

SCHEDULING ALGORITHMS FOR MULTIMEDIA APPLICATIONS

ANUJA HARIHARAN¹, BALAJI RANAN² AND SAMARJIT CHAKRABORTHY³

School of Computing, National University of Singapore
3 Science Drive 2, Singapore 117543

ABSTRACT

The aim of this project is to study scheduling algorithms used for Multimedia applications. The test bed used is S.Ha.R.K (Soft Hard Real-Time Kernel). Existing algorithms address specific issues, but there is no universal algorithm, which is without flaws, and is applicable in all kinds of devices. This is because they do not exploit the characteristics of the multimedia applications (which constitute significant workload) and therefore do not effectively use the available resources such as processor capacity, on-chip memory etc. Our research project thus aims to find one such algorithm, wherein our approach is guided by a mathematical framework that models execution of a streaming application on a given architecture. We use an operating system test-bed, to experiment with various scheduling policies.

1 INTRODUCTION

Multimedia applications have real-time constraints which are not handled well by today’s general-purpose operating systems. Primitive portable devices such as MP3 players, with only audio-playing capabilities, are not as complex as an operating system. But when we move to the next level of portable devices like I-pod video, MP4 players, Set Top Boxes, Gaming Devices (like PSP), Personal Digital Assistants (PDA), etc., which incorporate multimedia applications, we encounter many issues. These include video jitter, poor “lip-synchronization” between audio and video, and slow interactive response while running video applications [1]. The source of these problems can be traced to the scheduling algorithm in use, for that particular application.

¹ Student
² Ph.D Student Supervisor
³ Assistant Professor
1.1 Our Research

In order to understand how tasks are scheduled and run, tasks of various natures (periodic, aperiodic, soft and hard), are run simultaneously on an experimental OS. In addition, tasks are chosen such that they are similar to an Mpeg decoder, to study the behavior in various cases. Finally, we understand various parameters needed to describe the execution and running of a task. Information such as worst case execution time, available time for the job before the timer can fire, total number of instances of the task created, are obtained dynamically, when executing the task. The most important parameter would be the deadlines missed, for each task that is run concurrently. We obtain the numerical values for these parameters, by manipulating the S.Ha.R.K kernel’s C code.

2 EVALUATION

2.1 Implementation details

The experimental Operating system test bed employed in this project is S.Ha.R.K [2], chosen due to its modularity, dynamic re-configurability, support for device scheduling, and modular file system. S.Ha.R.K tasks are defined using standard C functions which return a void *type and can have one void * argument, which is passed when the task is created. A task is identified by a system-wide unique process identifier (PID) and meets a set of Quality of Service requirements specified in its Scheduling modules [3]. The typical task code consists of an optional initialization of local variables and resources (the scheduling policy for the task, the drivers, semaphores, mutexes etc), followed by a (finite or infinite) loop, representing the task’s body. The last instruction of such a loop must be the primitive task_endcycle() or the primitive task_sleep() which signals the end of a generic job.

2.1.1 Structure of S.Ha.R.K applications

For writing one’s own application, three kinds of files are needed:

1. The program file is the source file containing the main function: int main(int argc, char **argv)[4].

2. The initfile is a normal source file which contains the instruction for the program initialization. The initfile includes the system header, initializes the modules regarding the scheduling policies for tasks and shared resources, the graphical mode if needed, the keyboard and all the required devices. When the application is launched, before starting the multitasking mode the program must initialize the devices, the resources and the schedulers which will be used by the application. For doing so, the kernel calls the “kernel register levels” function, which usually registers the following modules:
• Scheduling modules: a scheduling module implements a particular scheduling algorithm, for example EDF, RM, Round Robin, etc.

• Resource Modules: a resource module implements a shared resource access protocol (for example the semaphores, the mutexes, etc.)

• Other devices, such as the file system, and other devices that has to be initialized before entering the multitasking mode.

3. The makefile, which mainly instructs the X.EXE extender to make the application, with the file path, source files, and libraries needed, etc.

2.2 Experimental Setup: Running a Multimedia Application

The BCA demo is used as an approximation to an mpeg decoder. The reason is that BCA is a demo which reads in image files, takes in input from the user, and displays new output. Hence the number of cycles needed by the processor is varied, which is exactly the case we are trying to study, in the case of MPEG decoder. The mpeg decoder has to be rewritten on the same lines, to suit the FAT32 system.

