Multimedia operating systems

- The operating system provides a comfortable environment for the execution of programs, and it ensures effective utilization of the computer hardware.
- The OS offers various services related to the essential resources of a computer: CPU, main memory, storage and all input and output devices.
- In multimedia applications, a lot of data manipulation (e.g. A/D, D/A and format conversion) is required and this involves a lot of data transfer, which consumes many resources.
- The integration of discrete and continuous multimedia data demands additional services from many operating system components.
- The major aspect in this context is *real-time processing* of continuous media data.

- Issues concerned:
  - Process management: a brief presentation of traditional real-time scheduling algorithms.
  - File systems: outlines disk access algorithms, data placement and structuring
  - Interprocess communication and synchronization

  - Memory management
  - Database management
  - Device management

- *Process management* must take into account the timing requirement imposed by the handling of multimedia data.
  - Concerns in process management (Scheduling):

	Traditional OS	MM OS
Timing requirement	No	Yes
Fairness	Yes	Yes

- Single components are conceived as resources that are reserved prior to execution to obey timing requirements and this *resource reservation* has to cover all resources on a data path.
- The *communication & synchronization between single processes* must meet the restrictions of real-time requirements and timing relations among different media.
- *Memory management* has to provide access to data with a guaranteed timing delay and efficient data manipulation functions. (e.g. should minimize physical data copy operations.)
- *Database management* should rely on file management services

9.2 Real-time

- A real-time process is a process which delivers the results of the processing in a given time-span.

- The main characteristic of real-time systems is the correctness of the computation.
  - Errorless computation
  - The time in which the result is presented

- Speed and efficiency are not the main characteristic of real-time systems.
  (e.g. the video data should be presented at the right time, neither too quickly nor too slowly)

- Timing and logical dependencies among different related tasks, processed at the same time, must also be considered.

Deadlines:

- A deadline represents the latest acceptable time for the presentation of a processing result.

- Soft deadline:
  - a deadline which cannot be exactly determined and which failing to meet does not produce an unacceptable result.

  - Its miss may be tolerated as long as (1) not too many deadlines are missed and/or (2) the deadlines are not missed by much.

- Hard deadline:
  - a deadline which should never be violated.
  - Its violation causes a system failure.
  - Determined by the physical characteristics of real-time processes.

Characteristics of real time systems

- The necessity of deterministic and predictable behavior of real-time systems requires processing guarantees for time-critical tasks.

- A real-time system is distinguished by the following features:
  - Predictably fast response to time-critical events and accurate timing information:
  - A high degree of schedulability: to meet the deadlines.
  - Stability under transient overload: critical task first.

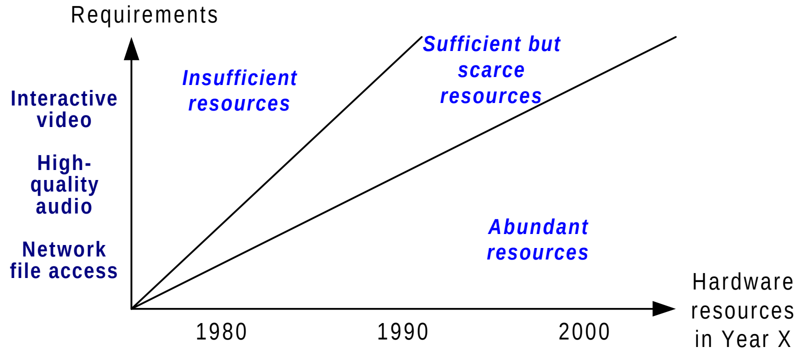
9.2.2 Real time and multimedia

- The real-time requirements of traditional real-time scheduling techniques usually have a high demand for security and fault-tolerance. (Most of them involve system control.)

- Real-time requirements of multimedia systems:
  - The fault-tolerance requirements of multimedia systems are usually less strict than those of real-time systems that have a direct physical impact.
  - For many multimedia system applications, missing a deadline is not a severe failure, although it should be avoided. (e.g. playing a video sequence)
  - In general, all time-critical operations are periodic and schedulability considerations for periodic tasks are much easier.
  - The bandwidth demand of continuous media is usually negotiable and the media is usually scalable.

9.3 Resource management

- Multimedia systems with integrated audio and video processing are at the limit of their capacity even with data compression and utilization of new technology. (Demand increases drastically.)



- No redundancy of resource capacity can be expected in the near future.
- In a multimedia system, the given timing guarantees for the processing of continuous media must be adhered to along the data path.
- The actual requirements depend on (1) the type of media and (2) the nature of the applications supported.

- The shortage of resources requires careful allocation.
- The resource is first allocated and then managed.

- At the connection establishment phase, the resource management ensures that the new 'connection;' does not violate performance guarantees already provided to existing connections.
- Applied to OS, resource management covers the CPU (including process management), memory management, the file system and the device management.
- The resource reservation is identical for all resources, whereas the management is different for each.

9.3.1 Resources

- A resource is a system entity required by tasks for manipulating data.
- A resource can be active or passive.
  - Active resource:
    - e.g. the CPU or a network adapter for protocol processing;
    - it provides a service.
  - Passive resource:
    - e.g. main memory, communication bandwidth or file systems;
    - It denotes some system capability required by active resources.

- A resource can be either used exclusively by one process at a time or shared between various processes.
- Active ones are often exclusive while passive ones can usually be shared.
- Each resource has a capacity in a given time-span. (e.g. processing time for CPU, the amount of storage for memory and etc.)
- For real-time scheduling, only the temporal division of resource capacity among real-time processes is of interest.

9.3.2 Requirements

- The requirements of multimedia applications and data streams must be served.
- The transmission/processing requirements of local and distributed multimedia applications can be specified according to the following characteristics:
  - *Throughput*: Determined by the needed data rate of a connection to satisfy the application requirements.
  - *Delay "at the resource"* (local): The maximum time span for the completion of a certain task at this resource.

- *End-to-end delay* (global): The total delay for a data unit to be transmitted from the source to its destination.
- *Jitter*: Determines the maximum allowed variance in the arrival of data at the destination.
- *Reliability*: Defines error detection and error correction mechanisms used for the transmission and processing of multimedia tasks.
  - How to handle errors: Ignored, indicated and/or corrected.
  - Retransmission may not be acceptable for time-critical data.

- These requirements are known as *Quality of Service* (QoS) parameters.

9.3.3 Components and phases

- Resource allocation and management can be based on the interaction between clients and their respective resource managers.
- The client selects the resource and requests a resource allocation by specifying its QoS specification.

