[Stei95b]. Ralf Steinmetz; A Multimedia File System Survey: Approaches for Continuous Media Disk Scheduling; erscheint in Computer Communications, April 1995.

Multimedia file systems survey: approaches for continuous media disk scheduling

Ralf Steinmetz

We understand multimedia data processing as the handling of audio and video data together with traditional data like text and images. This multimedia data is to be stored with and by a multimedia file system which comprises one or more of the following three issues: (1) The file system can rely on various types of different physical storage devices; however, we usually encounter the same devices as in any other high performance computers; (2) the organization of files in a contiguous order and the data structuring with ropes and strands improves the throughput at the expense of additional management effort; (3) the main goal of traditional disk scheduling is to reduce the cost of seek operations, to achieve a high throughput, and to provide a fair disk access. In multimedia disk scheduling the main goal is to meet all deadlines of the time critical tasks. The buffer requirement should be kept low, and aperiodic requests should not starve, i.e. a balance between the time constraints and efficiency must be found. This paper presents a survey of these three issues, with the focus on disk scheduling. It shows how the traditional disk scheduling techniques 'first come first serve', 'shortest seek time first', SCAN and C-SCAN are enhanced or substituted by EDF, SCAN-EDF, 'group sweeping scheduling', a 'mixed strategy' and a 'continuous media file system' approach.

Keywords: multimedia file systems, continuous media, disk scheduling

The operating system is the shield of the computer hardware to all other software components. It provides a comfortable environment for the execution of programs, and it ensures an effective utilization of the computer hardware. The operating system offers various services related to the essential resources of a computer: CPU, main memory, storage and all input and output devices. In this paper we focus on storage, i.e. file system aspects.

IBM European Networking Center, Creative Multimedia Studios, Vangerowstrae 18, 69115 Heidelberg, Germany (Email: steinmetz@vnet.ibm.com)

The file system is said to be the most visible part of an operating system. Most programs write or read on files – their program code as well as user data stored in files. The organization of the file system is an important factor for the usability and convenience of the operating system. A file is a sequence of information held as a unit for storage and use in computer systems[1]. Files are stored in secondary storage, and they can be used by different applications. The lifespan of files is usually longer than the execution of a program. In traditional file systems, the information types stored in files are sources, objects, libraries and executables of programs, numeric data, text, payroll records, etc.[2].

In multimedia systems, the stored information also covers digitized video and audio as being the continuous media data. As a prerequisite for the archiving and retrieval of this type of media data, the systems need to cope with the respective bandwidth demands and they must have sufficient storage capacity[3,4]. According to, for example, Doganata and Tantawy[5], today we can cope with these demands: We distinguish between:

- *expanded storage* (these are Random Access Modules – the bandwidth and capacity is limited by the surrounding system components. Typical values are a bandwidth of 100 Mbyte/s – 1 Gbyte/s and storage capacities up to 100 Gbyte);
- *disk storage* (with one disk of such a disk subsystem holding 0.5 – 2 Gbyte and a read throughput of each arm of 1–4 Mbyte/s); and
- *tape libraries* (which can hold 1–15 Tbyte of media data and have a limited bandwidth of a few Mbyte/s). In the following we will concentrate on disk storage, as this is the most widely used media for persistent storage in computing systems. There we encounter work on distributed hierarchical storage, including disk arrays[6]. The basic idea is to improve the throughput and capacity by storing data of each

audio and video file on several volumes (disk I/O bandwidth is maximized by striping; seek times are minimized by grouping and sorting). In multimedia file systems we do also have real-time 'read' and 'write' demands. Therefore, additional requirements in the design and implementation of file systems have to be considered.

Multimedia applications demand the processing of audio and video data such that humans perceive these media in a natural–error free and non-artificial–way. This continuous media data has its origin in sources like microphones, cameras and files. From these sources the data is transferred to destinations like loudspeakers, video windows and files located at the same computer or at a remote station. On the way from the source to the sink, the digital data is processed by at least some type of move, copy or transmit operation. Therefore, in this data manipulation process there are always many resources which are under the control of the operating system[7]. Files are such resources. The integration of discrete and continuous multimedia data provides a need for additional services. The main point in this context is the real-time processing of continuous media data. The process management must take into account the timing requirements imposed by the handling of multimedia data. Appropriate scheduling methods should thus be applied[3, 4].

The file system provides access and control functions for the storage and retrieval of files. From the user's viewpoint, the organization and structure allowed is important. The internals, which are more important in our context, i.e. the organization of the file system deals with the representation of information in files, their structure and organization in secondary storage. Because of the timing requirements of multimedia data, disk scheduling is essential.

The next section starts with a brief characterization of traditional file systems and disk scheduling algorithms. Subsequently, different approaches to organize multimedia files and disk scheduling algorithms for the use in multimedia systems are discussed.

TRADITIONAL FILE SYSTEMS

The two main goals of traditional files systems are to provide a comfortable interface for file access to the user, and to make efficient use of storage media. Whereas the first goal is still an area of interest for research (e.g. indexing for file systems[8] and intelligent file systems for the content-based associative access to file system data[9]), the structure, organization and access of data stored on disk have been extensively discussed and investigated over the last few decades. To understand the specific multimedia developments in this area, this section gives a brief overview on files, file system organizations and file access mechanisms. Later, disk scheduling algorithms for file retrieval are discussed.

File structure

We commonly distinguish between two methods of file organization: sequential and non-sequential.

In *sequential storage*, each file is organized as a simple sequence of bytes or records. Files are stored consecutively on the secondary storage media, as shown in *Figure 1*. They are separated from each other by a well defined 'end of file' bit pattern, character or character sequence. A file descriptor is usually placed at the beginning of the file, and is, in some systems, repeated at the end of the file. Sequential storage is the only possible way in which to organize the storage on tape, but it can also be used on disks. The main advantage is its efficiency for sequential access, as well as for direct access[1]. Disk access time for reading and writing is minimized.

Additionally, for a further improvement of performance with caching, the file can be read ahead of the user program[10]. In systems where file creation, deletion and size modification occur frequently, sequential storage has major disadvantages. Secondary storage is split, fragmented, through creation and deletion operations, and files cannot be extended without copying all of the files into a larger space. It is possible to copy the files such that all of them are located adjacently, i.e. without any 'holes' between them.

