Operating System Concerns for Multimedia

Ashish Ranjan

Jaydeep Punde

Arun Singal

Department of Computer Science

Florida International University

Miami, Florida

Operating System concerns for Multimedia

Abstract:

A multimedia system comprises of audio+video+data. The data characteristics of this system are quite different from the traditional data, which is well understood. But here the data is spatio-temporal in nature and hence involves issues like real time scheduling and synchronization. Obviously multimedia system needs additional support from the operating system.

In this context, this paper discusses “Operating System Concerns for Multimedia”. The paper will address the problems related to the operating systems issues in multimedia systems and will discuss various existing and proposed techniques researchers have developed to solve the problem. The paper discusses three aspect of the Multimedia system namely QoS support, CPU scheduling and File system.

This paper is divided into three sections. The first one deals with the QoS support for multimedia system. The second one deals with the File system and the last one deals with the CPU scheduling in Multimedia operating system.

Section 1

QoS in Multimedia Operating Systems

By: Ashish Ranjan

Introduction:

Multimedia has become an integral part of today's computing environment and so it needs to be integrated to the present computing systems without affecting its own as well as the system's performance. The behavior of multimedia application is quite different from the other applications which we commonly use such as text editor, browser, compilers etc. Multimedia has real time constraints, which demands real time support mechanism from operating systems. The real time characteristics of multimedia systems is that of "soft real time" [1] because unlike in "hard real time" systems, missing a deadline is not catastrophic, though it can effect the quality of presentation of multimedia stream. Supporting applications means allocating resources for them. This resource allocation is handled by operating systems. Researchers have tried to implement soft real time constraint characteristics in the existing operating systems and have got varied results. One thing that has come out of the various research is given the constraint that the multimedia streams have to be supported on the systems which also needs to support all traditional application, it is very difficult to process the multimedia stream as desired. One paradigm for allocating resource is resource allocation based on Quality of Service. Here application demands resources from the underlying system (which typically consists of operating systems and network communication systems). In this case applications needs to know what kind of resource (CPU time, network bandwidth etc) reservation can be made (if at all they can be made). But application are not aware of the resources, they just demand from the underlying

system resources by specifying parameters such as throughput, delay, jitter, synchronization skew etc. It is up to the system to understand these parameters and allocate resource for the application. These parameters are also called quality of service requirements. In the simplest sense, Quality of Service (QoS) means providing consistent, predictable data delivery services. In other words, satisfying application requirements. I will discuss QoS specification, QoS mechanisms and QoS management. And then I will discuss QoS model of "Nemesis" operating system.

QoS Paradigm: There are various paradigms for scheduling resources for multimedia tasks. As we have already stated that multimedia has real time requirements (for example in a video presentation the video data must be processed at regular interval and if a certain frame misses its deadline it should better be not presented to the user). HRT paradigm (Hard real time systems) provides stringent, deterministic guarantees that deadlines will always be met. The Earliest Deadline First (Layland & Liu) and Rate Monotonic Scheduling are some of the scheduling mechanism to achieve the stringent guarantees. But these algorithms perform poorly in case of overload [2][4][5]. As the researchers understood the importance of developing integrated approach for multimedia systems focus went on use of QoS to provide a probabilistic assurance that resource requirement will be satisfied a certain fraction of time. At the same time we should remember that this probabilistic assurance could be stringent as well. Hence this paradigm encompasses the HRT paradigm. This paradigm is very close to the soft real time paradigm where some failures are tolerated.

Distributed Multimedia: Most multimedia applications are distributed and have as one of its major goals the integration of continuous media data within a dirtibuted-computing environment. That is multimedia data has to pass through several layers such as network communication

system, protocol stack, devices etc. QoS guarantees should be provided at each layer. That is the approach is an integrated approach. This approach is really suitable for distributed multimedia application because what is the advantage of just guaranteeing CPU time for a multimedia application when the network cannot provide in time the data needed to process the task. That is why it is essential that the Quality of service is configurable, predictable and maintainable system wide including the distributed system platform, operating system, end-system devices, communication subsystem and network [8]. There are various model for QoS architecture and it is beyond the scope of the paper to discuss them. But for the sake of completeness i am mentioning a few QoS architecture. Later we discuss one of the frameworks developed by Washington University [7]. They are

1. Extended integrated reference model.
2. Quality of Service Architecture (developed at Lancaster University)
3. OSI QoS framework. Being developed by the ISO SC21 QoS working group.
4. Heidelberg QoS model. Developed by IBM European Networking center.

There are many more, in [8] various QoS architectures have been dealt in detail.

Operating System and QoS:

Operating system manages resources. The resources are Memory (cache, Main memory, disk), Bandwidth (Disk input output, Bus, network) and last but not the least CPU. An operating system based on QoS must manage the system resources in such a manner that adequate resources are available at the right time to perform the operation with requested QoS. Resource management in operating system for Quality of service has to perform the following tasks.

1.Specification: It is concerned with capturing application level quality of service requirements and management policies. This specification is passed from one layer of the system to the other. Obviously QoS specification is generally different at each system layer. QoS specification consists of flow synchronization specification (degree of synchronization between related flows), flow performance specification (This is most important for the multimedia systems and from the user perspective. Flow performance parameters consists of traffic throughput rates, delay, jitter and loss rates. These parameters vary from one application from other. For example humans are more sensitive to Audio than to Video. So loss rates in audio stream should definitely be lesser than the loss rates of a video frame.) , Level of service (As discussed above the QoS paradigm encompasses both SRT and HRT paradigm and level of service precisely does that. That is level of service expresses a degree of certainty that the QoS levels requested at flow establishment or re-negotiation will actually be adhered) and Cost of service which specifies the price the user is willing to pay for the level of service. (As stated in this article, no operating system can achieve all that is required by the multimedia streams, tradeoffs will have to be made. This parameter comes in handy while making the tradeoff or choice between several alternatives.)

2.QoS mapping: This is concerned with converting the high level specification into actual resource level parameters (low level parameters). Because of the need to keep the application level specification simple and also because application may not know about the resources involved in supporting the application, mapping is done at various system levels.