- The resource manager checks its own resource utilization and decides if the reservation request can be served or not.
- Performance can be guaranteed once it is accepted.

- Phases of the resource reservation and management process

1. *Schedulability*
   - The resource manager checks with the given QoS parameters (e.g. throughput and reliability).
2. *QoS calculation*
   - The resource manager calculates the best possible performance (e.g. delay) the resource can guarantee for the new request.
3. *Resource reservation*
   - Allocates the required capacity to meet the QoS guarantees for each request.
4. *Resource scheduling*
   - Incoming messages (i.e. LDUs) from connections are scheduled according to the given QoS guarantees.

9.3.4 Allocation Scheme

- Reservation of resources can be made either in a pessimistic or optimistic way:
- The pessimistic approach avoids resource conflicts by making reservations for the worst case. (It's very conservative.)
- The optimistic approach reserves resources according to an average workload only.

	Pessimistic approach	Optimistic approach
Account for	Worst case	Average case
QoS	Guaranteed	Best effort
Utilization	Low	High
Remarks		May need a monitor to detect overload situation and act

9.3.5. Continuous media resource model

- A model is frequently adopted to define QoS parameters and the characteristics of the data stream.
- It is based on the model of linear bounded arrival process (LBAP).
- A distributed system is decomposed into a chain of resources traversed by the messages on their end-to-end path.
- The data stream consists of LDUs (messages).
- Various data streams are independent of each other.

- The model considers a burst of messages consists of messages that arrived ahead of schedule.

- LBAP is a message arrival process at a resource defined by 3 parameters:
  M = maximum message size (byte/message)
  R = maximum message rate (message/second)
  B = maximum burstiness (message)

Example: Single channel audio data are transferred from a CD player attached to a workstation over the network to another computer

- As CD audio is handled, the bit rate is constant.
- The audio signal: sampled at 44.1 kHz, each sample is coded with 16 bits.
- Samples are grouped into 75 frames of equal size (corresponds to messages) in a second and transmitted in CD-format standard.
- Assume up to 12000 bytes are assembled into 1 packet and transmitted over the LAN.

Then, we have

data rate	=	44100x16/8	=	88200 bytes/s
R	=	75 messages/second		
M	=	88200/75	=	1176 bytes/message
B	=	10 messages	≤	12000/1176

That B=10 means we receive 1 packet at a time and each of them carries 10 messages, which implies we receive no more than 10 messages at a time.

Burst:

- Bursts are generated
  - When data is transferred from disk in a bulk transfer mode
  - When messages are assembled into larger packets
  - When traffic congestion is experienced
- In the model, it is assumed that, during a time interval of length t, the maximum number of messages arriving at a resource must not exceed $\overline{M} = B + R \times t$ (message)

Maximum buffer size

- Messages arriving ahead must be queued in a buffer.
- The buffer size is $S = M \times (B+1)$ (bytes)

Logical backlog

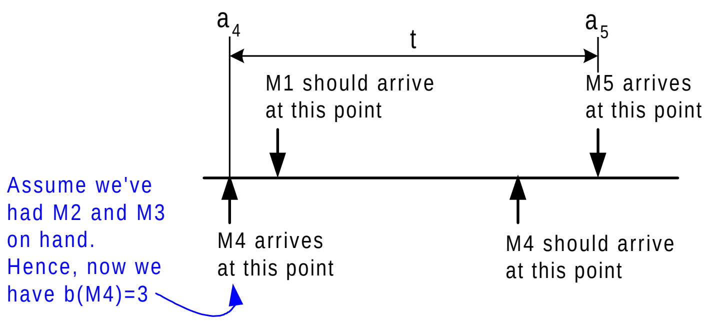
- Logical backlog is the number of messages which have already arrived "ahead of schedule" at the arrival of message m.
- Let a_i be the actual arrival time of message m_i for $0 \le i \le n$. Then b(i) is defined by

$$\begin{cases} b(m_0) = 0 \\ b(m_i) = \max(0, b(m_{i-1}) - (a_i - a_{i-1})R + 1) \end{cases}$$

  Unit: message

- Example:

When message M5 arrives, the time increases by t. In this period, it should consume tR messages, so the number of message on hand (in buffer) should be b(M5)=b(M4)-tR+1. The last one is the new arrival.



Logical arrival time

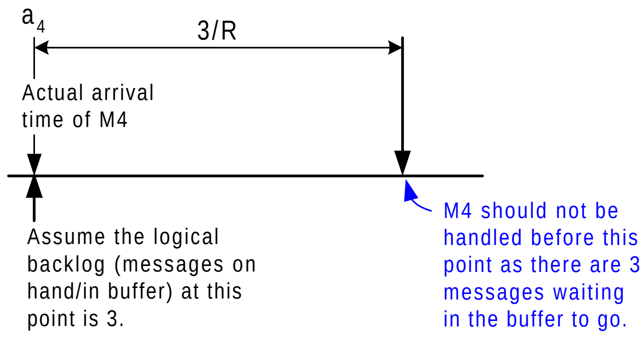
- The logical arrival time defines the earliest time a message m_i can arrive at a resource when all messages arrive according to their rate.

$$l(m_i) = a_i + \frac{b(m_i)}{R}$$

or

$$\begin{cases} l(m_0) & = & a_0 \\ l(m_i) & = & \max\left(a_i, l(m_{i-1}) + \dfrac{1}{R}\right) \end{cases}$$

- Example:



Guaranteed logical delay

- The guaranteed logical delay of a message m denotes the maximum time between the logical arrival time of m and its latest valid completion time (deadline).

Workahead messages

- If a message arrives "ahead of schedule" and the resource is in an idle state, the message can be processed immediately and it's called a workahead message.
- If a message is processed 'ahead of schedule' the logical backlog is greater than the actual backlog.

9.4 Process management

- It's handled by the process manager.
- The process manager maps single processes onto resources according to a specified scheduling policy such that all processes meet their requirements.

- A process under control of the process manager can be in one of the 4 states:
  1. *Idle state*: No process is assigned to the program.
  2. *Blocked state*: The process is waiting for an event, i.e., it lacks one of the necessary resources for processing.
  3. *Ready-to-run state*: All necessary resources except the processor are assigned to the process.
  4. *Running state*: A process is running as long as the system processor is assigned to it.

- The process manager is the scheduler.
- The scheduler transfers a process into the ready-to-run state by assigning it a position in the respective queue of the dispatcher.
- Dispatcher is the essential part of the operating system kernel.

