Kernel Support in a Distributed Multimedia Environment

Pierre G. Jansen and Paul Sijben
University of Twente, Dept. of Computer Science
PO Box 217, 7500 AE Enschede, the Netherlands
e-mail: jansen@cs.utwente.nl and sijben@pegasus.esprit.ec.org

Keywords: continuous media, ATM networks, periodic processes, scheduling, resource management

This research[1] is part of the multimedia Huygens project [1].

Abstract

We refer to a multimedia environment, consisting of workstations and servers connected by a network dealing with video and audio streams. These streams require Real-Time (RT) handling. Fortunately the requirements are not so demanding as those in Hard Real Time (HRT) systems. Therefore resources are easier to obtain and less restrictive in use. This implies a more relaxed behaviour of workstation kernels with respect to timeliness and opens the possibility of having a better average use of resources. Such a kernel may (1) respond in an opportunistic way by serving the most demanding process immediately and not caring about the others or it may (2) make flexible Quality of Service (QoS) contracts with applications. In case of emergency it should be possible to break a contract. Both types of kernels are considered and (dis)advantages are compared.

1. Introduction

The widespread availability of multi-service networks opens the possibility of offering new types of electronic traffic such as video and audio. On the other hand we have the availability of workstations capable of handling video and audio streams, mostly by controlling the streams indirectly via dedicated hardware and sometimes by handling them by system or application software. Since this requires timeliness we are dealing straightforwardly with Real Time (RT) systems. However we certainly do not refer to HRT systems in which a catastrophe occurs when timing requirements are exceeded. Instead we deal with systems in which timeliness is maintained without having costly resource reservations, expensive timing administrations or rigid delivery contracts. Such a contract could be made between several parties such as a client, a network and a server. Of course we would like to maintain synchronisation of sound and image, both of high quality, however we occasionally could tolerate the loss of a display image or a sound sample. Under pressing circumstances we could even tolerate the breaking of a delivery contract.
We will take a closer look at the environment in which a multimedia workstation will operate. Here we expect a typical session, that is a user looking at a window in which a movie is playing or video conferencing is proceeding.
We may expect the Continuous Media (CM) stream to be received and unpacked by the ATM AAL5 interface [8]. The unpacked but still compressed video (moving JPEG or MPEG [ISO /IEC91]) is sent to the decompression unit and from there straightforwardly to a video processor that displays the images in time in its window.

The immediate question is now: How and where should processing be done. This might be a dedicated processor, taking care of the necessary buffering and synchronisation. The sound could be processed in a similar way. This implies that the main processor (or processors) - which does the general purpose task of the workstation - does not have to handle the RT CM streams itself but only the set-up of these. On the other hand the dedicated processor must do the RT work now. This processor is probably better equipped to meet the timeliness requirements. We expect this processing to be executed in hardware for a great deal. This has already proved to work within the Pandora project [2].

Shifting the RT obligations from the main processor to the dedicated processor however does not free us from the responsibility of paying attention to the RT aspects of CM stream handling. Also, we might want a general purpose processor to intervene in a CM stream in order to do some filtering. An extreme example for this type of filtering is

[1] A less recent description has been presented also by the same authors in "Real-Time in Multimedia: Opportunistic Scheduling or Quality of Service Contracts?", which is accepted for publication at the ISMM conference on Distributed Multimedia System and Applications, Hawaii, August 1994.

given by Massalin in [3], where the processor is part of a Phase Locked Loop configuration in order to add a beat to a piece of music.

Filtering indeed requires RT support of the kernel. Since we deal with CM streams, in which packets are transferred and handled periodically, the nature of this filtering support must be periodic too. On the other hand we also need to support aperiodic events, for instance caused by the arrival of an IP-package, or an interrupt from keyboard or file system. These may require handling immediately.