In *non-sequential storage* the data items are stored in a non-contiguous order. Two main approaches exist:

- One way is to use linked blocks, where physical blocks, containing consecutive logical locations, are linked using pointers. The file descriptor must contain the number of blocks occupied by the file, the pointer to the first block, and it may also have the pointer to the last block. A serious disadvantage of this method is the cost of implementation for random access, because all prior data has to be read. In MS-DOS a similar method is applied. A file allocation table (FAT) is associated with each disk. One entry in the table represents one disk block. The directory entry of each file holds the block number of the first block. The number in the slot of an entry refers to the next block of a file. The slot of the last block of a file contains an end-of-file mark[11].
- Another approach is to store block information in mapping tables. Each file is associated with a table where, apart from the block numbers, information

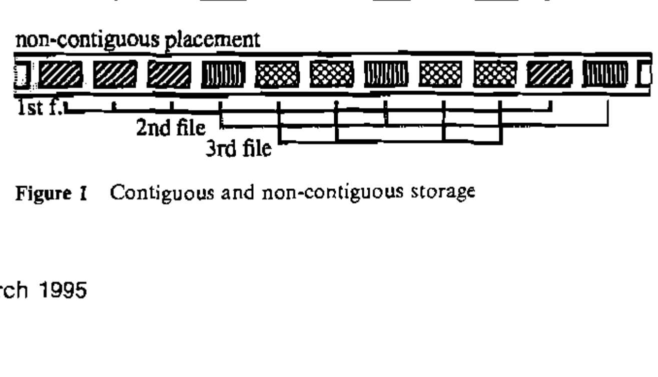

Figure 1 Contiguous and non-contiguous storage

like owner, file size, creation time, last access time, etc. are stored. Those tables usually have a fixed size, which means that the number of block references is bounded. Files with more blocks are referenced indirectly by additional tables assigned to the files. In UNIX, with each file a small table (on disk) called an i-node is associated (see *Figure 2*). The indexed sequential approach is an example of multi-level mapping; where logical and physical organization is not clearly separated[1].

Directory structure

Files are usually organized in directories. Most of today's operating systems provide tree structured directories, where the user can organize the files according to his personal needs. In multimedia systems, it is important to organize files in a way that allows an easy, fast and contiguous data access.

Disk management

Disk access is a slow and costly transaction. In traditional systems, a common technique used to reduce disk access times is block caching. Using a block cache, blocks are kept in memory because it is expected that future read or write operations will access this data again. Thus, performance is increased due to the shorter access time. Another way in which to increase performance is to reduce disk arm motion. Blocks that are likely to be accessed in sequence are placed together on one cylinder. To refine this method, the rotational positioning can be taken into account. Consecutive blocks are placed on the same cylinder but in an interleaved way, as shown in *Figure 3*.

Another important issue is the placement of the mapping tables (e.g. I-nodes in UNIX) on the disk. If they are placed near the beginning of the disk then the distance between them and the blocks will be, on average, half the number of cylinders. To improve this, they can be placed in the middle of the disk, hence the

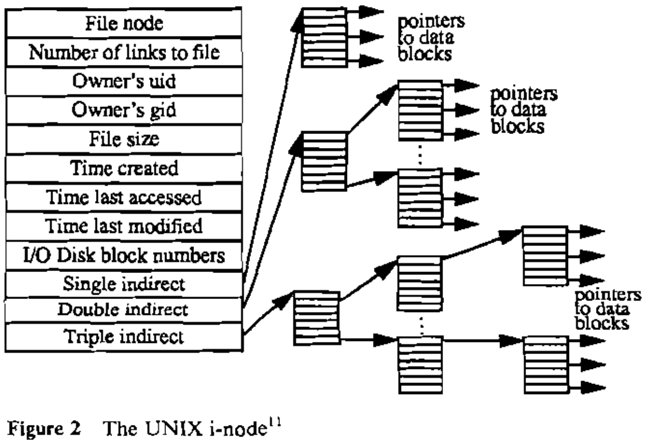

Figure 2 The UNIX i-node[11]

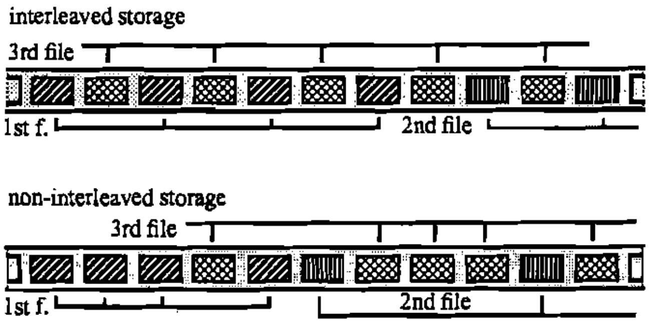

Figure 3 Interleaved and non-interleaved storage

average seek time is roughly reduced by a factor of two. In the same way, consecutive blocks should be placed on the same cylinder. The use of the same cylinder for storage of the mapping tables and the referred blocks also improves the performance.

Disk scheduling

Whereas strictly sequential storage devices (e.g. tapes) do not have a scheduling problem, for random access storage devices every file operation may require movements of the read/write head. This operation, known as 'to seek', is very time consuming, i.e. a seek time in the order of 200 ms for CDs is still state-of-the-art. The actual time taken to read or write a disk block is determined by:

- the seek time (the time required for the movement of the read/write head);
- the latency time or rotational delay (the time delay, during which the transfer cannot proceed, for the right block or sector to rotate under the read/write head);
- the actual data transfer time needed for the data copy from disk into main memory.

Usually, the seek time is the largest factor of the actual transfer time. Most systems try to keep the cost of seeking low by applying special algorithms for the scheduling of disk read/write operations. The access of the storage device is a problem greatly influenced by the file allocation method. For instance, a program, reading a contiguously allocated file, generates requests which are located close together on a disk, thus head movement is limited. Linked or indexed files with blocks, which are widely scattered, cause many head movements. In multiprogramming systems, where the disk queue may often be non-empty, fairness is also a criterion for scheduling. Most systems apply one of the following scheduling algorithms:

- *First-Come-First-Served (FCFS)*
  With this algorithm the disk driver accepts requests one at the time, and serves them in the incoming order. This is an easy to program – and an intrinsically fair – algorithm. However, it is not optimal with respect to head movements because it does not

consider the location of the other queued requests. This results in a high average seek time. *Figure 4* shows an example of the application of FCFS to a request of three queued blocks.

- *Shortest-Seek-Time First (SSTF)*
  At every point in time when a data transfer is requested, SSTF selects from all the requests that one with the minimum seek time from the current head position. Therefore the head is moved to the closest track in the request queue. This algorithm was developed to minimize seek time, and in this sense it is optimal. SSTF is a modification of Shortest Job First (SJF), and like SJF, it may cause starvation of some requests. Request targets in the middle of the disk will get immediate service at the expense of requests in the innermost and outermost disk areas. *Figure 5* demonstrates the operation of the SSTF algorithm.
- *SCAN*
  Like SSTF, scan orders the requests to minimize seek time. In contrast to SSTF, it takes the direction of the current disk movement into account. It first serves all requests in one direction until it no longer has any requests in this direction. The head movement is then reversed and service is continued. SCAN provides a very good seek time, because the edge tracks get better service times. Note that middle tracks still get a better service than edge tracks. When the head movement is reversed, it first serves tracks that are near the most recently serviced tracks. *Figure 6* shows an example of the SCAN algorithm.
- *C-SCAN*
  C-SCAN also moves the head in one direction, but it offers a fairer service with more uniform waiting times. It does not alter the direction as in SCAN, but instead scans in cycles, always increasing or decreasing, with one idle head movement from one edge to the other between two consecutive scans. The performance of C-SCAN is somewhat less than that of SCAN. *Figure 7* shows the operation of the C-SCAN algorithm.

Traditional file systems are not designed for employment in multimedia systems. They do not, for example, consider requirements like real-time, which are important to the retrieval of stored audio and video. To serve this requirement, new policies in the structure and organization of files, and in the retrieval of data from the disk, have to be applied. The next section outlines the most important developments in this area.

