System Programming
Loaders and Linkers

Introduction:
In this chapter we will understand the concept of linking and loading. As discussed earlier the source program is converted to object program by assembler. The loader is a program which takes this object program, prepares it for execution, and loads this executable code of the source into memory for execution.

Definition of Loader:
Loader is utility program which takes object code as input prepares it for execution and loads the executable code into the memory. Thus loader is actually responsible for initiating the execution process.

Functions of Loader:
The loader is responsible for the activities such as allocation, linking, relocation and loading
1) It allocates the space for program in the memory, by calculating the size of the program. This activity is called allocation.

2) It resolves the symbolic references (code/data) between the object modules by assigning all the user subroutine and library subroutine addresses. This activity is called linking.

3) There are some address dependent locations in the program, such address constants must be adjusted according to allocated space, such activity done by loader is called relocation.

4) Finally it places all the machine instructions and data of corresponding programs and subroutines into the memory. Thus program now becomes ready for execution, this activity is called loading.

Loader Schemes:
Based on the various functionalities of loader, there are various types of loaders:
1) **"compile and go" loader:** in this type of loader, the instruction is read line by line, its machine code is obtained and it is directly put in the main memory at some known address. That means the assembler runs in one part of memory and the assembled machine instructions and data is

directly put into their assigned memory locations. After completion of assembly process, assign starting address of the program to the location counter. The typical example is WATFOR-77, it's a FORTRAN compiler which uses such "load and go" scheme. This loading scheme is also called as "assemble and go".

Advantages:

• This scheme is simple to implement. Because assembler is placed at one part of the memory and loader simply loads assembled machine instructions into the memory.

Disadvantages:

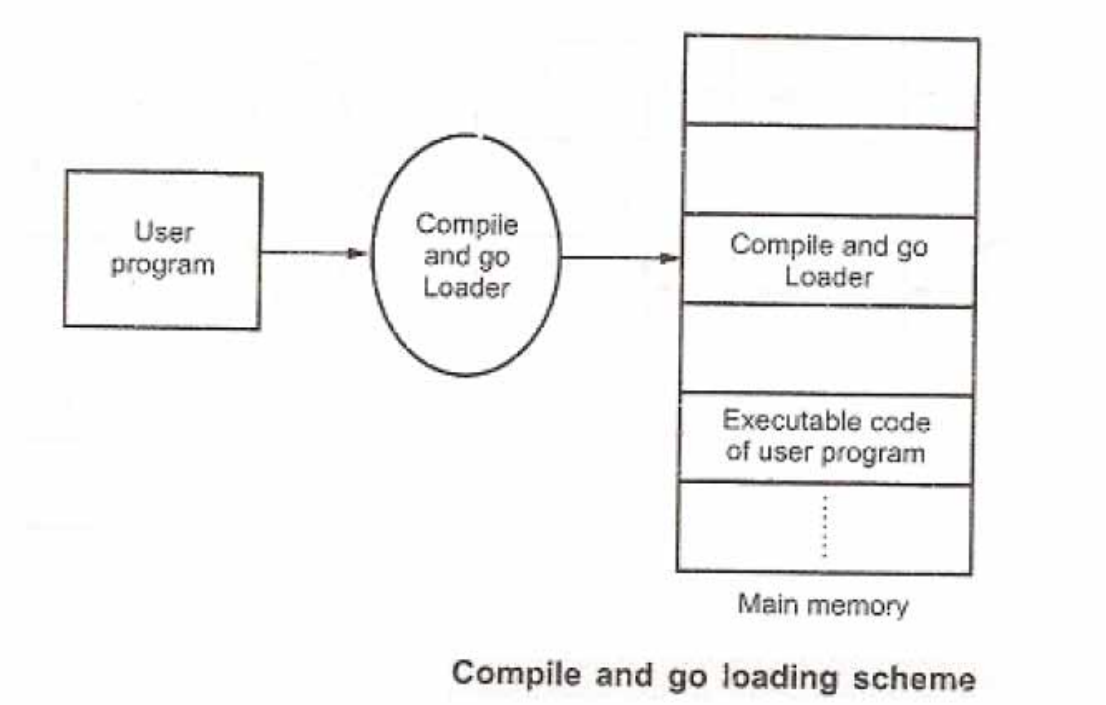
• In this scheme some portion of memory is occupied by assembler which is simply a wastage of memory. As this scheme is combination of assembler and loader activities, this combination program occupies large block of memory.