In this paper we will consider two types of RT kernels:

- the opportunistic kernel
- the contract kernel

Sometimes these kernels are also referred to as dynamic or static respectively, as for instance in [4]. In a dynamic kernel no information about the arrival of tasks is known a priory. Therefore the scheduling decisions have to be made at the moment a task arrives. In a static kernel however, arrival times and resource usage are known *a priori*. So the scheduling decisions can be done off-line. We will give an overview of both kernel types and will compare the advantages and disadvantages.

2. The Opportunistic kernel

In the opportunistic kernel we think of a workstation with a RT kernel that will adapt itself to the immediate needs of the system as well as possible. The kernel has no knowledge about the behaviour of its environment in the future. It simply serves the most important process until this process has completed or until a more important process is requesting service. Therefore the kernel is called opportunistic.

In the opportunistic kernel jobs will arrive without *a priori* knowledge. These are scheduled according to a priority. This priority is typically derived from parameters as used in classical scheduling policies such as *Static Priority Scheduling, Shortest Process Time*, *Shortest Slack Time* (elapsed time minus its estimated completion time) *or Earliest Deadline First.* Among others Liu and Layland [5] proved that a given set of processes, scheduled by any satisfying scheduling algorithm, could also be scheduled by a deadline driven algorithm. Without any precautions these techniques may lead to the phenomenon of *priority inversion.* This could happen when a high priority task is blocked for a resource that is held by a low priority task. The latter may not proceed due to its low priority, thus blocking the high priority task.

Depending on the synchronisation policy such as the *Fixed Priority Protocol*, *the Basic Inheritance Protocol,* the *Priority Ceiling Protocol* [6] or the *RT Transaction Protocol* (RTTP) [7] a dispatcher assigns processes to the processor(s).

The *Fixed Priority Protocol* generally suffers from priority inversion. The *Basic Inheritance Protocol*, the *Priority Ceiling Protocol* and the *RT Transaction Protocol* provide methods to avoid priority inversion.

The *Basic Inheritance Protocol* does this by inheritance of priority. Low priority processes, owning shared resources that are also requested by high priority processes, inherit the high priority from the waiting processes. A primary disadvantage of this scheme is the impossibility of avoiding *transitive waiting*.

The *Priority Ceiling Protocol* avoids priority inheritance and also transitive waiting. The basic idea is to make way for high priority jobs, even if it is not certain that they will become active. The rule is that a medium priority job may not pre-empt a low priority job if the low priority job holds resources that could be claimed by a high priority job. The priority ceiling associated with a resource is the highest priority of a job that ever can claim this resource.

For an opportunistic kernel we expect most from a combination of a deadline driven scheduling policy and a RTTP as a synchronisation policy. Since this is our preference we explain this combination in more detail.

The RTTP works roughly as follows:

Given a set of processors, which are executing a number of concurrent real-time jobs. Each of them has a priority, derived from dynamic parameters such as a deadline or a periodic interval, delay and execution time.

A job is a sequence of alternately free-running and resource-using *transactions*. A resource stores, manipulates, or communicates continuous media data. An important aspect of a resource is that it can be shared by concurrent jobs under mutual exclusion. When a job is free running, it uses no resources (except the processor). Resource-using transactions are the interesting ones and are simply referred to as “transactions”.

When a transaction starts, it simultaneously acquires all resources it needs to complete the transaction. During the transaction resources can only be released. A transaction has completed when it has released all of them and becomes free running. During execution of transactions, external synchronisation may take place, for instance to do I/O or an RPC. For the moment however, we only consider simple transactions that do not synchronise with external events.

A transaction manager takes care of managing resources. It keeps track of the state of all resources, of the

identities of requesting transactions and of their requested resources. A transaction is assigned a processor if it has the highest priority and when it can acquire all its requested resources. The manager assigns them to the transaction in question at once in an atomic action, after which the transaction starts running, possibly by pre-empting lower priority transactions. Priority inheritance is used when a high priority transaction is waiting for a low priority transaction to release one or more of its resources. The transaction manager knows about the requested resources and the priorities of their requesters. With this information it can execute the inheritance of priority dynamically.