3.Admission control includes a test whether enough resources are available to satisfy the request without interfering with the previously granted request.

4.Allocation and Scheduling: This is where actual resources are allocated and scheduled. The system has to adhere to the temporal characteristics of multimedia data and hence various

scheduling mechanisms are adapted for different application. In some operating systems application are divided between various classes and scheduled according to the type of class [4]. Arun Singal has dealt various scheduling mechanisms for multimedia operating systems in detail in this paper.

5.Accounting/Policing: As the word states, it implies tracking down the resources consumed by the task. This is essential to make a fair allocation (which may or may not be acceptable), or to make sure that task consumes more resources than the previously negotiated value.). This can also lead to system initiated adaptation. It should be noted that accounting is a big problem by itself. The problem here is how to identify which application used what amount of resources because there is no one to one mapping between an application task and a process. One of the abstraction called reserve abstraction [6] does the job of accounting quite well.

6.Adaptation: The very paradigm of QoS based scheduling of resources requires adaptation. When a particular resource cannot be scheduled as desired then the QoS can be negotiated. That is, this can result in to a downgraded QoS and hence the resource requirement. Adaptation hence leads to a set of new resource parameters. Adaptation is critical because resource requirement is hard to predict partly because of the varied nature of multimedia application and user interaction. Moreover resource availability cannot be guaranteed in a time sharing, best effort operating systems (Unix, windowsNT). It is possible to provide more resource to an application if the system can afford and if that is not possible then application will have to adapt itself (Tradeoff will occur). Adaptation can be with or without feedback. We shall see later that in Nemesis operating system approach is adaptation with feedback. In case where there is no feedback application can change only the functionality of user interface and cannot change any resource parameters.

7.Deallocation: After the application dies resources should be freed reallocated among the live applications.

It is not possible to discuss in detail all of the above mentioned techniques and how are they actually achieved in a system. So, I will discuss briefly how specification and mapping is done, i.e. how a high level parameter is chosen and then actually expressed in resource parameter such as delay, CPU time etc**.**

Specification & Mapping: Parulkar and GopalaKrishna developed a QoS [7] framework for providing QoS guarantees within the end system for networked multimedia applications. There are four components of their end system QoS framework: QoS specification, QoS mapping, QoS enforcement and protocol implementation. Their model for QoS mapping is presented here.

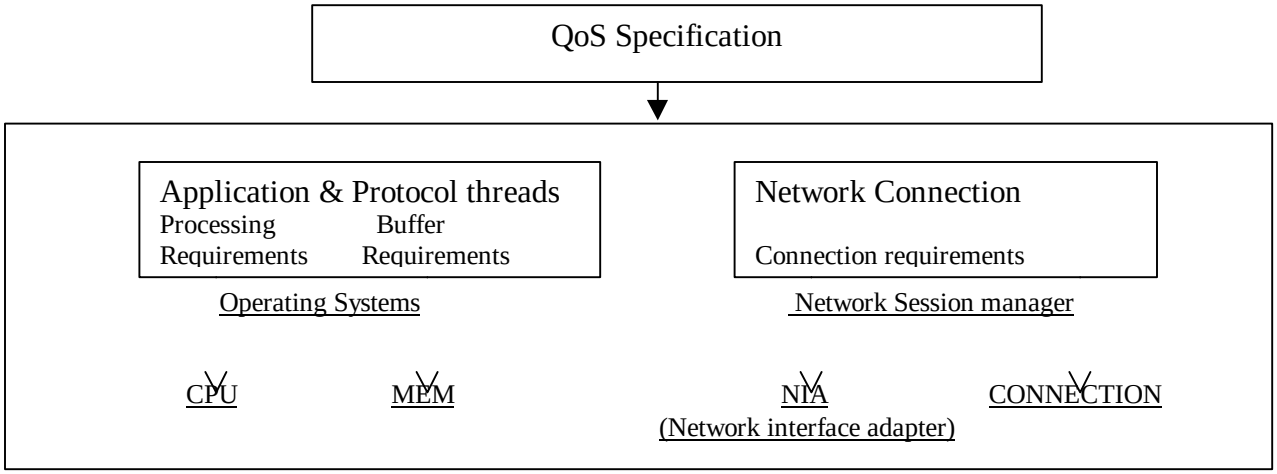

Fig 1.0

Three classes of application (Which encompasses most multimedia application) were chosen for the above-mentioned model. They are as follows:

1.Isochronous (Continuous media) 2. Burst (Bulk data transfer is required) 3. a) Message delay bounded b) Message rate bounded (Includes application that require low response time such as RPC requests).

The next step is to choose QoS parameters for these classes. The choice of parameters for the above mentioned classes are Maximum size of the frame, Average frame size, frame rate and maximum delay or maximum Bandwidth. The choice of parameters as stated in their paper has been driven by whether the application can give meaningful values to these parameters. For example Maximum frame size can determine the maximum application buffer requirement. The average frame size allows us to reserve less than the maximum bandwidth that would be dictated by the maximum frame size. The delay refers to the time shift in the multimedia stream at the destination with respect to the source. The point to be noted here is that not all application can give values to the above mentioned parameters. For example in burst type, the applications do not have accurate estimates average values for their bandwidth requirements. The steps for QoS mapping are as follows:

As the data passes trough different layers of protocol, protocol headers are attached to the frame, which increases its size, and hence scaling of required application bandwidth is done. Scaling gives a multiplying factor to account for the increase in the required bandwidth. The delay is partitioned across each module. Delay and scaling factor is then used to derive the parameters for network connection. From the network connection peak bandwidth and the delay budget scheduling attributes are derived.

After having seen the method for QoS specification and mapping lets focus on the operating systems design consideration for QoS support.

QoS problem in existing systems:

Some of the reasons for QoS problems in existing systems are as follows:

1.Data packets from the network are processed in First in first out basis for all connection. When some connection have real time requirement then waiting for packets from other connection becomes a problem. That is high priority packet has to wait behind low priority packets.

2.Kernel do a lot of hidden processing with high priority. These unaccounted for kernel activities make QoS guarantees difficult.