• There is no production of .obj file, the source code is directly converted to executable form. Hence even though there is no modification in the source program it needs to be assembled and executed each time, which then becomes a time consuming activity.

• It cannot handle multiple source programs or multiple programs written in different languages. This is because assembler can translate one source language to other target language.

• For a programmer it is very difficult to make an orderly modulator program and also it becomes difficult to maintain such program, and the "compile and go" loader cannot handle such programs.

• The execution time will be more in this scheme as every time program is assembled and then executed.



Compile and go loading scheme

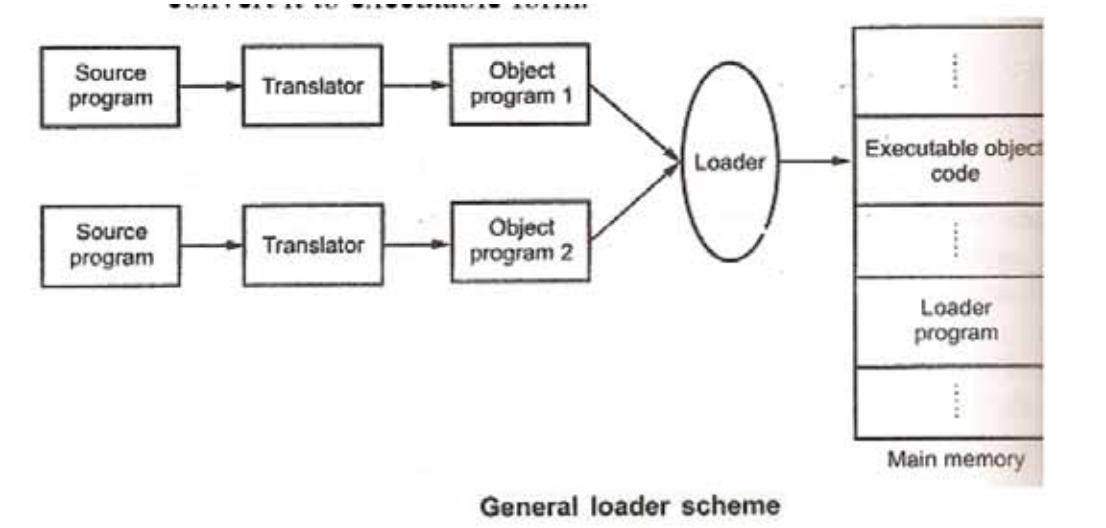
2) **General Loader Scheme:** in this loader scheme, the source program is converted to object program by some translator (assembler). The loader accepts these object modules and puts machine instruction and data in an executable form at their assigned memory. The loader occupies some portion of main memory.

Advantages:

• The program need not be retranslated each time while running it. This is because initially when source program gets executed an object program gets generated. Of program is not modified, then loader can make use of this object program to convert it to executable form.

• There is no wastage of memory, because assembler is not placed in the memory, instead of it, loader occupies some portion of the memory. And size of loader is smaller than assembler, so more memory is available to the user.

• It is possible to write source program with multiple programs and multiple languages, because the source programs are first converted to object programs always, and loader accepts these object modules to convert it to executable form.