In RTTP transactions can be waiting for resources to be released. Because of the simultaneous resource acquisition strategy, runnable transactions can run to completion without further acquisition of resources. When the highest priority transaction, say T_h, needs resources of runnable transactions, priority inheritance brings these to execution if they were not executing already. From then on they can run until the release of the requested resource(s).

This implies that T_h never has to wait for *more transactions* than its number of *requested resources*. Note also that the runnable transactions could be completed in parallel in case there should be adequate hardware to do this.

If the acquisition of resources were not atomic, transitive waiting could be the consequence. Such a situation occurs for instance if a transaction
$T_h(R_1,R_2)$ *waits for* $T_1(R_2,R_3)$ *waits for* $T_2(R_3,R_4)$ *waits for* $T_3(R_4,R_5)$ *... waits for ...* $T_n(R_x)$,
where $T_i(R_j,R_k)$ denotes the transaction T_i waiting for the resources R_j and R_k to be released. Then there exists *no* clear upper bound for the number of transactions T_h is waiting for and hence no upper bound of waiting time.

Advantages

In RTTP waiting relations of transactions cannot be transitive and an upper bound of waiting time can be estimated easily.

RTTP transactions are opportunistic and always try to fulfil the requirements of the highest priority process. Having no knowledge about the future behaviour of the environment, this is probably the best way to react to aperiodic events.

Disadvantages

Arrival of a transaction involves re-scheduling, which may take some overhead and may cause pre-emption.

Since the kernel does not make use of statistical information, it can not anticipate the future behaviour of the environment.

3. The Contract kernel

In the contract kernel a distributed application tries to establish a contract between several parties, such as a producer (workstation), a network and a consumer (workstation). Of course this can be extended to a client/server environment. The network is typically an Asynchronous Transfer Mode (ATM) network [8]. Such a network can give some statistical guarantee of bandwidth and it can support RT behaviour under the assumption that both end systems keep to the contract. The service required by the network is specified by QoS requirements such as bandwidth, delay, jitter and error rate. However, it might happen that some thrashing still occurs occasionally. In this type of network HRT guarantees cannot be given. For multimedia application this is not necessary, as we have indicated already in the introduction.

For the periodic processes we can estimate the future needs of processor time and resource usage. A rational technique would be to compute a schedule at the time a contract is made. However, sporadic unexpected events might make it difficult to keep to this schedule. This requires some extra adaptibility of our kernel. We might have to push the processor beyond the average agreed values in the contract. This certainly does not mean that the schedulability criteria of Liu and Layland [5] are not valid any more. On the contrary, their Rate Monotonic algorithm[2] is still very useful and it can give an easy first hand impression for *admission control* of a task.

The interesting question for us is how to introduce the needed flexibility: (1) how can we temporarily adapt the schedule when the processor becomes overloaded due to sporadic events and (2) how do we deal with fluctuations or missing information from the network.

3.1 Admission control

From the considerations above it follows that it makes sense to split up the jobs in two classes: (1) the periodic jobs and (2) the aperiodic jobs also called sporadic jobs.

Tindell, as mentioned in [4], exercised with a mix of periodic multimedia processes and sporadic processes. When the sporadic jobs consume no more than 40% of the available processing power and the multimedia processes consume not more than 50%, then the response times of the sporadic events are almost equal to those that would be observed if the machine was running with a zero multimedia load.

Observations of overload behaviour are also given by Sha [9]. He states that the task with the longest period will

[2] Under this algorithm a process with the shorter period has the higher priority and will get the processor first.

be the first to miss the deadline. As a solution he proposes period transformation. This implies a shorter period - and run-time, with as a consequence a higher priority. This is probably not a good solution in a multimedia environment.

