coulson

Computing Department,
Lancaster University,
Lancaster LA1 4YR, UK
e.mail: [andreas, geoff]@comp.lancs.ac.uk

1 Introduction

In this short paper, we address the issue of real-time CPU scheduling in operating systems. Our approach is to design new admission tests for periodic real-time threads which guarantee that a run time scheduler, say an earliest deadline first (EDF) or a fixed priority scheduler, will be able to honour all specified deadlines *where the specified deadlines may be earlier than the end of the current period*. We refer to threads whose deadlines and periods may differ as *jitter constrained threads* because they exhibit less jitter (i.e. variation in periodicity) than conventional periodic threads. In the extreme case, where the deadline is specified to be identical to the execution time, a jitter constrained thread executes perfectly isochronously. Jitter constrained threads are particularly appropriate for high quality continuous media applications such as video and animation playouts as the display jitter of high quality video or animation can be reduced to an arbitrary degree without relying on a hardware clocked display device.

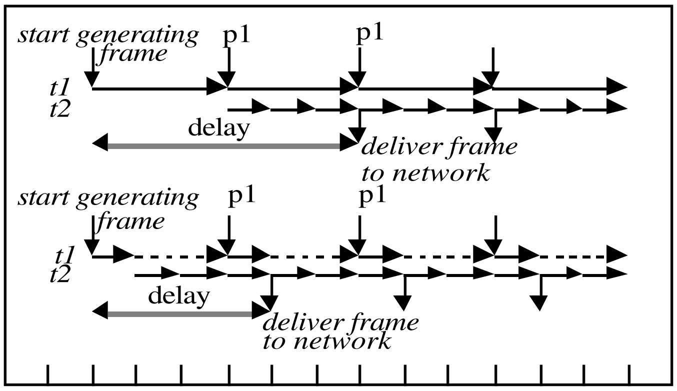

Fig. 1. Reducing delay with jitter constrained threads

The provision of jitter constrained threads also helps end-systems to honour delay and jitter-related QoS parameters. To see how delay and buffering requirements are affected, consider an application that sends large video frames over a network using a rate based transport protocol. This scenario may be realised as a periodic thread t_1 which executes application code to generate video frames, and another periodic thread t_2 which performs fragmentation and rate control. With conventional real-time periodic threads, the delay incurred by this two thread system would be the sum of the period of t_1 plus three times the period of t_2 (assuming three packets per frame as illustrated in figure 1). However, if we are able to specify an earlier deadline for t_1, the delay (and hence buffering requirement) can be significantly reduced. The top part of figure 1 shows the timing of the scenario in the normal case. In the bottom part of figure 1, t_1 is a jitter constrained thread which completes the generation of a frame within one time unit of the start of a frame period. It can be seen that significant delay reductions are achieved.

The scheduling work introduced in this paper is part of a broader resource management framework carried out within SUMO, a collaborative project involving Lancaster University and CNET, France Telecom [1]. The aim of this project is the design and implementation of a Chorus microkernel based operating system infrastructure for supporting distributed multimedia applications.

2 Admission Testing of Jitter Constrained Threads

Our scheduling approach is principally based on the dynamic *earliest deadline first* (EDF) algorithm [2]. The algorithm can be simply expressed as follows:

> *Every time a thread becomes runnable, or a periodic thread invocation finishes, select the runnable thread with the earliest deadline to run next.*

EDF is an attractive policy as it is an optimal, dynamic algorithm, i.e. it is *guaranteed* to find a valid schedule (i.e. one that honours the deadlines of all threads) *if such a schedule exists* [3]. In addition, it fully utilises the CPU resource. For periodic threads with deadlines equal to periods a simple admission test exists: it only has to be checked if the processor utilisation of all threads is less or equal than 100%. More formally, the admission test is expressed as follows [2]:

$$\sum_{i=1}^{\#t} e_i / p_i \leq 1 \qquad \begin{array}{l} e_i = \textit{execution time} \\ p_i = \textit{period of thread } i \end{array}$$

2.1 Extension Utilising Deadlines and Periods

A straightforward extension of the above admission test is to consider explicit deadlines smaller than the period, i.e. to replace p_i with d_i in the above formula. The problem with this test is that it will often refuse to admit thread sets for which EDF could find a valid schedule. Consider the following thread set:

```
t1 : p = 3, d = 1, e = 1
t2 : p = 5, d = 4, e = 2
```

This set will not pass the above admission test even though a valid schedule can easily be found that will honour the deadlines of both threads: in figure 2, it can be seen that every time the scheduling time of t_1 is reached, it is scheduled immediately because either it has a closer deadline than t_2, or t_2 has already finished processing when t_1 becomes ready.

We can use the above example to extract the following general insight:

> *A candidate thread can be accepted if, in the period length of each of its invocations, there is sufficient spare resource to schedule the required number of invocations of all other threads with an earlier or equal deadline, even when the scheduling times of all threads concerned coincide.*

Relating this to figure 2, we can see that, in a system currently supporting only t_1, it should also be possible to support t_2. This is because in each instance of t_2 it is possible to fit the required number of invocations of t_1 (i.e. 2 invocations of 1 time unit each) plus the per-period invocation execution time of t_2 itself (i.e. 2 time units) within the deadline of t_2 (i.e. 4 time units).