- The next process to run is chosen according to a priority policy.
- The process with the longest ready time is chosen if more than one processes have equal priority.

9.4.2 real-time processing requirements

- The real-time process manager determines a schedule for the resource CPU that allows it to make reservations and to give processing guarantees.
- Each of them can meet its deadlines.
- In a multimedia system, continuous and discrete media data are processed concurrently.

- There are 2 conflicting goals for scheduling of multimedia tasks.
  - An uncritical process should not suffer from starvation because time-critical processes are executed. (e.g. should handle text while handling video.)
  - A time-critical process must never be subject to priority inversion.

- One should minimize (1) The overhead caused by the schedulability test and the connection establishment (2) The costs for the scheduling of every message.

- The latter is more critical because they occur periodically.

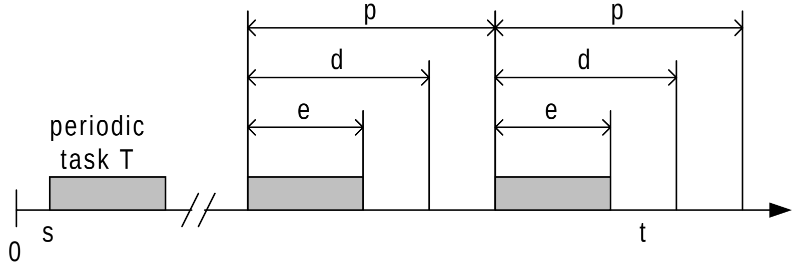
9.4.3 Traditional real-time scheduling

- The goal of traditional scheduling on time-sharing computers is Optimal throughput, optimal resource utilization and fair queuing.
- The main goal of real-time tasks is to provide a schedule that allows all, respectively, as many time-critical processes as possible, to be processed in time, according to their deadline.
- Two basic algorithms for solving real-time scheduling problems: *Earliest deadline first algorithm* and *Rate monotonic scheduling*.

9.4.4 real-time scheduling: system model

- A task is a schedule entity of the system.
- In a hard real-time system, a task is characterized by its timing constraints and its resource requirements.
- Here, only periodic tasks without precedence constraints are discussed. i.e. the processing of 2 tasks is mutually independent.
- The time constraints of the periodic task T are characterized by the following parameters (s,e,d,p)

  s : Starting point
  e : Processing time of T
  d : Deadline of T
  p : Period of T
  r : Rate of T (r=1/p) whereby $0 \le e \le d \le p$



Characterization of periodic tasks

- For continuous media tasks, it is assumed that the deadline of the period (k-1) is the ready time of period k (i.e. d=p), which is known as *congestion avoiding deadlines*.

- All tasks processed on the CPU are considered as preemptive unless otherwise stated.

- In a real-time system, the scheduling algorithm must determine a schedule for an exclusive, limited resource that is used by different processes concurrently such that all of them can be processed without violating any deadlines.

- If a scheduling algorithm guarantees a task, it ensures that the task finishes processing prior to its deadline.

- Performance metrics:
  - *Guarantee ratio*: The total number of guaranteed tasks versus the number of tasks which could be processed.
  - *Processor utilization*: The amount of processing time used by guaranteed tasks versus the total amount of processing time: $U = \sum_{i=1}^{n} \frac{e_i}{p_i}$

- Note, for each task i, the processor utilization is $e_i \,/\, p_i$. Many tasks are running in parallel, so we've $U = \sum_{i=1}^{n} \frac{e_i}{p_i}$.

9.4.5 Earliest deadline first (EDF) algorithm

- The highest priority is assigned to the task with the earliest deadline.
- EDF is an optimal, dynamic algorithm.
- At every new ready state, the scheduler selects the task with the earliest deadline among the tasks that are ready and not fully processed.
- At any arrival of a new task, EDF must be computed immediately leading to a new order and the new task is scheduled according to its deadline.
- The running task is preempted.
- No guarantee about the processing of any task can be given.
- It has a lot of overhead.

- Extension of EDF (*Time-Driven Scheduler* (TDS)):
  - If an overload situation occurs the scheduler aborts tasks which cannot meet their deadlines anymore.
  - If there is still an overload situation, the scheduler removes task which is less important for the system.

- Another variant of EDF:
  - Every task is divided into a mandatory and an optional part.
  - A task is terminated according to the deadline of the mandatory part, even if it is not completed at this time.
  - Tasks are scheduled with respect to the deadline of the mandatory parts.
  - The optional parts are processed if the resource capacity is not fully utilized.
  - In an overload situation, the optional parts are aborted.
  - This implementation favors scalable video.

9.4.6 Rate monotonic algorithm

- It is an optimal, static, priority-driven algorithm for preemptive, periodic jobs.

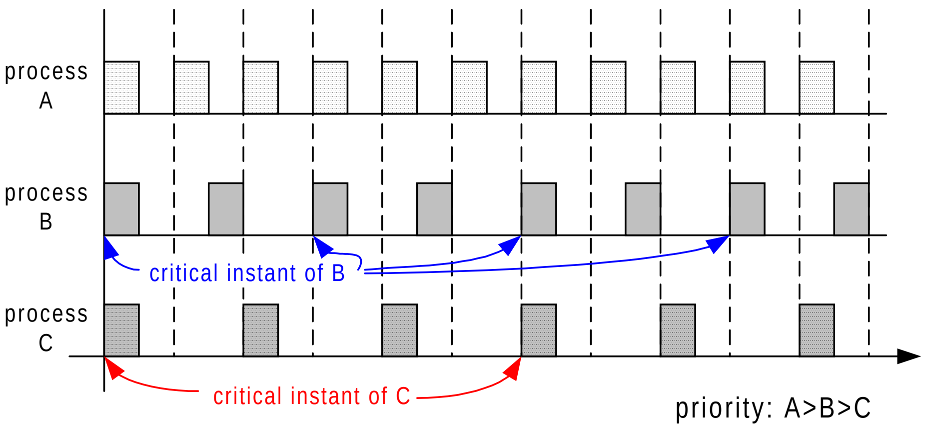
- It is optimal in a way that it can provide the same schedule any static algorithm can provide.

- There are 5 necessary prerequisites to apply the rate monotonic algorithm:
  - The requests for all tasks with deadlines are periodic.
  - Each task must be completed before the next request occurs.
  - All tasks are independent.
  - Run-time for each request of a task is constant
  - Any non-periodic task in the system has no required deadline.