Figure 1 Screenshot of BCA demo

3 RESULTS

On the S.Ha.R.K kernel, we run the following demos to study how the scheduling algorithm for various tasks work:

• BCA, a game demo, with a few soft periodic tasks, and a few hard tasks. (not necessarily periodic)
• Advtimer, which shows the S.Ha.R.K. Timer with TSC and APIC. This classifies as a hard periodic task.

• Ego, a simple demo, which displays graphical text, and comprises of three hard periodic tasks.

• Aster, a hard periodic task as well, which runs for 10 seconds.

Some of the parameters studied are listed, as below:
1. Avail time – The time the task can execute before a timer fire.
2. Number of deadlines missed
3. Number of queued activations
4. Number of skipped activations
5. Number of execution overruns missed
6. Job Control info : Sum – Total execution time since it was created
7. Job Control info : Max – Maximum time used by a task instance since it was created
8. Job Control info : Current – Total execution time of the current instance

Case 1 Run of three demos simultaneously, namely, the BCA demo, the Advtimer demo, and the Ego demo. The results are presented below. On printing out the real level that owns the task, we find that the hard tasks have higher priority (by being in Level 1) than the soft tasks, which are owned by level 2. For all the following graphs, the x-axis labels are the tasks that are scheduled, when running the demos. Update is the name of the task for Advtimer demo, Ego1, Ego2, Ego3 are the tasks for the Ego demo, and all the other tasks belong to the BCA demo.

The total execution time for every task varies according to the time period of the task, and the nature of the task. The y-axis is in milliseconds. The graph on the right shows the available time for a task, before the timer can fire, for all instances of that task. If the available time is high, the probability of the task missing its deadline is low, and vice versa.

The most interesting graph would be the deadlines related parameters, as shown below. We observe that most of the tasks have not missed their deadlines, especially the hard tasks (update, ego1, ego2, ego3). Only two soft tasks have missed their deadlines, and skipped activations. The criticality of
these tasks is low, and hence did not affect the successful running of all three demos simultaneously.

**Figure 4 Deadlines missed**

**Case 2** In the second case, we run the same three demos, but we change the nature of the tasks. The Advtimer and BCA are left unchanged, but the Ego tasks are all made soft periodic, instead of hard periodic. We observe that the Advtimer does not get scheduled, and the run aborts after creating ego1 task and the initial tasks of BCA demo (aereo_creator, hard_aereo, cannone_creator, hard_cannone)

**Case 3** In the third case, we run again the same three demos, but here, the Ego demo, and BCA demos are left unchanged. The advtimer is now a soft periodic clock. Again, the initial tasks of BCA demo run successfully, and so do the tasks of Ego (Ego1, Ego2, Ego3), but the advtimer is not schedulable again.

**Case 4** In this case, we run 4 demos, the fourth one being the Aster1 demo. The graph below shows their worst case execution times, and available time (The time the task can execute before a timer fire). Thus, if a task’s available time is high, (and is close to its worst case execution time) then the likelihood of the task missing its deadline is small. The tasks, aereo_creator, cannone_creator, soft_missile have very less available time, and thus are liable to miss their deadlines.
The deadlines missed and other related parameters, track trends similar to the above graph. In accordance with the graph above, the tasks which have a low available time are the ones which miss deadlines. When there is a scheduling crisis, (due to deadline or period of task), the priority algorithm ensures that the hard tasks complete their execution, and do not miss deadlines. BCA demo (similar to an mpeg decoder) has soft periodic, hard periodic, soft aperiodic and hard aperiodic tasks. In case of scheduling crises, the soft aperiodic tasks (Soft_missile, esplo) either miss deadlines, or are made to skip their activations. Another measure would be to queue the activations. (Like in the case of above, for Aereo_creator, and cannone creator).

4 CONCLUSION
We have analyzed various parameters of scheduling, with particular attention to Multimedia – like applications. Tasks of various natures (periodic, aperiodic, soft and hard), were run simultaneously on S.Ha.R.K. The most important information, the number of deadlines missed, was obtained when executing the tasks. The S.Ha.R.K kernel has functionalities to allow the application to change the CPU speed on hardware which has frequency scaling capabilities. It is possible to get and set the current frequency and obtain the theoretical value of the transition (from one task to another) duration. This is a good place for future work and can be used to analyze the effects of increasing the playout delay of a multimedia task, and verify the results which were mathematically proven in the paper, *Meeting CPU Constraints by Delaying Playout of Multimedia Tasks, written by Balaji Raman, Samarjit Chakraborty, Wei Tsang Ooi.*
5 REFERENCES


2. www.shark.ssup.it.