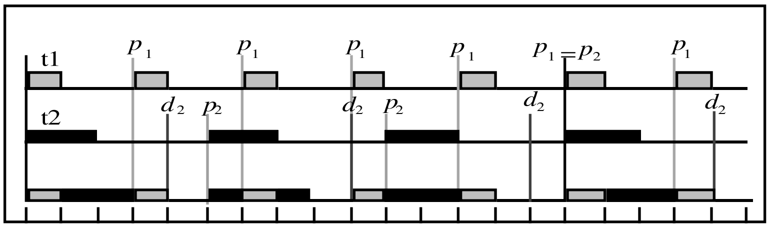

Fig. 2: Deadline scheduling of jitter constrained threads

This argument is more formally captured in the following admission test:

$$\forall c \;\; in \; \left\langle c_{\min}, \; \ldots, \; c_{\max} \right\rangle \; \left(\sum_{j=1}^{k} etime(j, \; d_c) + \sum_{i=1}^{l} e_i \right) \Big/ d_c \leq 1$$

$$where \;\; etime(j, \; t) = e_j \left\lfloor t/p_j \right\rfloor + \begin{cases} e_j, \;\; if \;\; t \geq \mathrm{p_j} \left\lfloor t/\mathrm{p}_j \right\rfloor + d_j \\ \dfrac{e_j}{d_j}(t - \mathrm{p_j} \left\lfloor t/\mathrm{p}_j \right\rfloor), \;\; otherwise \end{cases}$$

In the above, the inequality is applied iteratively over an ordered sequence of *deadline equivalence classes* and must hold for all these classes. Each deadline equivalence class c contains all threads in the system that have a deadline d_c, and the sequence of classes as a whole is ordered smallest deadline first to largest deadline last.

The function *etime*(j, t) gives the maximum processing time required by a thread j in some time span t. Given this definition of *etime*, it can be seen that the first term in the numerator of the top equation sums the required execution times of some number of threads within the period d_c. In fact, the summation ranges over $(1..k)$ which is defined to contain all threads in deadline classes *lower* in the ordered sequence than the current class c. The second term, ranging over $(1..l)$, sums the execution time required by all threads in the current deadline class c within the interval d_c. The numerator as a whole thus evaluates to the total CPU time required, in the worst case, in the time interval d_c by all threads with a deadline $\leq d_c$.

2.2 Extension Utilising Predictable Scheduling Times

A limitation of the above test is that it pessimistically assumes that all threads can become ready at the same time. Where it can be demonstrated that this cannot occur, a further degree of freedom in admission testing is possible. For example, due to the implicit timing relationship between threads t_1 and t_2 in figure 1, it is only possible for their invocations to coincide at certain fixed instants. This is so because the scheduling times of invocations of t_2 are always located at a fixed offset from the deadlines of invocations of t_2.

One situation where it is possible to assume fixed timing relationships is in the context of so called *harmonic sets*. A harmonic set [4] is a thread set in which the constituent threads can be ordered such that the period of each thread divides its successor; e.g. 3 divides 6 divides 12. Harmonic sets can, in principle, be recognised *automatically* by the infrastructure without requiring the user to explicitly specify scheduling time relationships. The special case of harmonic sets containing multiple threads with the same period is likely to be quite common as applications frequently open multiple instances of a given connection time (e.g. multiple video connections). In the revised admission test (shown below) we recognise harmonic sets with pre-defined timing relationships as described above; the test is a relatively straightforward extension of our first test:

$$\forall c \ \ in \ \left\langle c_{\min},\ \ldots,\ c_{\max} \right\rangle \left(\sum_{j=1}^{k} etime(j,\ d_c) + \sum_{i=1}^{l} e_i + \sum_{H=1}^{\#H} E_H \right) \Big/ d_c \le 1$$

The differences are, first, that the original two terms now only consider those threads that are *not* placed in a harmonic set, and, second, that a new term is added to express the maximum required processing time of any harmonic sets in the thread set. The expression E_H represents the maximum processing time requirement over any interval d_c for the subset of threads in a harmonic set H whose deadlines are $\le d_c$.

Due to lack of space we are not able to present the derivation and the full formula for E_H. Basically, we use an iterative function over scheduling times and deadlines to determine the required maximum processing time.

3 Summary and Conclusions

In conclusion, we have briefly presented admission tests that ensure that a run time EDF algorithm will be able to maintain the required semantics of jitter constrained threads. The second test involves more admission time overhead, but is able to admit a wider range of candidate thread sets. In the full version of this paper [5], the tests are derived fully and proofs are given. In addition, the (minimal) necessary adaptations to the run time scheduler are discussed. We also generalise our tests to accommodate fixed priority scheduling schemes.

References

1 Coulson, G., Campbell, A., Robin, P., Blair, G. S., Papathomas, M. and D. Shepherd, “The Design of a QoS Controlled ATM Based Communications System in Chorus”, *To be published in IEEE JSAC, Special Issue on ATM Local Area Networks*, 1994.

2 Liu, C. L. and Layland, J. W., “Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment”, *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, pp 46-61, February 1973.

3 Mauthe, A., Schultz, W. and Steinmetz, R., “Inside the Heidelberg Multimedia Operating System Support: Real-Time Processing of Continuous Media in OS/2, Technical Report No. 43.9214, IBM European Networking Center, Tiergartenstrasse 8, D-6900 Heidelberg 1, 1992.

4 Kuo, T. W. and Mokthe, A. K., Load Adjustment in Adaptive Real-Time Systems", *Proc. 12th IEEE Real Time System Symposium*, 1991.

5 Mauthe, A and G. Coulson, “Scheduling and Admission Testing for Jitter Constrained Periodic Threads”, Internal Report MPG-95-04, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, January 1995.