3.In the popular existing systems scheduling are optimized for time sharing system. The CPU bound process gets lower priority than an IO bound process.

4.The layered architecture of the communication systems may imply considerable data movement in the protocols. This leads to bottleneck because copying data physically is expensive [9]. One of the solution to this problem is the Zero-Copy architecture [14].

5.In a microkernel environment, an application is typically implemented by a number of process. Most of these process are servers performing work on behalf of more than one client [12]. This creates huge problems for accounting. On the other hand in kernel based system multimedia applications spend most of their time in the kernel leading to the same problem of accounting.

A number of other bottlenecks in high-speed communications are operating system related, and they are well documented. These primarily relate to avoiding unnecessary context switches, and avoiding extra data copying. We would like to have if possible, only a single context switch pair. One when the application requests the communication operation, and one when the operation is successful and the application can resume. (In the case of a request for information this is when the first complete piece of information the application can use is in place on the host and available in the application user space).

In considering integrated communications, we note that we know how to engineer operating systems for data reasonably well. So although we know how to deal with data in the host

memory hierarchy (cache, main store, backing store), what to do with audio and video objects is very much an open question. Furthermore, we know how to schedule conventional processes reasonably well, but work on how to deal with the real time constraints of mixed media and stream synchronization is another big problem.

After understanding the bottlenecks and the QoS paradigm lets discuss one operating system which has been designed with a view to support QoS paradigm. One such operating system is Nemesis.

Nemesis: [11] The design of nemesis supports temporal data (such as Multimedia data) which requires a consistent quality of service. Nemesis provides fine-grained guaranteed levels of all system resources including CPU, memory bandwidth and disk bandwidth. As stated above frequent context switches are one of the major bottlenecks. Nemesis made this operation cheap by providing a single address space for all application. Some of the features of Nemesis are given below

1.Provide QoS guarantees to applications for all real resources.

2.Separate control and data path. Multiplex data path operations once at the lowest possible level.

3.Reduce quality of service crosstalk.

3.Share text and data as much as possible

Lets focus on the QoS control in Nemesis.

Nemesis uses a feedback control mechanism for adaptation.

The feedback model is shown schematically in the figure below:

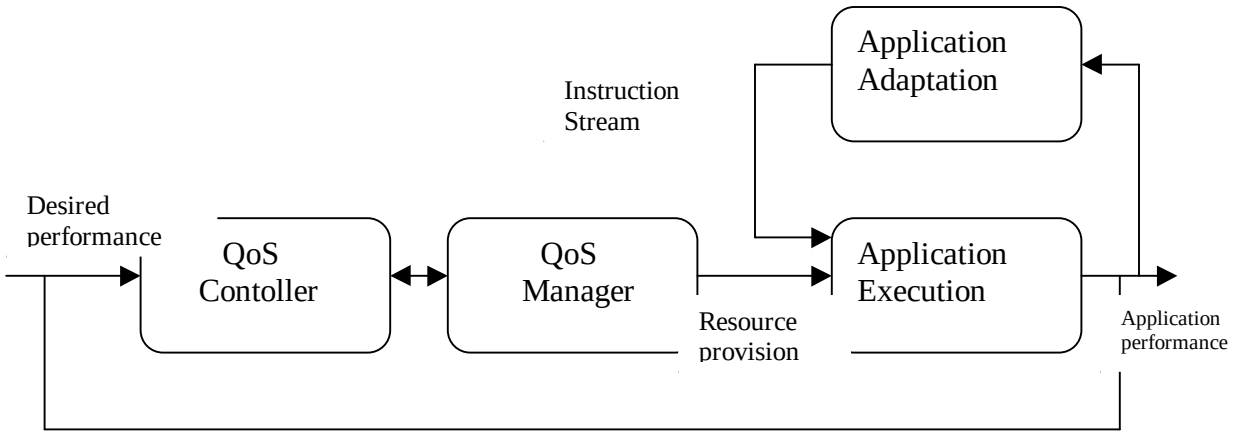

Fig.1.1 Source [16]

The QoS Controller states the policy to be adopted. This controller can be directly dictated by the user or by an agent running on the user's behalf or both. QoS manager implements the allocation of resources (as far as it can) according to the policies dictated by the QoS controller and then informs the operating system which actually allocated the resources and the application which adapts to the new resource allocation. In this approach application need not know exactly about its resource requirement but application has to implement algorithm to adapt to the changing scenario of resource. It should be noted that in this feedback scheme is different from the common scheme where application just degrades itself when resources are over committed.

Conclusion: The QoS paradigm encompasses most of the other paradigms (HRT, SRT, Workahead model) and is very flexible. This is particularly suited for multimedia application given the fact that there is not adequate existing scheduling mechanism for multimedia system.

There are quite a number of Operating system designed with a view to provide QoS such as HeiTs, Nemeis, AQUA ,SwiFT and Odyssey. This paper just presented the QoS model and explained its generality. Further works can be done on Specifications,mapping and on QoS manager.

References:

[1] Ralf Steinmetz ,"Analyzing the Multimedia Operating System", IEEE MultiMedia, 2, 1, pp 68-84 (Spring 1995).

[2] T.Plagemann, V.Goebel, P.Halvorsen, O. Anshus, "Operating system support for multimedia systems",Computer communications,23,3,pp 267-289,(2000).

[4] P. Goyal and X. Guo and H. Vin "A hierarchical CPU scheduler for multimedia operating systems ", In Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI). USENIX, October 1996.

[5] Schulzrinne, H., "Operating System Issues for Continuous Media," Multimedia Systems, vol. 4, pp. 269--280, Oct. 1996.

[6] Clifford W. Mercer, “Operating System Resource Reservation for Real-Time and Multimedia Applications”, PHD thesis, School of Computer Science Carnegie Mellon University, June !997.

[7] R. Gopalakrishna, G. Parulkar, “Efficient QoS Support in Multimedia Computer Operating Systems”, Washington university report WUCS_TM_94_04, August 94.

[8] Aurrecoechea, C., Campbell, **A**., and L. Hauw “ A Survey of Quality of Service Architecture”, Multimedia Systems Journal, November, 1995 (to be published).