- A process is scheduled by a static algorithm at the beginning of the processing.
- Each task is processed with the priority calculated at the beginning.
- No further scheduling is required.

- Static priorities are assigned to tasks, once at the connection set-up phase, according to their request rates.
- How to determine the priority?
  - Tasks with higher request rates will have higher priorities.
  - The task with the shortest period gets the highest priority.

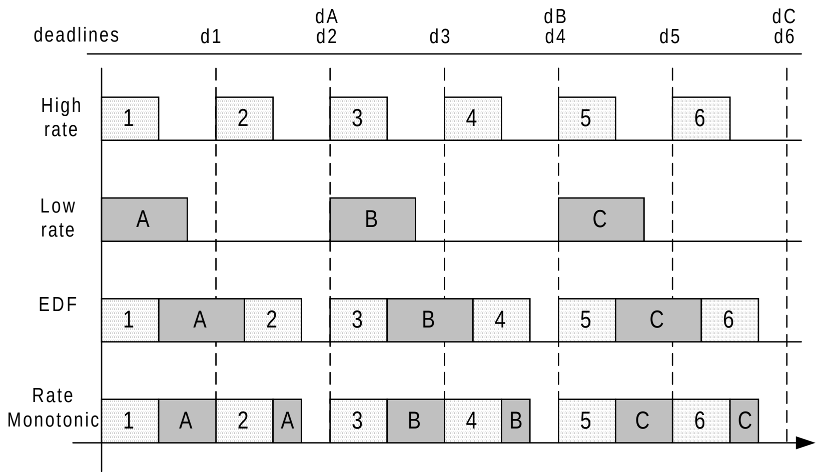
- The response time is the time span between the request and the end of processing the task.
- This time span is maximal when all processes with a higher priority request to be processed at the same time.
- The time instant when this happens is known as the *critical instant*. (the fewer, the better.)



- The *critical time zone* is the time interval between the critical instant and the completion of a task. (the shorter, the better.)

9.4.7 EDF and Rate Monotonic: Context switches

- If more than one stream is processed concurrently in a system, it is very likely that there might be more context switches with a scheduler using the rate monotonic algorithm than one using EDF.



Rate monotonic vs. EDF: context switches in preemptive systems

9.4.8 EDF and Rate Monotonic: Processor Utilization

- Rate monotonic algorithm
  - The processor utilization depends on the number of tasks which are scheduled, their processing times and their periods.
  - It is upper bounded.
  - The upper bound of the processor utilization is determined by the critical instant.
  - A set of m independent, periodic tasks with fixed priority will always meet its deadline if

    $$U(m) = m\left(2^{1/m} - 1\right) \geq \frac{e_1}{p_1} + \cdots + \frac{e_m}{p_m}$$

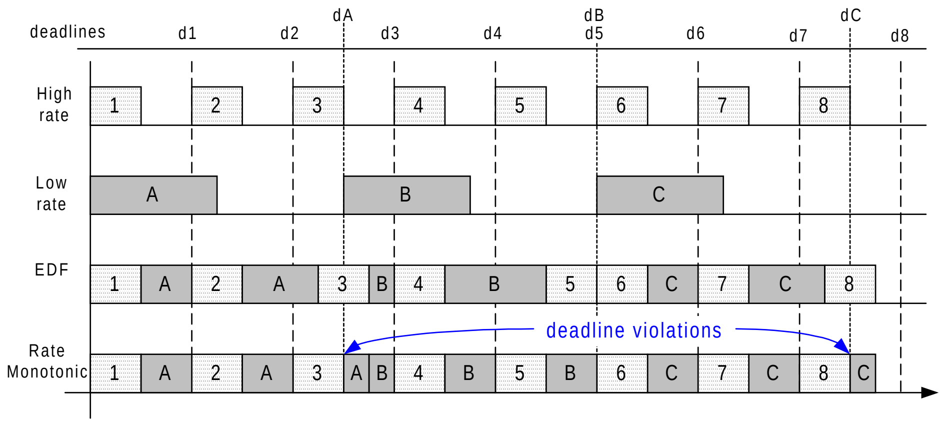
    Note: $U(m) = m\left(2^{1/m} - 1\right) \geq \ln 2 = 0.6931$

  - When a new job is coming, to save effort, just check if $U_k = U_{k-1} + \frac{e_k}{p_k} \leq \ln 2$, where $U_k = \sum_{i=1}^{k} \frac{e_i}{p_i}$. If yes, go ahead, else deny it.
  - The problem of underutilizing the processor is aggregated by the fact that, in most cases, the average task execution time is considerably lower than the worst case execution time.

  - The rate monotonic algorithm on average ensures that all deadlines will be met even if the bottleneck utilization is well above 80%.
  - No other static algorithm can achieve a higher processor utilization. (That's why it is optimal.)

- EDF
  - U(m) of 100% can be achieved with EDF as all tasks are scheduled dynamically according to their deadlines.

Example: Where the CPU can be utilized to 100% with EDF, but where rate monotonic scheduling fails.



Rate monotonic vs. EDF: processor utilization

9.4.9 Extensions to Rate Monotonic Scheduling

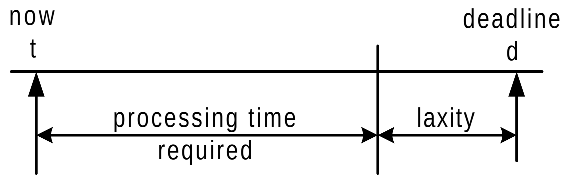
- divides a task into a mandatory and an optional part.
- The processing of the mandatory part delivers a result which can be accepted by the user.
- The optional part only refines the result.
- The mandatory part is scheduled according to the rate monotonic algorithm.
- The optional part is scheduled with other polices.
- What can we do if there are aperiodic tasks in some systems?
  - If the aperiodic request is an aperiodic continuous stream, we transform it into a periodic stream if possible.
  - If the stream is not continuous, we can apply a sporadic server to respond to aperiodic requests.
- Pros of the rate monotonic algorithm
  - It is particularly suitable for continuous media data processing because it makes optimal use of their periodicity.
  - No scheduling overhead as it is a static algorithm.

- Potential problems:
  - Problems emerge with data streams which have no constant processing time per message
  - The simplest solution is to schedule these tasks according to their maximum data rate, but this decreases processor utilization.

9.4.10 Other approaches for In-time scheduling

<u>Least Laxity First (LLF)</u>

- The task with the shortest remaining laxity is scheduled first.
- The laxity is the time between the actual time t and the deadline minus the remaining processing time



- LLF is an optimal, dynamic algorithm for exclusive resources.