General loader scheme

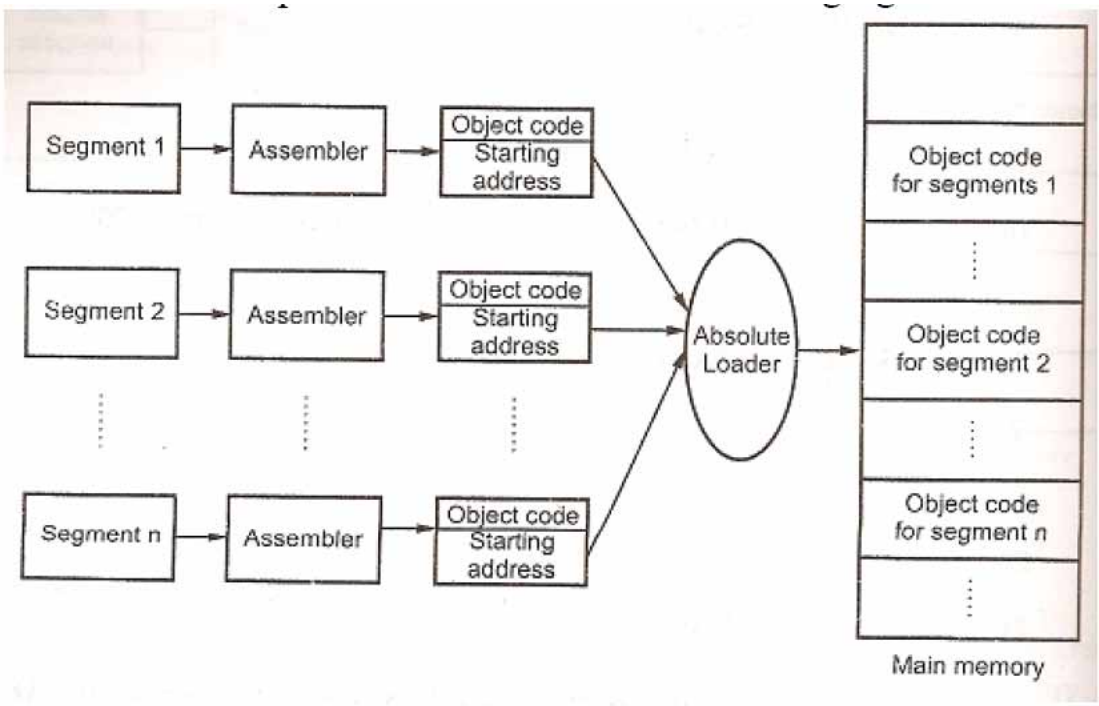
3) **Absolute Loader:** Absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at specified locations in the memory. This type of loader is called absolute because no relocation information is needed; rather it is obtained from the programmer or assembler. The starting address of every module is known to the programmer, this corresponding starting address is stored in the object file, then task of loader becomes very simple and that is to simply place the executable form of the machine instructions at the locations mentioned in the object file. In this scheme, the programmer or assembler should have knowledge of memory management. The resolution of external references or linking of different subroutines are the issues which need to be handled by the programmer. The programmer should take care of two things: first thing is : specification of starting address of each module to be used. If some modification is done in some module then the length of that module may vary. This causes a change in the starting address of immediate next . modules, its then the programmer's duty to make necessary changes in the starting addresses of respective modules. Second thing is ,while branching from one segment to another the absolute starting address of respective module is to be known by the programmer so that such address can be specified at respective JMP instruction. For example

```
Line number
1          MAIN          START    1000
.                          .
.                          .
.                          .
1                        JMP      5000
16                       STORE    ;instruction at location 2000
                         END
```

```
1                       SUM     START 5000
2
20                      JMP     2000
21                      END
```

In this example there are two segments, which are interdependent. At line number 1 the assembler directive START specifies the physical starting address that can be used during the execution of the first segment MAIN. Then at line number 15 the JMP instruction is given which specifies the physical starting address that can be used by the second segment. The assembler creates the object codes for these two segments by considering the stating addresses of these two segments. During the execution, the first segment will be loaded at address 1000 and second segment will be loaded at address 5000 as specified by the programmer. Thus the problem of linking is manually solved by the programmer itself by taking care of the mutually dependant dresses. As you can notice that the control is correctly transferred to the address 5000 for invoking the other segment, and after that at line number 20 the JMP instruction transfers the control to the location 2000, necessarily at location 2000 the instruction STORE of line number 16 is present. Thus resolution of mutual references and linking is done by the programmer. The task of assembler is to create the object codes for the above segments and along with the information such as starting address of the memory where actually the object code can be placed at the time of execution. The absolute loader accepts these object modules from assembler and by reading the information about their starting addresses, it will actually place (load) them in the memory at specified addresses.

The entire process is modeled in the following figure.



Process of absolute loading

Thus the absolute loader is simple to implement in this scheme-
l) Allocation is done by either programmer or assembler
2)Linking is done by the programmer or assembler
3)Resolution is done by assembler
4)Simply loading is done by the loader
As the name suggests, no relocation information is needed, if at all it is required then that task can be done by either a programmer or assembler

Advantages:

1. It is simple to implement
2. This scheme allows multiple programs or the source programs written different languages. If there are multiple programs written in different languages then the respective language assembler will convert it to the language and a common object file can be prepared with all the ad resolution.
3. The task of loader becomes simpler as it simply obeys the instruction regarding where to place the object code in the main memory.
4. The process of execution is efficient.

