OVERVIEW OF MULTIMEDIA OPERATING SYSTEMS

Report submitted in partial fulfillment of the requirement for the degree of B.Sc.

In

Electrical and Electronic Engineering

Under the Supervision of

Dr. ElRasheed Osman Khidir

By

Awad El.Rahman Mohamed El.Sadig Ali

To

Department of Electrical and Electronic Engineering

University of Khartoum

July 2005

To my parents,

my family,

my colleagues and

my friends.

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude to my supervisor
Dr. ElRasheed Osman Khidir.
for his advise, encouragement and support.

Also I want to express my warm thanks to the Department of Electrical and Electronic Engineering, University of Khartoum staff members; for their appreciated efforts to enhance the department.

I wish to thank Ali Yasin, one of my Java references; for his patience and useful programming advices.

TABLE of CONTENTS

ABSTRACT

The operating system is the most important program in the computer that manages its resources. Nowadays multimedia (video, sound, text, etc…) becomes very popular and has many useful applications, but it has certain characteristics that must be accounted for when dealing with it.

What makes the operating system able to handle multimedia? This survey outlines the main features that the operating system must have when handling multimedia: process scheduling, file system and disk scheduling, to answer this question.

A simulation of a multimedia operating system that reflects some of its important functions is designed and implemented. An application program that plays video and audio files with displaying segments of a text file as subtitles written, the program plays all in synchronization with each other. Also the program let the user to switch between two predetermined video files, to be as a kernel for multimedia editing system. Program is written in Java because it is a multithreaded-language that can perform many tasks at the same time, also it has a good support for multimedia.

مستخلص

إنّ نظامَ التشغيل هو البرنامجُ الأكثر أهميةً في الحاسوبِ، وهو الذي يقوم بإدارة مصادره. في الوقت الحاضر أصبحت الوسائط المتعددة (فيديو، صوت، نص، ...) ذات شعبية كبيرة ولها العديد مِنْ التطبيقاتِ المفيدة، غير أنها تتمتع ببَعْض الخصائصِ التي يجب أنْ تؤخذ في الاعتبار عند الَتعاملُ معها.

ما الذي يَجْعلُ نظام التشغيل قادر على مُعَالَجَة الوسائط المتعددة؟ يُلخّصُ هذا المسحِ المميزات الرئيسية التي يجب أن يمتلكها نظامَ التشغيل عند معالجته للوسائط المتعددة: جدولة العمليات، ونظامِ الملفات وجدولةالقرصِ للإجابة على هذا السؤالِ.

تم تصميم برنامج لمحاكاة نظام تشغيل متعدد الوسائط يَعْكسُ البعض مِنْ وظائفِه المهمةِ . البرنامج يقوم بتَشْغيل الفيديو والملفاتِ السمعيةِ وبعَرْض قِطَعِ من ملف نَصِّ مكتوبُ، يشغل البرنامجَ كُلّ هذه الملفات في تزامنِ مَع بعضها البعض. يتيح البرنامج أيضاًَ للمستخدم امكانية التنقل بين ملفي فيديو محدّدين مسبقاً كنواة لنظام لتحرير الوسائط المتعددة. البرنامج مكتوبُ بلغة الجافا لمقدرتها على تنفيذ العديد من المهام في نفس الوقت، كما أن لها أيضاً دعم جيد للوسائط المتعددة. تَعْكسُ هذه المحاكاةِ بَعْض الوظائفِ المهمةِ لأنظمةِ التشغيل متعددة الوسائط.

Modern computer systems consist of one or more processors, some main memory, disks, printers, a keyboard, a display, network interfaces, and other input/output devices. All in all, a complex system. Writing programs that keep track of all these components and use them correctly, let alone optimally is an extremely difficult job. For this reason, computers are equipped with a layer of software called the **operating system**, whose job is to manage all these devices and provide user programs with a simpler interface to the hardware.[1]

After we have known why computers are equipped with operating systems, in this chapter some kinds of operating systems are briefly discussed, also some important operating system concepts are discussed; in the last section an overview of the following chapters is given.

1.1 WHAT IS AN OPERATING SYSTEM?

Operating system is an interface between the computer hardware and the user; figure 1-1 shows the layers of the computer system.

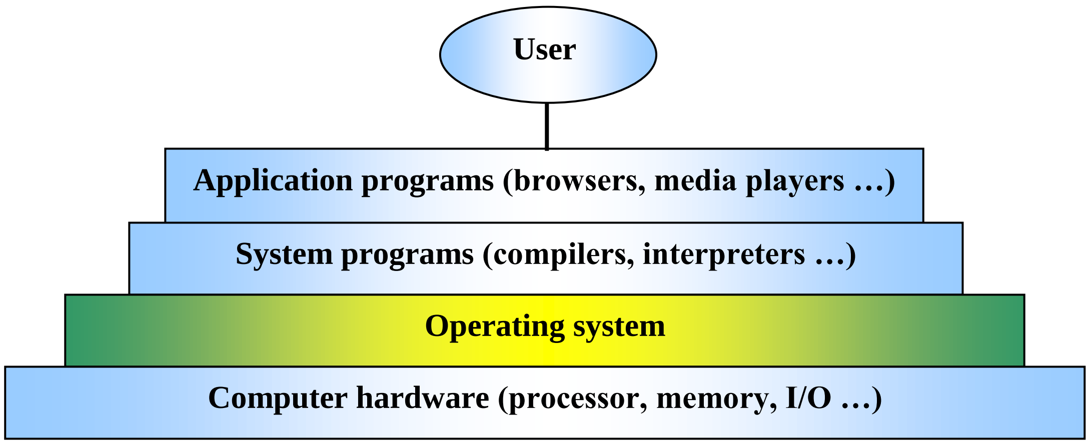

User
Application programs (browsers, media players …)
System programs (compilers, interpreters …)
Operating system
Computer hardware (processor, memory, I/O …)

Figure 1-1 Computer system layers

It is obviously that the operating system layer lies between the hardware and the user programs.

The operating system performs two basically functions: extending the machine and managing resources.

The architecture (instruction set, memory organization, I/O, and bus structure) of most computers at the machine language level is primitive and awkward to program, especially for input/output. For example to perform I/O using floppy disk we have to know the correct bytes format to open, read and write from a file in the floppy disk, i.e. we must drive the derive motor, also to know how to make the head to move to the correct track, to seek for the correct sector, to read the data and so on. All these detailed information could be hidden from the user by the operating system, which presents a nice, simple view of named files that can be read and written.

The function of the operating system is to present the user with the equivalent of an **extended machine** or **virtual machine** that is easier to program than the underlying hardware.

The second function of the operating system is to act as a **resource manager**, the computer has many resources such as CPU, memory, hard disks, printers, network interfaces and so forth. Many programs (processes) competing for these resources, for example if three programs want to use printer at the same time, the output will be chaos. Here the operating system can solve this problem by buffering all the outputs on the disk, and then print them orderly, other programs can work while printing. All resources can be managed in same manner [1].

1.2 THE OPERATING SYSTEM KINDS:

Due to the rapidly development of operating systems, variety of operating systems are in present. Here we will mention a few of them:

1.2.1 Server operating systems:

Servers are very large personal computers, workstations, or even mainframes. They serve multiple users at once over a network and allow the user to share hardware and software resources. Typical server operating systems are UNIX and Windows 2000.

1.2.2 Multiprocessor operating systems:

In order to increase the computing power multiple CPUs are connected into a single system. This kind needs special operating systems with special features for communication and connectivity.

1.2.3 Personal Computer Operating Systems:

Their job is to provide a good interface to a single user. They are commonly used for word processing and Internet access. Common example is Windows 98.

1.2.4 Real-Time Operating Systems:

Another type of operating systems is the real-time system. These systems are characterized by having time as a key parameter. If the action must occur at a certain moment, then we have a **hard real-time system.**

Another type of real-time systems is a **soft real-time system**, in which missing an occasional deadline is acceptable. Digital audio or multimedia systems fall in this category. [1]

1.3 SOME OPERATING SYSTEM CONCEPTS:

1.3.1 PROCESS:

1.3.1.1 Process Concept:

Process is simply a program in execution, i.e. it is a dynamic version of a static program. The process may be in one of four states as shown in Figure 1-2:

- New: when the process is being created.
- Running: the instructions are being executed.
- Waiting: process is waiting for an event.
- Ready: the process is ready to be assigned to processor.
- Terminated: the process has finished the execution.

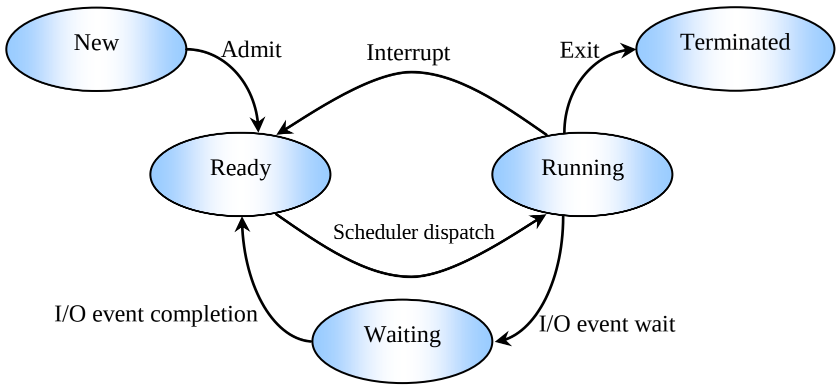


Figure 1-2 Process state

1.3.1.2 Process Scheduling:

As process enters the system, they are put into *job queue*, which consists of all processes in the system. Ready and waiting processes are kept into the *ready queue*. There are other queues in the system. When a process is allocated the CPU, it executes for awhile and eventually quits, is interrupted or waits for occurrence for a particular event, such as completion of I/O request. The list of processes that wait for a particular I/O device is called a *device queue*. Each device has its own queue. Figure 1-3 shows the queue-diagram representation of process scheduling.

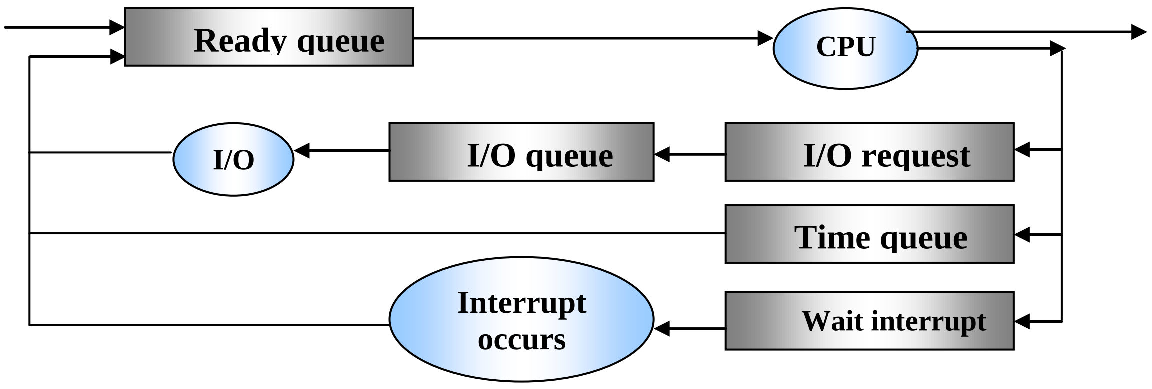


Figure 1-3 Queueing-diagram representation for process scheduling

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select processes from these queues in some fashion. The process selection is carried out by the appropriate *scheduler*.[2]

1.3.2 CPU SCHEDULING:

By switching CPU among processes, the operating system can make the computer more productive. A fundamental function of the operating system is resource scheduling. Hence the CPU is a primary computer resource, thus its scheduling is central to operating-system design. CPU scheduling deals with deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU.[2]

1.3.2.1 Scheduling Algorithms:

There are many different CPU scheduling algorithms; in this section some of these algorithms are briefly described.

1.3.2.1.1 First-Come, First-Served Scheduling (FCFS):

It is the simplest CPU scheduling algorithm. By this algorithm the process that requests the CPU first is allocated to CPU first. Its implementation is easily managed with a FIFO queue. This algorithm is nonpreemptive. Once the CPU has been given to a process, that process keeps the CPU until it releases it.

1.3.2.1.2 Priority Scheduling:

Here a priority is associated with each process; the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

1.3.2.1.3 Shortest-Job-First Scheduling:

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. That means the SJF algorithm is a special case of the priority scheduling.

1.3.2.1.4 Round-Robin Scheduling:

This algorithm is similar to FCFS scheduling, but preemption is added to switch between processes. A small time unit called *time quantum* is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.[2]

1.3.3 FILE SYSTEM:

Computer can store information on several different storage media, such as magnetic disks and optical disks. So that the computer system will be convenient to use, the operating system can provide a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the *file*. Files are mapped, by the operating system, onto physical devices. Thus the file is a named collection of related information that is recorded on secondary storage.

Many different types of information may be stored in a file: source programs, text, graphic images, sound recordings, and so on. Files have many attributes: name, type, location, and size. The operating system provides system calls to create, write, read, reposition, delete, and truncate files.

The file system of computers can be extensive. To manage all these data, we need to organize them. This organization is done in two parts. First, the file system is broken into *partitions*. In this way the user can ignore completely the problems of physically allocating space for files. Second, each partition contains information about files within it. This information is kept in entities in a *device directory*. Figure 1-4 shows the typical file-system organization.[2]

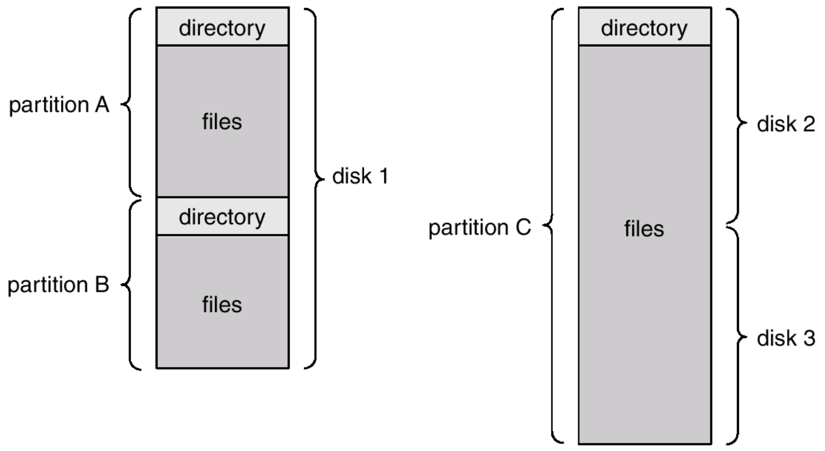


Figure 1-4 A typical file system organization

To access a file, a process first issues an open system call. If this succeeds, the caller is given some kind of token, called a file descriptor in UNIX or handle for Windows to be used for future calls. At that point the process can issue a read system call, providing the token, buffer address, and byte count as parameters. The operating system then returns the requested data in the buffer. Additional read calls can then be made until the process is finished, at which time it calls close to close the file and return its resources.[1]

1.3.4 DISK SCEDULING:

One of the operating system responsibilities is to use the hardware efficiently. For disk derives, this means having fast access time and disk bandwidth. The access time has two major components: the first is the *seek time* is the time for the disk arm to move the heads to the cylinder sector; the second is the *latency time* is the additional time waiting for the disk to rotate the desired sector.

In many cases the disk queue contains many requests want to be serviced. Thus when one request is completed, the operating system has an opportunity to choose which pending request to service next. Many forms of disk scheduling forms are available; next some of them are briefly discussed.[2]

1.3.4.1 FCFS Scheduling:

First-come first-served is the simplest, fair but it generally does not provide the fastest service.

1.3.4.2 SSTF Scheduling:

To achieve faster access it seems reasonable to service all the requests close to the current head position, before moving the head far away to service other requests. This assumption is the basis for the *shortest-seek-time-first.*

1.3.4.3 SCAN Scheduling:

Here the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.[2]

1.4 WHY MULTIMEDIA OPERATING SYSTEMS?

[The importance of multimedia operating systems comes from the importance of multimedia on a side and the importance of the operating system on the other side. [Multimedia is one of the most important aspects of the information area [3]. Nowadays, multimedia generally indicates a rich sensory interface between humans and computers or computer-like devices-an interface that in most cases gives the user control over the pace and sequence of the information. We all know multimedia when we see and hear it [4]. The operating system is the most important program in the computer; it is the program that manages all the resources of the computer as we have seen in the previous sections. Now how much the multimedia operating systems are important can be clearly observed.