MULTIMEDIA FILE SYSTEMS

The storage subsystem is a major component of any information system. Due to the immense storage space

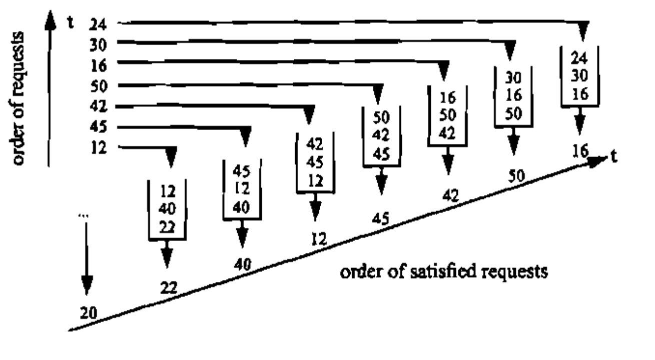

Figure 4 FCFS disk scheduling

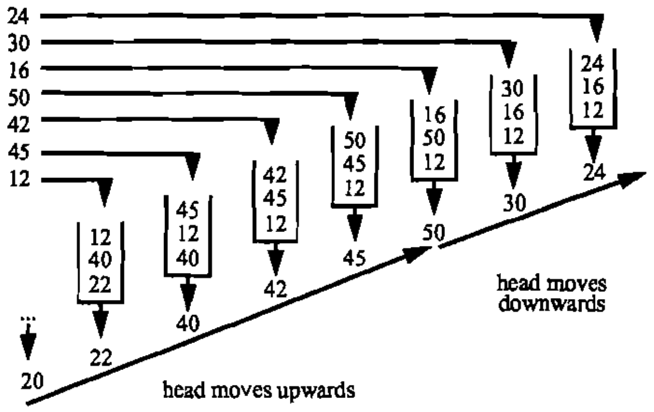

Figure 6 SCAN disk scheduling

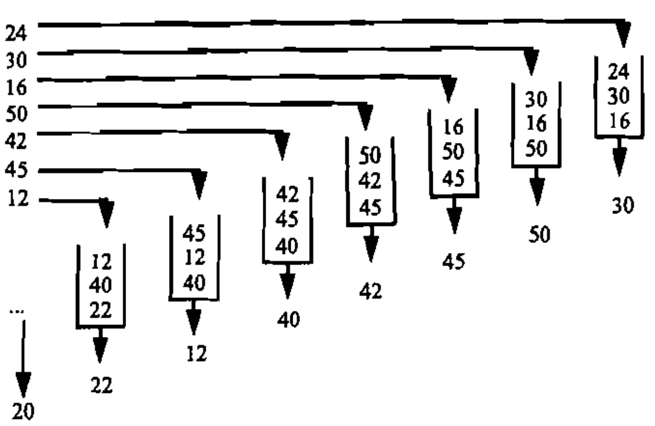

Figure 5 SSTF disk scheduling

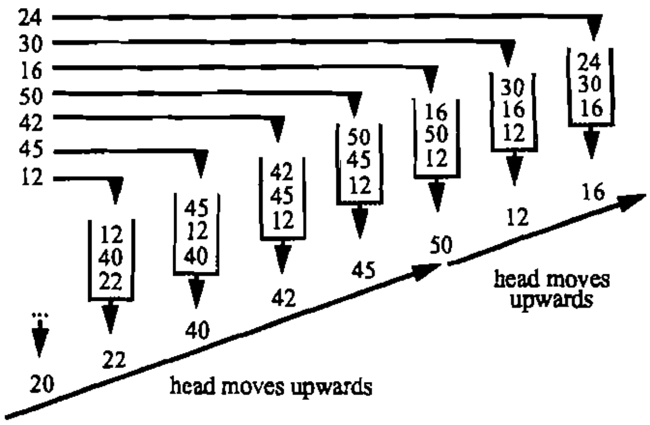

Figure 7 C-SCAN disk scheduling

requirements of continuous media, conventional magnetic storage devices are often no longer sufficient. Tapes, still in use in some traditional systems, are inadequate for multimedia systems because they cannot provide independent accessible streams, and random access is slow and expensive.

Apart from common disks with a large capacity, some multimedia applications, such as kiosk systems, use CD-ROMs to store data. In general, disks can be characterized in two different ways:

1. How information is stored on them. There are rewriteable (magnetic and optical) disks, write-once (WORM) disks and read-only disks like CD-ROMs.
2. Method of recording. The distinction is between magnetic and optical disks. The main difference is the access time and their track capacity. The seek time on magnetic disks is typically above 10 ms, whereas on optical disks 200 ms is a common lower bound. Magnetic disks have a constant rotation speed (constant angular velocity, CAV). Thus, while the density varies, the storage capacity is the same on the inner and outer tracks. Optical disks have a varying rotation speed (constant linear velocity, CLV), and hence the storage density is the same on the whole disk.

Therefore different algorithms for magnetic and optical disks are necessary. File systems on CD-ROMs are defined by ISO 9660. It is considered to be closely related to CD-ROMs and CD-ROM-XA. Very few variations are possible, thus we will focus the description on algorithms applicable to magnetic storage devices.

Compared to the increased performance of processors and networks, storage devices have become only marginally 'faster'[12]. The effect of this increasing speed mismatch is the search for new storage structures, access and retrieval mechanisms with respect to the file system. Continuous media data is different to discrete data in:

- real-time characteristics: as mentioned above, the retrieval, computation and presentation of continuous media is time-dependent. The data has to be presented (read) before a well defined deadline with only a small jitter. Thus, algorithms for the storage and retrieval of such data have to consider time constraints, and additional buffers to smooth the data stream have to be provided.
- file size: compared to text and graphics, video and audio have very large storage space requirements. Since the file system has to store information ranging from small unstructured units like text files to large, highly structured data units like video and associated audio, it has to organize the data on disk in a way that efficiently uses the limited storage. For example, the storage requirement of uncompressed CD quality stereo audio is 1.4 Mbit/s. Low but acceptable quality compressed video still requires about 1 Mbit/s using, for example, MPEG-1.
- multiple data streams: a multimedia system has to support different media at the same time. It not only has to ensure that each medium gets a sufficient share of the resources, but it also has to consider the tight relationships between different streams arriving from different sources. The retrieval of a movie, for example, requires the processing and synchronization of audio and video.

There are two different approaches to supporting continuous media in file systems. With the first approach, the organization of files on disk remains as it is. The necessary real-time support is provided through special disk scheduling algorithms and sufficient buffers to avoid jitter. In the second approach, the organization of audio and video files on disk is optimized for their use in multimedia systems. Supporting continuous media in file systems still remains an issue of research, especially the scheduling of multiple data streams.

In the next section, the different approaches to the organization of files on disks employed in multimedia systems are outlined. Subsequently, different disk scheduling algorithms for the retrieval of continuous media are presented.

FILE STRUCTURE AND PLACEMENT ON DISK

Whereas, in conventional file systems, the main goal of the file organization is to make efficient use of the storage capacity (i.e. to reduce internal and external fragmentation) and to allow arbitrary deletion and extension of files, in multimedia systems the main goal is to provide a constant and timely retrieval of data. Internal fragmentation occurs when blocks of data are not entirely filled. On average, the last block of a file is only half utilized. The use of large blocks leads to a larger waste of storage due to this internal fragmentation. External fragmentation mainly occurs when files are stored in a contiguous way. After the deletion of a file the gap can only be filled by a file of the same or of a smaller size. Therefore, there are usually small fractions between two files that are not used; storage space for continuous media is wasted.