[9] K. Nahrstedt and R. Steinmetz. “Resource Management in Networked Multimedia Systems”. IEEE Computer, May 1995.

[10] H. Kitamura, K. Taniguchi, H. Sakamoto, and T. Nishida, "A New OS Architecture for High Performance Communication over ATM Networks," Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video (April 1995): 87--91

[11] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. IEEE Journal on Selected Areas in Communications, 14(7):1280--1297, September 1996.

[12] Distributed Operating Systems, Tanenbaum.

Section 2

Multimedia File Systems

By : Jaydeep Punde

Introduction: -

Multimedia imposes unique demands on the underlying file system in terms of storage, continuity of data retrieval, real-time constraints, and synchronization.

Most of the existing computing environments treat multimedia as a special application domain. However recent trends in computing clearly indicate that multimedia will become an integral part of many disparate applications. Unfortunately, the performance of multimedia applications that are built on top of generic file systems usually leaves much to be desired.

Multimedia Data Characteristics: -

This section describes the unique characteristics of multimedia data and outlines their demands on the file and storage subsystems.

- Large Data Size-

One of the primary differences between conventional files and multimedia data is that of size, multimedia data is typically orders of magnitude larger.

The table below shows an example of some common multimedia data sizes.

Type	Format	Parameters	Bandwidth	1-Hour clip
Video	MPEG -1	320 x 240, 30 fps	1.2 – 3 Mbps	550 – 1350 MB
Video	MPEG-2 (CCIR 601)	720 x 480, 30 fps	5 – 10 Mbps	2.24 – 4.5 GB
Video	MPEG-2 (HDTV)	1920 x 1080, 30 fps	20 – 40 Mbps	9 – 18 GB
Audio	Uncompressed	Stereo, 44.1 KHz 16 bits/sample	1.4 Mbps	630 MB
Audio	MPEG-1 Layer II	Stereo, 44.1 KHz	64-768 Kbps	30 – 350 MB

Table1. Common multimedia data sizes.

- Continuous Storage and retrieval –

A significant difference between multimedia data types such as audio and video and their traditional counterparts is that they possess a temporal dimension as well. The real-time constraints imposed on data retrieval affects the design of various file system modules such as disk scheduling, admission control, on-disk allocation schemes, bandwidth reservation policies etc.

- Heterogeneous data -

Each medium has inherently different characteristics. Considering video and audio, we see that video streams require a very high bandwidth whereas the QoS issues are not that stringent, in contrast audio consumes less bandwidth, but quality degradation is more perceptible to the user. From the point of view of the file system, this leads to the situation where design decisions and optimizations must be made with a view towards accommodating the idiosyncrasies of all the different media that are supported by the system.

- Synchronization -

Multimedia compositions usually contain multiple media streams, which have temporal dependencies between them. Temporal dependencies can be modeled using absolute timing or relative timing. The former scheme borrows from the motion picture industry where unique time codes are assigned to each frame. Another approach for modeling synchronization constraints uses relative timing between temporal intervals.

Allen’s temporal relations [1] define thirteen different ways in which two intervals can be related. These are as shown in Figure 1.