1.5 WHAT IS NEXT?

In this chapter we have known what the operating system is, and why it is introduced. Also we have listed some kinds of operating systems; the reader can observe that the multimedia operating system, which is our point of discussion in the next chapters, is categorized under the real-time system kind as a soft real-time system. Some important operating systems concepts are discussed briefly.

1.6 OVERVIEW:

Chapter 2: SOME ABOUT MULTEMEDIA: this chapter talks about multimedia, and illustrates some of its important characteristics that are the reasons of treating it differently, some multimedia compression standard are discussed briefly.

Chapter 3: MULTEMEDIA OPERATING SYSTEMS: here multimedia operating systems are discussed; showing the different characteristics that they hold. This chapter talks about processing scheduling, file system, and disk scheduling schemes that are found in multimedia operating systems**.**

Chapter 4: **SIMULATION**: This chapter shows the design of a program that simulates some of the multimedia operating system functions. It plays video and audio files, and displays text segments as subtitles, all these are done synchronously. Also the program represents a seed for multimedia editing system. Algorithms and flowcharts for the program are given.

Chapter 5: ANALYSIS: In this chapter the outputs of the program designed in chapter 4 is shown. Also some discussions are made.

Chapter 6: CONCLUSION: Here the main issues of the project have been concluded. Problems that have been faced by the project have been stated. Some Recommendations for future work are suggested.

In this chapter we will know some about multimedia; what multimedia is, what are its uses and applications, its characteristics, its files and some about compression.

2.1 WHAT IS MULTIMEDIA?

Multimedia literally means more than one medium. Under this definition this document is a multimedia work, because it contains two media: text and images (the figures).However most people use the term "multimedia" to mean a document containing two or more *continuous* media, that is media that must played back over some time interval [1].

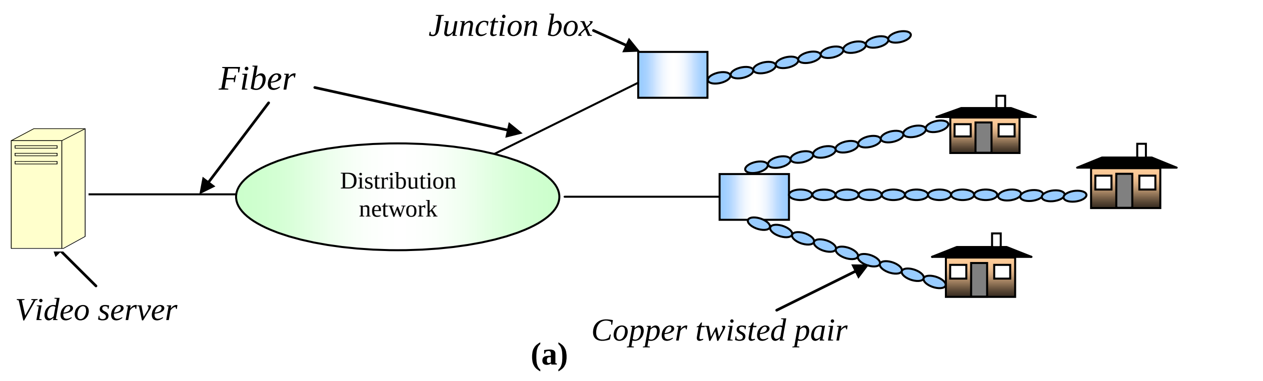
Another term is "video", which technically is just the image portion of a movie (sound is another portion). In this text the media which will act as multimedia are: video, audio and text.

2.2 MULTIMEDIA USES AND APPLICATIONS:

Multimedia applications are primarily existing applications that can be made less expensive or more effective through the use of multimedia technology. In addition, new, speculative applications, like video on demand, can be created with the technology. A few of these applications are presented here.

2.2.1 Video on demand:

Video on demand (VOD), also called movies on demand, is a service that provides movies on an individual basis to television sets in people's homes (Figure 2-1 in the next page). The movies are stored in a central server and transmitted through a communication network which must be capable of transmitting data at high speed and in real time for last few kilometers the transmission medium might be ADSL or cable TV. A set-top box (STB) connected to the communication network converts the digital information to analog and inputs it to the TV set. STB device, in fact, is a normal computer containing as minimum a CPU, RAM, ROM and network interface, in addition to special chips for video decoding and decompression. The viewer uses a remote control device to select a movie and manipulate play through start, stop, rewind, and visual fast forward buttons. The capabilities are very similar to renting a video at a store and playing it on a VCR. VOD differs from pay per view by providing any of the service's movies at any time, instead of requiring that all purchasers of a movie watch its broadcast at the same time.[1]



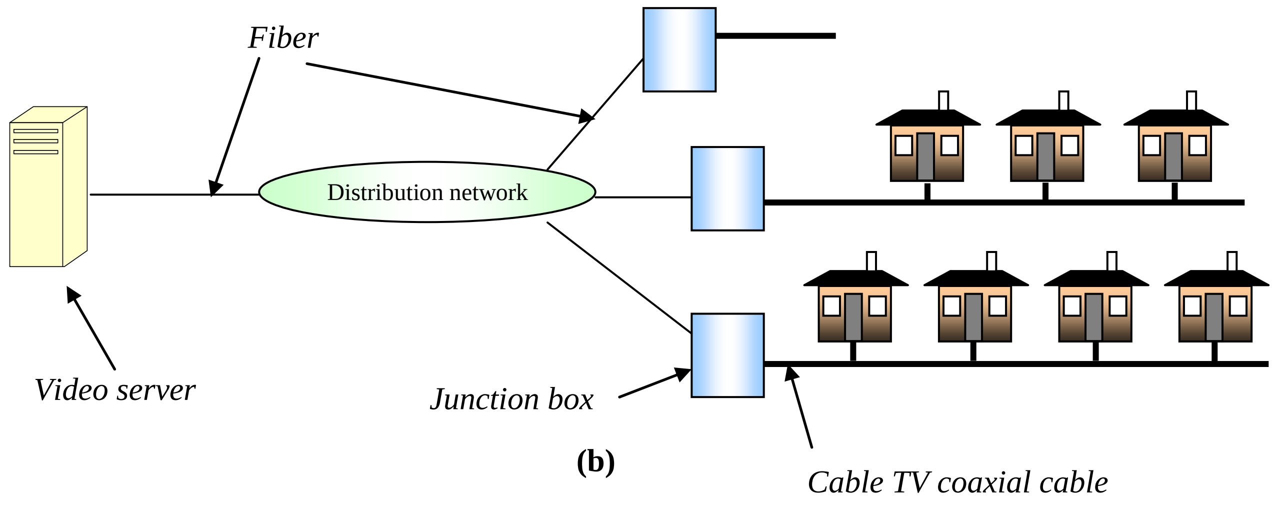


Figure 2-1 VOD (a) using ADSL (b) using Cable TV

2.2.2 Video conferencing:

Currently, most video conferencing is done between two specially set-up rooms. In each room, one or more cameras are used, and the images are displayed on one or more monitors. Text, images, and motion video are compressed and sent through telephone lines. Recently, the technology has been expanded to allow more than two sites to participate.[4]

2.2.3 Education

A wide range of individual educational software employing multimedia is available on CD-ROM. One of the chief advantages of such multimedia applications is that the sequence of material presented is dependent upon the student's responses and requests. Multimedia is also used in the classroom to enhance the educational experience and augment the teacher's work. Multimedia for education has begun to employ servers and networks to provide for larger quantities of information and the ability to change it frequently.[4]

2.2.4 Distance learning

Distance learning is a variation on education in which not all of the students are in the same place during a class. Education takes place through a combination of stored multimedia presentations, live teaching, and participation by the students. Distance learning involves aspects of both teaching with multimedia and video conferencing.[4]

2.2.5 Telemedicine

Multimedia and telemedicine can improve the delivery of health care in a number of ways. Digital information can be centrally stored, yet simultaneously available at many locations. Physicians can consult with one another using video conference capabilities, where all can see the data and images, thus bringing together experts from a number of places in order to provide better care. Multimedia can also provide targeted education and support for the patient and family.[4]

2.3 MULTIMEDIA CHARACTERISTICS:

Multimedia has two key characteristics that must be well understood to deal with it successfully:

1. Multimedia uses extremely high data rates.
2. Multimedia requires real-time playback.

The high data rates come from the nature of visual and acoustic information. Eye and ear can process enormous amounts of information per second, and have to be fed at that rate to produce an acceptable viewing experience. For example data rate of uncompressed HDTV (1280×720) is 288 GB/hr, that means a 2-hour of this movie type fills a 570-GB file, a video server which stores 1000 such movies needs 570 TB of disk space.

The second demand that multimedia puts on a system is the need for real-time data delivery. If this condition is not satisfied we will get some problems. For video portion of a digital movie which consists of some number of frames per second, these frames must be delivered at a certain rate; say 25 frames/sec, or the movie will look choppy. The ear is more sensitive than the eye, so a variance of even a few milliseconds in delivery times will be noticeable. Variability in delivery rates is called jitter. The real-time properties required to play back multimedia acceptably are often described by quality of service parameters (QoS). They include average bandwidth available, peak bandwidth available, minimum and maximum delay, and a bit loss probability.[1]

2.4 MULTIMEDIA FILES:

In most systems an ordinary text file consists of a linear sequence of bytes without any structure that the operating system knows about or cares about. With multimedia, the situation is more complicated. To start with, video and audio are completely different. They are captured by distinct devices (CCD chip versus microphone), have a different internal structure (video has 25-30 frames/sec; audio has 44,100 samples/sec), and they are played back by different devices(monitor versus loudspeakers).

Movies that have a lot of audience that do not speak the language which is in the original movie, two ways are used to deal with this problem. For some countries, an additional sound track is produced, with the voices dubbed into the local language. In addition many movies now provide closed-caption subtitles to allow hearing-impaired people to watch the movie.

The net result is that a digital movie may actually consist of many files: one video file, multiple audio files, and multiple text files with subtitles in various languages. The most important thing here is to keep the subfiles synchronized so that when the selected audio track is played back it remains in sync with the video, if is not satisfied the viewer may hear an actor's words before or after his lips move. A simple set of multimedia files is shown in figure 2-2.[2]

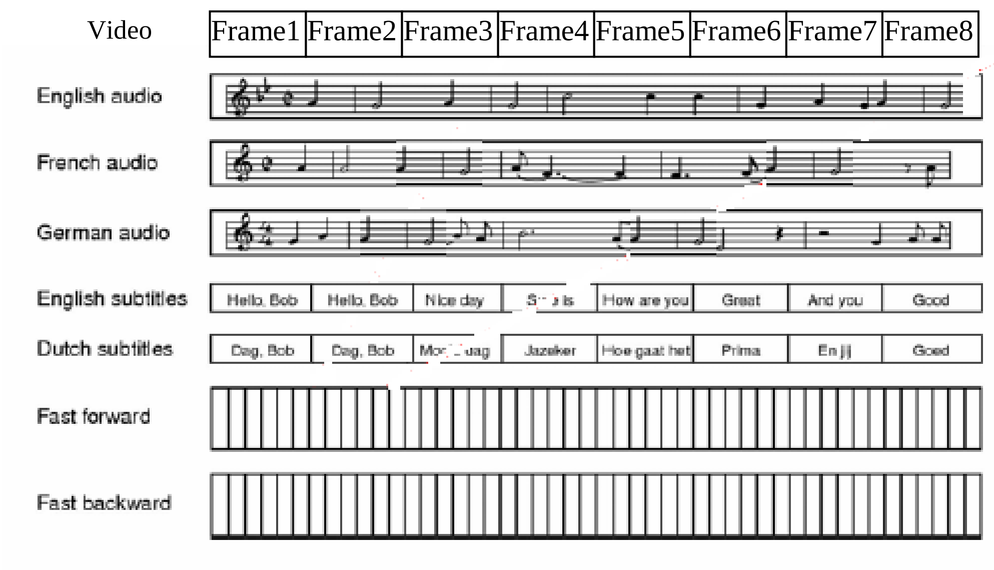


Figure 2-2 A movie may consist of several files

2.5 VIDEO COMPRESSION:

As we have seen multimedia uses extremely high data rates and we have said that a ten 2-hour uncompressed movies need 570 TB of disk space, this is a sufficient reason to think about a massive data compression. [Fortunately, a large body of research over the past few decades has led to many compression techniques and algorithms that make multimedia transmission feasible.[1]

2.5.1 The JPEG Standard:

JPEG which stand for Joint Photographic Experts Group is the first international digital image compression standard for still images, both grayscale and color. JPEG has four modes and many options, but we will concentrate on the way it is used for 24-bit RGB video, without long details.[5]

Six steps are followed to get the compressed image:

Step 1 is block preparation. For the sake of specificity, assume that the JPEG input is a 640×480 RGB image with 24-bit/pixel, as shown in figure 2-3(a) in the next page, the picture is represented by three signals one is called luminance(*Y*), and two are chrominance signals(*I* & *Q*). Each signal has a separate matrix, each with elements in the range between 0 to 255. Next square blocks of four pixels are averaged in the *I* and *Q* matrices to reduce them to 320×240. This reduction is lossy, but the eye barely notices it since the eye responds to the luminance more than to chrominance. Finally each matrix (*Y*,*I* & *Q*) is divided up into 8×8 blocks, so the *Y* matrix has 4800 blocks and the other two each have 1200 blocks each, as shown in fig 2-3(b).

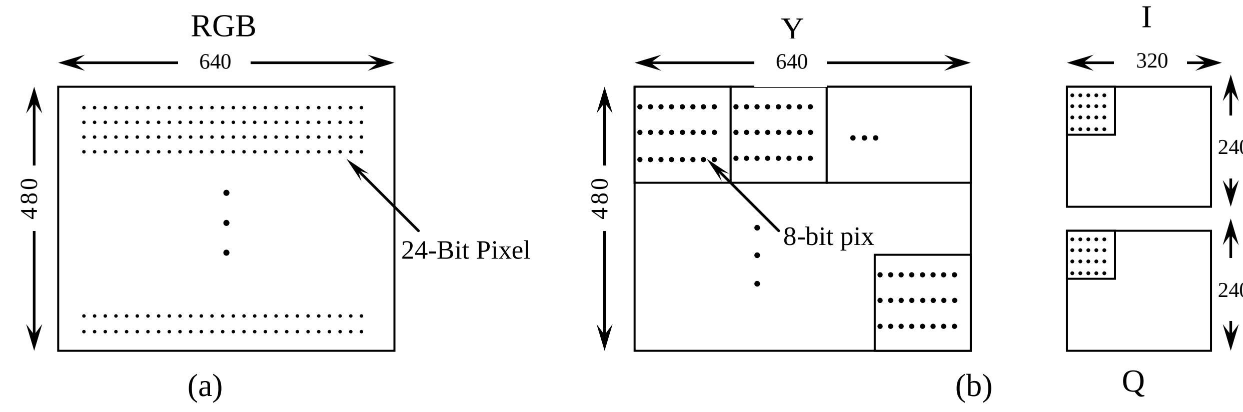


Figure 2-3 (a) RGB input data. (b) After block preparation.

Step 2 is to apply DCT (Discrete Cosine Transformation) to each 7200 blocks separately. The output of each DCT is an 8×8 matrix of DCT coefficients. DCT element (0,0) is the average value of the block.

Step 3 is called quantization, in which less important DCT coefficients are wiped out. This lossy transformation is done by dividing each of the coefficients in the 8×8 DCT matrix by a weight taken from a table. If all weights are 1, the transformation do nothing. As an example this step is given in figure 2-4.