As mentioned above, the goals for multimedia file systems can be achieved by providing enough buffer for each data stream and through the employment of disk scheduling algorithms that are especially optimized for the real-time storage and retrieval of data. The advantage of this approach (where data blocks of single files are scattered) is flexibility. External fragmentation is avoided, and the same data can be used by several streams (via references). Even when only using one stream this might be of advantage. For instance, it is possible to access one block twice, e.g. when a phrase in a sonata is repeated. However, due to the large seek operations during playback, even with optimized disk scheduling, large buffers have to be provided to smooth jitter at the data retrieval phase. Therefore, there are also large initial delays in advance of the actual retrieval of continuous media data.

Another problem in this context is the restricted transfer rate. With upcoming disk arrays, which might have 100 or more parallel heads, the projected seek and latency times are less than 10 ms, and a block size of 4 Kbytes at a transfer rate of 0.32 Gbit/s will be achieved. But this is not, for example, sufficient for the simultaneous retrieval of four or more production level MPEG-2 videos compressed in HDTV-quality that may require transfer rates of up to 100 Mbit/s[13, 14].

Approaches which use a specific disk layout take the specialized nature of continuous media data into account to minimize the cost of retrieving and storing streams. The much greater size of continuous media files and the fact that they will usually be retrieved sequentially, because of the nature of operation performed on them (such as play, pause, fast forward, etc.), are reasons for an optimization of the disk layout. The author's multimedia application related experience has shown that continuous media streams predominantly belong to the write-once-read-many nature, and streams that are recorded at the same time are likely to be played back at the same time (e.g. audio and video of a movie)[15]. Hence, it seems to be reasonable to store continuous media data in large data blocks contiguously on disk. Files that are likely to be retrieved together are grouped together on the disk. Thus, interference due to concurrent access of these files is minimized. With such a disk layout, the buffer requirements and the seek times decrease.

The disadvantage of the contiguous approach is external fragmentation and copying overhead during insertion and deletion. To avoid this, without scattering blocks in a random manner over the disk, a multimedia file system can provide constrained block allocation of the continuous media. Different placement strategies have been compared elsewhere[16]. The size of the blocks (M) and the size of the gaps (G) between them can be derived from the requirement of continuity. The size is measured in terms of sectors. We assume that the data transfer rate r_{dt} is the same as the disk rotation rate (sectors/s). The continuity requirement in this case is met if the time to skip over a gap and to read the next media block does not exceed the duration of its playback time $T_{\mathrm{play}}(\mathrm{s})$[17].

$$T_{\mathrm{play}}(s) \geqslant \frac{M(\mathrm{sectors}) + G(\mathrm{sectors})}{r_{\mathrm{dt}}(\mathrm{sectors/s})}$$

Since there are two variables in the equation, the storage pattern (M, G) is not unique. There are several combinations possible to satisfy the above equation. Problems occur if the disk is not sufficiently empty, so that single data streams cannot be stored exactly according to their storage pattern. In this case the continuity requirements for each block are not strictly maintained. To serve the continuity requirements, read-ahead and buffering of a determined number of blocks has to be introduced (e.g. see elsewhere for a detailed description of this storage method)[17–19].

Some systems using scattered storage make use of a special disk space allocation mechanism to allow fast and efficient access. Abbott[20] performed the pioneer work in this field. He was especially concerned about the size of single blocks and their positions on disk. Another topic to be considered is the placement of different streams. With interleaved placement the nth blocks of each stream are in close physical proximity on disk. A contiguous interleaved placement is possible as well as a scattered interleaved placement. With interleaved data streams synchronization is much easier to handle. On the other hand, the insertion and deletion of single parts of data streams become more complicated.

A layout algorithm has been developed and analysed which provides a uniform distribution of media blocks on the disk after copying or writing audio and video files[21]. It takes into account the fact that further files will be merged. Therefore, a set of non-filled gaps is left. This uniform distribution is achieved by storing new blocks at the centre of existing – so far – non-filled gaps. With this 'central merging method' gaps are successively split into two new equal gaps. It was shown that the mean efficiency of the secondary storage usage with this algorithm is about 75% without violation of any real-time constraint.

DISK SCHEDULING ALGORITHMS

The main goal of traditional disk scheduling algorithms is to reduce the cost of seek operations, to achieve a high throughput, and to provide fair disk access for every process. The additional real-time requirements, introduced by multimedia systems, make traditional disk scheduling algorithms, as described before, inconvenient for multimedia systems. Systems without any optimized disk layout for the storage of continuous media depend far more upon reliable and efficient disk scheduling algorithms than others. In the case of contiguous storage, scheduling is only needed to serve requests from multiple streams concurrently. A round-robin scheduler is employed in Lougher and Shepherd[15] that is able to serve hard real-time tasks. Here, additional optimization is provided through the close physical placement of streams that are likely to be accessed together.

The overall goal of disk scheduling in multimedia systems is to meet the deadlines of all time-critical tasks. The goal of keeping the necessary buffer space requirements low is closely related. As many streams as possible should be served concurrently, but aperiodic requests should also be schedulable without delaying their service for an infinite amount of time. The scheduling algorithm has to find a balance between time constraints and efficiency.

Earliest Deadline First

Let us first look at the EDF (Earliest Deadline First) scheduling strategy, as known from CPU scheduling, but used for the file system issue as well. The EDF algorithm is one of the best known algorithms for real-time processing. At every new ready state, the scheduler selects from the tasks that are ready and not fully processed the one with the earliest deadline. The requested resource is assigned to the selected task. At the arrival of any new task, EDF must be computed immediately, heading to a new order, i.e. the running task is preempted and the new task is scheduled according to its deadline. The new task is processed immediately if its deadline is earlier than that of the interrupted task. The processing of the interrupted task is continued according to the EDF algorithm later on. EDF is not only an algorithm for periodic tasks, but also for tasks with arbitrary requests, deadlines and service execution times[22]. In this case, no guarantee about the processing of any task can be given.

In file systems the block of the stream with the nearest deadline would be read first. The employment of EDF as shown in *Figure 8* in the strict sense results in poor throughput and an excessive seek time; no buffer space is optimized. Further, as EDF is usually applied as a preemptive scheduling scheme, the costs for preemption of a task and scheduling of another task are considerable. The overhead caused by this is in the same order of magnitude as at least one disk seek. Hence EDF has to be adapted or combined with file system strategies.

SCAN-Earliest Deadline First

The SCAN-EDF strategy is a combination of the SCAN and the EDF mechanisms. The seek optimization of SCAN and the real-time guarantees of EDF are combined in the following way. As in EDF, the request with the earliest deadline is always served first. Among requests with the same deadline, the specific one, that is first according to the scan direction, is served first. This principle is repeated among the remaining requests until no request with this deadline is left.