Disadvantages:

1. In this scheme it is the programmer's duty to adjust all the inter segment addresses and manually do the linking activity. For that, it is necessary for a programmer to know the memory management.
If at all any modification is done the some segments, the starting addresses of immediate next segments may get changed, the programmer

has to take care of this issue and he needs to update the corresponding starting addresses on any modification in the source.

Algorithm for absolute Loader

Input: Object codes and starting address of program segments.

Output: An executable code for corresponding source program. This executable code is to be placed in the main memory

Method: Begin

```
For each program segment
 do Begin
        Read the first line from object module to
        obtain information about memory location. The
        starting address say S in corresponding object
        module is the memory location where executale
        code is to be placed.
        Hence
        Memory_location = S
        Line counter = 1; as it is first line While (!
        end of file)
        For the curent object code
do Begin
1. Read next line
2. Write line into location S
3. S = S + 1
4. Line counter Line counter + 1
```

Subroutine Linkage: To understand the concept of subroutine linkages, first consider the following scenario:

"In Program A a call to subroutine B is made. The subroutine B is not written in the program segment of A, rather B is defined in some another program segment C"

Nothing is wrong in it. But from assembler's point of view while generating the code for B, as B is not defined in the segment A, the assembler can not find the value of this symbolic reference and hence it will declare it as an error. To overcome problem, there should be some mechanism by which the assembler should be explicitly informed that segment B is really defined in some other segment C. Therefore whenever segment B is used in segment A and if at all B is defined in C, then B must -be declared as an external routine in A. To declare such subroutine as external, we can use the assembler directive EXT. Thus the statement such as EXT B should be added at the beginning of the segment A. This actually helps to inform assembler that B is defined somewhere else. Similarly, if one subroutine or a variable is defined in the current segment and can be referred by other segments then those should be declared by using pseudo-ops INT. Thereby the assembler

could inform loader that these are the subroutines or variables used by other segments. This overall process of establishing the relations between the subroutines can be conceptually called a_ subroutine linkage.
For example

```
MAIN START
          EXT B
            .
            .
            .
         CALL B
           .
             .
          END
B       START
            .
            .
          RET
        END
```

At the beginning of the MAIN the subroutine B is declared as external. When a call to subroutine B is made, before making the unconditional jump, the current content of the program counter should be stored in the system stack maintained internally. Similarly while returning from the subroutine B (at RET) the pop is performed to restore the program counter of caller routine with the address of next instruction to be executed.

Concept of relocations:

Relocation is the process of updating the addresses used in the address sensitive instructions of a program. It is necessary that such a modification should help to execute the program from designated area of the memory.
The assembler generates the object code. This object code gets executed after loading at storage locations. The addresses of such object code will get specified only after the assembly process is over. Therefore, after loading,
address of object code = Mere address of object code + relocation constant.
There are two types of addresses being generated: Absolute address and, relative address. The absolute address can be directly used to map the object code in the main memory. Whereas the relative address is only after the addition of relocation constant to the object code address. This kind of adjustment needs to be done in case of relative address before actual execution of the code. The typical example of relative reference is : addresses of the symbols defined in the Label field, addresses of the data which is defined by the assembler directive, literals, redefinable symbols.

Similarly, the typical example of absolute address is the constants which are generated by assembler are absolute.
The assembler calculates which addresses are absolute and which addresses are relative during the assembly process. During the assembly process the assembler calculates the address with the help of simple expressions.
For example
LOADA(X)+5
The expression A(X) means the address of variable X. The meaning of the above instruction is that loading of the contents of memory location which is 5 more than the address of variable X. Suppose if the address of X is 50 then by above command we try to get the memory location 50+5=55. Therefore as the address of variable X is relative A(X) + 5 is also relative. To calculate the relative addresses the simple expressions are allowed. It is expected that the expression should possess at the most addition and multiplication operations. A simple exercise can be carried out to determine whether the given address is absolute or relative. In the expression if the address is absolute then put 0 over there and if address is relative then put lover there. The expression then gets transformed to sum of O's and l's. If the resultant value of the expression is 0 then expression is absolute. And if the resultant value of the expression is 1 then the expression is relative. If the resultant is other than 0 or 1then the expression is illegal. For example:

Expression	Computation	Relocation attribute
A-B	1-1 = 0	Absolute
A+B-C	1+1-1= 1	Relative
A-B+5	1-1+0 = 0	Absolute
A+B	1+1 = 2	Illegal

In the above expression the *A,* Band C are the variable names. The assembler is to c0l1sider the relocation attribute and adjust the object code by relocation constant. Assembler is then responsible to convey the information loading of object code to the loader. Let us now see how assembler generates code using relocation information.