DCT Coefficients

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized Coefficients

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization Table

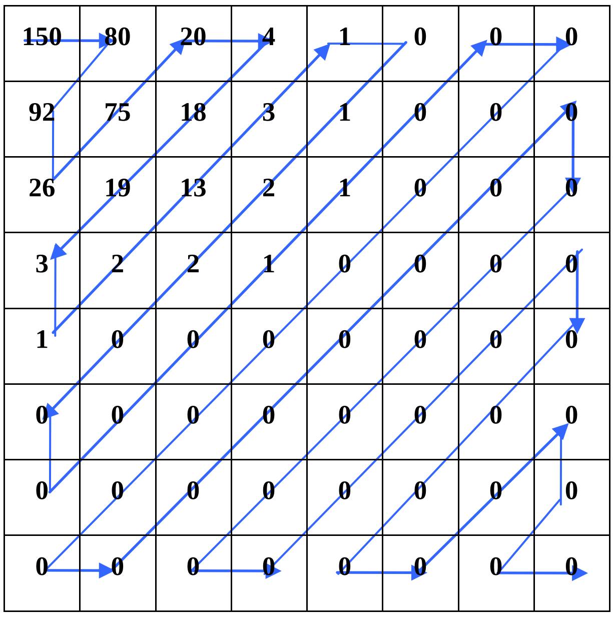
1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Figure 2-4 Computation of the quantized DCT coefficients

Step 4 reduces the (0,0) value of each block by replacing it with the amount it differs from the corresponding element with the previous block. Now differentials are computed from the other values. The (0,0) values are referred to as the DC components; the other values are the AC components.

Step 5 linearizes the 64 elements and applies run-length encoding to the list. Zigzag scanning pattern is used to make a lot of zeros be adjacent to each other, as shown in figure 2-5. Just we have to say that there are 38 zeros, rather than to send them all.

Step 6 Huffman encode the numbers for storage or transmission.[1]



150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 2-5 The order in which the quantized values are transmitted

2.5.2 The MPEG Standard:

The greatest amount of data redundancy in video is not the neighboring pixels in a single frame, it's the vast number of similar pixels in successive frames that do not change or change very slowly. By predicting that a pixel will remain the same, and then encoding the difference, you end up with a string of zeroes or very small numbers suitable for efficient run-length encoding.

MPEG (Motion Picture Experts Group), there are three kinds of frames as an output of MPEG standard:

- I (Intracoded) frames: Self-contained JPEG-encoded still pictures.
- P (Predictive) frames: Block-by-block difference with the last frame.
- B (Bidirectional) frames: Differences with the last and next frame.[1]

Figure2-6 shows the three frame kinds and the relationships between them.

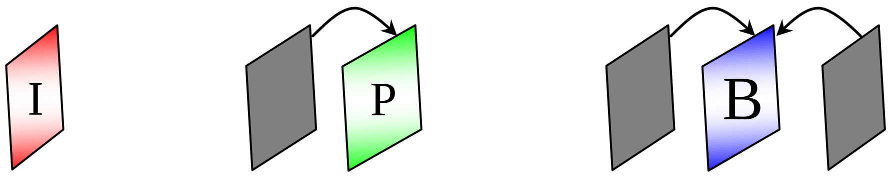


Figure 2-6 I, P and B frames

MPEG firstly compresses the first frame using JPEG standard illustrated in the previous section. Some times frames are very slightly different from each other it is wasteful to bandwidth to send all the frame many times because of that P-frames are used, P-frames based on the idea of macroblocks, which cover 16×16 pixels in luminance space and 8×8 pixels in chrominance space. A macroblock is encoded by searching the previous frame for it or for something slightly different from it. B-frames are similar to P-frames, except that they allow the reference to be in either a previous frame or in a succeeding frame, either in an I-frame or in a P-frame.[6]

2.6 SUMMARY:

Multimedia means a document containing two or more continuous media. Multimedia has many applications and uses in many fields. Multimedia has two characteristics: using extremely high data rates and requiring real-time playback. Multimedia files have a different structure from ordinary text files. Due to multimedia usage of high data rates multimedia files need a massive compression, many standards used for compression, JPEG for still pictures and MPEG for moving pictures.

In chapter 1 we have defined the operating system, what is it, why it is used, and knew some operating systems kinds. Also we have described some operating system concepts. In chapter 2 we have discussed some about multimedia: what is it, its uses and characteristics.

In this chapter we will combine the previous two chapters by applying the operating system concepts to the multimedia; this is because of its different characteristics from ordinary files.

As we have known in subsection 1.2.5 multimedia systems are categorized under real-time systems as a soft real-time system, thus additional efforts must be done by the operating system to meet the multimedia requirements, and this is about what this chapter focuses.

3.1 WHAT IS MULTIMEDIA OPERATING SYSTEM?

For the processing of audio and video, multimedia application demands that humans perceive these media in a natural, error-free way. These continuous media data originate at sources like microphones, cameras and files. From these sources, the data are transferred to destinations like loudspeakers, video windows and files located at the same computer or at a remote station. On the way from source to sink, the digital data are processed by at least some type of move, copy or transmit operation. In this data manipulation process there are always many resources which are under the control of the operating system. The integration of discrete and continuous multimedia data demands additional services from many operating system components [1]. [The operating system that its primary job is servicing multimedia is called **multimedia operating system**. [7]

Multimedia operating systems differ from traditional ones in three main ways: process scheduling, the file system, and disk scheduling. In subsequent sections we will discuss these main differences. [1]

3.2 MULTIMEDIA PROCESS SHEDULING:

We have agreed that multimedia operating system is a real-time system; the major aspect in this context is real-time processing of continuous media data. Process management must take into account the timing requirements imposed by the handling of multimedia data. Appropriate scheduling methods should be applied. As we have said multimedia operating system is a soft real-time-system, thus multimedia operating systems also have to consider tasks without hard timing restrictions under the aspect of fairness. The communication and synchronization between single processes must meet the restrictions of real-time requirements and timing relations among different media. The main memory is available as a shared resource to single processes.

For scheduling of multimedia tasks, two conflicting goals must be considered:

• An uncritical process should *not suffer from starvation* because time-critical processes are executed. Multimedia applications rely as much on text and graphics as on audio and video.

Therefore, not all resources should be occupied by the time-critical processes and their management processes.

• On the other hand, a time-critical process must never be subject to *priority inversion*. The scheduler must ensure that any priority inversion (also between time-critical processes with different priorities) is avoided or reduced as much as possible.[8]

Consider figure 2-1 in chapter 1, the operating system that is in the video server is a multimedia operating system. In the following sections we will refer to this operating system.

3.2.1 Scheduling Homogeneous Processes:

The simplest kind of server is one that can support the display of a fixed number of movies, all using the same frame rate, video resolution, data rate, and other parameters. Under these circumstances, a simple, but effective scheduling algorithm is as follows. For each movie, there is a single process whose job is to read the movie from disk one frame at a time and transmit the frame to the user. Since all the processes are equally important, have the same amount of work to do per frame, and block when they have finished processing the current frame, round-robin scheduling does the job just fine. The only addition needed to standard scheduling algorithms is timing mechanism to make sure each process runs at the correct frequency.[1]

3.2.2 General Real-Time Scheduling:

Unfortunately, the above model is rarely applicable in reality. The viewers come and go, frame sizes vary widely due to nature of video compression, and different movies may have different resolutions. As a sequence, different processes may have to run at different frequencies, with different amount of work, and with different deadlines by which the work must be completed. This leads to multiple processes competing for the CPU, each with its own work and deadlines. The scheduling of multiple competing processes, some or all of which have deadlines that must be met is called **real-time scheduling.**

As an example of the kind of environment a real-time multimedia scheduler works in, consider the three processes, A, B, and C shown in figure 3-1 in the next page. Process A runs every 30 msec; each frame requires 10 msec of CPU time. In the absence of competition, it runs in the bursts A1, A2, A3, etc... each one starting 30 msec after the previous one. Each CPU burst handles one frame and has a deadline: it must complete before the next one is to start.

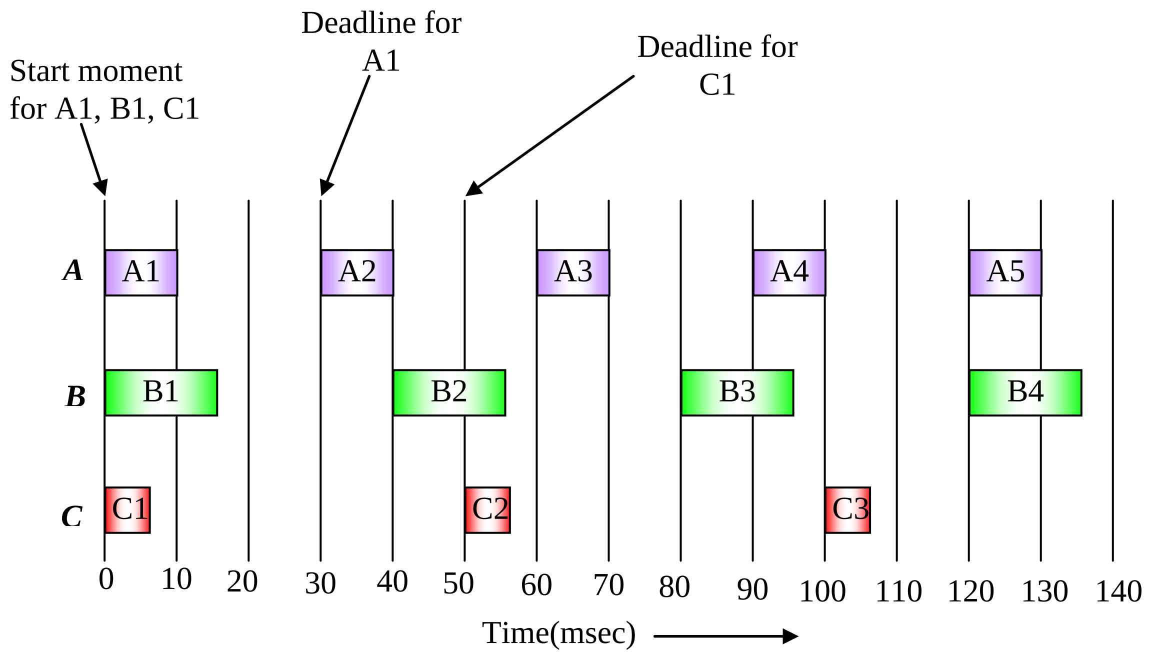


Figure 3-1 Three nonhomogeneous periodic processes each displaying a movie.

Also shown in the figure, two other processes B and C. Process B runs 25 times/sec and process C runs 20 times/sec. The computation time per frame is shown as 15 msec and 5 msec for B and C, respectively, just to make the scheduling problem more general than having all of them the same. The problem is now how to schedule A, B and C to make sure they meet their respective deadlines. Before that, if this set of processes is schedulable. The system schedulable if and only if:

$$\sum_{i=0}^{m} \frac{C_i}{P_i} \le 1$$

Where,

m: is the number of processes

C_i: is the processing time of process i, and
P_i: is the period of process i.

In this case m=3. A is eating 10/30 of the CPU, B is eating 15/40 of the CPU and C is eating 5/50 of the CPU. Together these fractions add to 0.808 of the CPU, so the system is schedulable.

In real-time systems processes might be either preemptable or not. In multimedia processes are generally preemptable to allow the process that is in danger of missing its deadline may interrupt the running process before the running process has finished with its frame. When it is done the previous process can continue. Preemptable real-time algorithms give better performance than nonpreemptable ones. The only concern is that the buffer should not be completely filled with processes.

Real-time algorithms can be either static or dynamic. Static algorithms assign each process a fixed priority in advance and then do prioritized preemptive scheduling using those priorities. Dynamic algorithms do not have fixed priorities. Below an example of each type is studied.[1]

3.2.3 Rate Monotonic Scheduling:

The rate monotonic scheduling algorithm is an optimal, static, priority-driven algorithm for preemptive, periodic jobs. It can be used for processes that meet the following conditions:

- Each periodic process must complete within its period.
- No process is dependent on any other process.
- Each process needs the same amount of CPU time on each burst.
- Any nonperiodic processes have no deadline.
- Process preemption occurs instantaneously and with no overhead.

Static priorities are assigned to tasks, once at the connection set-up phase, according to their request rates. For example, a process that must run every 30 msec (33 times/sec) gets priority 33, a process that must run every 40 msec (25 times/sec) gets priority 25, and a process that must run every 50 msec (20 times/sec) gets priority 20. At run time, the scheduler always runs the highest priority ready process, preempting the running process if need be.

Figure 3-2 shows how rate monotonic scheduling works in the example of Figure 3-1. Processes A, B and C have static priorities, 33, 25 and 20, respectively, which mean that whenever A needs to run, it runs preempting any other process currently using the CPU. Process B can preempt C, but not A. Process C has to wait until the CPU is otherwise idle in order to run.

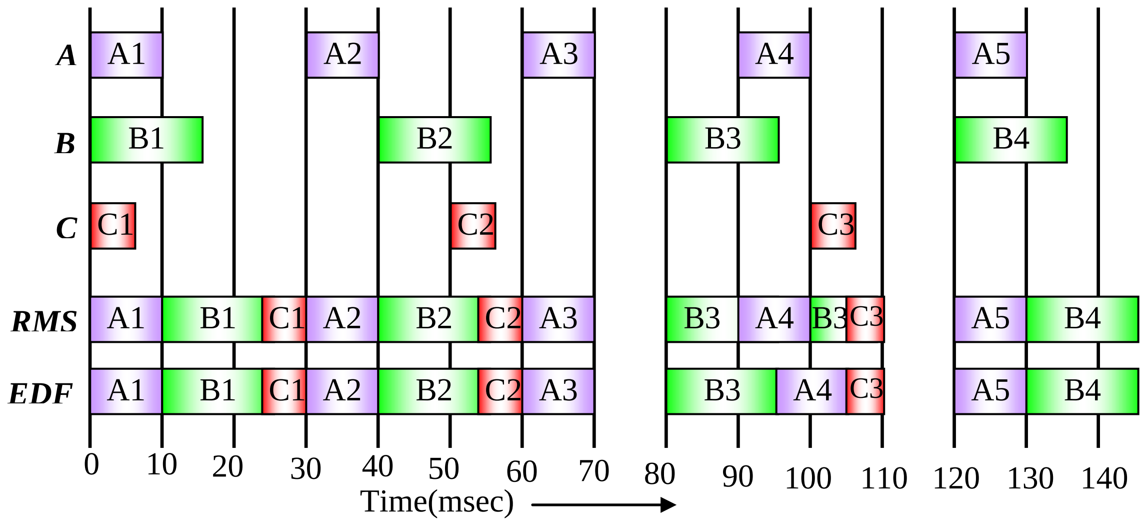


Figure 3-2 An example of RMS and EDF real-time scheduling

In Figure 3-2, initially all three processes are ready to run. The highest priority one, A, is chosen, and allowed to run until it completes at 15 msec, as shown in the RMS line. After it finishes, B and C are run in that order. Together, these processes take 30 msec to run, so when C finishes, it is time for A to run again. This rotation goes on until the system goes idle at t=70.

At t=80, B becomes ready and runs. However, at t=90, a higher process, A, becomes ready, so it preempts B and runs until it is finished, at t=100. At that point the system can choose between finishing B or starting C, so it chooses the highest priority process, B.[1]

3.2.4 Earliest Deadline First Algorithm:

Another popular real-time scheduling algorithm is Earliest Deadline First. EDF is a dynamic algorithm that does not require processes to be periodic, as does the RMS algorithm. Nor does it require the same run time per CPU burst, as does RMS. Whenever a processes needs CPU time, it announces its presence and its deadline. The scheduler keeps a list of runnable processes, sorted on deadline. The algorithm runs the first process on the list, the one with the closet deadline. Whenever a new process becomes ready, the system checks to see if its deadline occurs before that of the currently running process. If so, the new process preempts the current one.

An example of EDF is given in Figure 3-2. Initially all three processes are ready. They are run in the order of their deadlines. A must finish by t=30, B must finish by t=40, and C must finish by t=50, so A has the earliest deadline and thus goes first. Up until t=90 the choices are the same as RMS. At t=90, A becomes ready again, and its deadline is t=120, the same as B's deadline. The scheduler could legitimately choose either one to run, but since in practice, preempting B has some nonzero cost associated with it, it is better to let B continue to run.[1]

3.2.5 RMS vs. EDF:

To dispel the idea that RMS and EDF always give the same result let us look at Figure 3-3 in the next page shows another example. The periods of A, B and C are the same as before, but now A needs 15 msec of CPU time per burst instead of only 10 msec. the schedulability test computes the CPU utilization as

$$0.500+0.375+0.100 = 0.975$$

That means CPU theoretically should be possible to find a legal schedule, and will not oversubscribe.