Since the optimization only applies for requests with the same deadline, its efficiency depends upon how often it can be applied (i.e. how many requests have the same or a similar deadline). To increase this probability, the following sophisticated technique can be used: all requests have release times that are multiples of the period p. Hence, all requests have deadlines that are multiples of the period p. Therefore the requests can be grouped together and served accordingly. For requests with different data rate requirements, in addition to SCAN-EDF, the employment of a periodic fill policy is proposed[24] to let all requests have the same deadline. With this policy, all requests are served in cycles. In every cycle each request gets an amount of service time that is proportional to its required data rate. The cycle length is equal to the sum of the service times of all requests. Thus, in every cycle all requests can be given a deadline at the end of the cycle.

SCAN-EDF can easily be implemented, therefore EDF has to be modified slightly. If D_i is the deadline of task i and N_i is the track position, then the deadline can be modified to be $D_i + f(N_i)$. Thus the deadline is deferred. The function $f(\ )$ converts the track number of i into a small perturbation of the deadline as shown in the example of *Figure 9*. It has to be small enough so that $D_i + f(N_i) \geqslant D_j + f(N_j)$ holds for all $D_i \geqslant D_j$. For $f(N_i)$ the following function was proposed[23] (Note: this mapping works according to SCAN in most of the cases, but not always; *Figure 9* shows an example where, according to pure SCAN-EDF, the resulting series of block numbers would be ...12, 22, 45, 42, 40, 16..., with this function we get ...12, 22, 45, 40, 16, 42,..):

$$f(N_i) = \frac{N_i}{N_{\max}}$$

where $N_{\max}$ is the maximum track number on disk.

We enhanced this mechanism by proposing a more accurate perturbation of the deadline which takes into account the actual position of the head (N) and always leads to the exact SCAN-EDF results (e.g. ... 12, 22, 45, 42, 40, 16, ... in our example). This position is measured in terms of block numbers and the current direction of the head movement (see also *Figures 10* and *11*):

1. If the head moves toward $N_{\max}$, i.e. upwards (according to *Figure 10*), then:
   1A. for all blocks N_i located between the actual position N and $N_{\max}$ the perturbation of the deadline is:
   $$f(N_i) = \frac{N_i - N}{N_{\max}} \forall N_i \geqslant N$$
   1B. for all blocks N_i located between the actual position and the first block (no. 0):
   $$f(N_i) = \frac{N_{\max} - N_i}{N_{\max}} \forall N_i < N$$
2. If the head moves downwards (according to *Figure 11*) towards the first blocks then:
   2A. for all blocks located between the actual position and $N_{\max}$
   $$f(N_i) = \frac{N_i}{N_{\max}} \forall N_i > N$$

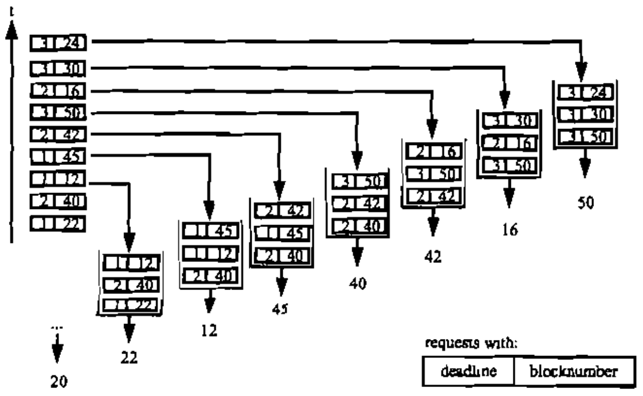

Figure 8 EDF disk scheduling

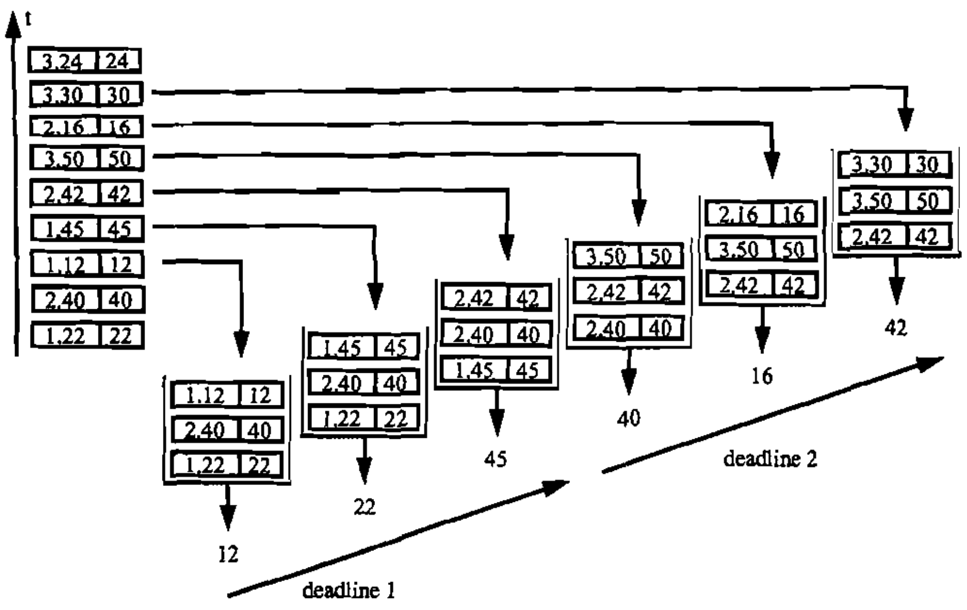

Figure 9 SCAN-EDF disk scheduling with $N_{max} = 100$ and $f(N_i) = N_i/N_{max}$

2B. for all blocks located between the first block (block number 0) and the actual position:

$$f(N_i) = \frac{N - N_i}{N_{max}} \forall N_i \leqslant N$$

Our algorithm is more computationally intensive than those with the simple calculation of Reddy and Wyllie[23]. In cases with only a few equal deadlines, our algorithm provides improvements and the expenses of the calculations can be tolerated. In situations with many, i.e. typically more than five) equal deadlines, the simple calculation provides sufficient optimization, and additional calculations should be avoided. Buffer requirements are not optimized, but they are lower than in the EDF scheduling technique; the throughput is also higher than applying the EDF method.

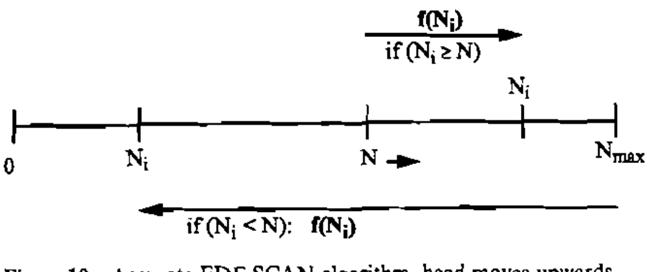

Figure 10 Accurate EDF-SCAN algorithm, head moves upwards

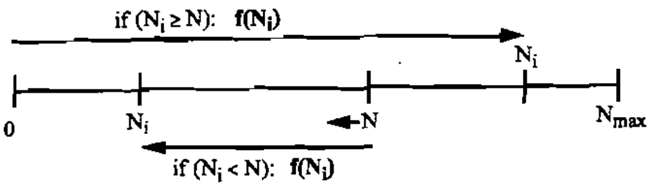

Figure 11 Accurate EDF-SCAN algorithm, head moves downwards

SCAN-EDF was compared with pure EDF and different variations of SCAN. It was shown that SCAN-EDF with deferred deadlines performs well in multimedia environments[23].