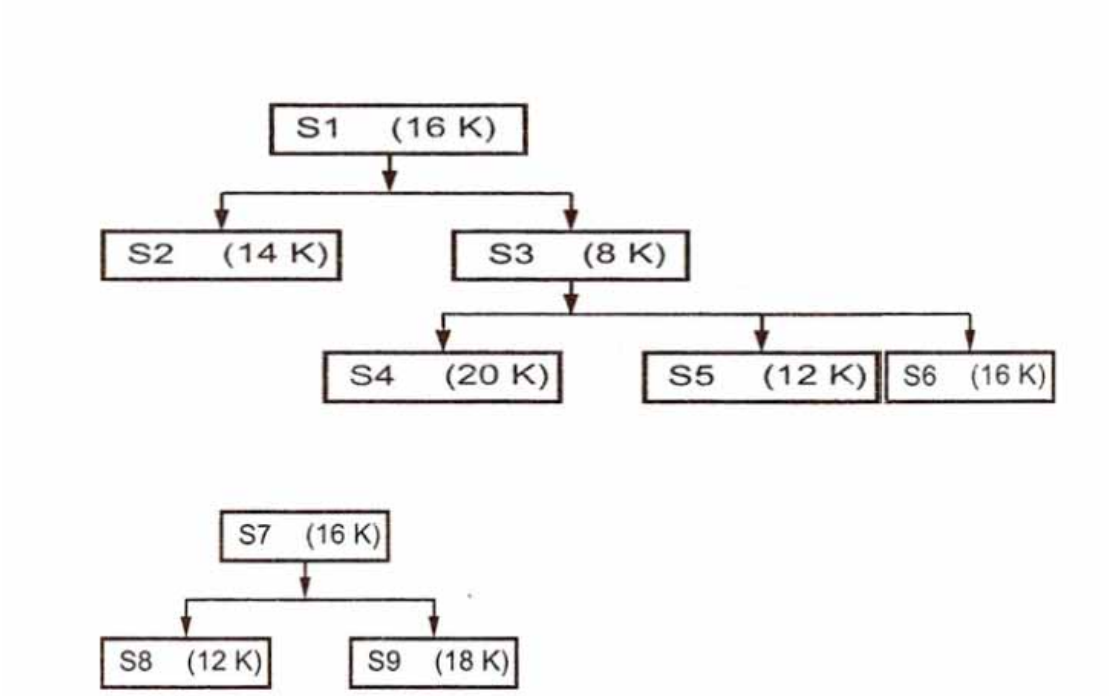
Direct Linking Loaders

The direct linking loader is the most common type of loader. This type of loader is a relocatable loader. The loader can not have the direct access to the source code. And to place the object code in the memory there are two situations: either the address of the object code could be absolute

which then can be directly placed at the specified location or the address can be relative. If at all the address is relative then it is the assembler who informs the loader about the relative addresses.
The assembler should give the following information to the loader
1)The length of the object code segment
2) The list of all the symbols which are not defined 111 the current segment but can be used in the current segment.
3) The list of all the symbols which are defined in the current segment but can be referred by the other segments.
The list of symbols which are not defined in the current segment but can be used in the current segment are stored in a data structure called USE table. The USE table holds the information such as name of the symbol, address, address relativity.
The list of symbols which are defined in the current segment and can be referred by the other segments are stored in a data structure called DEFINITION table. The definition table holds the information such as symbol, address.

Overlay Structures and Dynamic Loading:

Sometimes a program may require more storage space than the available one Execution of such program can be possible if all the segments are not required simultaneously to be present in the main memory. In such situations only those segments are resident in the memory that are actually needed at the time of execution But the question arises what will happen if the required segment is not present in the memory? Naturally the execution process will be delayed until the required segment gets loaded in the memory. The overall effect of this is efficiency of execution process gets degraded. The efficiency can then be improved by carefully selecting all the interdependent segments. Of course the assembler can not do this task. Only the user can specify such dependencies. The inter dependency of the segments can be specified by a tree like structure called static overlay structures. The overlay structure contain multiple root/nodes and edges. Each node represents the segment. The specification of required amount of memory is also essential in this structure. The two segments can lie simultaneously in the main memory if they are on the same path. Let us take an example to understand the concept. Various segments along with their memory requirements is as shown below.