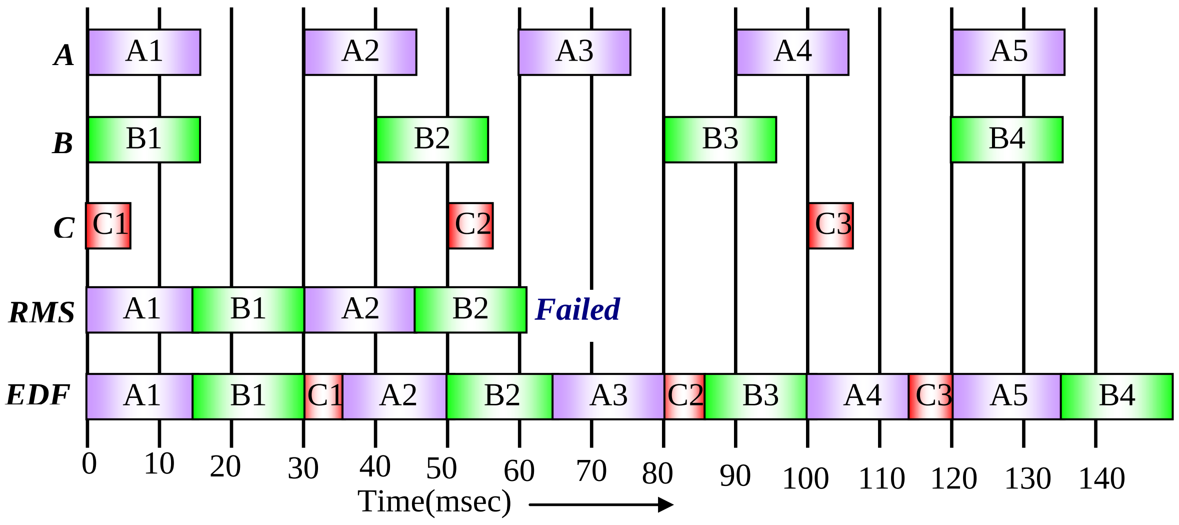


Figure 3-3 Another example of RMS and EDF real-time scheduling

With RMS, the priorities of the three processes are still 33, 25 and 20 as only the period matters, not the run time. This time, B1 does not finish until t=30, at which time A is ready to roll again. By the time A is finished, at t=45, B is ready again, so having a higher priority than C, it runs and C misses its deadline. RMS fails.

An interesting question is why RMS is failed? Basically using static priorities only works if the CPU utilization is not too high. It is proved that for any system of periodic processes, if:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

then RMS is guaranteed to work. For 3, 4, 5, 10, 20, and 100, the maximum permitted utilization is 0.780, 0.757, 0.743, 0.718, 0.705 and 0.696. As $m \to \infty$, the maximum utilization is asymptotic to ln2.

In contrast, EDF always works for any schedulable set of processes. It can achieve 100% CPU utilization.[1]

3.3 MULTIMEDIA FILE SYSTEM PARADIGMS:

The file system model has been discussed in section 1.3.3 does not work well for multimedia on account of the need for real-time behavior. One problem is that the user must make the read calls fairly precisely spaced in time. A second problem is that the video server must be able to supply the data blocks without delay, something that is difficult for it to do when the requests come in unplanned and no resources have been reserved in advance.

To solve these problems, multimedia file servers act like VCRs (Video Cassette Recorders). To read a multimedia file, a user process issues a start system call, specifying the file to be read and various other parameters, such as the sound and subtitles tracks. Server starts to send the frames until the user sends a stop system call.

In order to implement standard VCR control functions, including pause, fast forward, and rewind. Pause is fairly straightforward. The user sends a message back to the video server to tell it to stop. The server should only know which frame goes out next, to continue from when the user instructs it to start again.

True rewind is actually easy. All the server has to do operating systems note that the next frame to be sent is zero. However fast forward and fast backward are much trickier. If it were not for compression, one way to go forward at k times normal speed would require displaying every k^{th} frame.

Compression makes rapid motion either way more complicated. In some techniques each frame is compressed independently of all others, here it is possible to use above strategy. But with MPEG this scheme does not work, due to the use of I-, P-, and B-frames. Skipping ahead k frames might land on a P-frame that is based on an I-frame, this frame becomes useless.

To attack this problem there are many suggestions: the first is to pull data from the disk at k times normal speed, server decompress the frames, and then recompress every k^{th} frame as an I-frame. This approach makes puts a huge load on the server. It also requires the server to understand the compression format.

Another approach is to ship all the data to the user over the network and let the correct frames to be selected out, this require running the network at 10x speed. It is not easy given the high speed at which it normally has to operate.

The only feasible strategy requires advance planning. What can be done is build a special file containing every k^{th} frame, and compress this file with normal MPEG algorithm. To switch fast forward mode, the server must figure out where in the fast forward file the user currently is. Then frames are sent from this file at its normal speed, and then return to the original file. This approach has some disadvantages. First, some extra disk space is required to store the additional files. Second, fast forward and rewinding can only be done at speeds corresponding to the special files. Third, extra complexity is needed to switch back and forth between the regular, fast forward, and fast backward.[1]

3.4 FILE PLACEMENT:

Multimedia files are very large, are often written only once but read many times, and tend to be accessed sequentially. Their playback must also meet strict quality of service criteria. Together these requirements suggest different file system layouts than traditional operating system use.

3.4.1 Placing a File in a Single Disk:

One way to eliminate interfile seeks on video servers is to use contiguous files. Here we need to seek from video file to audio file to text file. Another possible storage arrangement is to interleave the three files as shown in figure 3-4, the entire file is still contiguous; but here, the video for frame 1 is followed directly by the various audio tracks for frame 1 and then the various text tracks for frame 1.

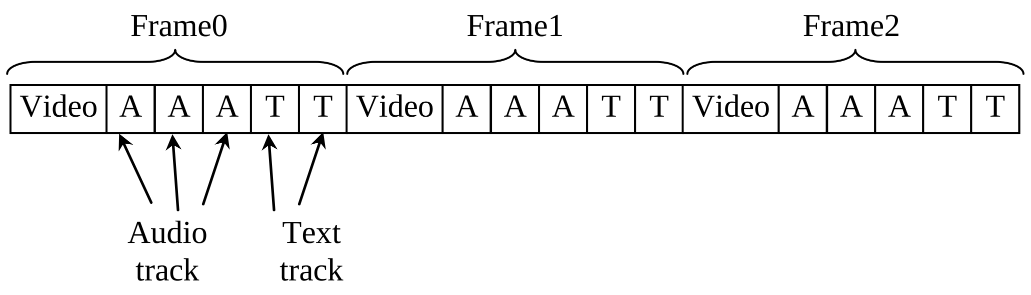


Figure 3-4 Interleaving video, audio, and text in a single contiguous file per movie

This organization requires extra disk I/O for reading in unwanted audio and text, and extra buffer space in memory to store them. However it eliminates all seeks and does not require any overhead for keeping track of which frame is where on the disk since the whole movie is in one contiguous file.[1]

3.4.2 Two Alternative File Organization Strategies:

The above observations lead to two other file placement organizations for multimedia files. Namely **small block model** and **large block model,** these models are shown in figure 3-5, in small block level the disk block is chosen to be considerably smaller than the frame size. There is a frame index for each movie with one entry for each frame pointing to the start of the frame.

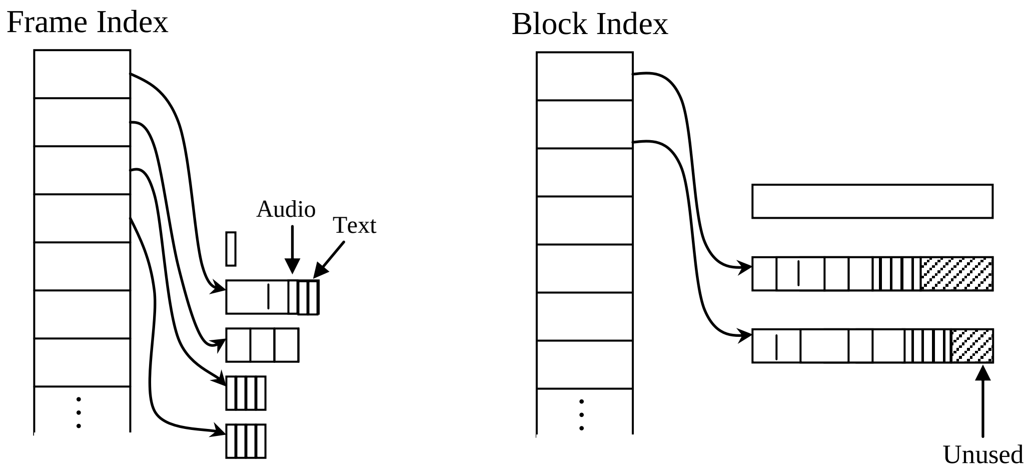


Figure 3-5 Noncontiguous movie storage (a) Small blocks (b) large blocks

In large block model the disk blocks are considerably larger than the frame size, and we have block index rather than frame index. The block can hold more than one frame; because frames have different sizes the block can not hold an integral number of frames. This problem can be solved by either: the rest of the block is just left empty, this wasted space is internal fragmentation, but make the advantage of no seeking in the middle of the frame. Another solution is to split the frame over blocks, this introduces the seeks in the middle of frames, but saves the disk space. Also it is observed that the frame index is larger than the block index. Thus the trade-offs involves RAM usage during playback, wasted disc space all the time, and performance loss during playback due to extra seeks.[1]

3.4.3 Placing Multiple Files on a Single Disk:

In the previous section we have looked only at the placement of a single movie. Actually video sever contains many movies. The idea is to find a way to place files in order to minimize wasted time of moving the disk head from movie to movie.

This situation is improved by observing the relative popularity between movies; this can be held by Zipf's law which states that if the movies are ranked on their popularity, the probability that the next customer will choose the item ranked k-th in the list is C/k, where C is normalization constant. This means that the most popular movie is chosen twice as often as the second most popular movie, three times as often as the third most popular movie, and so on.

Knowing the relative popularities of the different movies makes it possible to model the performance of a video server and to use that information for placing files. **Organ-pipe algorithm** is used to distribute files; it places the most popular movie in the middle of the disk, with the second and third most popular movies on their either side of it, then four and five and so on, as shown in figure 3-6.[1]

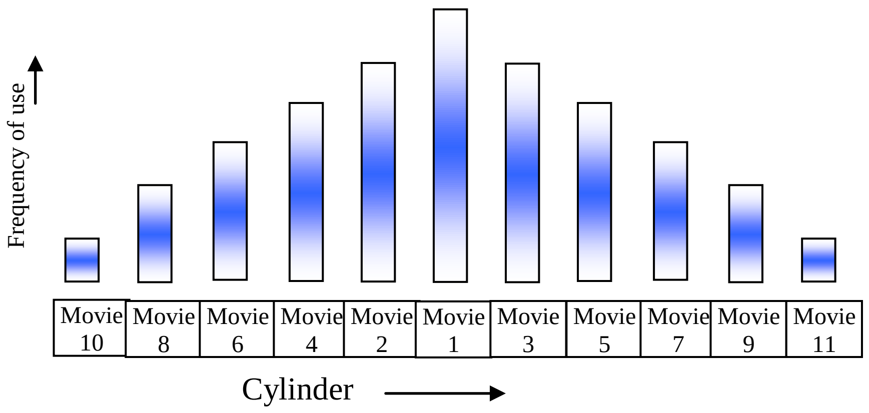
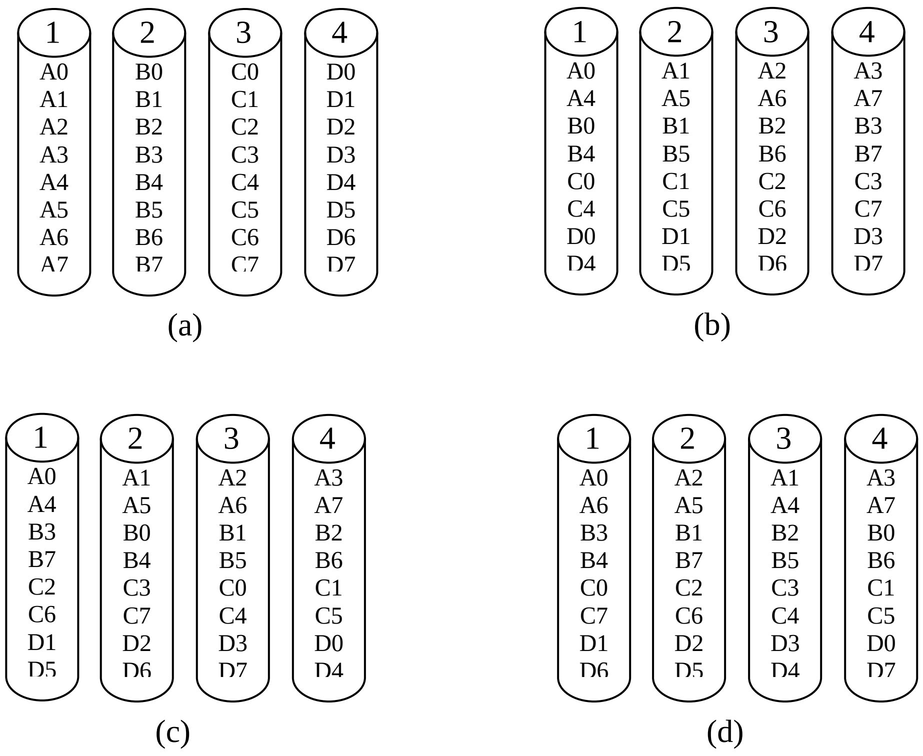


Figure 3-6 Organ-pipe Distribution

Placing Files on Multiple Disks:

Having many disks running in parallel is a good way to increase performance. A common configuration is simply a large number of disks, referred to as a **disk farm.** Here disks do not rotate synchronously and do not contain any parity bits, as RAID do. One possible configuration is to put movie A on disk 1, B on 2, and so on, as shown in figure 3-7(a). This organization is simple and has straight forward failure characteristics. Its disadvantage is that the load may not be well balanced.



(a)
1 A0 A1 A2 A3 A4 A5 A6 A7
2 B0 B1 B2 B3 B4 B5 B6 B7
3 C0 C1 C2 C3 C4 C5 C6 C7
4 D0 D1 D2 D3 D4 D5 D6 D7
(b)
1 A0 A4 B0 B4 C0 C4 D0 D4
2 A1 A5 B1 B5 C1 C5 D1 D5
3 A2 A6 B2 B6 C2 C6 D2 D6
4 A3 A7 B3 B7 C3 C7 D3 D7
(c)
1 A0 A4 B3 B7 C2 C6 D1 D5
2 A1 A5 B0 B4 C3 C7 D2 D6
3 A2 A6 B1 B5 C0 C4 D3 D7
4 A3 A7 B2 B6 C1 C5 D0 D4
(d)
1 A0 A6 B3 B4 C0 C7 D1 D6
2 A2 A5 B1 B7 C2 C6 D2 D5
3 A1 A4 B2 B5 C3 C4 D3 D4
4 A3 A7 B0 B6 C1 C5 D0 D7

Figure 3-7 Four ways to organize multimedia files over multiple disks
(a) No stripping (b) Same stripping pattern for all files
(c) Staggered stripping (d) Random stripping

A second possible organization is to stripe each movie over multiple disks, as shown in figure 3-7(b). Its disadvantage is that because all movies start at the first disk, the load across the disks may not be balanced. One way to balance the load is to stagger the starting disks, as shown in figure 3-7(c). Yet another way to attempt to balance the load is to use a random striping pattern for each file, as shown in figure 3-7(d).[1]

3.5 DISK SCHEDULING:

The third difference between multimedia operating systems and traditional one is the disk scheduling. This is due to multimedia extremely high data rates and real-time delivery of data.

3.5.1 Static Disk Scheduling:

In multimedia, each active stream puts a well defined load on the system that is highly predictable. For example, playback made every 33.3 msec each client wants the next frame in its file and the system has 33.3 msec to provide all the frames the system needs to buffer at least one frame per stream so that the fetching of frame k+1 can proceed in parallel with the playback of frame k.

This predictable load can be used to schedule the disk using algorithms modified to multimedia operation. Assume that there are 10 users, each one viewing a different movie. Furthermore, we will assume that all movies have the same resolution, frame rate, and other properties.

Depending on the rest of the system, the computer may have 10 processes, one per video stream, or one process with 10 threads, or even one process with one thread that handles the 10 streams in round-robin fashion. The details are not important. What is important is that time is divided up into rounds, where a round is the round time (e.g. 33.3 msec). At the start of each round, one disk request is generated on behalf of each user, as shown in figure 3-8.