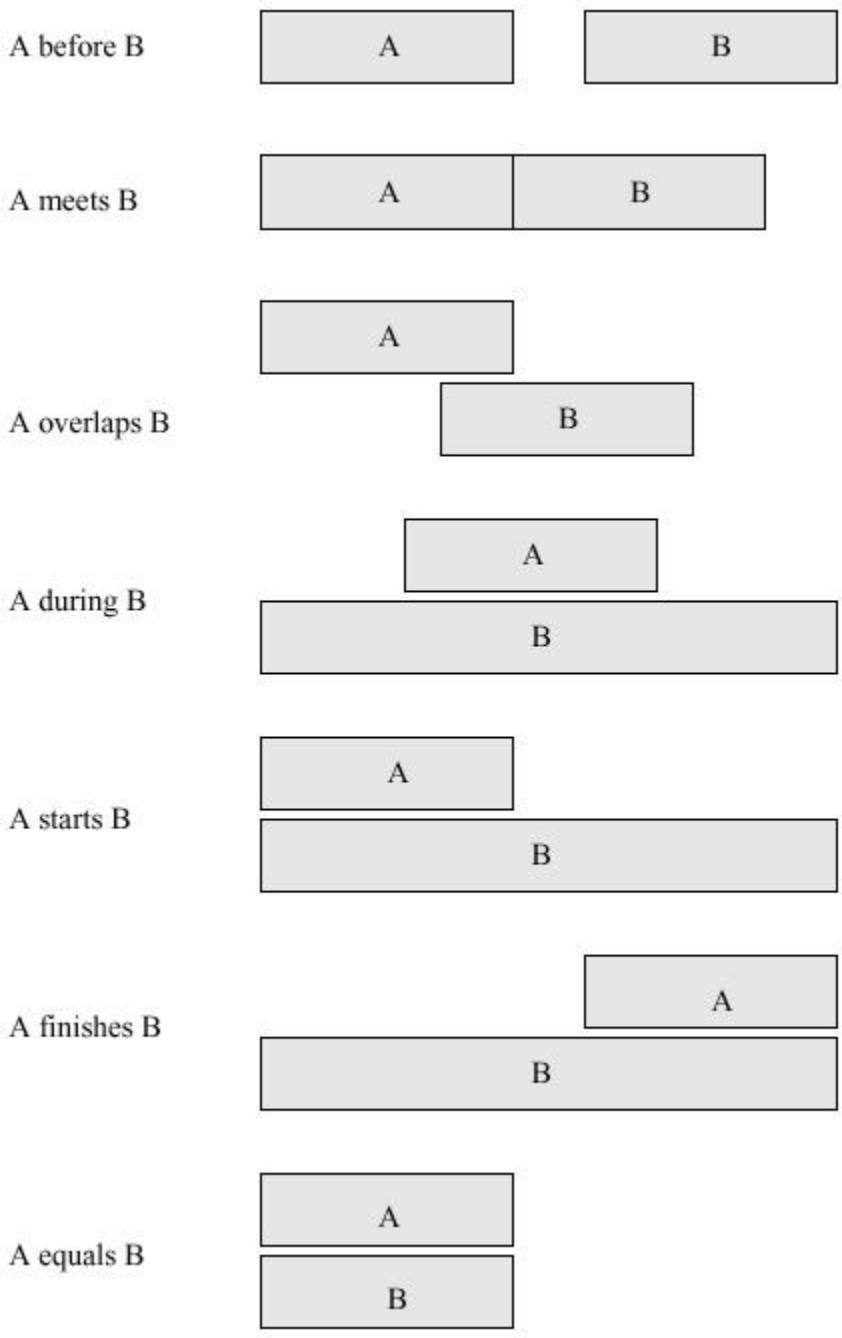

Figure 2.1. Source [8]:Allen’s temporal relation

Synchronization constraints that are commonly encountered in multimedia compositions can be easily specified using two of the thirteen temporal relations, namely equals and the meets relations. While the *meets* relation specifies an ordering of intervals that represent the continuous playback of a single medium, the *equals* relation temporally binds two different media together.

A representation of a synchronized video and an audio stream using the *equals* and the *meets* relations is shown graphically in Figure 2.

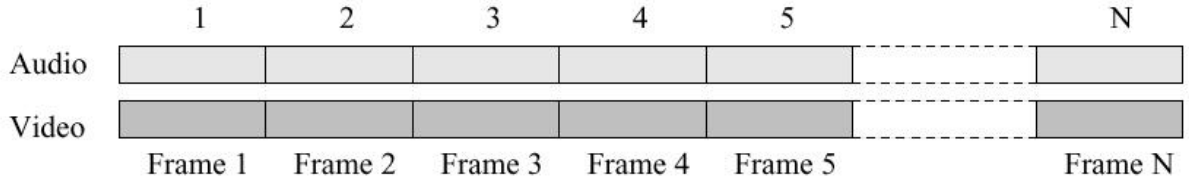

Figure 2.2. Source[8]: Representation of synchronized audio and video stream using equals and meets relation

Existing work done on multimedia file systems-

From the architectural perspective, multimedia file systems can be classified as [3]

- *Partitioned file systems* consists of multiple subfile systems, each tailored to handle data of a specific data type. An integration layer may provide transparent access to the data handled by the different subfile systems. There are multiple examples of systems using this approach, e.g. Tiger Shark [4] and the combination of UFS and CMFS [5].

- *Integrated file systems* multiplex all available resources (storage space, disk bandwidth, and buffer cache) among all multimedia data.

Another way to classify multimedia file systems is to group the systems according to the supported multimedia data characteristics.

General file systems capable of handling multimedia data to a certain extend. E.g. *log-structured file systems* [6].

Multimedia file systems optimized for continuous multimedia data. E.g. CMFS [5], Tiger Shark [4].

Multimedia file systems handling mixed-media workloads (continuous and discrete multimedia data). E.g. Fellini [7], MMFS [8].

Fellini [7] supports storage and retrieval of continuous and discrete multimedia data. The system provides rate guarantees for active clients by using admission control to limit the number of concurrent active clients.

MMFS [8] handles interactive multimedia applications by extending the UNIX file system. MMFS has a two-dimensional file structure for single medium editing and multimedia playback: (1) a single medium strand abstraction and (2) a multimedia file construct which ties together multiple strands that belong logically together. MMFS uses application-specific information for performance optimization of interactive playback. This includes intelligent prefetching, state-based caching, prioritized real-time disk scheduling, and synchronized multi-stream retrieval.

Data Placement -

Data placement and disk scheduling are responsible for the actual values of seek time, rotation time, and transfer time, which are the 3 major components, which determine the disk efficiency. There are a few general data placement strategies for multimedia applications in which read operations dominate and only few non-concurrent write operations occur:

- *Scattered placement:* In scattered placement blocks are allocated at arbitrary places on the disk. Because of this sequential access to data will usually cause a large number of intra-file seeks and rotations resulting in high disk read times.
- *Contiguous placement:* all the data blocks of a file are successively stored on disk. Contiguous allocation will mostly result in better performance compared to scattered allocation. The problem with contiguous allocation is that it causes external fragmentation.
- *Locally contiguous placement:* the files are divided into multiple fragments. All blocks of a fragment are stored contiguously, but fragments can be scattered over 1-n disks.
- *Constrained placement:* this strategy restricts the average distance measured in tracks, between a finite sequence of blocks [9].

To optimize the write operations, *log-structured placement* has been developed to reduce disk seeks for write intensive applications [6]. When modifying blocks of data, log structured systems do not store modified blocks in their original positions. Instead, all writes for all streams are performed sequentially in a large contiguous free space. As a result instead of requiring a seek for each stream writing, only one seek is required prior to a number of write operations.

For systems managing multiple storage devices, there exist two possibilities of distributing data among disks [10].

- *Data striping:* to realize a larger logical sector, many physical sectors from multiple disks are accessed in parallel.
- *Data interleaving:* requests of a disk are handled independent of requests from other disks. All fragments of a request can be stored on 1-n disks [11].

Some of the existing multimedia file systems e.g. Tiger Shark [4], use striping techniques to interleave both continuous and non-continuous multimedia data across multiple disks. There are two factors crucially determining the performance of multi-disk systems [10].

Efficiency in using each disk. The amount of seek and rotation times should be reduced as much as possible in order to have more time available for data transfer.

Fairness in distributing the load over all disks.

These two factors largely depend on the data distribution strategy and the application characteristics. They are very important to achieve *synchronization*, which means the temporal relationship between different multimedia data streams [12].

Disk Scheduling –

Traditional disk scheduling algorithms for current file systems focused mainly on reducing seek times [13], e.g., *Shortest Seek Time First (SSTF)* or *SCAN*. SSTF has high response time variations and may result in starvation of certain requests. SCAN reduces the response time variations and optimizes seek times by serving the requests in an elevator like way.