Automatic Library Search:

Previously, the library routines were available in absolute code but now the library routines are provided in relocated form that ultimately reduces their size on the disk, which in turn increases the memory utilization. At execution time certain library routines may be needed. Keeping track of which library routines are required and how much storage is required by these routines, if at all is done by an assembler itself then the activity of automatic library search becomes simpler and effective. The library routines can also make an external call to other routines. The idea is to make a list of such calls made by the routines. And if such list is made available to the linker then linker can efficiently find the set of required routines and can link the references accordingly.

For an efficient search of library routines it desirable to store all the calling routines first and then the called routines. This avoids wastage of time due to winding and rewinding. For efficient automated search of library routines even the dictionary of such routines can be maintained. A table containing the names of library routines and the addresses where they are actually located in relocatable form is prepared with the help of translator and such table is submitted to the linker. Such a table is called subroutine directory. Even if these routines have made any external calls the -information about it is also given in subroutine directory. The linker searches the subroutine directory, finds the address of desired library routine (the address where the routine is stored in relocated form).Then linker prepares a

load module appending the user program and necessary library routines by doing the necessary relocation. If the library routine contains the

external calls then the linker searches the subroutine directory finds the address of such external calls, prepares the load module by resolving the external references. **Linkage Editor:** The execution of any program needs four basic functionalities and those are allocation, relocation, linking and loading. As we have also seen in direct linking loader for execution of any program each time these four functionalities need to be performed. But performing all these functionalities each time is time and space consuming task. Moreover if the program contains many subroutines or functions and the program needs to be executed repeatedly then this activity becomes annoyingly complex .Each time for execution of a program, the allocation, relocation linking and -loading needs to be done. Now doing these activities each time increases the time and space complexity. Actually, there is no need to redo all these four activities each time. Instead, if the results of some of these activities are stored in a file then that file can be used by other activities. And performing allocation, relocation, linking and loading can be avoided each time. The idea is to separate out these activities in separate groups. Thus dividing the essential four functions in groups reduces the overall time complexity of loading process. The program which performs allocation, relocation and linking is called binder. The binder performs relocation, creates linked executable text and stores this text in a file in some systematic manner. Such kind of module prepared by the binder execution is called load module. This load module can then be actually loaded in the main memory by the loader. This loader is also called as module loader. If the binder can produce the exact replica of executable code in the load module then the module loader simply loads this file into the main memory which ultimately reduces the overall time complexity. But in this process the binder should knew the current positions of the main memory. Even though the binder knew the main memory locations this is not the only thing which is sufficient. In multiprogramming environment, the region of main memory available for loading the program is decided by the host operating system. The binder should also know which memory area is allocated to the loading program and it should modify the relocation information accordingly. The binder which performs the linking function and produces adequate information about allocation and relocation and writes this information along with the program code in the file is called linkage editor. The module loader then accepts this rile as input, reads the information stored in and based on this information about allocation and relocation it performs the task of loading in the main memory. Even though the program is repeatedly executed the linking is done only once. Moreover, the flexibility of allocation and relocation helps efficient utilization of the main memory.

Direct linking: As we have seen in overlay structure certain selective subroutines can be resident in the memory. That means it is not necessary to resident all the subroutines in the memory for all the time. Only necessary routines can be present in the main memory and during execution the required subroutines can be loaded in the memory. This process of postponing linking and loading of external reference until execution is called dynamic linking. For example suppose the subroutine main calls A,B,C,D then it is not desirable to load A,B,C and D along with the main in the memory. Whether A, B, C or D is called by the main or not will be known only at the time of execution. Hence keeping these routines already before is really not needed. As the subroutines get executed when the program runs. Also the linking of all the subroutines has to be performed. And the code of all the subroutines remains resident in the main memory. As a result of all this is that memory gets occupied unnecessarily. Typically 'error routines' are such routines which can be invoked rarely. Then one can postpone the loading of these routines during the execution. If linking and loading of such rarely invoked external references could be postponed until the execution time when it was found to be absolutely necessary, then it increases the efficiency of overhead of the loader. In dynamic linking, the binder first prepares a load module in which along with program code the allocation and relocation information is stored. The loader simply loads the main module in the main memory. If any external ·reference to a subroutine comes, then the execution is suspended for a while, the loader brings the required subroutine in the main memory and then the execution process is resumed. Thus dynamic linking both the loading and linking is done dynamically. **Advantages**

1. The overhead on the loader is reduced. The required subroutine will be load in the main memory only at the time of execution.
2. The system can be dynamically reconfigured.