After all the requests have come in at the start of the round, the disk knows what it has to do during that round. It also knows that no other requests will come in until these have been processed and the next round has begun. Consequently, can sort the requests in the optimal way, probably in cylinder order and then process them in the optimal order. In Figure 3-8, the requests are shown sorted in cylindrical order.

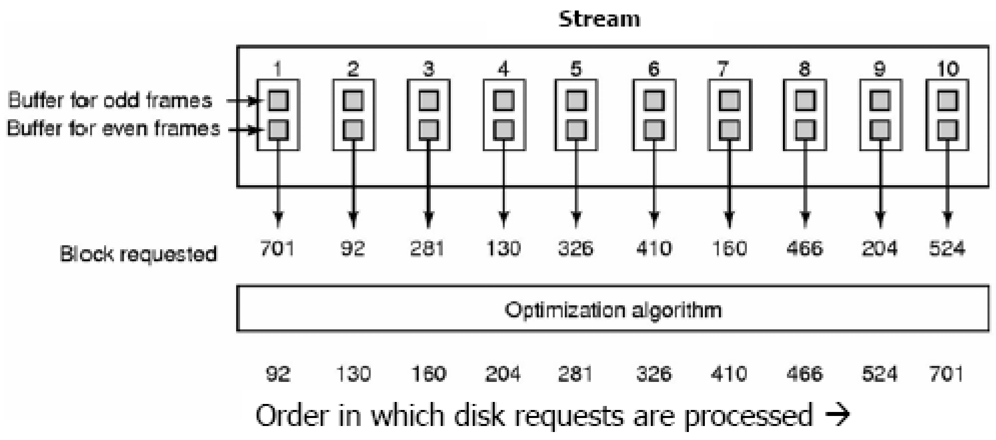


Figure 3-8 In one round, each movie asks for one frame

At first glance, one might think that optimizing the disk in this way has no value because as long as the disk meets the deadline, it does not matter if it meets it with 1 msec to spare or 10msec to spare. However, this conclusion is false. By optimizing seeks in this fashion, the average time to process each request is diminished, which means that the disk can handle more streams per round on the average. In other words, optimizing disk requests like this, increases the number of movies the server can transmit simultaneously. Extra time at the end of the round can also bee used to service any nonreal-time requests that may exist.

If a server has too many streams, once in a while when it is asked to fetch frames from distant parts of the disk and miss a deadline. But as long as missed deadlines are enough, they can be tolerated in return for handling more streams at once. Note that what matters is the number of streams being fetched. Having two or more clients per stream does not affect disk performance or scheduling.

To keep the flow of data out to the clients moving smoothly, double buffering is needed in the server. During round 1, one set off buffers is used, one buffer per stream. When the round is finished, the output process or processes are unblocked and told to transmit frame 1. At the same time, new requests come in for frame 2 of each movie (there might be a disk thread and output thread for each movie). These requests must be satisfied using a second set of buffers, as the first ones are still busy. When round 3 starts, the first set of buffers are now free and can be reused to fetch frame 3.[1]

3.5.2 Dynamic Disk Scheduling:

It was assumed that all streams have the same resolution, frame index and other properties. In fact, different movies may have different data rates, so it is not possible to have one round every 33.3 msec and fetch one frame for each stream. Requests come in to the disk more or less at random.

Each read request specifies which block is to be read and in addition at what time the block is needed, that is, the deadline. For simplicity, it is assumed that the actual service time for each request is the same. In this way we can subtract the fixed services time from each request to get the latest time the request can initiated and still meet the deadline. This makes the model simpler because what the disk scheduler cares about is the deadline for scheduling the requests.

When the system starts up, there are no disk requests pending. When the first request comes in, it is serviced immediately. While the first seek is taking place, other requests may come in, so when the first request is finished, the disk driver may have a choice of which request to process next. Some request is chosen and started. When that request is finished, there is again a set of possible requests: those that were not chosen the first time and the new arrivals that came in while the second request was being processed. In general, whenever a disk request completes, the driver has some set of requests pending from which it has to make a choice. The question is "What algorithm does it use to select the next request to service?"

Two factors play a role in selecting the next disk request: deadlines and cylinders. From a performance point of view, keeping the requests sorted on cylinder and using the elevator algorithm minimizes total seek time, but may cause requests on outlying cylinders to miss their deadline. From a real-time point of view, sorting the requests on deadline and processing them in deadline order, earliest deadline first, minimizes the chance of missing deadlines, but increases total seek time.

These factors can be combined using scan-EDF algorithm. The basic idea of this algorithm is to collect requests whose deadlines are relatively close together into batches and process these in cylinder order. As an example, consider the situation of Figure3-9 at t=700. The disk driver knows it has 11 requests pending for various deadline and various cylinders. It could decide, for example to treat five requests with the earliest deadline as batch, sort them on cylinder number and use the elevator algorithm to service these in cylinder order. The order would then be 110, 330, 440, 676 and 680. As long as every request is completed before its deadline, the requests can be safely rearranged to minimize the total seek time required.

When different streams have different data rates, a serious issue arises when a new customer shows up: should the customer be admitted. If admission of the customer will cause other streams to miss their deadlines frequently, the answer is probably no. there are two ways to calculate weather to admit the new customer or not. One way is to assume that each customer needs a certain amount of resources on the average, for example, disk bandwidth, memory buffers, CPU time, etc. If enough of each left for an average customer, the new one is admitted.[1]

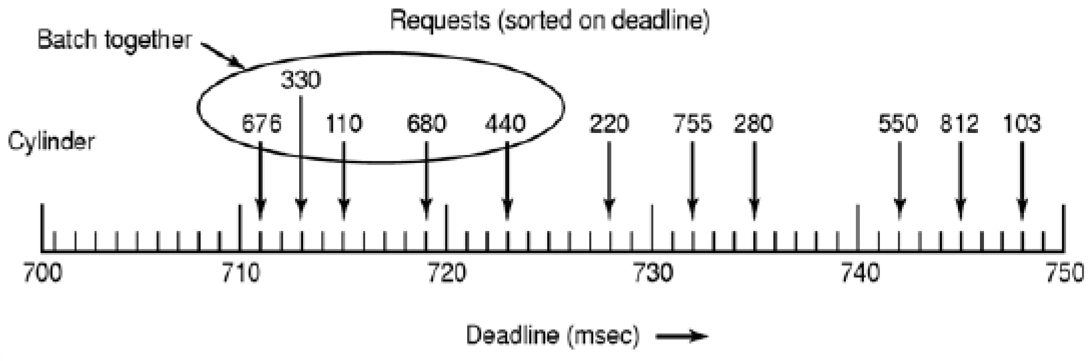


Figure 3-9 The scan-EDF algorithm uses deadlines and cylinder numbers for scheduling

As we have seen in section 1.2.1 video on demand (VOD) and video editing are two of the most popular applications of multimedia operating systems, this of course in addition to playing multimedia files and synchronize between them. In this chapter a design of a program that demonstrates the above applications is shown.

4.1 Program specifications:

Firstly, According to video on demand, it has been assumed that there are three types of files resident on the disk: a single silent movie file, multiple sound files with multiple languages, and multiple text files also with different languages. The program has to get the video file which the user wants to play, the sound language in which the user wants to hear the sound in, also it gets the text file in which the user wants to see the subtitles in. Then the program should play the video and sound files, and show the subtitles periodically one segment at a time (the segment used is a line). The most important requirement while playing these files is to synchronize between them, so that if the user scrolls forward or backward, then all files go to the correct positions, i.e. the correct frame is displayed, the correct sound should be heard and the correct text is shown.

Secondly, according to video editing; the program has to let the user to input another video file and wait for the user to instruct it to replace the previous video file by the new one from the point, keeping the sound and text files as they are, and reserving the synchronization. While playing the data should be transferred to a single file. The interface is expected to be as shown in figure 4-1

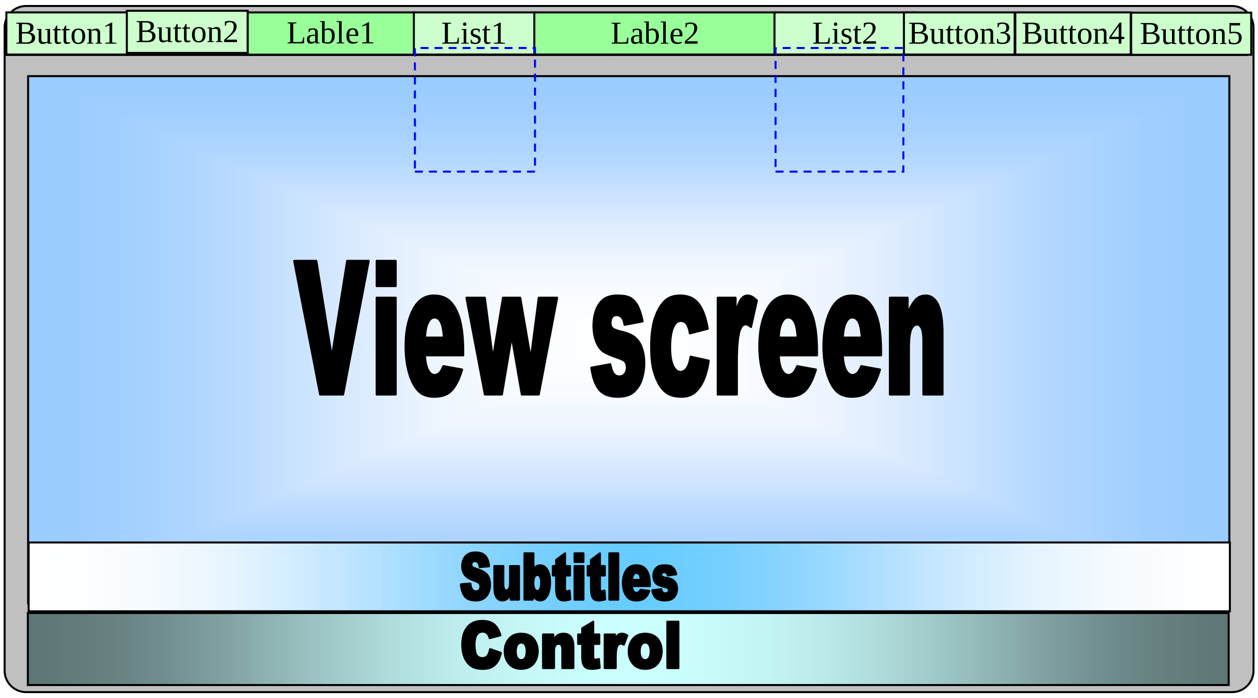


Figure 4-1 program interface elements

Button1: allows the user to choose a video file.

Button2: allows the user to choose another video file.

List1: contains language that can be used for sound stream.

List2: contains language that can be used for subtitles.

Lable1 and lable2: tells the user about the contents of list1 and list2.

View screen: the area where the video will be displayed.

Subtitles: Where the text will be displayed.

Control: control components from where the user can scroll forward and backward.

4.2 Program Design:

The program is written in JAVA for two main reasons:

1. As we have seen in the previous section, that we want to work on three files concurrently and read from them at the same time, this which means in programming jargon *multithreading*, that means we have to use a language that support this concept. Java is unique among popular general-purpose programming language is that it makes concurrency primitives available to the applications programmer. Each thread designating a portion of a program that may execute concurrently with other threads. *Multithreading* gives the Java programmer powerful capabilities not available in C and C++ the languages on which Java is based[9]. In our application we have to control each file with a different thread.
2. [Java has a good support for multimedia applications, java has been built in multimedia capabilities and many languages are not, Development of powerful multimedia applications is faster.[10] A very powerful package called JMF (Java Media Framework) which contains very good functions that work on multimedia files.

The following algorithms and the subsequent flowcharts illustrate the main steps that are done by the program, algorithm 1 shows that. Algorithm 2 shows the steps followed by the program if the user instructs it to switch between video movies.

Algorithm 1:

```
1.  Start.
2.  Input first video file.
3.  Input the second video file.
4.  Choose sound stream language.
5.  Get the .wav file that contains the sound in the language that specified in 4.
6.  Input the subtitles stream language.
```

7. Get the .txt file contains the text in the language that specified in 6.
8. Check if there are current playing files.
   - If yes: remove the visual and control components of the players and remove the subtitles.
   - If no: go to 9.
9. Create player for the first video file.
10. Create player for sound file.
11. Get visual and control components of the created player in 9.
12. Open the txt file that got in 7 for reading.
13. Count the lines of the file opened in 12.
14. Determine the first video file length in seconds.
15. Calculate sleep time = $\left(\frac{\textit{total}\ \text{media time}}{\textit{number}\ \text{of the lines}} \right)$
16. Let the sound player created in 10 controls the video player created in 9 to achieve the synchronization between sound and video.
17. While there is a playing movie do the following:
    17.1 Get the current player time.
    17.2 The appropriate line number = $\textit{round}\left(\frac{\textit{current}\ \text{time}}{\textit{sleep}\ \text{time}} \right)$
    17.3 Open the text file.
    17.4 Get the appropriate line and display it.
18. End.

Algorithm 2 (switching between movies):

If one of the buttons: "play 1st" or "play 2nd" the following algorithm is used:

1. Start.
2. Get time of the current playing player.
3. Get the video file for which the button is pressed.
4. Create player for the video that gotten in 3.
5. Let the sound player created in 10 in algorithm 1 controls the video player created in 4 to achieve the synchronization between sound and video.
6. Return to 17 in algorithm 1.
7. End.

The flowchart of algorithm 1 is shown in figure 4.2 and the flowchart of algorithm 2 is shown in figure 4.3. The code is appended in the appendix.