However, disk-scheduling algorithms for multimedia data requests need to optimize, besides the traditional criteria, also other criteria special for multimedia data including QoS guarantees [12]. Following are a few multimedia disk-scheduling algorithms

- *EDF strategy* [14] serves the block request with the nearest deadline first. Strict EDF may cause low throughput and very high seek times. Thus, EDF is often adapted or combined with other disk scheduling strategies.
- *SCAN-EDF strategy* [15] combines the seek optimization of SCAN and the real-time guarantees of EDF. Requests are served accordingly to their deadline. If multiple requests have the same (or similar) deadline, SCAN is used to define the order to handle the requests.
- *Group Sweeping Strategy (GSS)* [16] optimizes disk arm movement by using a variation of SCAN handling the requests of continuous media streams into multiple groups. The groups are handled in a fixed order. Within a group, SCAN is used to determine time and order of request serving. To guarantee continuity of playout, a smoothing buffer is needed. GSS represents a trade-off between optimizations of buffer space and disk arm movement.

Conclusions-

Commodity operating systems do not presently support all the requirements of multimedia systems. New OS abstractions need to be developed to support a mix of applications with real-time and best effort requirements and to provide the necessary performance. Thus, management of all system resources, including processors, main memory, network, disk space, and disk I/O, is an important issue.

The management needed encompasses admission control, allocation and scheduling, accounting, and adaptation.

References: -

[1] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11): 832-843, November 1983.

[2] Venkat Rangan, P., et al. A testbed for managing digital video and audio storage. In *Proc. Of the USENIX summer conference, 1991.*

[3] Shenoy, P.J., Goyal, P., Vin, H.M.: Architectural Considerations for Next Generation File Systems.

[4] Haskin, R.L., Schmuck, F.B.: The Tiger Shark File System, Proc. of 41st IEEE Int. Conf.: Technologies for the Information Superhighway (COMPCON'96), Santa Clara, CA, USA, February 1996, pp. 226-231

[5] Ramakrishnan, K.K., Vaitzblit, L., Gray, C., Vahalia, U., Ting, D., Tzelnic, P., Glaser, S., Duso, W.: Operating System Support for a Video-on-Demand File Service, Proc. of 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93), Lancaster, U.K., 1993, pp. 216-227

[6] Wang, R.Y., Anderson, T.E., Patterson, D.A.: Virtual Log Based File Systems for a Programmable Disk, Proc. of 3rd USENIX Symp. On Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 29-43.

[7] Martin C., Narayanan, P.S., Özden, B., Rastogi, R., Silberschatz, A.: The Fellini Multimedia Storage Server, in: Chung, S.M. (Ed.): Multimedia Information and Storage Management, Kluwer Academic Publishers, 1996, pp. 117-146.

[8] Niranjan, T.N., Chiueh, T., Schloss, G.A.: Implementation and Evaluation of a Multimedia File System, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'97), Ottawa, Canada, June 1997.

[9] Anderson, D., Osawa, Y., Govindan, R.: A File System for Continuous Media, ACM Transactions on Computer Systems, Vol. 10, No. 4, November 1992, pp. 311-337.

[10] Garcia-Martinez, A., Fernadez-Conde, J., Vina, A.: Efficient Memory Management in VoD Servers, to appear in: Computer Communications, 2000.

[11] Abbott, C.: Efficient Editing of Digital Sound on Disk, Journal of Audio Engineering, Vol. 32, No. 6, June 1984, pp. 394-402.

[12] Steinmetz, R.: Analyzing the Multimedia Operating System, IEEE Multimedia, Vol. 2, No. 1, Spring 1995, pp. 68-84.

[13] Denning, P.J.: Effects of Scheduling on File Memory Operations, Proc. AFIPS Conf., April 1967, pp. 9-21.

[14] Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment, Journal of the ACM, Vol. 20, No. 1, January 1973, pp. 46-61.

[15] Reddy, A.L.N., Wyllie, J.: Disk Scheduling in a Multimedia I/O System, Proc. of 1st ACM Multimedia Conf. (ACM MM'93), Anaheim, CA, USA, August 1993, pp. 225-233.

Section 3

CPU Scheduling in Multimedia Systems

By: Arun Singal

Introduction:

Integration of discrete and continuous multimedia data command additional services from the existing operating system components. Emerging multimedia applications demand real-time delivery of information over computer networks. To provide such a time guarantee in a dynamic environment needs efficient admission test for data transmission requests. There is no meaning to an audio clip, which is played out of turn during a multimedia presentation, which hints that data has some deadlines to meet.

Operating systems like Unix and Windows NT have time-sharing algorithm which are designed to provide fair allocation of resources among users. Users are given equal importance and no guarantee is provided regarding the timely scheduling of the task, thus it fails to meet the requirements of multimedia data. Real time operating systems have addressed the problem of meeting deadlines and so most of the present scheduling algorithm for multimedia is some variation of the real time scheduling algorithm. Since multimedia data is spatio-temporal, lets first discuss the timing characteristics and deadline characteristics.

Temporal Characteristics:

A task is a schedulable system entity, corresponding to the notion of a thread or a process. The timing characteristics of a task in a real-time system specify when the computation may begin, for how long it may execute and the deadline for its completion [4]. The figure below shows the timing characteristics of a task.

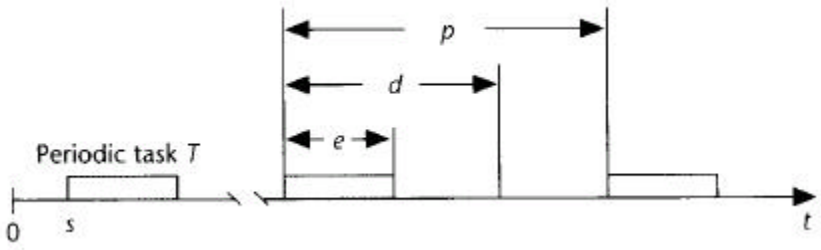

Fig.3.1 Source [1]

s: Indicates the starting point

e: Indicates the processing time

d: Indicates the deadline

p: Indicates the period

The start event for the computation may be periodic (so can be predicted) or it may be nonperiodic (but predictable) or may be unpredictable. The computation may be non preemptible (some other computation is schedules to process) or non preemptible (no other computation can remove this computation for the CPU). Multimedia streams are generally periodic in nature. For example video has to be sampled at a rate of 30 frames per sec for a good presentation.