Disadvantages The linking and loading need to be postponed until the execution. During the execution if at all any subroutine is needed then the process of execution needs to be suspended until the required subroutine gets loaded in the main memory.

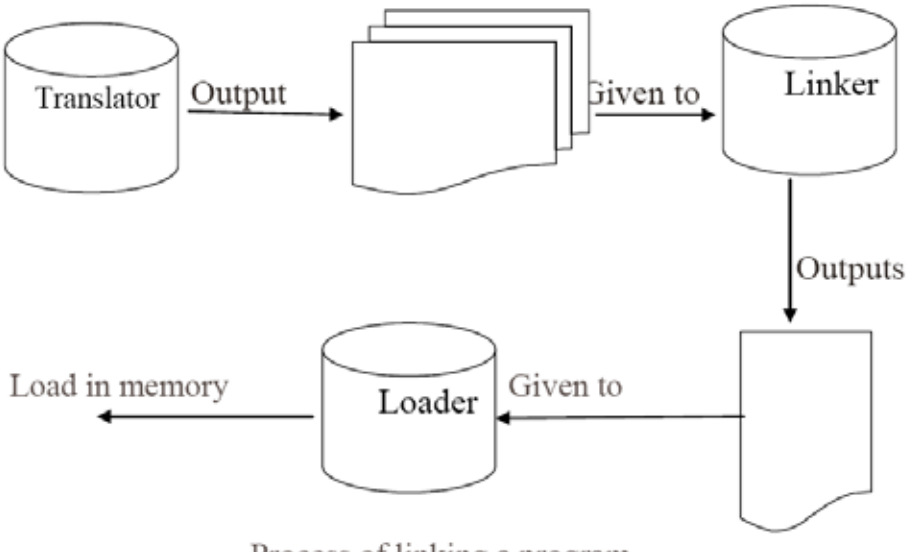
Bootstrap Loader: As we turn on the computer there is nothing meaningful in the main memory (RAM). A small program is written and stored in the ROM. This program initially loads the operating system from secondary storage to main memory. The operating system then takes the overall control. This program which is responsible for booting up the system is called bootstrap loader. This is the program which must be executed first when the system is first powered on. If the program starts from the location x then to execute this program the program counter of this machine should be loaded with the value x. Thus the task of setting

the initial value of the program counter is to be done by machine hardware. The bootstrap loader is a very small program which is to be fitted in the ROM. The task of bootstrap loader is to load the necessary portion of the operating system in the main memory .The initial address at which the bootstrap loader is to be loaded is generally the lowest (may be at 0th location) or the highest location. .

Concept of Linking: As we have discussed earlier, the execution of program can be done with the help of following steps

1. Translation of the program(done by assembler or compiler)
2. Linking of the program with all other programs which are needed for execution. This also involves preparation of a program called load module.
3. Loading of the load module prepared by linker to some specified memory location.

The output of translator is a program called object module. The linker processes these object modules binds with necessary library routines and prepares a ready to execute program. Such a program is called binary program. The "binary program also contains some necessary information about allocation and relocation. The loader then load s this program into memory for execution purpose.



Process of linking a program

Various tasks of linker are -

1. Prepare a single load module and adjust all the addresses and subroutine references with respect to the offset location.
2. To prepare a load module concatenate all the object modules and adjust all the operand address references as well as external references to the offset location.

3. At correct locations in the load module, copy the binary machine instructions and constant data in order to prepare ready to execute module.

The linking process is performed in two passes. Two passes are necessary because the linker may encounter a forward reference before knowing its address. So it is necessary to scan all the DEFINITION and USE table at least once. Linker then builds the Global symbol table with the help of USE and DEFINITION table. In Global symbol table name of each externally referenced symbol is included along with its address relative to beginning of the load module. And during pass 2, the addresses of external references are replaced by obtaining the addresses from global symbol table.