Group Sweeping Scheduling

With Group Sweeping Scheduling (GSS) requests are served in cycles, in a round-robin manner. To reduce disk arm movements, the set of *n* streams is divided into *g* groups. Groups are served in fixed order. Individual streams within a group are served according to SCAN; therefore it is not fixed at which time or order individual streams within a group are served. In one cycle a specific stream may be the first to be served, in another cycle it may be the last in the same group. A smoothing buffer which is sized according to the cycle time and data rate of the stream assures continuity. If the SCAN scheduling strategy is applied to all streams of a cycle without any grouping, then the playout of a stream cannot be started until the end of the cycle of the first retrieval at that stream (where all requests are served once) because the next service may be in the last slot of the following cycle. As the data must be buffered in GSS, the playout can be started at the end of the group in which the first retrieval takes place. Whereas SCAN requires buffers for all streams, in GSS the buffer can be reused for each group. Further optimizations of this scheme are proposed by Chen *et al.*[25]. This method ensures that each stream is served once in each cycle. GSS is a trade-off between the optimization of buffer space and arm movements. To provide the requested guarantees for continuous media data we propose here to introduce a 'joint deadline' mechanism: we assign to each group of streams one deadline, the 'joint deadline'.

This deadline is specified as being the earliest of the deadlines of all streams in the respective group. Streams are grouped such that all of them have similar deadlines. *Figure 12* shows an example of group sweeping scheduling.

Mixed strategy

In Abbott[20] a mixed strategy was introduced based on the shortest seek (also called the greedy strategy) and the balanced strategy. As shown in *Figure 13*, every time data is retrieved from disk it is transferred into buffer memory allocated for the respective data stream. From there the application process removes it piece by piece. The goal of the scheduling algorithm is:

- to maximize transfer efficiency by minimizing seek time and latency, and
- to serve process requirements with a limited amount of buffer space.

With shortest seek the first goal is served, i.e., the process whose data block is closest is served first. The balanced strategy chooses the process which has the least amount of buffered data for service, because this process is likely to run out of data. The crucial part of this algorithm is the decision between the two strategies to be applied (shortest seek or balanced strategy). For the employment of shortest seek two criteria must be fulfilled: the number of buffers for all processes should be balanced (i.e. all processes should have almost the same amount of buffered data) and the overall required bandwidth should be sufficient for the number of active processes, so that none of them will try to read data immediately out of an empty buffer. In Abbott[20] the *urgency* is introduced as an attempt to measure both. The urgency is the sum of the reciprocals of the current 'fullness' (amount of buffered data). This number measures both the relative balance of all read processes and the number of them. If the urgency is large then the balance strategy will be used, if it is small it is safe to apply the shortest seek algorithm.

Continuous Media File System

The CMFS Disk Scheduling is a non preemptive disk scheduling scheme designed for the continuous media File System (CMFS) at UC-Berkeley[26]. Different policies ean be applied in this scheme. Here the notion of the slack time H is introduced. The slack time is the time during which CMFS is free to do non-real-time operations or workahead for real-time processes, because the current workahead of each process is sufficient so that no process would starve even if it was not served for H seconds. The considered real-time scheduling policies are:

- The *Static/Minimal Policy* is based on the minimal workahead-augmenting set (WAS). A process p_i reads a file at a determined rate R_i. To each process a positive integer M_i is assigned which denotes the time overhead required to read a block covering, e.g., the seek time. The CMFS performs a set of operations (i.e. disk operations required by all processes) by seeking the next block of a file and reading M_i blocks of this file. Note: we consider only read operations; the same also holds with minor modifications for write operations. This seek is done for every process in the system. The data read by a process during this operation 'lasts':

  $$\frac{M_i \times A}{R_i}$$

  where A is the block size in bytes. The WAS is a set

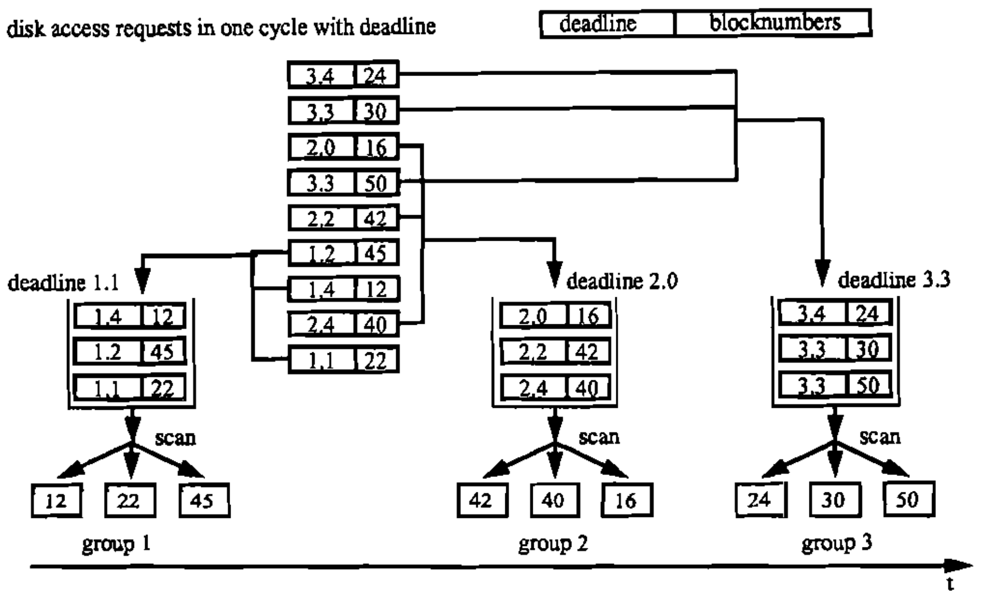

Figure 12 Group sweeping scheduling as disk access strategy

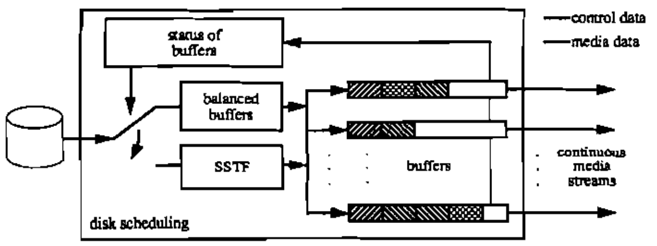

Figure 13 Mixed disk scheduling strategy

of operations where the data read for each process 'lasts longer' than the worst-case time to perform the operations (i.e. the sum of the read operations of all processes is less than the time read data lasts for a process). A schedule is derived from the set that is workahead-augmenting and feasible (i.e. the requests are served in the order given by the WAS). The 'Minimal Policy', the minimal WAS, is the schedule where the worst-case elapsed time needed to serve an operation set is the least (i.e. the set is ordered in a way that reduces time needed to perform the operations, e.g. by reducing seek times). The Minimal Policy does not consider buffer requirements. If there is not enough buffer this algorithm causes an buffer overflow. The 'Static Policy' modifies this schedule such that no block is read if this would cause a buffer overflow for that process. With this approach starvation is avoided but its use of short operations causes high seek overhead.