Deadline Characteristics:

There are 2 types of deadlines. They may be either hard of soft.

A computation which is completed before the deadline is considered to be successful otherwise a failure is called to have a hard deadline. Traditional real-time scheduling techniques used for

command and control systems in application areas such as factory automation or aircraft piloting, demand high security and fault tolerance.

A computation, which gives some freedom i.e., missing a deadline does not result into a total failure of the computation, is called to have a soft deadline e.g. Multimedia systems outside of traditional real-time scenarios have more flexible real time constraints. The fault tolerance requirements of a multimedia system are usually less strict than those of real-time systems with a direct impact. A temporary failure of a continuous-media system, such as a delay in delivering video-on-demand, will not directly lead to the destruction of technical equipment or constitute a threat to human life. For many multimedia applications, missing a deadline is not a severe failure.

Fundamentals of Multimedia Scheduling:

The multimedia data processing has to occur in precisely predetermined, periodic intervals [1]. As discussed above the data is compared within a certain deadline. The task is to find a solution that allows all time critical, continuous streams to meet their deadlines. This must be guaranteed for all tasks in every period for the whole run time of the system. The crucial point is that the system has to make sure that both real time processes as well as other processes get fair amount of CPU time.

Tasks can be scheduled in a preemptive as well as nonpreemptible manner. It has been proved that non-preemptible continuous media can also be scheduled but is less favorable [1]. Moreover preemption usually gives greater benefits than nonpreemption in scheduling complexity.

Classification of scheduling techniques for Multimedia Systems:

Current general-purpose and real time operating system is insufficient for the desired quality of delivery of continuous media [2]. Current UNIX systems can only service a single audio or video stream assuming there is not other applications and that the continuous media stream is only piped though the system with moderate processing.

System-V and BSD-derived schedulers use a multilevel feedback queue, wherein the processes are server in a round-robin fashion within a priority level, but moved from one level to the other based on the CPU time accumulated and a preference for I/O intensive processes. Other processes in the system can intervene the kernel if they prove that they are more input output sensitive. Real-time Unix systems generally implement a fixed priority, preemptive schedulers and are thus not suited for general-purpose system shared by real-time, interactive and batch applications. By implementing a thread based fully preemptible kernel newer versions of Unix SVR4 and Mach OS are developed to reduce the problem of low priority processes from preventing the high priority processed from running. Or else another approach is proposed wherein they introduce preemption points. These points are milestone marks such as fork, exec or exit where after these checkpoints it is checked if any of the high priority process is waiting to run. UNIX SVR4 also introduced the concept of real time class with fixed priorities greater than those processes of the timesharing class. Any process in the real time class executed until it finishes or it may be blocked or every process may be associated with a time value. After the elapsed time the process would be forced to end. It is obvious that since the priority of the processes of the real time class is higher than the system processes it may have disastrous effects on the system performance. A high priority class [2] could only be successful if the high priority

processes don’t deprive the low priority processes from running. They should release the CPU so that low priority threads don’t just cast away. Nieh et.al [5] introduces a new concept of a time-sharing class, which has a better balance between interactive, real time and batch applications.

There are 2 most popular paradigms for resource allocation and scheduling.

1. Reservation-Based algorithms
2. Proportional share resource allocation

1. Reservation-Based algorithms:

A scheduling algorithm should guarantee that for any newly arrived packet it can find a schedule in the whole run time, so that the new task as well all the other tasks could finish their deadlines. In order to achieve this the scheduling algorithm should be able to check the schedulabililty of the current new task. As a performance metric, the algorithm could also use the processor utilization, which is the amount of processing time used by guaranteed tasks over the total amount of processing time.

In most cases the attempt to solving real-time scheduling is just a variation to two of the basic standard algorithms for multimedia systems: earlier deadline first and rate monotonic scheduling [1].

1.1 Earlier Deadline First:

The earlier deadline first is best suited for real time processing. As the name suggests it schedules the jobs according to its deadline. A job having the earliest deadline has the highest priority. At any arrival of a new task, EDF immediately computes a new order. If the job presently being scheduled needs to be preempted it is preempted. EDF is optimal, dynamic

algorithm. A dynamic scheduling algorithm has complete knowledge of currently active set of tasks but new arrivals may occur in the future not known to the algorithm at the time it is scheduling the current set. The schedule therefore changes over time. In the worst case the priorities of all the processes have to be rearranged and that can cause considerable overhead.

An extension of EDF is a time driven scheduler (TDS). TDS schedules tasks by deadlines and not by priorities. TDS handles the overload situation by discarding all those processes that cannot meet the deadline. Even then if the situation of overloading continues then it deletes the processes with a low value density, which means processes with low importance.

Another extension of priority driven EDF scheduling algorithm divides every task into 2 parts. The mandatory part and the optional part. The deadlines of the mandatory part decide the scheduling of the tasks. For a set of tasks their respective mandatory parts are checked for the deadline condition. If it is satisfied then the set can be scheduled. If the deadline of the mandatory part is ended then the task is forced to end even if the task is not completed. The optional part is processed if the resource capacity is underutilized.

This method can easily be combined with the encoding of data according to their importance [1]. E.g. suppose a pixel of a BMP format picture is each represented by 16 bits. Then the processing of 8 bits can be made mandatory whereas the last 8 as optional.

The advantage using this method is that more processes can be scheduled concurrently. In case of an overload the optional part is aborted according to QoS (quality of service) requirements.

1.2 Rate Monotonic Algorithm:

The Rate Monotonic Algorithm was introduced by Liu and Layland, which is a static algorithm, applied to real time systems. It guarantees that a certain amount of computation time will be

available for a reserve period of time with a delay bound equal to the length of the period. It assigns static priorities to tasks at the connection set up phase according to their request rates. Subsequently each task is processed with the priority at the beginning. It is an optimal, static, priority-driven algorithm for preemptive, periodic jobs. It is called optimal because there is no other scheduling algorithm, which can schedule tasks, which rate monotonic algorithm cannot. The proof of rate monotonic algorithm stating that all of the tasks would successfully meet their deadlines and compute at the associated rates if

$$\sum_{i=1}^{n} C_i / T_i <= n(2^{1/n} - 1) \qquad \text{[from 1]}$$

Where C is the computation time and T is the period.