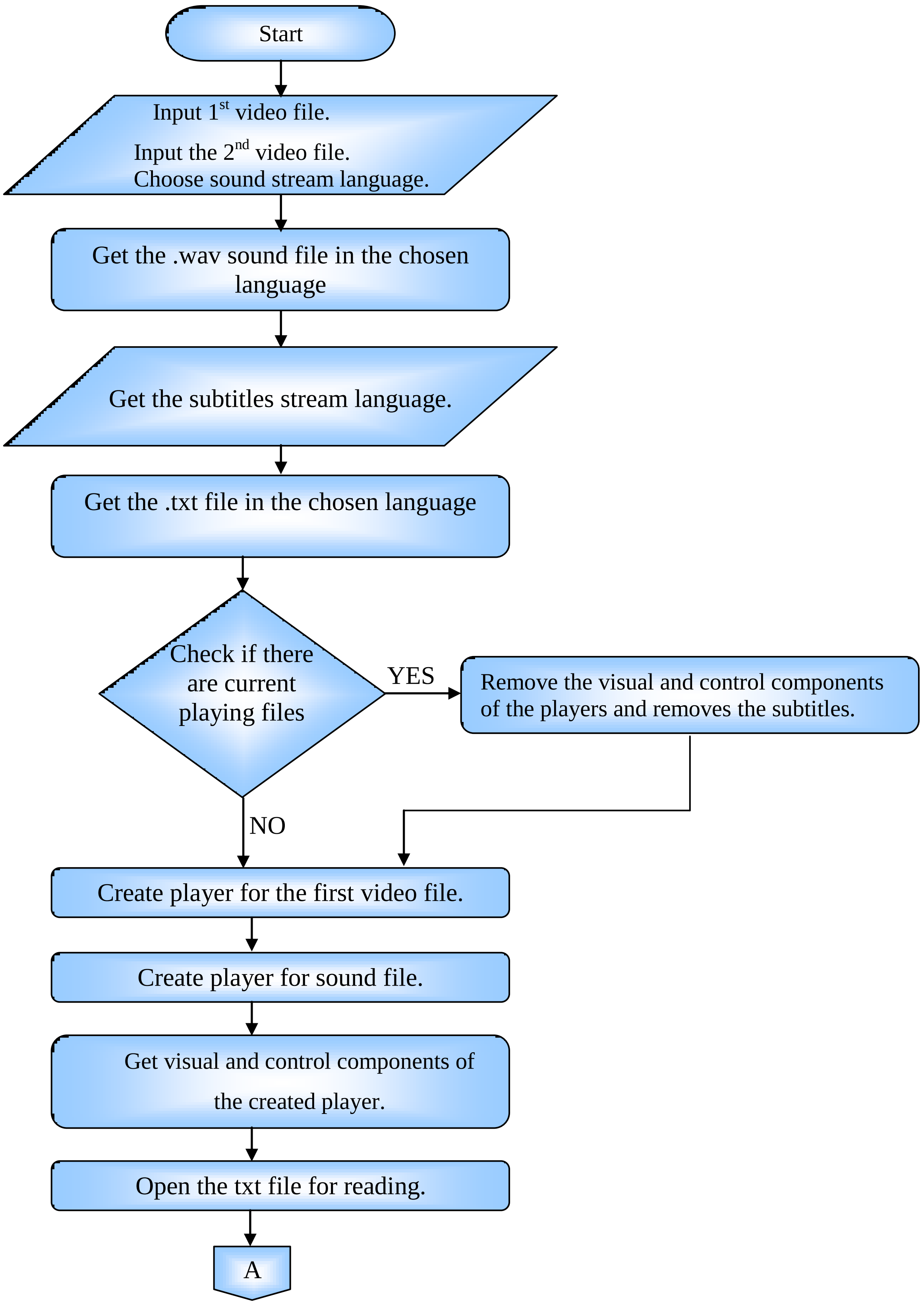
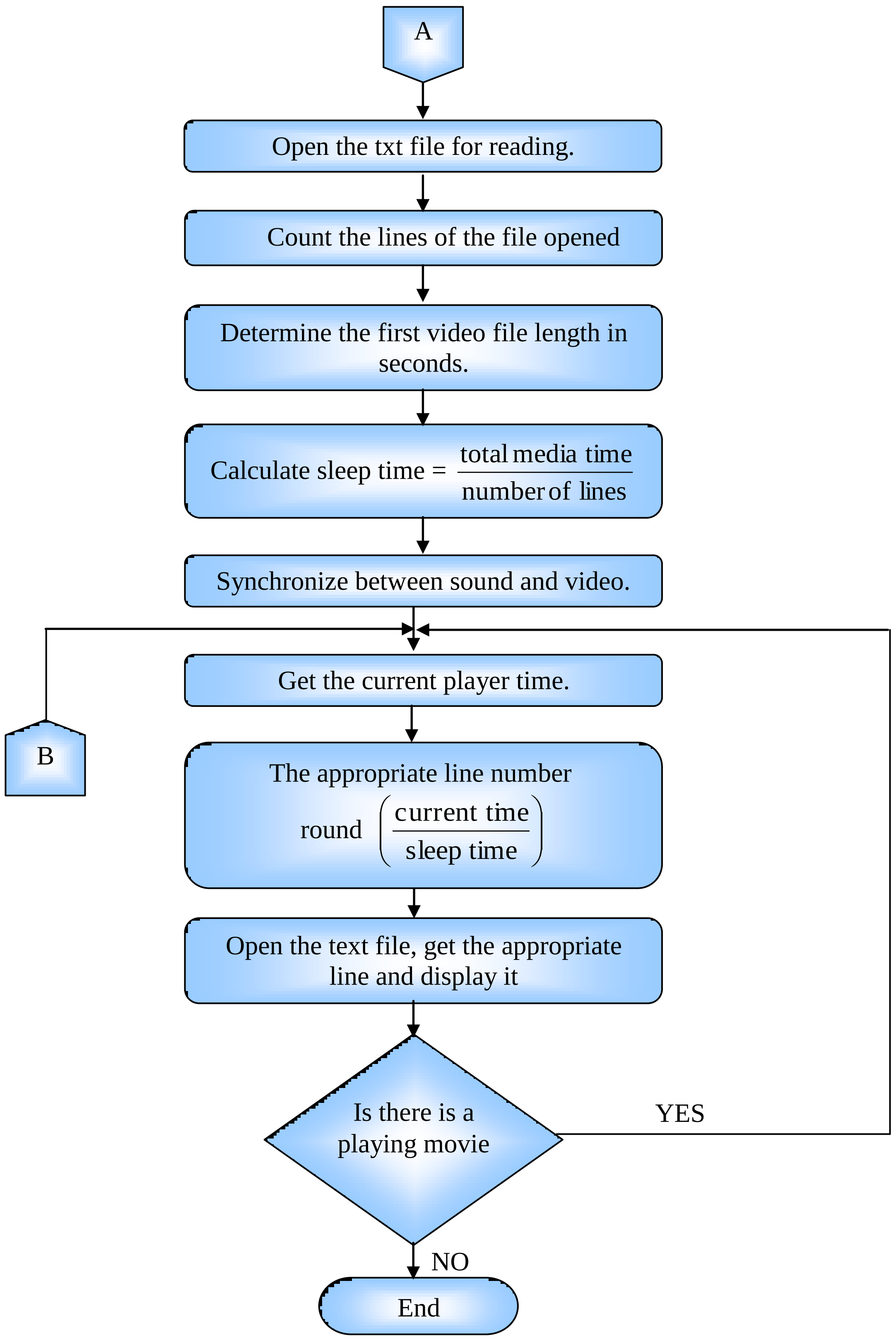

Start
Input 1st video file.
Input the 2nd video file.
Choose sound stream language.
Get the .wav sound file in the chosen language
Get the subtitles stream language.
Get the .txt file in the chosen language
Check if there are current playing files
YES
Remove the visual and control components of the players and removes the subtitles.
NO
Create player for the first video file.
Create player for sound file.
Get visual and control components of the created player.
Open the txt file for reading.
A

Figure 4.2 algorithm 1 flowchart



A
Open the txt file for reading.
Count the lines of the file opened
Determine the first video file length in seconds.
Calculate sleep time = $\frac{\text{total media time}}{\text{number of lines}}$
Synchronize between sound and video.
Get the current player time.
B
The appropriate line number
round $\left(\frac{\text{current time}}{\text{sleep time}}\right)$
Open the text file, get the appropriate line and display it
Is there is a playing movie
YES
NO
End

Figure 4.2 (continued) Algorithm 1 flowchart

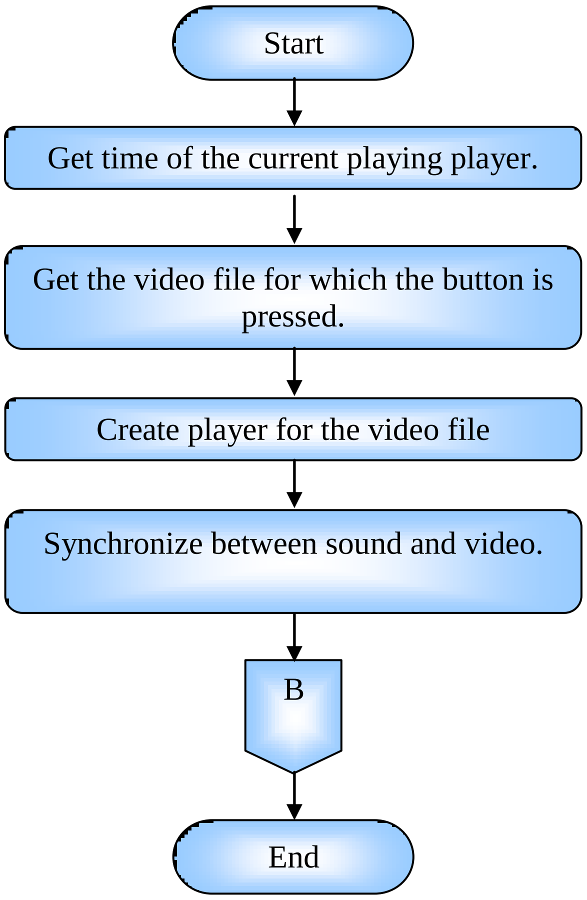

Start
Get time of the current playing player.
Get the video file for which the button is pressed.
Create player for the video file
Synchronize between sound and video.
B
End

Figure 4.3 Algorithm 2 flowchart

In this chapter the screen captures of the program that discussed and designed in the previous chapter are shown. Also some discussions are demonstrated about the results and problems are faced.

Figure 5-1 shows the interface of the program,"1st video" and "2nd video" buttons enabled, "play 1st", "play 2nd" and "stop" buttons are disabled. "Sound stream language" and "subtitles language" lists also are displayed.

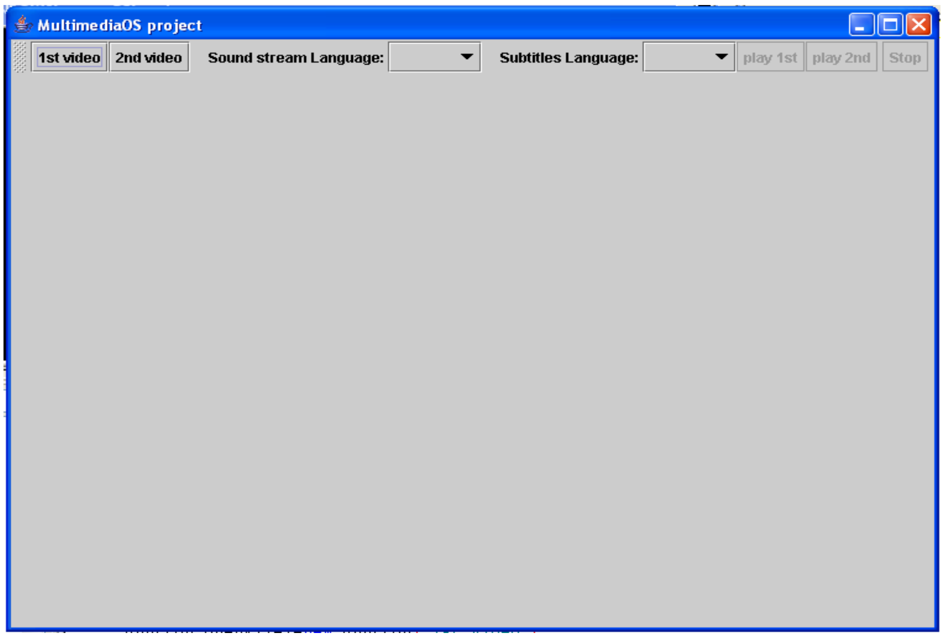


Figure 5-1 Program interface

Figure 5-2 shows the files as it appear in an example folder, "video1.mpg" and "video2.mpg" are silent video files. "English.wav","عربي.wav" and "French.wav" are audio files. "English.text","عربي.text" and "French.text" are text files.

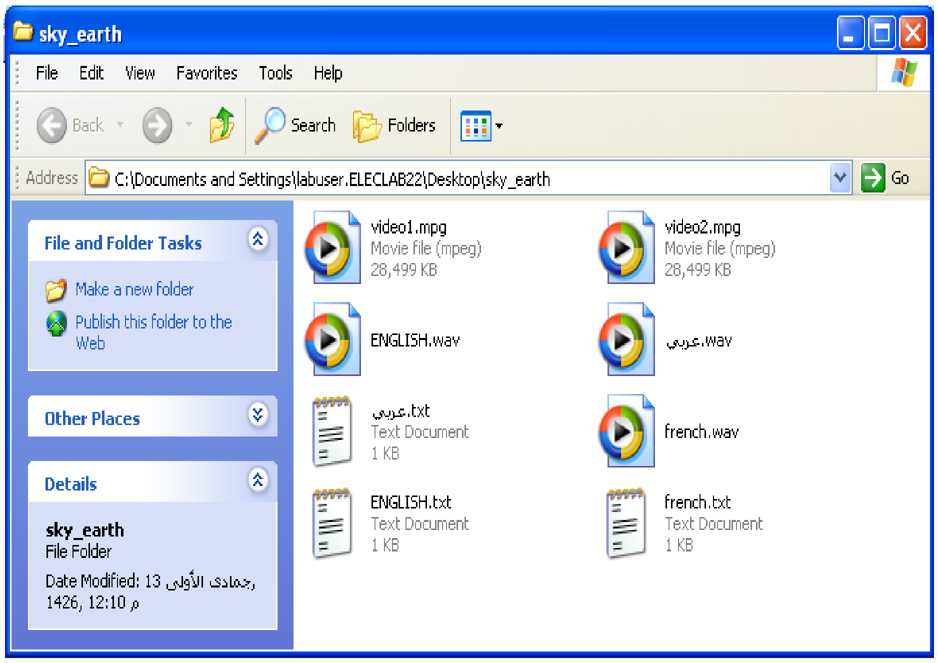


Figure 5-2 Example files

Figure 5-3 shows the response of the program when "1st video" or "2nd video" buttons are pressed. The sub window "open" is displayed.

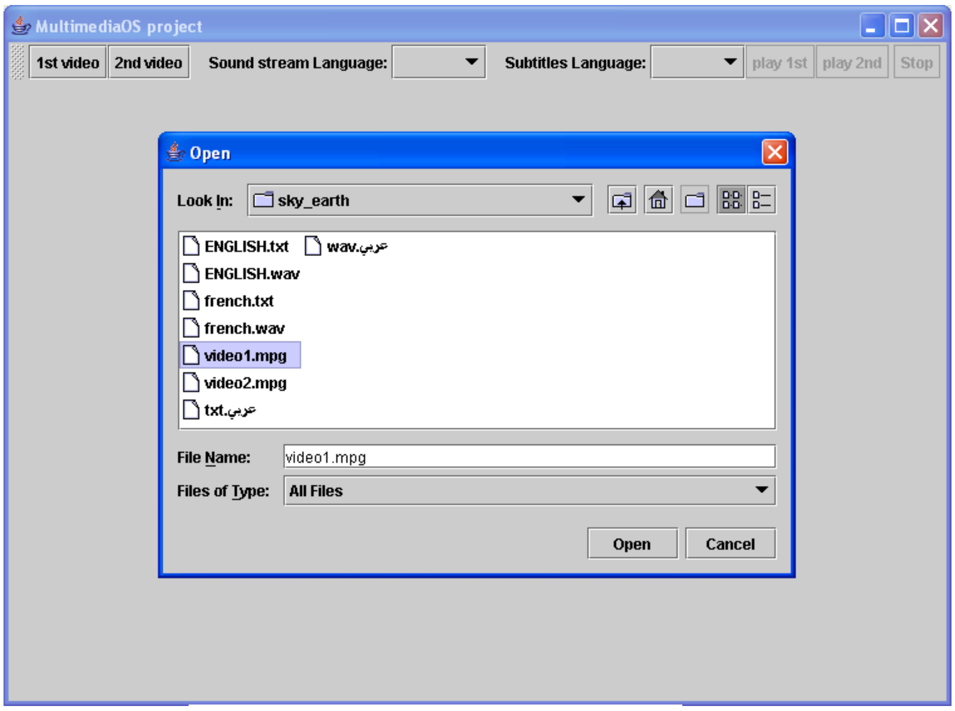


Figure 5-3 Open window

Figure 5-4 and 5-5 show the "sound stream language" and "subtitles language" lists.