- With the *Greedy Policy* a process is served as long as possible. Therefore it computes at each iteration the slack time H. The process with the smallest workahead is served. The maximum number n of blocks for this process is read; n is determined by H (the time needed to read n blocks has to be less than or equal to H) and the currently available buffer space.
- The *Cyclical Plan Policy* distributes the slack time among processes in order to maximize the slack time. It calculates H and increases the minimal WAS with H milliseconds of additional reads; an additional read for each process is done immediately after the regular read determined by the minimal WAS. This policy distributes workahead by identifying the process with the smallest slack time and schedules an extra block for it; this is done until H is exhausted. The number of block reads for the least-workahead is determined. This procedure is repeated every time the read has completed.

The *Aggressive* version of the Greedy and the Cyclical Plan Policy calculates H of all processes except the least-workahead process that is immediately served by both policies. If the buffer size limit of a process is reached, all policies skip to the next process. Non-real-time operations are served if there is enough slack time. First performance measurements of the above introduced strategy showed that Cyclical Plan increases system slack faster at low values of the slack time (which is likely to be the case at system set up). With a higher system slack time, apart from the Static/Minimal Policy, all policies perform about the same.

All of the disk scheduling strategies described above have been implemented and tested in prototype file systems for continuous media. Their efficiency depends on the design of the entire file system, the disk layout, tightness of deadlines, and last but not least on the application behaving. Which algorithm is the 'best' method for the storage and retrieval of continuous media files is not yet common sense. Further research must show which algorithm serves the timing requirements of continuous media best and ensures that aperiodic and non-real-time requests are efficiently served.

DATA STRUCTURING

Continuous media data is characterized by consecutive, time dependent logical data units. The basic data unit of a motion video is a *frame*. The basic unit of audio is a sample. Frames contain the data associated with a single video image; a sample represents the amplitude of the analog audio signal at a given instance. Further structuring of multimedia data was suggested in the following way[18,27,28]: a *strand* is defined as an immutable sequence of continuously recorded video frames, audio samples, or both, i.e., it consists of a sequence of blocks which contain either video frames, audio samples or both. Most often it includes headers and further information related to the compression used. The file system holds primary indices in a sequence of 'Primary Blocks'. They contain mappings from media block numbers to their disk addresses. Pointers to all Primary Blocks are stored in 'Secondary Blocks'. The 'Header Block' contains pointers to all secondary blocks of a strand. General information about the strand like recording rate, length etc. is also included in the header block.

Media strands that together constitute a logical entity of information (e.g. video and associated audio of a movie) are tied together by synchronization to form a multimedia *rope*. A rope contains the name of its creator, its length and access rights. For each media strand in this rope the strand ID, the rate of recording, the granularity of storage, and the corresponding block-level is stored; this information is used for the synchronization of the playback start for all media at the strand interval boundaries. Editing operations on ropes manipulate pointers to strands only. Strands are regarded as immutable objects because editing operations like insert or delete may require substantial copying which can consume significant amount of time and space. Intervals of strands can be shared by different ropes. Strands that are not referenced by any rope can be deleted, and their storage is reclaimed[18]. The following interfaces are the operations that file systems provide for the manipulation of ropes:

- RECORD [media] → [requestID, mmRopeID]
  A multimedia rope, represented by mmRopeID and consisting of media strands, is recorded until a STOP operation is issued.
- PLAY [mmRopeID, interval, media] → requestID
  This operation plays a multimedia rope consisting of one or more media strands.
- STOP [requestID]
  This operation stops the retrieval or storage of the corresponding multimedia rope.
- Additionally the following operations are supported:
  INSERT [baseRope, position, media, withRope, withInterval]
  REPLACE [baseRope, media, baseInterval, withRope, withInterval]
  SUBSTRING [baseRope, media, interval]
  CONCATE [mmRopeID1, mmRopeID2]
  DELETE [baseRope, media, interval]

Figure 14 provides an example of the INSERT operation while *Figure 15* shows the REPLACE operation.

The storage system is divided into two layers:

- The *rope server* is responsible for the manipulation of multimedia ropes. It communicates with applications, allows the manipulation of ropes, and communicates with the underlying storage manager to record and play back multimedia strands. It provides the rope abstraction to the application. The rope access methods were designed to be similar to UNIX file access routines. Status messages about the state of the play or record operation are passed to the application.
- The *storage manager* is responsible for the manipulation of strands. It places the strands on disk to ensure continuous recording and playback. The interface to the rope server includes four primitives for manipulating strands:

1. 'PlayStrandSequence' takes a sequence of strand intervals and displays the given time interval of each strand in sequence.
2. 'RecordStrand' creates a new strand and records the continuous media data either for a given duration or until StopStrand is called.
3. 'StopStrand' terminates a previous PlayStrandSequence or RecordStrand instance.
4. 'DeleteStrand' removes a strand from storage.

The experimental Video File Server introduced by Rangan[27] supports integrated storage and retrieval of video. The 'Video Rope Server' presents a device independent directory interface to users (Video Rope). A Video Rope is characterized as a hierarchical directory structure constructed upon stored video frames. The 'Video Disk Manager' manages frame oriented motion video storage on disk, including audio and video components.

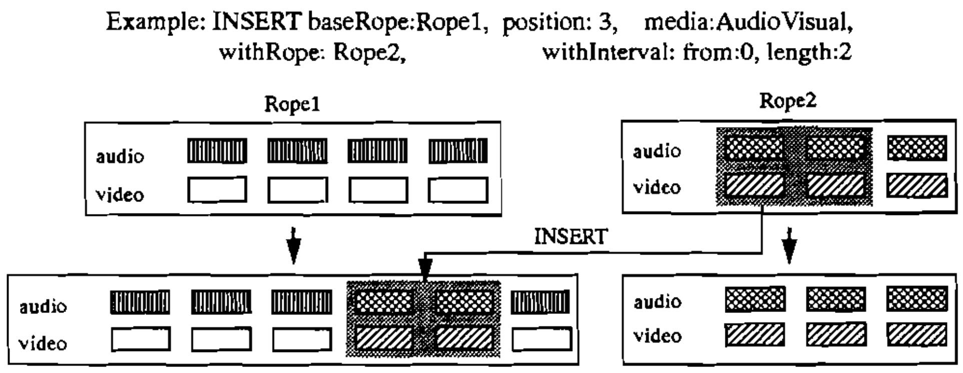
Example: INSERT baseRope:Rope1, position: 3, media:AudioVisual,
withRope: Rope2, withInterval: from:0, length:2
Rope1
Rope2
audio
video
INSERT