- Problems of laxity-dependence:
  - The determination of the laxity is inexact as the processing time can't be exactly specified in advance.
  - The laxity of waiting processes dynamically changes over time.
  - ⇨ The worst case is always taken into account.
- Disadvantages of LLF:
  - This may cause numerous context switches.
  - The laxity of each task must be newly determined at each scheduling point.

<u>Shortest Job First (SJF)</u>

- The task with the shortest remaining computation time is chosen for execution.
- It guarantees that as many tasks as possible meet their deadlines under an overload situation if all of them have the same deadline.
- In most multimedia systems with preemptive tasks, the rate monotonic algorithm in different variations is employed.

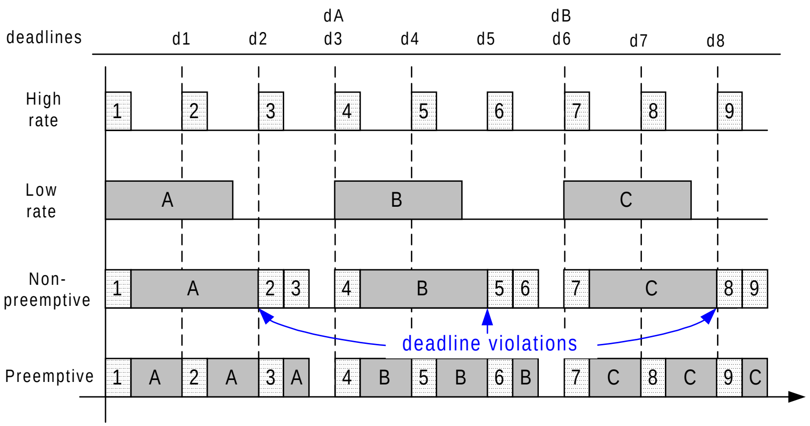
9.4.11 preemptive versus Non-preemptive task scheduling

- A *non-preemptive* task is processed and not interrupted until it is finished or requires further resources.
- In most cases where tasks are treated as non-preemptive, the arrival times, processing times and deadlines are arbitrary and unknown to the scheduler until the task actually arrives.
- The best algorithm is the one which maximizes the number of completed tasks.
- It is not possible to provide any processing guarantees or to do resource management in this case.
- For periodic processes, to guarantee their processing and to get a feasible schedule for periodic task sets, tasks are usually treated as preemptive.
- Reasons:
  - High preemptability minimizes priority inversion.
  - No feasible schedule can be found for some non-preemptive task sets, but it's possible for preemptive scheduling

Scheduling of preemptive tasks

- A task set of m periodic, preemptive tasks with processing times e_i and request periods p_i for all $i \in \{1,2\cdots m\}$ is schedulable
  - With fixed priority assignment if $\sum \frac{e_i}{p_i} \le \ln 2$
  - For deadline driven scheduling if $\sum \frac{e_i}{p_i} \le 1$

- The preemptiveness of tasks is a necessary prerequisite to check their schedulability.



Preemptive vs. Non-preemptive scheduling

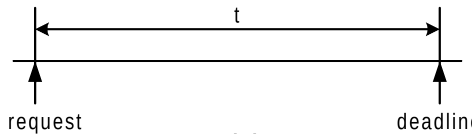
Scheduling of non-preemptive tasks:

- A set of m periodic streams with processing times e_i, deadlines d_i and request periods p_i for all $i \in \{1,2\cdots m\}$ is schedulable with the non-preemptive fixed priority scheme if

$$d_m \ge e_m + \max_{1\le i\le m} e_i$$

$$d_i \ge e_i + \max_{1\le j\le m} e_j + \sum_{j=i+1}^{m} e_j F(d_i - e_j, p_j)$$

where $F(x,y) = ceil\left(\frac{x}{y}\right) + 1$



Task i

$$t \ge e_i + \sum_{k\in\Omega} e_k + e_m$$

e_i: the processing time of task i
Ω: all tasks of higher priority than task i
e_m: longest task among all

- The time between the logical arrival time and the deadline of a task t_i has to be larger or equal to the sum of its processing time e_i and the processing time of any higher-priority task that requires execution during that time interval, plus the longest processing time of all lower- and higher- priority tasks $\max_{1\le j\le m} e_j$ that may be services at the arrival of task t_i

Schedulability test for the following tasks:

- Given m periodic tasks with periods p_i and the same processing time E per message, let $d_i = p_i + E$ be the deadline for task t_i. Then, the streams are schedulable
  - With the non-preemptive rate monotonic scheme with $\sum \frac{E}{p_i} \le \ln 2$
  - With deadline-based scheduling, the same holds with $\sum \frac{E}{p_i} \le 1$

- Non-preemptive tasks are less favorable because the number of schedulable task sets is smaller compared to preemptive tasks.
- It must be more conservative as it has to be play save, so the number of schedulable task sets is smaller.

9.5 File systems

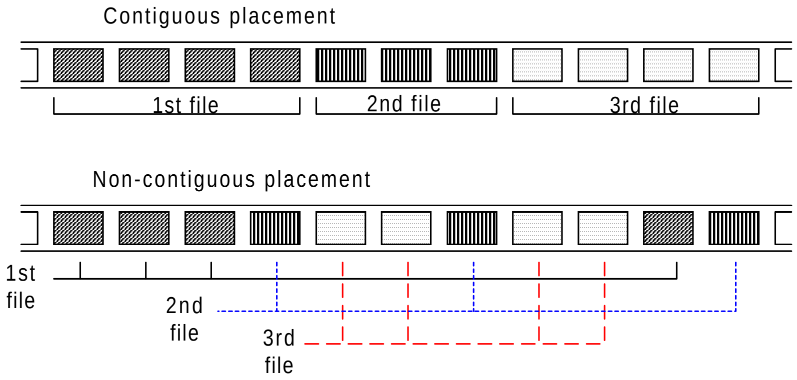
- A file is a sequence of information held as a unit for storage and use in a computer system.
- Files are stored in secondary storage.
- In traditional file systems, the information types stored in files are sources, objects, libraries and executables of programs, numeric data, text, payroll records, etc.
- In multimedia systems, the stored information also covers digitized video and audio with their related real-time 'read' and 'write' commands.

- The file system provides access and control functions for the storage and retrieval of files.
- Two issues are addressed here: the organization of the file system and disk scheduling.