Miller [10] presented a completely different solution to an overloaded processor. He proposes a weight for each task, thus giving a measure of flexibility. He does this in an Earliest Deadline First algorithm. This causes more flexible tasks to receive less computation time when the system comes under pressure. In this context we consider a video task more flexible than an audio task.

ARTS [11] is a real-time tool set, which includes a *schedulability analyser* and an *Advanced Real Time* (ART) monitor for analysing the run-time behaviour of the system. The system on which ARTS runs includes a FDDI network. An application can specify QoS requirements to the network. A request is accepted if the network can meet the requirements. In case of more requests later on, the original QoS requirements can be changed dynamically until some lower-bound is reached. There are plans to integrate the ARTS ideas in RT Mach [12], where a manager decides how much to degrade the services to applications.

YARTOS [13] has an admission control similar to ARTS. The kernel decides *a priori* if the application's QoS can be met. In case of rejection, an application may try to renegotiate with less demanding requirements. After a job has been admitted, YARTOS uses an opportunistic strategy for scheduling and adapted an earliest deadline first strategy based on pre-emption. In YARTOS priority inversion is avoided by changing deadlines, which is quite similar to a priority inheritance protocol.

Hyden [4] states that the use of QoS for scheduling CM within networks gives enough flexibility in order to accommodate a wide range of scheduling demands. Also he presented means of mapping high level QoS to low level QoS. Applications are provided with the knowledge of the current availability of the resources via a so called Virtual Processor Interface. This gives applications the possibility to control the manner in which their performance degrades. A little experimental kernel called NEMO has been developed. It incorporates basic scheduling techniques as well as accounting and policing mechanisms in order to maintain QoS contracts.

3.2 TUKKER[3]

The basics of the NEMO kernel serve as a starting point for further development of TUKKER. Here follows a global description of TUKKER.

[3] TUKKER is an acronym for Twenty University miKro KERnel.

The network QoS specifications of an application are used to derive a kernel QoS specification. These specifications include process period, arrival time, release time, computation time and deadline, thereby taking into account the use of resources.

With this information we start a schedulability analysis, much in the same way as is done in HRT systems. We will however not base our analysis on a hard guarantee of utilisation of resources but on the probable use of them. This analysis is performed on all end stations involved in a CM stream and computes the possibility of concluding a contract. No schedulability means no contract.

A contract is assigned a weight according to its importance. In case of refusing an important contract there is the possibility of breaking an existing contract in favour of the more important one. Rules should be developed in order to determine when breaking the contract is allowed and what the penalty should be.

If a contract can be concluded, the process schedules are updated in background. A dispatcher brings them to execution.

There are two reasons why an application cannot deliver a requested QoS:

1. The network (or another contract party) does not keep to its contract and the required information is not delivered in time or not at all.
2. Due to overload of a-periodic processes there is not enough processor time and/or resources for the periodic processes.

This probably will have consequences to the application that will experience lack of processing time, indispensable resources or lack of information.

Consequently these causes must lead to a lesser quality of moving image. This could proceed until the picture freezes. Sound cannot easily deal with decreasing quality. A sudden implosion of sound to silence is not likely appreciated. So the QoS of sound will probably not have a flexible lower bound, unless smart filtering techniques are developed which can deal with decreasing sound information.

Case 1 is easy to handle for a kernel. In first instance the application does not need to know from the kernel that there is not enough information. It can discover itself that there are empty or partly filled buffers and it could do some effort to produce the optimal result in the assigned time slice.

Case 2 is more difficult to handle. This is because the dispatcher has to decide from where to pick the lacking processor time. Moreover the victims - periodic processes we assume - need to know that they are assigned less time than agreed earlier. The kernel bases its decision for the selection of a victim on the QoS lower bounds of periodic processes. These bounds indicate to which extend the QoS

of the stream might drop. Dropping beyond this point will involve exception handling in which operator intervention might be needed.