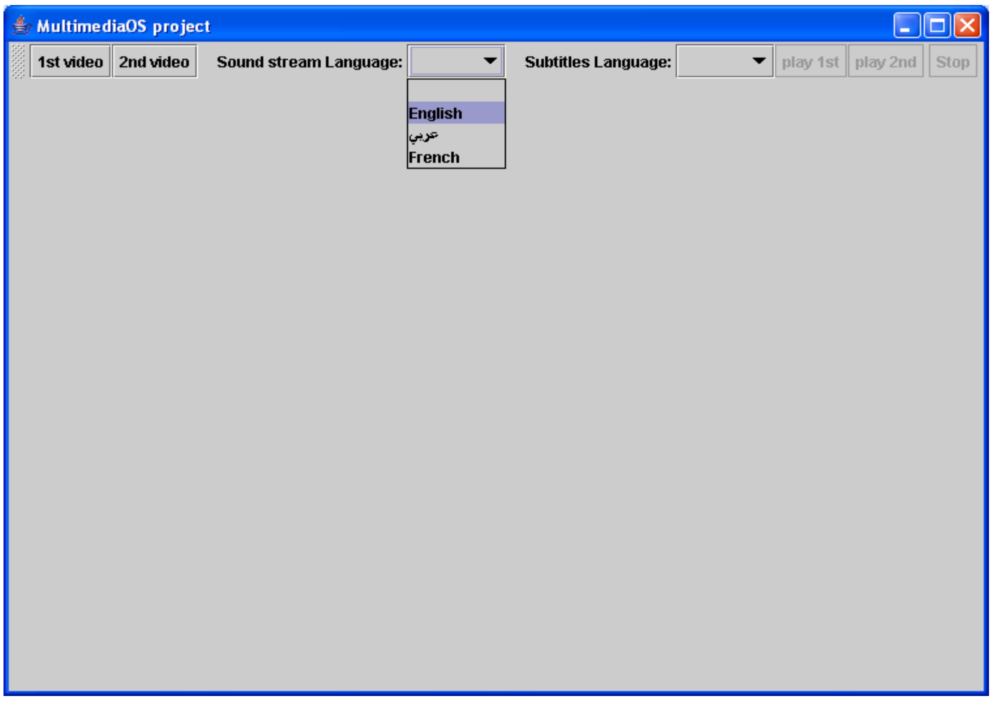


Figure 5-4 Sound language lists

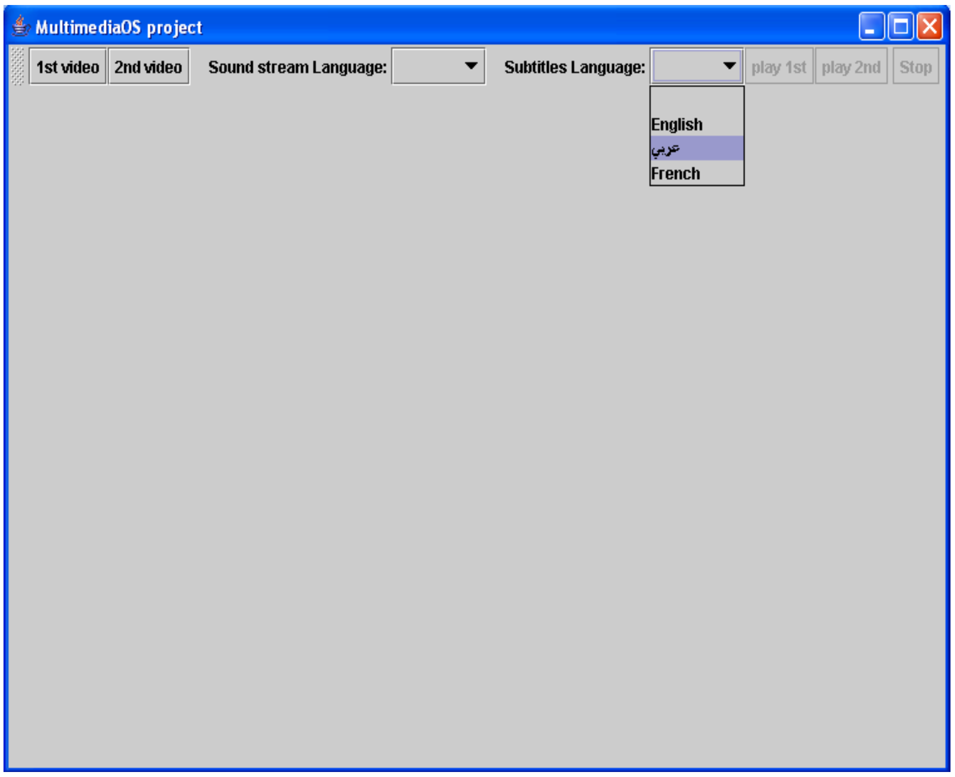


Figure 5-5 Subtitles language list

Figure 5-6 shows a movie when it is played synchronized with sound and subtitles.

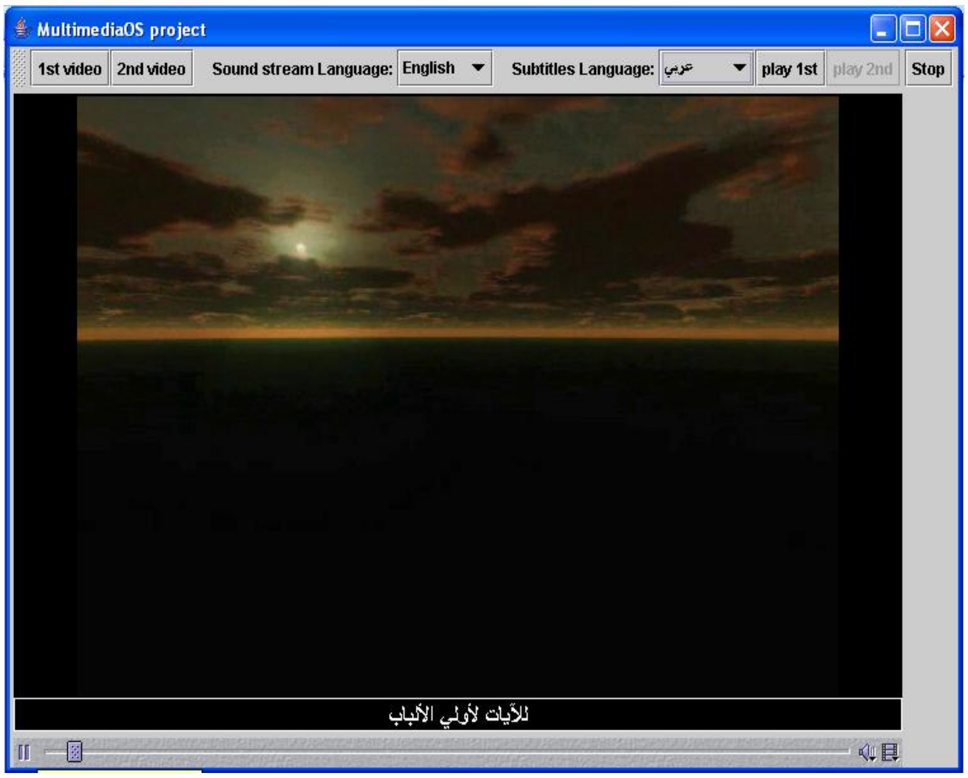


Figure 5-6 A video file is playing

Figure 5-7 shows the synchronization of the video, audio and subtitles after scrolling forward.

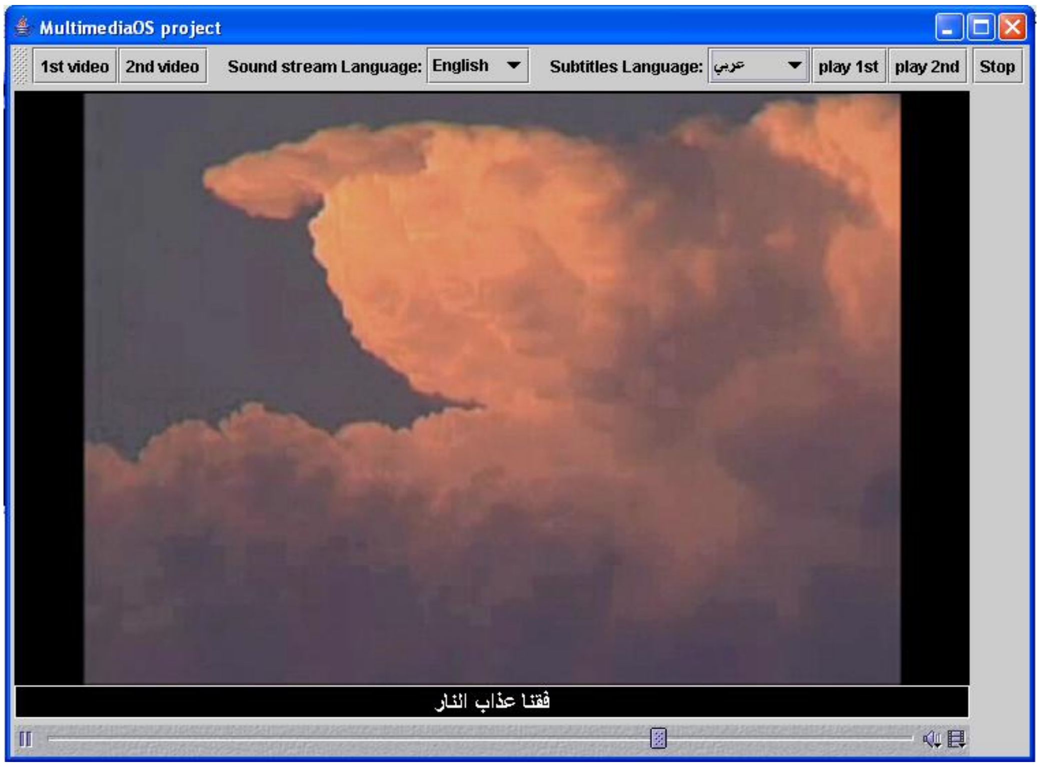


Figure 5-7 Movie after scrolling forward

Figure 5-8 shows a movie with another subtitle language, at another instance of the movie. The buttons "play 1st","play 2nd" and "stop" are enabled.

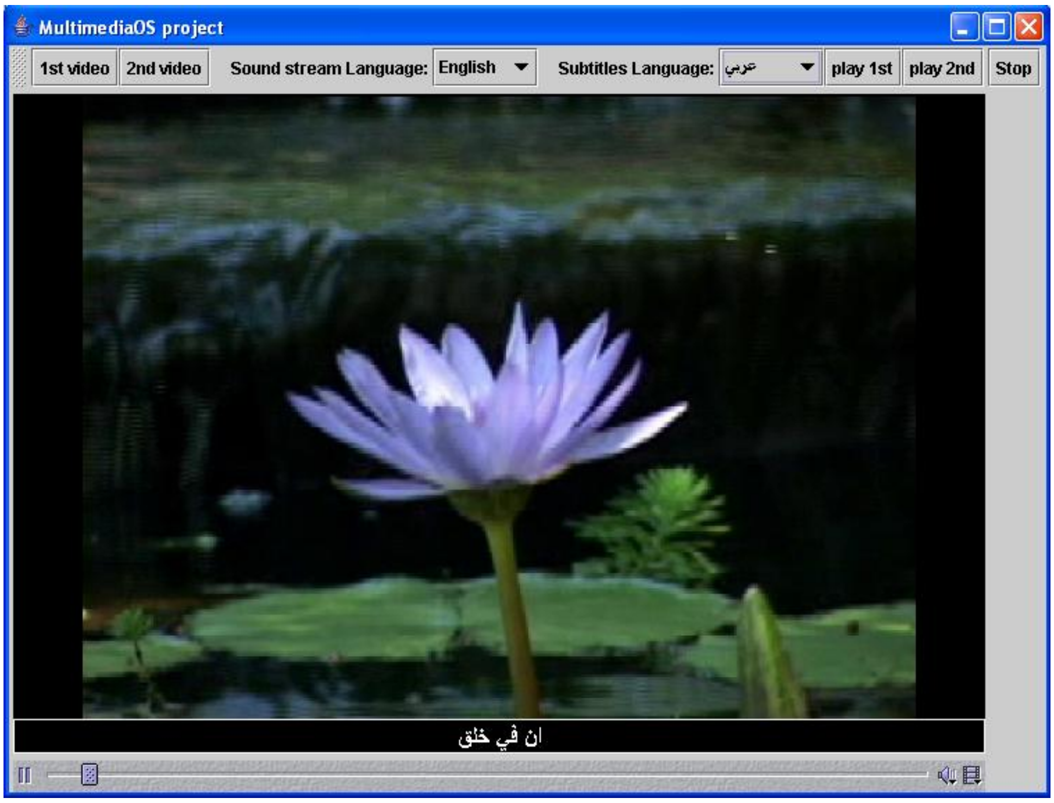
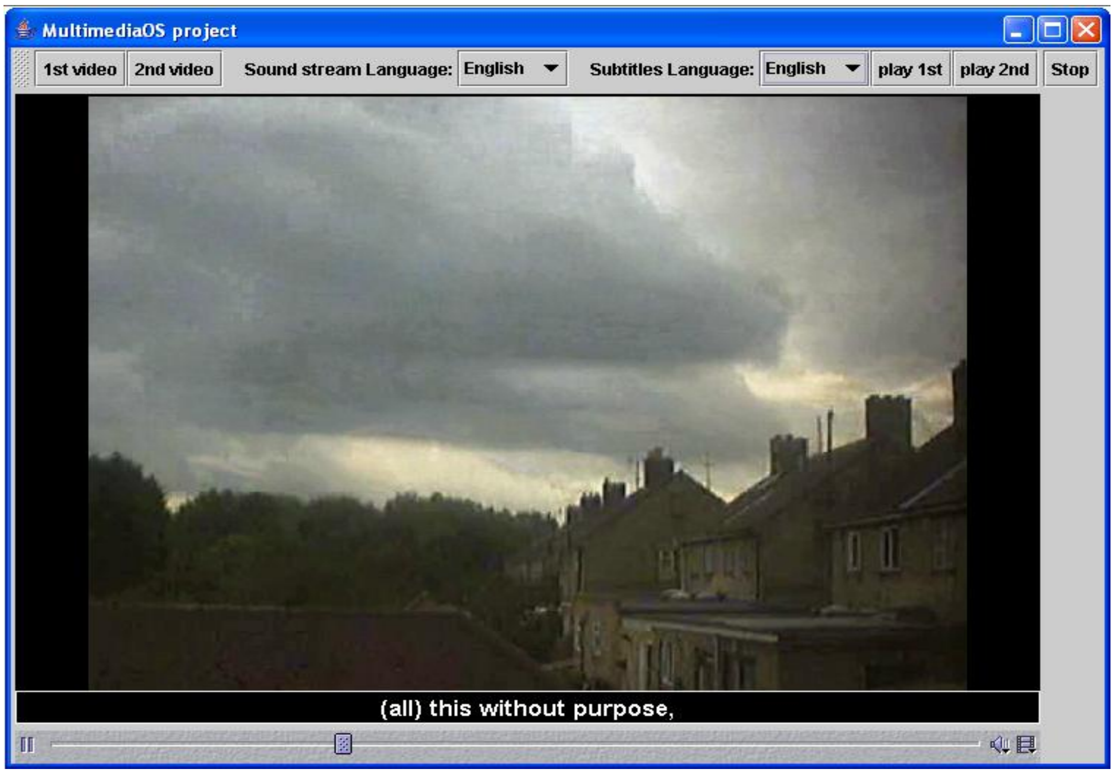


Figure 5-8 Movie after scrolling backward

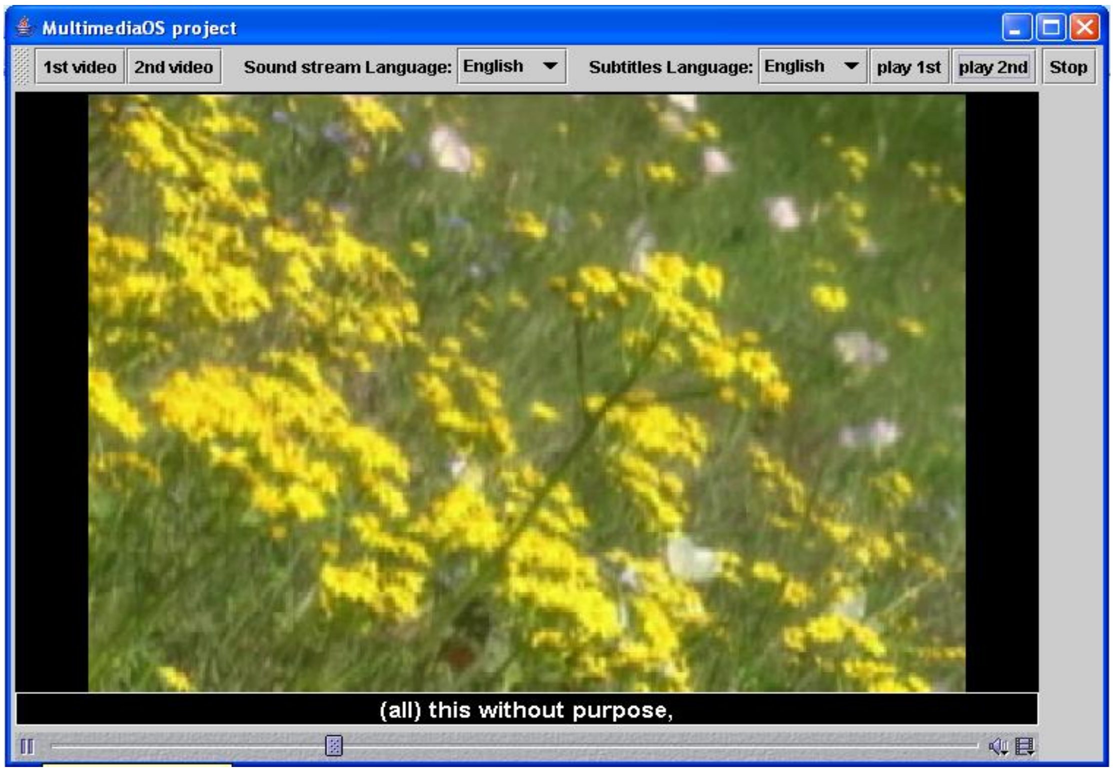
Figure 5-9 shows the movie with another subtitles language and before pressing "play 2nd" button.