9.5.1 traditional file systems

- Main goals of traditional files systems are:
  - To provide a comfortable interface for file access to the user.
  - To make efficient use of storage media.

File structure



Contiguous and Non-contiguous Storage

Sequential storage:

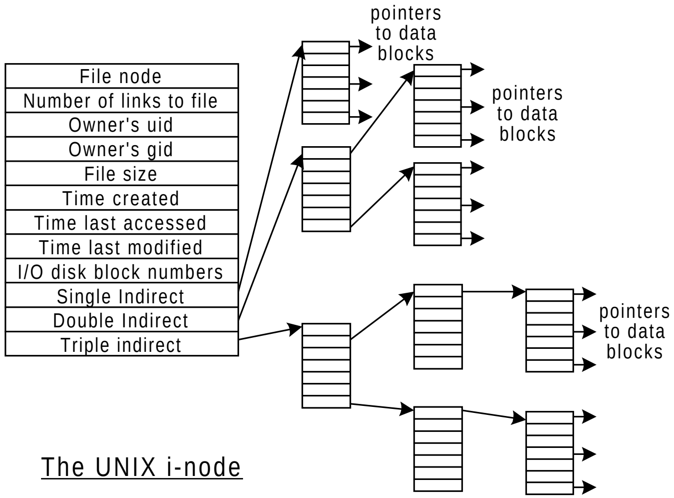
- Each file is organized as a simple sequence of bytes of records.
- Files are stored consecutively on the secondary storage media.
- A file descriptor is usually placed at the beginning of the file.
- Main advantages:
  - It's efficient for sequential access and direct access.
  - Disk access time for reading and writing is minimized.
  - Performance can be further improved with caching.

- Disadvantages:
  - Files cannot be extended without copying the whole files into a larger space.
  - Second storage is split and fragmented after a number of creation and deletion operations.

Non-sequential storage:

- The data items are stored in a non-contiguous order.
- Two approaches:
  - To use linked blocks:
    - 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 1st and the last blocks.
    - Disadvantage: The cost for random access is high as all prior data must be read.
    - Example: In MS-DOS, a file allocation Table (FAT) is associated with each disk.
- To store block information in mapping tables:
    - Each file is associated with a table where information like owner, file size, creation time, last access time, etc., are stored.

- Example: In UNIX, a small table called an i-node is associated with each file.



The UNIX i-node

Directory structure

- Files are usually organized in directories (especially, tree-structured directories).
- In multimedia systems, it's important to organize the files in a way that allows easy, fast and contiguous data access.

Disk management

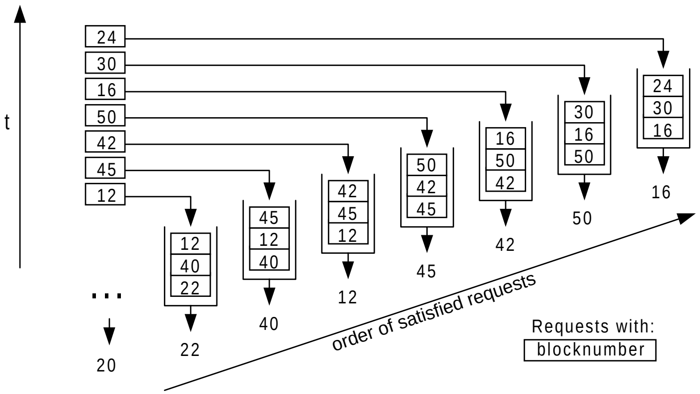
- Disk access is a slow and costly transaction.
- Common techniques used in traditional systems to reduce disk access:

  1. *Use block caches*: Blocks are kept in memory as it's expected that future read or write operations access these data again.
  2. *Reduce disk arm motion*: Take rotational positioning into account.
     - Blocks that are likely to be accessed in sequence are placed together on one cylinder.
     - The mapping tables on the disk should be placed in the middle of the disk to minimize the average seek time.
     - Use the same cylinder for the storage of mapping tables and referred blocks.

Disk scheduling

- For random access storage devices, every file operation may require movements of the read/write head (seek operation).
- The actual time 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 during which the transfer cannot proceed until the right block or sector rotates under the read/write head).
  - The actual *data transfer time* needed for the data to copy from disk to main memory.
- Usually, seek time is the largest factor of the actual transfer time.

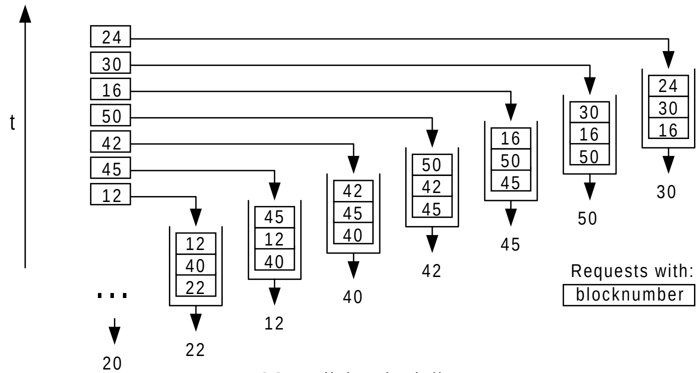
First-Come-First-Served (FCFS)



FCFS disk scheduling

- The disk driver accepts requests one-at-a-time and serves them in incoming order.
- Advantages: easy to program, intrinsically fair
- Disadvantages: it is not optimal with respect to head movement.

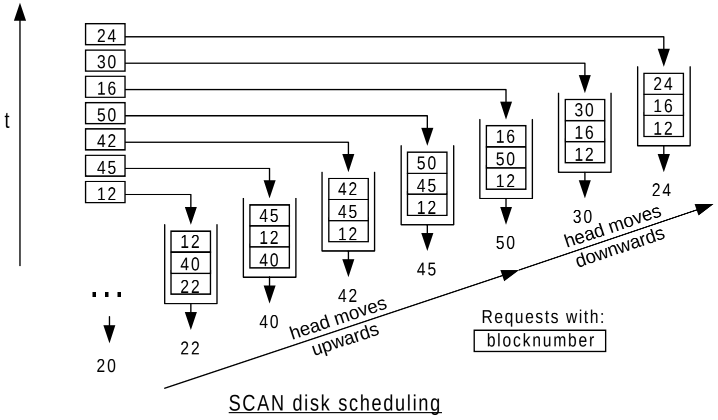
Shortest-Seek-Time First (SSTF)