Advantages

Since the network QoS requirements of continuous media streams are directly translated to the QoS requirements for multimedia kernels, the contract kernel is expected to adapt well to this type of applications. Consequently we expect a better average use of the processor(s) and other resources.

Disadvantages

Schedulability analysis is, in fact, np-complete. This implies that it could be quite time consuming to find feasible schedules. On-line analysis opens a new research area for finding a good compromise between a adequate scheduling scheme and an acceptable computation time.

4. Conclusion

Both proposed types of kernels are expected to behave well in RT systems where requirements are not too hard. The *opportunistic* kernel reacts immediately to important events and keeps resources at hand for handling the involved processing. Consequently it is a promising kernel for an environment in which aperiodic events play a major role. In case of continuous media such as audio and video the *contract* kernel directly fits to the Quality of Service requirements as used in ATM networks. Therefore this kernel is a natural candidate for handling CM streams. We are currently building a prototype kernel indicated as TUKKER. It is of the contract type and strongly based on a periodic thread model.

Of course we still have a number of aspects of RT systems which do not only address CM oriented systems. Among others we mention here (1) the implementation of efficient context switching and (2) lowering the number of kernel boundary crossings. They remain valid as research subjects, also in the multimedia context.

Acknowledgements

Acknowledgements are due to Andrej Koelewijn, Danny Havenith, Michiel Pelt and Koos Wiegman for contributions in the Tukker-meetings and to Richard Earnshaw for giving comments on the manuscript.

References

[1] Mullender S.J., Leslie I.M., McAuly D. “Operating Support for Distributed Multimedia”, Proceedings of Summer Usenix Conference, Boston MA, June 1994.

[2] Hopper A.: “Pandora, an experimental system for multimedia applications”, ACM Operating Systems Review, 24 (2), April 1990, pp. 19-34.

[3] Massalin H.: “Synthesis: An efficient implementation of fundamental operating system services”, Ph.D. Thesis, Columbia University, 1992.

[4] Hyden E.A.: “Operating System Support for Quality of Service”, Ph.D. Thesis, University of Cambridge, Febr. 1994.

[5] Liu C.L., Layland J.W.: “Scheduling algorithms for multi-processing in a hard-real-time environment”, J.Ass. Comput. Mach., Vol. 20, Jan. 1973, pp. 46-61.

[6] Sha L., Rajkamur R., Lehoczky J.P.: “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”, IEEE Trans. on Comp., Vol. 39, No. 9, Sept. 1990, pp. 1175-1185.

[7] Jansen P.G., Gansevles F.: “Priority Inheritance in Real-Time Micro Kernels”, Proceedings International Symposium on Computer and Information Sciences VII, Antalya, Turkey, Nov. 1992, pp. 211-220

[8] Le Boudec J.Y.: “The Asynchronous Transfer Mode: A tutorial”, Comp. Networks and ISDN Systems, V 24, 1992, pp. 279-309

[9] Sha L., Lehockzky J. and Rajkumar R.: “Solutions for some practical problems in Prioritized Preemtive Scheduling”, Real-Time Systems Symposium, The Computer Society of the IEEE, IEEE Computer Society Press, Dec. 1986, pp. 181-191.

[10] Miller L.L.: “Predictive Deadline Multi-Processing”, ACM Operating Systems Review, Vol. 24, No. 4, October 1990, pp. 52-63.

[11] Tokuda H., Tobe Y., Chou S.T.C., Moura J.M.F.: “Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network”, Proceedings of the ACM SIGCOMM' 92Symposium on Architectures and Protocols, Oct. 1992, pp. 88-98

[12] Oikawa S., Tokuda H.: “User-Level Real-Time Threads: An Approach Towards Performance Multimedia Threads”, Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, Nov. 1993, pp. 61-71,

[13] Jeffay K., Stone D. and Poririer D. “YARTOS: Kernel Support for efficient predictable Real-Time Systems”, Real-Time Programming, Pergamode Press, Oxford UK, 1992, pp. 7-12.