Figure 14 INSERT operation

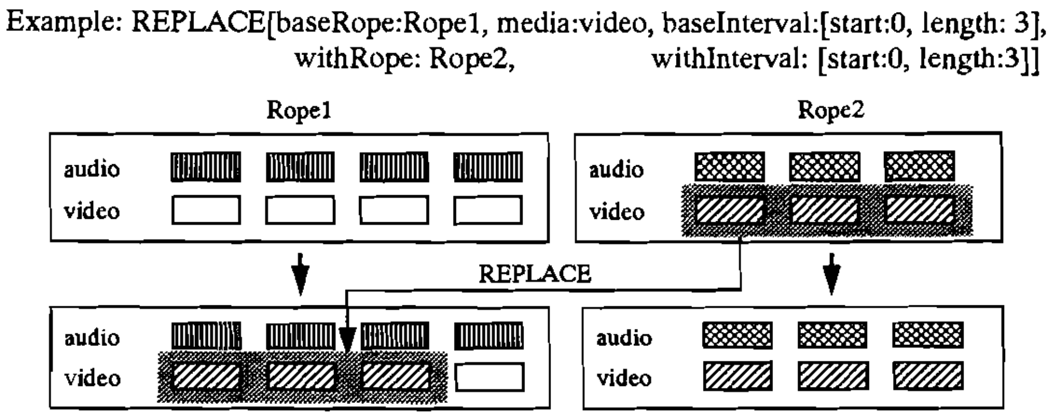
Example: REPLACE[baseRope:Rope1, media:video, baseInterval:[start:0, length: 3],
withRope: Rope2, withInterval: [start:0, length:3]]
Rope1
Rope2
audio
video
REPLACE

Figure 15 REPLACE operation

CONCLUSION

Multimedia data handling which comprises the media audio and video has mainly been addressed in the application, hypermedia, communication and operating system domain. The key issue - from the system's point of view - is the merging of real-time data stream manipulation with traditional data handling. Therefore several operating system extensions with this capability and respective file systems have been developed, are being developed, or are still a matter of research. A multimedia file system must preserve the required real-time demands and deliver or consume data streams. In this paper we extensively showed the form disk scheduling must take.

Unfortunately most existing multimedia file systems do either only make use of appropriate storage devices (because it is done by, e.g., storage manufacturers), or they only apply data structuring correctly (because it is done by, e.g., system programmers), or they only perform the disk scheduling according to the multimedia demands (because it is done by, e.g., system software developers). However only with the application of all these techniques together, multimedia filesystems will preserve real-time demands, keep the buffer management as well as end-to-end delays low, and handle many streams concurrently.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the many valuable advices, days and nights of discussions, and the specific work performed by Andreas Mauthe in this area. Marcus Beattle, Klara Nahrstedt, Lars Wolf and the anonymous reviewers suggested many improvements. Thank you.

REFERENCES

1 Krakowiak, S *Principles of Operating Systems*. MIT Press, Cambridge, MA (1988)

2 Peterson, J and Silberschatz, A *Operating System Concepts*. Addison-Wesley, Reading, MA (1983)

3 Steinmetz, R *Multimedia Technology: Fundamentals and Introduction* (in German), Springer-Verlag, Berlin (1993)

4 Steinmetz, R and Nahrstedt, K *The Fundamentals in Multimedia Systems*. Prentice-Hall, Englewood Cliffs, NJ (February 1995)

5 Doganata, Y N and Tantawy, A 'A cost/performance study of video servers with hierarchical storage', *IEEE Proc. Int. Conf. Multimedia Computing and Systems*, Boston, MA (May 1994) pp 393–402 (To appear in revised form as 'Cost effectiveness of video servers', *IEEE Multimedia Mag.*)

6 Tobagi, F A, Pang, J, Baird, R and Gang, M 'Streaming RAID - A disk array management system for video files', *Proc. Ist Int. ACM Conf. Multimedia*, Anaheim, CA (August 1993) pp 393–400

7 Mauthe, A, Schulz, 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, IBM Heidelberg (1992)

8 Salzer, J H 'File systems indexing and backup', *Operating Systems of the 90s and Beyond, Int. Workshop Dagstuhl Castle*, Germany (July 1991) pp 13–19

9 Gifford, D W and O'Toole, J W 'Intelligent file systems for object repositories', *Operating Systems of the 90s and Beyond, Int. Workshop*, Dagstuhl Castle, Germany (July 1991) pp 20–24

10 Janson, P A *Operating Systems, Structures and Mechanisms*. Academic Press, Orlando, FL (1985)

11 Tanenbaum, A S *Operating System, Design and Implementation*. Prentice-Hall, Englewood Cliffs, NJ (1987)

12 Mullender, S J 'Systems of the nineties - Distributed multimedia systems; systems of the 90s and beyond', *Int. Workshop Dagstuhl Castle*, Germany (July 1991)

13 Steinmetz, R 'Data compression in multimedia computing: principles and techniques', *Multimedia Systems*, Vol 1 No 4 (February 1994) pp 166–172

14 Steinmetz, R 'Data compression in multimedia computing: standards and systems', *Multimedia Systems*, Vol 1 No 5 (March 1994)

15 Lougher, P and Shepherd, D 'The design of a storage service for continuous media', *The Computer J*, Vol 36 No 1 (1993) pp 32–42

16 Gemmell, J and Christodoulakis, S 'Principles of delay sensitive multimedia data storage and retrieval', *ACM Trans. Infor. Syst.*, Vol 10 No 1 (January 1992)

17 Rangan, P V, Käppner, T and Vin, H W 'Techniques for efficient storage of digital video and audio', *Proc. Workshop on Multimedia Information Systems*, Tempe, AZ (February 1992)

18 Rangan, P V and Vin, H M 'Designing file systems for digital video and audio', *Proc. 13th ACM Symposium on Operating Systems Principles*, Monterey CA *Operating Systems Review*, Vol 25 No 5 (October 1991)

19 Vin, H M and Rangan, P V 'Techniques for efficient storage of digital video and audio', *Comput. Commun.*, Vol 16 (March 1993) pp 168–176

20 Abbott, C 'Efficient editing of digital sount on disk', *J. Audio Eng. Soc.*, Vol 32 No 6 (June 1984) pp 394–402

21 Karmouch, A, Wang, R and Yeap, T *Design and Analysis of a Storage Retrieval Model for Audio and Video Data*. Technical Report, Multimedia Information Systems, Department of Electrical Engineering, University of Ottawa, Canada (1994)

22 Dertouzos, M L 'Control robotics', *The Procedural Control of Physical Processing*. Information Processing 74, North Holland (1974) pp 807–813

23 Reddy, A L N and Wyllie, J 'Disk scheduling in a multimedia I/O system', *Proc. ACM Int. Conf. Multimedia*, Anaheim, CA (1993) pp 225–233

24 Yee, J and Vatarya, P V *Disk scheduling policies for Real-Time Multimedia Applications*. Technical Report, University of California, Berkeley (August 1992)

25 Chen, M-S, Kandlur, D D and Yu, P S 'Optimization of the group sweeping scheduling (GSS) with heterogeneous multimedia streams', *Proc. 1st ACM Int. Conf. Multimedia*, Anaheim, CA (1993) pp 235–241

26 Anderson, D P, Osawa, Y and Govindan, R *Real-Time Disk Storage and Retrieval of Digital Audio/Video Data*. Technical Report No. UCB/CSD 91/646, Computer Science Division, University of California, Berkeley (September 1991)

27 Rangan, P V 'Video conferencing, file storage, and management in multimedia computer systems', *Comput. Networks & ISDN Syst.* (March 1993)

28 Steinmetz, R and Fritzsche, C 'Abstractions for continuous media programming', *Comput. Commun.*, Vol 15 No 6 (July/August 1992)