SSTF disk scheduling

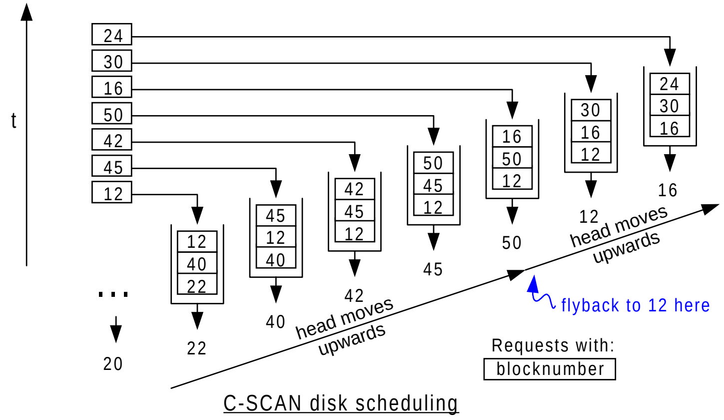
- At every point in time, when a data transfer is requested, SSTF selects among all requests the one with the minimum seek time from the current head position.
- Minimize the seek time and it's optimal in this sense.
- May cause starvation of some requests.
- It favors middle tracks.

SCAN



- Like SSTF.
- It takes the direction of the current disk movement into account.
- It serves all requests in one direction until it does not have any requests in this direction anymore.
- It provides a very good average seek time.
- The service for edge tracks is improved as c.w. SSTF.

C-SCAN



- It also moves the head in one direction, but it offers fairer service with more uniform waiting times.
- It scans in cycles, always increasing or decreasing, with one idle head movement from one edge to the other between 2 consecutive scans.

- Traditional file systems are not designed for employment in multimedia systems.
- They do not consider requirements like real-time which are important to the retrieval of stored audio and video.

9.5.2 Multimedia file systems.

- Continuous media data are different from discrete data in:
  1. Real time characteristics:
     - The data must be presented (read) before a well-defined deadline with small jitter only.
  2. File size:
     - The size of video and audio files are usually very large.
     - The file system must organize the data on disk in a way that efficiently uses the limited storage.
  3. Multiple data streams
     - Must support different media at one time.
     - Must consider tight relations between different streams arriving from different sources.

<u>Storage devices:</u>

- Tapes are inadequate for multimedia systems because they cannot provide independent accessible streams, and random access is slow and expensive.
- Disks can be characterized in 2 different ways:
  1. How information is stored on them: re-writeable, write-once or read-only
  2. The method of recording: magnetic or optical

	Seek time	Rotation speed
optical	~200ms	CLV
magnetic	~10ms	CAV

- Different algorithms for magnetic and optical disks are necessary.
- File systems on CD-ROMs are defined in ISO 9660.

<u>File Structure and placement on disk</u>

- Main goal of the file organization in conventional file systems:
  - To make efficient use of the storage capacity (i.e. to reduce internal and external fragmentation)
  - To allow arbitrary deletion and extension of files
- Main goal of the file organization in multimedia file systems:
  - To provide a constant and timely retrieval of data.
- *Internal fragmentation* occurs when blocks of data are not entirely filled.
- *External fragmentation* occurs when files are stored in a contiguous way in where the gap cannot be filled after deleting a file.

- Two basic approaches to support continuous media in file systems:
  - Use special disk scheduling algorithms and sufficient buffer to avoid jitter.
  - Optimize the organization of audio and video files on disk for their use in multimedia systems.

Approach 1 : providing enough buffer

- Advantages:
  - It is flexible. (Don't need to store files in a contiguous way.)
  - External fragmentation can be avoided.
  - The same data can be used by several streams (via reference)
- Disadvantage:
  - There are long initial delays at the retrieval of continuous media.
  - Large buffers must be provided.
  - Transfer rate is restricted

Approach 2: Using specific disk layout

- Aim: Try to minimize the cost of retrieving and storing streams
- Features of continuous media data support this approach.
  - Continuous media streams predominantly belong to the write-once-read-many nature.
  - Streams that are recorded at the same time are likely to be played back at the same time.
- It's reasonable to store continuous media data in large data blocks contiguously on disk.
- Disadvantages:
  - External fragmentation and copying overhead during insertion and deletion.
- Solution: To provide constrained block allocation of the continuous media.
  - The continuity requirement is met if,

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

where $T_{play}(s) =$ the duration of its playback
M = the size of the blocks
G = the size of the gaps

    - Here, we assume that the data transfer rate r_{dt} is the same as the disk rotation rate (sectors/s)

- This makes sure that the data required can be retrieved before it's played

- Sometimes, to serve the continuity requirements, read-ahead and buffering of a determined number of blocks must be introduced.

Disk scheduling algorithms

- Main goals of traditional disk scheduling algorithms:
  - To reduce the cost of seek operations
  - To achieve a high throughput
  - To provide fair disk access for every process

- Systems without any optimized disk layout for the storage of continuous media depend far more on reliable and efficiency disk scheduling algorithms than others.

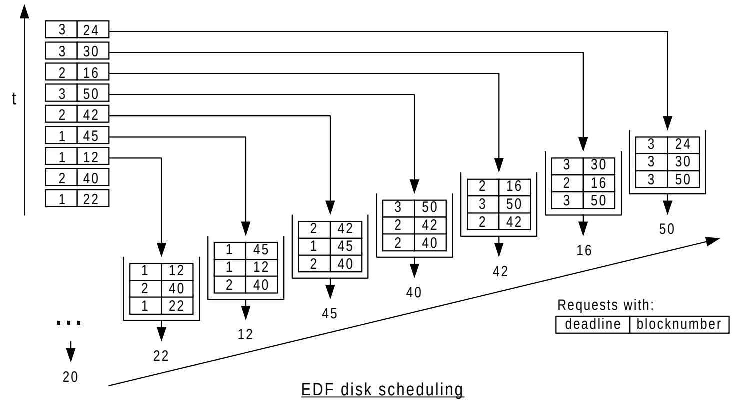
- The overall goal of disk scheduling in multimedia systems is to meet the deadlines of all time-critical tasks.

- The scheduling algorithm must find a balance between time constraints and efficiency.

Earliest deadline first (EDF)

- The block of the stream with the nearest deadline would be read first.