There are a few assumptions to rate monotonic algorithm [1]:

1. The requests for all tasks with deadlines are periodic.
2. The processing of a single task must finish before the processing of the next task in the same data stream.
3. All tasks are independent. The request of one task does not depend on the initiation or completion of requests for any other task.
4. Runtime for each request of a maximum time a processor requires to execute the task without interruption-is constant.
5. Any nonperiodic task in the system has no required deadline. Typically such tasks initiate periodic or failure recovery tasks. They usually displace periodic tasks.

Ongoing research claims that it is not necessary to make all these assumptions to employ monotonic algorithm.

The extension of rate monotonic is called deadline monotonic. The difference here from rate monotonic is that the in deadline monotonic there is an additional parameter called the deadline which specifies the duration of time by which the computation released at the beginning of the period must be completed.

1.3 Least-laxity-first algorithm:

This algorithm schedules the task with the shortest remaining laxity-the time between the current time and the deadline minus the remaining processing time [6]. Since laxity is a function of deadline, processing time and current time and since the processing time cannot be specified in advance, the calculation of laxity assumes the worst case. This makes the algorithm inexact.

This algorithm is optimal for exclusive resources. It is also optimal for multiple resources if the ready times of the real-time tasks are the same.

2 Proportional share resource allocation:

Demands on the available resources are made by specifying the relative share (weight) of the resource. In a dynamic system where tasks join and leave in real time a share depends both on the current system state and the current time. A resource is then allocated based on the shares of the competing requests. Proportional share resource allocation is somewhat similar to packet switched networks and hence implemented using algorithms scheduled for packet switched networks. A list of various such algorithms is weighted fair queuing, virtual clock, and packet-by-packet generalized processor sharing [7].

Researchers have come to a conclusion that CPU scheduling for multimedia systems should be based on a combination of proportional share allocation and reservation based algorithms. In

addition to the combination the following criteria should also be considered while classifying CPU scheduling approaches

1. Whether admission control is performed

2. Whether adaptation is supported.

3. Whether a new abstraction for resource principle is introduced.

4. What is the context of the scheduler i.e. is it part of a new kernel, integrated in an existing kernel, or implemented on top of an existing kernel in user-space.

Let's discuss the scheduler in the Rialto system, which combines EDF and round robin scheduling [7] and the SMART scheduler, which uses proportional share CPU allocation.

Rialto scheduler:

The Rialto scheduler is implemented on the Rialto OS. It is based on the simple abstractions:

1. Task as defined before is a program or an application, which comprises of multiple threads of control. When requested the resources are allocated to these tasks. After the resources are freed the usage of resources is charged to these tasks.
2. Tasks request for CPU reservation.
3. Time constraints are dynamic requests from threads to the scheduler to run a certain code segment within a specified start time and deadline to completion.

To decide on which thread to activate the scheduler refers to a scheduling graph. The scheduler graph is dynamically recomputed for each iteration. The scheduling graph is basically a graph of a set of nodes, which represent activity with a CPU reservation.

For each period of CPU reservation the scheduler traverses the graph in the depth-first manner, but backtracks always to the root after visiting a leaf in the tree. Each node, i.e. activity that is

crossed during the traversal is marked for scheduling that CPU period. Threads cannot request for time constraints during blocking except for short blocking intervals for synchronization and I/O. In normal situations when a thread request for time constraints then the scheduler analysis the situation and see if it is possible by checking the time interval assignment data structure. This data structure is in correspondence to the scheduling graph and checks whether enough free computation time is available between start time and the deadline. When during the course of a scheduling graph traversal an interval assignment record for the current time is encountered, a thread with an active time constraint is selected according to EDF. Otherwise, threads of a task are scheduled according to round robin.

SMART Scheduler:

SMART scheduler works on the idea of differentiating the importance to determine the overall resource allocation for each task and the urgency to determine when each task is given it's allocation [7]. The scheduler distinguishes the importance and the urgency characteristics. It identifies all tasks that are important enough to execute and collects them in a candidate set. Afterwards, it orders the candidate set according to urgency consideration. The candidate set is scheduled according to the best-effort real time scheduling algorithm. The algorithm finds the task with the earliest deadline that can be executed without violating deadlines of task with higher value-tuples. There is no admission control implemented in SMART. It can only enforce real-time behavior in underload situations.

Conclusions:

Even today there are huge concerns about scheduling in multimedia systems.

1.The scheduling algorithms for real time systems are good for multimedia system but perform poorly in the case of overloading.

2.New OS abstractions need to be implemented which can provide applications with real-time and best-effort requirements and to provide the necessary performance.

3.There is an immediate need to integrate real-time processing and non-real-time processing in the native system kernel.

References:

[1] Ralf Steinmetz, " Analyzing the Multimedia Operating System", IEEE Multimedia, 2, 1, ppp 86-84(Spring 1995).

[2] Schulzrinne, H., "Operating System Issues for Continuous Media," Multimedia Systems, vol.4, pp.269-280, Oct.1996.

[4] Clifford W.Mercer, "Operating System Resource Reservation for Real-Time and Multimedia Applications", PHD thesis, School of Computer Science Carnegie Mellon University, June 1997.

[5] J.Nieh, J.G.Hanko, J.D.Northcutt, and G.A.Wall, "SV4 UNIX scheduler unacceptable for multimedia applications," in Proceedings of the 4th International Workshop on Network and Operating Sytem Support for Digital Audio and Video, (Lancaster , U.K.), pp.35-47, Lancaster University, Nov. 1993.

[6] D.W. Craig and C.M.Woodside, “The Rejection Rate for Tasks with Random Arrivals, Deadlines, and Preemptive Scheduling, “IEEE Trans. on Software Eng., Vol.16, No.10, Oct.1990, pp. 1,198-1,208.

[7] T.Plagemann, V.Goebel, P.Halvorsen, O.Anshus, “ Operating System support for multimedia systems”, Computer communications,23,3,pp 267-289(2000).