(all) this without purpose,

Figure 5-9 First video is playing

Figure 5-10 shows the program after pressing "play 2nd", it is observed that the position of the scroller, and the text are the same. Also the sound is the same.



(all) this without purpose,

Figure 5-10 Second video is playing

Also we can return to the first movie if the second one is playing, figure 5-11 shows an instance of the second movie, and figure 5-12 shows the same instance in the first movie after pressing "play 1st" button.

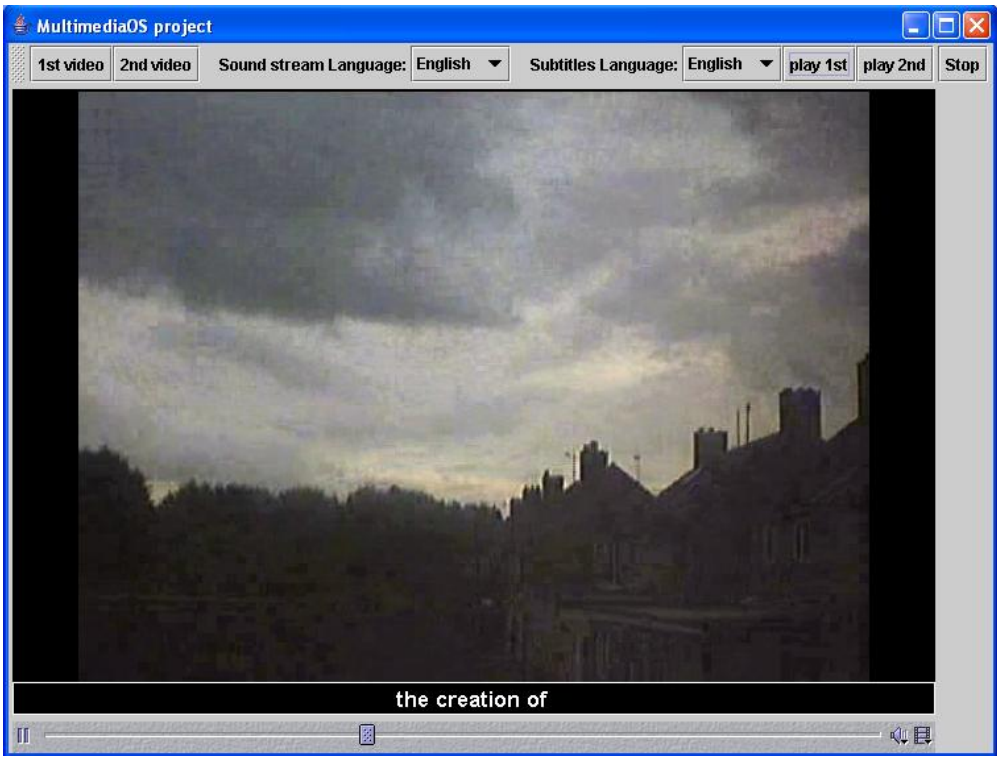


Figure 5-11 Second video is playing

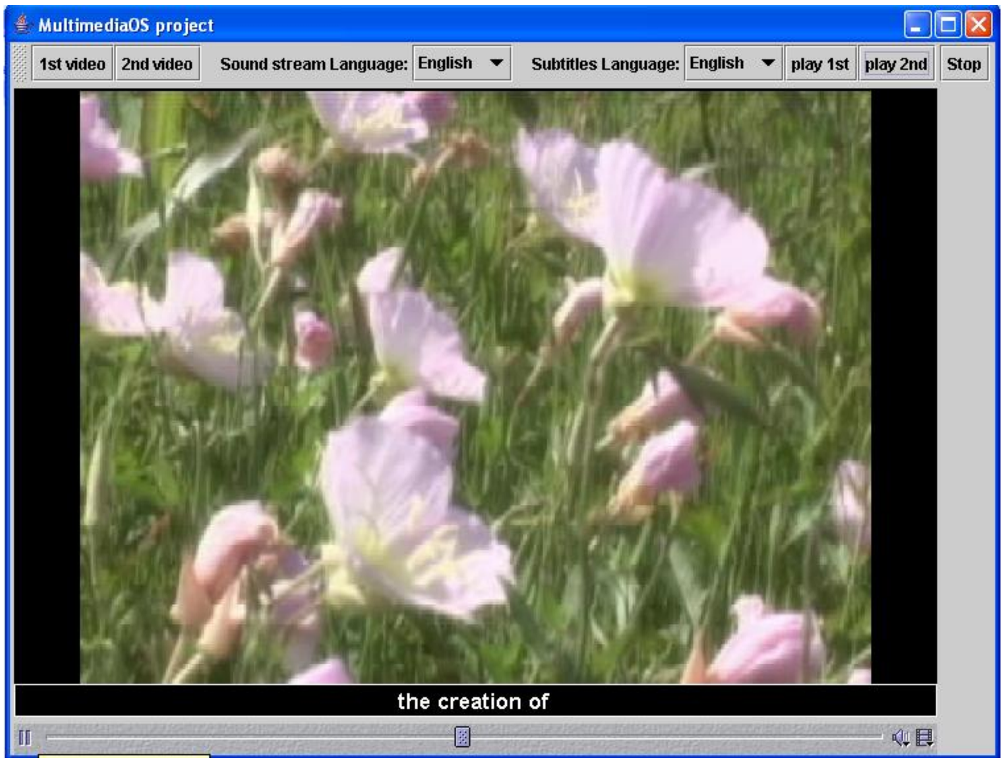


Figure 5-12 First video is playing

Figure 5-13 shows the program frame after pressing "stop" button.

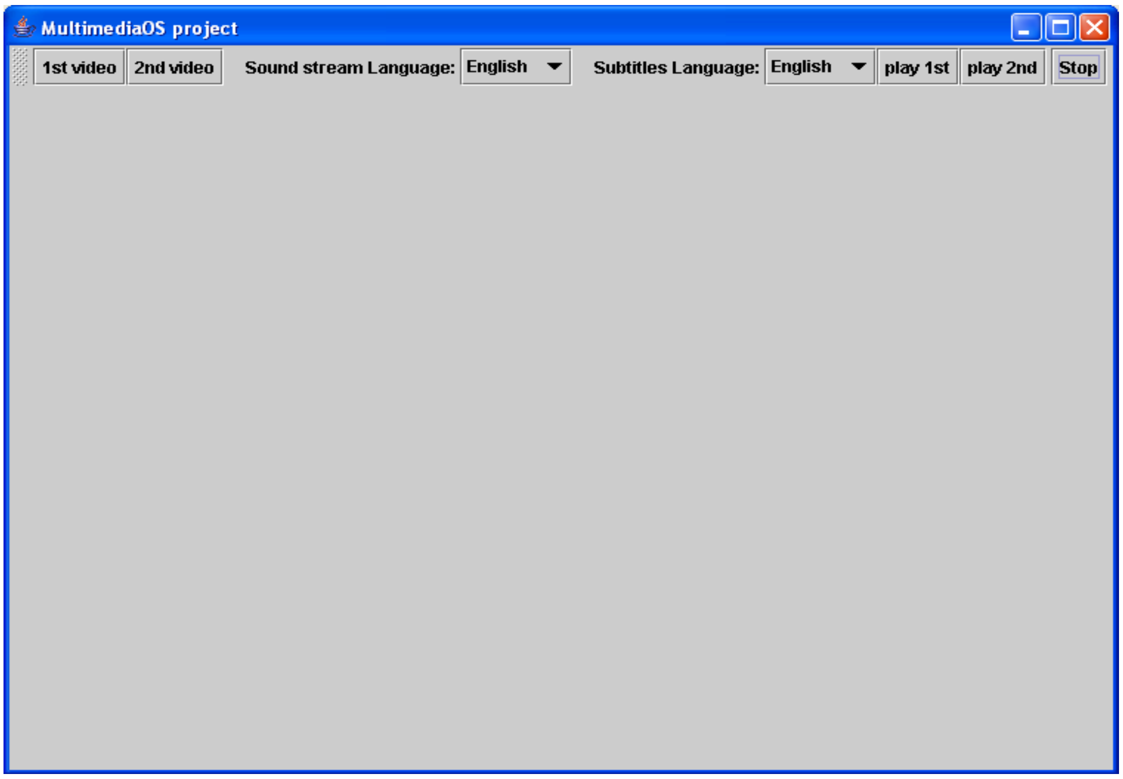


Figure 5-13 After pressing "stop"

Discussions:

The program is written under Windows operating system which provides many operating system tasks such as: process management, memory management and secondary storage management. Then the program provides some tasks that are held by the multimedia operating systems, namely: playing multimedia files, control the playing of files and synchronization between three files.

The above figures are showing screen captures of the program in some instances when playing sample files. It can be observed that there is synchronization between the three opened files: movie, audio and text. Also switching between video files is shown with the same sound and text files. The video formats that can be played are: .MPEG, .AVI and .WME. The sound file format supported is: .WAV . The text format is .TXT.

This can reflect some of the multimedia operating system tasks. The tasks that reflected are: the ability of the multimedia operating system to play video and sound files. Also the ability of dealing with multimedia file system which has different characteristics that distinguish it from ordinary file system.

The problem that is faced, is when switching between two video files a flicker is seen, it is about 150-250 msec, which is noticeable by the human being eye. The instance of the switching is shown in figure 5-14 in the next page.

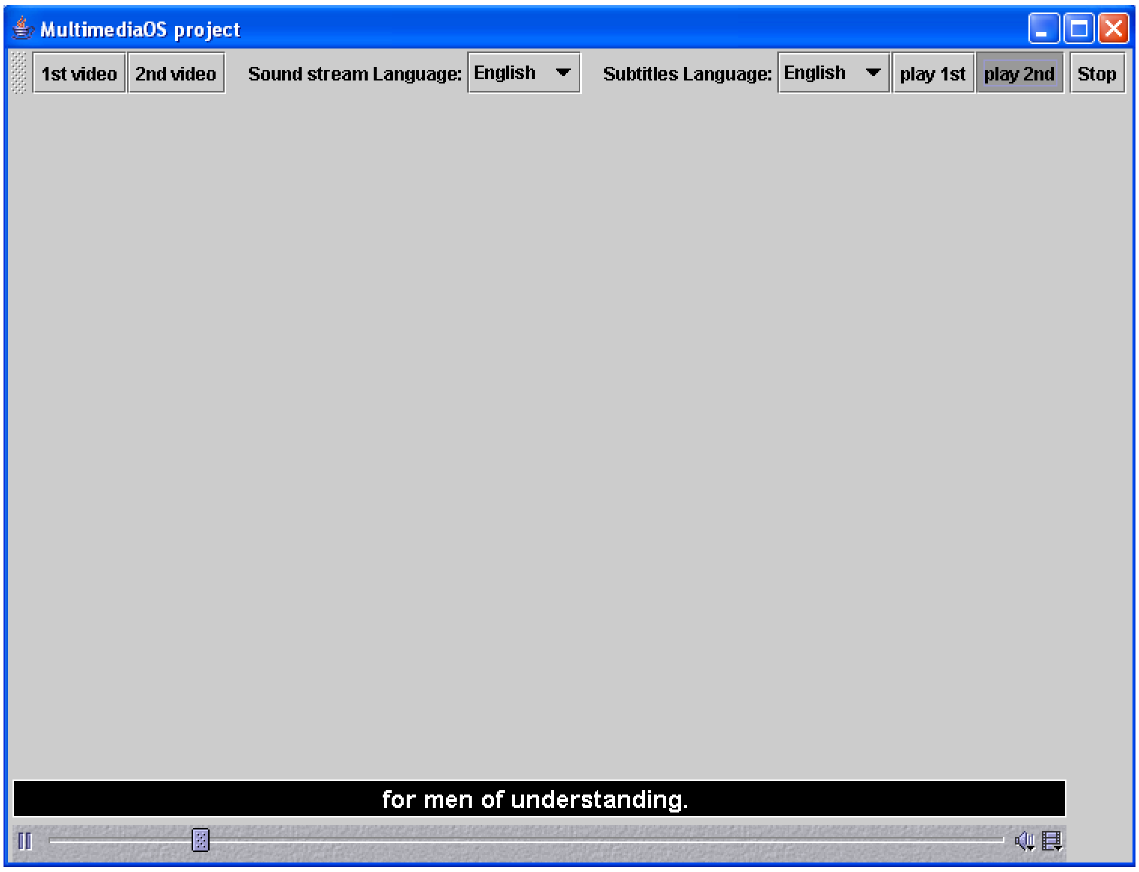


Figure 5-14 Flickering

The reason of the flickering is that: in JMF each player steps over many states until it is displayed:

- Unrealized state: when a player is just created.
- Realizing state: the player is switched to this state after calling realize() state. Here the resources required by the player are defined.
- After the realization then the player can start playing by calling start() method.

The realizing state takes time. During this time there is no something displayed in the viewing screen [11].

The operating system is the most important program that can function as an extended machine and a resource manager. It has many important concepts to gain the highest utilization of computer resources.

Multimedia has special characteristics that distinguish it from ordinary text files. Multimedia uses extremely high data rates and requires real-time playback. These characteristics must be taken into account when dealing with it.

Multimedia operating systems are those their primary function is to handle multimedia. In the previous chapters, a discussion about multimedia operating systems is held. Multimedia operating systems differ from traditional ones in three main ways: process scheduling, file system and disk scheduling.

In process scheduling: multimedia operating systems use Rate Monotonic Scheduling and Earliest Deadline First scheduling algorithms, to meat the time requirements of the process. The second difference is the file system, and the organization of placing files and to access them. The third is disk scheduling where multimedia operating systems apply static and dynamic disk scheduling algorithms to speed up the data access and achieve the wanted data rates.

A program that plays video and audio files and displaying subtitles read from text file has been designed and implemented. The program does all that in synchronization. Also the program allows switching between two movies selected by the user.

It has been stated that when the user switches between two movies, no picture appears on the view screen for a while. This flickering is noticeable by the user.

RECOMMENDATIONS FOR FUTURE WORK:

It is recommended that this project to be improved by others in the future through these points:

1. Trying to remove the flickering problem that has been discussed in chapter 5, this can be done by improving the Java multimedia package.
2. Try increasing the file formats that can be played by the program specially the video and sound files.
3. Due to multimedia editing, the program is hoped to transfer the data of the played files to another single file is consisted of chosen instances of the two video files to be played independently.

4. Also it can be added that the program let the user to switch between two different sound files. Also the number of video files used can be determined by the user himself.
5. Two videos can be played at the same time and the same view screen, the program can also mix sound files at the same time.
6. As we have seen the program depends on an existing operating system that make the ordinary operating system functions to it, it is hoped to use the concepts has been discussed to develop an independent operating system, by introducing operating systems concepts.

REFERENCES

1. Andrew Tanenbaum; **Modern Operating Systems**, 2nd ed., Prentice Hall, 2001.
2. Abraham Silberschatz, Peter Baer Galvin; **Operating System Concepts,** 5th ed. Addison Wesley Longman Inc., 1998.
3. Ling Guan, Sun-Yuan Kung, Jan Larsen; **Multimedia Image and Video Processing**, CRC Press, 2000.
4. IBM Journal of Research and Development Mar 1998 www.findarticles.com/p/articles/mi_qa3751.
5. Gregory K. Wallace; **The JPEG still picture compression standard**, April 1991/vol.34 No.4
6. chanae.walon.org/pub/docs/technical.info/codec1.htm
7. http://www.cs.rpi.edu/academics/courses/fall04/os/c20/
8. www.cairo.cs.uiuc.edu/~klara/os-vol2.pdf
9. H. M. Deitel, P. J. Deitel, Harvey M. Deitel, Paul J. Deitel; **Java How to Program,** 5th ed., Prentice Hall, 2002.
10. www. zeus.it.uom.gr/pdp/teaching/ deitel/chtp3e24to30ppt/C_chap30.ppt
11. www.java.sun.com/products/java-media/jmf/2.1.1/guide/understandingJMF.html