- It results in poor throughput and excessive seek time



EDF disk scheduling

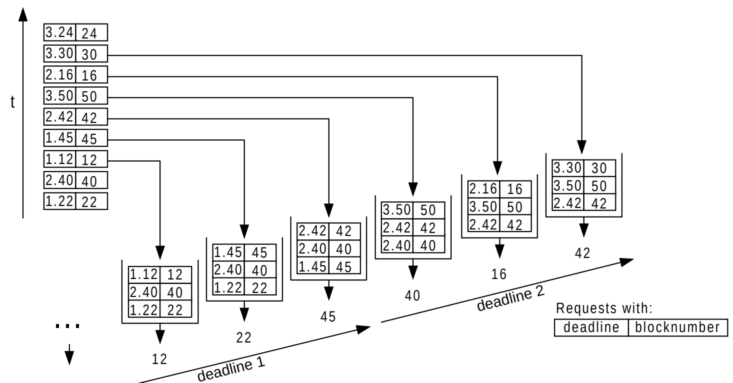
SCAN-Earliest Deadline First (SCAN-EDF)

- 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.

- Modified version 1: (SCAN-EDF with deferred deadline)

  - If D_i is the deadline of task i and N_i is the track position, the deadline can be modified to be $D_i + f(N_i)$, where $f(\bullet)$ is a function which converts the track number of i into a small perturbation of the deadline.



SCAN-EDF disk scheduling

  - It is selected to be $f(N_i) = \dfrac{N_i}{N_{\max}}$, where $N_{\max}$ is the maximum track number on disk, such that $D_i + f(N_i) \le D_j + f(N_j)$ holds for all $D_i \le D_j$.

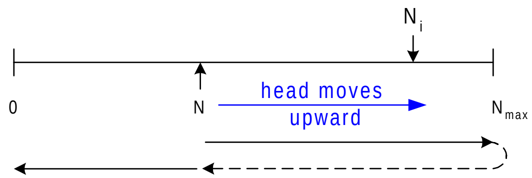
  - After this modification
    - It can directly use EDF to do the job and can do it easily.
    - It cannot achieve the original goal.

- Modified version 2:

  - A more accurate perturbation of the deadline is proposed to enhanced the previous mechanism.
  - The original goal can be achieved by doing so.

  - Let N be the actual position of the head.

  Case 1: If the head moves upward (toward $N_{\max}$)



$$f(N_i) = \begin{cases} \dfrac{N_i - N}{N_{\max}} & \text{for all } N_i \ge N \\ \dfrac{N_{\max} - N_i}{N_{\max}} & \text{for all } N_i < N \end{cases}$$

Example

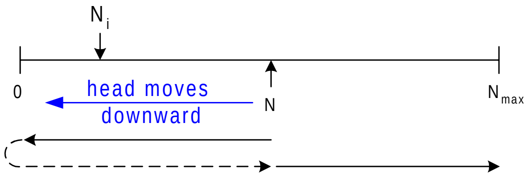
Direction $\longrightarrow$

current position, N

↓

N_i	0	1	2	3	4	5	6	7
$f(N_i)$	8/8	7/8	6/8		1/8	2/8	3/8	4/8

Case 2: If the head moves downward



$$f(N_i) = \begin{cases} \dfrac{N_i}{N_{\max}} & for \quad all \quad N_i > N \\ \dfrac{N - N_i}{N_{\max}} & for \quad all \quad N_i \le N \end{cases}$$

Example:

← Direction

current position, N

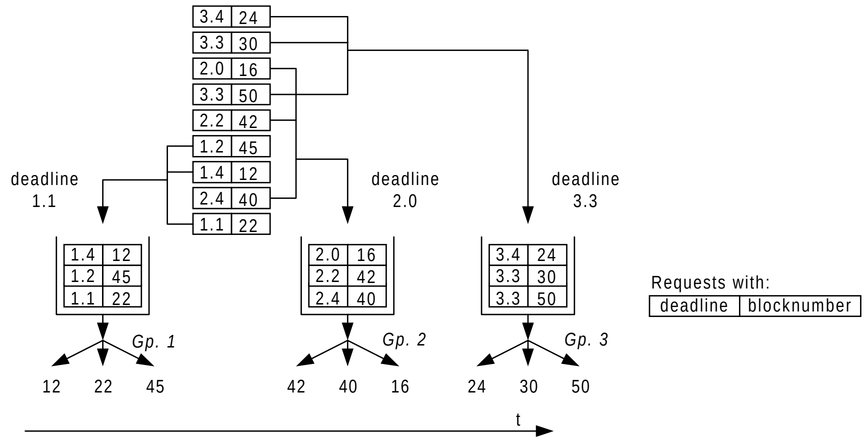
↓

N_i	0	1	2	3	4	5	6	7
$f(N_i)$	3/8	2/8	1/8		4/8	5/8	6/8	7/8

- SCAN-EDF with deferred deadlines performed well in multimedia environments.

Group Sweeping Scheduling (GSS)

- Requests are served in cycles, in round-robin manner.
- Streams are grouped in such a way that all of them comprise similar deadlines.
- Groups are served in fixed order.
- Individual streams within a group are served according to SCAN.
- GSS is a trade-off between the optimization of buffer size and arm movements.



Group sweeping scheduling

Mixed Strategy

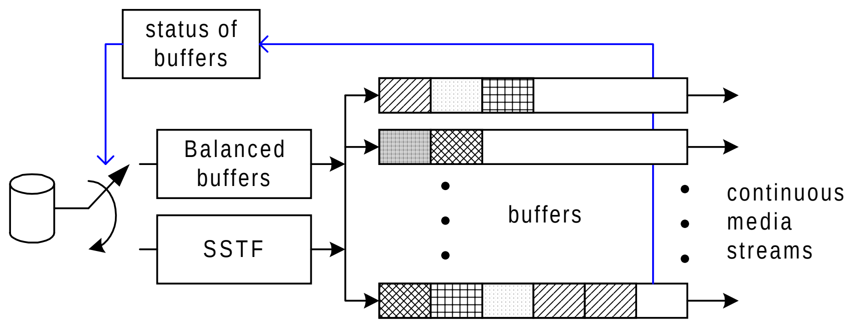
- It's a mixed strategy based on the shortest seek and the balanced strategy.

- The goal is
  - To maximize transfer efficiency by minimizing seek time and latency
  - To serve process requirements with a limited buffer space.

- With shortest seek, the process of which 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.

- Two criteria must be fulfilled:
  - The number of buffers for all processes should be balanced.
  - The overall required bandwidth should be sufficient for the number of active processes



Mixed disk scheduling strategy

- The *urgency* is the sum of the reciprocals of the current 'fullness' (amount of buffered data).

- If the urgency is large, the balance strategy will be used, if it is small, it is safe to apply the shortest seek algorithm.