(Operating System)

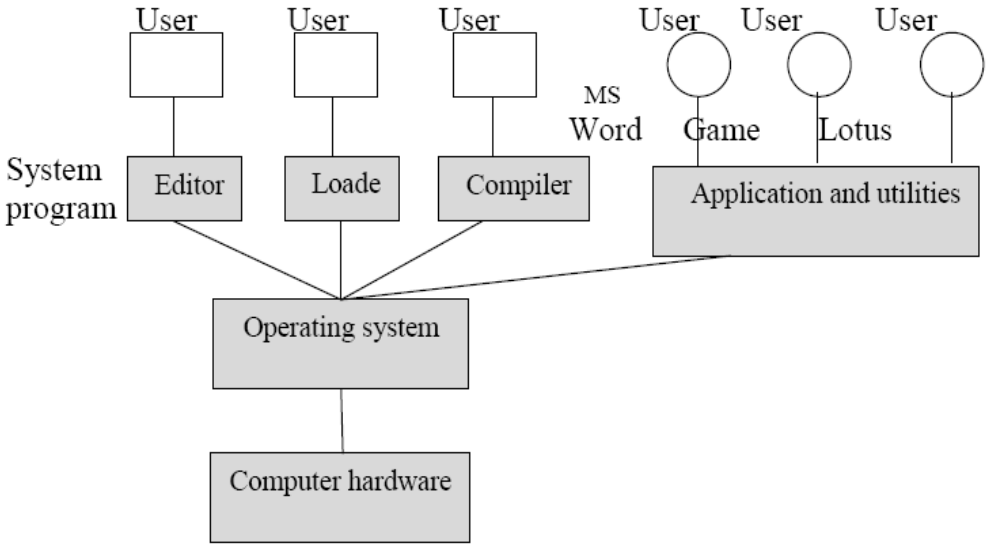
Introduction:

An operating system is a program that control the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

• Operating system performs three functions:
1. Convenience: An as makes a computer more convenient to use.
2. Efficiency: An as allows the computer system resources to be used in an efficient manner.
3. Ability to evolve : An as should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfaring with service .

Operating System as a User Interface:

• Every general purpose computer consists of the hardware, operating system, system programs, application programs. The hardware consists of memory, CPU, ALU, I/O devices, peripheral device and storage device. System program consists of compilers, loaders, editors, as etc. The application program consists of business program, database program.
• The Figure below shows the conceptual view of a computer system.



(Conceptual view of a computer system)

Every computer must have an operating system to run other programs. The operating system controls and co-ordinates the use of the hardware among the various system programs and application program for a various users. It simply provides an environment within which other programs can do useful work.

• The operating system is a set of special programs that run on a computer system that allow it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling a peripheral devices.
• OS is designed to serve two basic purposes :
1. It controls the allocation and use of the computing system's resources among the various users and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

• The operating system must support the following tasks:
1. Provides the facilities to create, modification of program and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.

Operating System Services:

• An operating system provides services to programs and to the users of those programs. It provides an environment for the execution of programs. The services provided by one operating system is different than other operating system.
• Operating system makes the programming task easier. The common services
Provided by the operating system is listed below.
1. Program execution
2. I/O operation
3. File system manipulation
4. Communications
5. Error detection.
1. **Program execution**: Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.

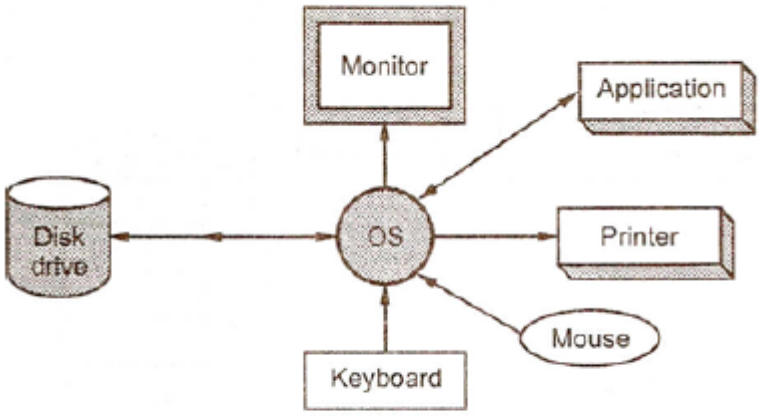
2. ***I/O* operation:** I/O means any file or any specific I/O device. Program may require any I/O device while running. So operating system must provide the required I/O.
3. **File system manipulation**: Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.
4. **Communication:** Data transfer between two processes is required for some time. The both processes are on the one computer or on different computer but connected through computer network. Communication may be implemented by two methods: shared memory and message passing.
5. **Error detection:** Error may occur in CPU, in I/O devices or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing.
Operating system with multiple users provides following services.

1. Resource allocation

2. Accounting

3. Protection •

Figure below shows the view of OS with components.



(view of OS with components)

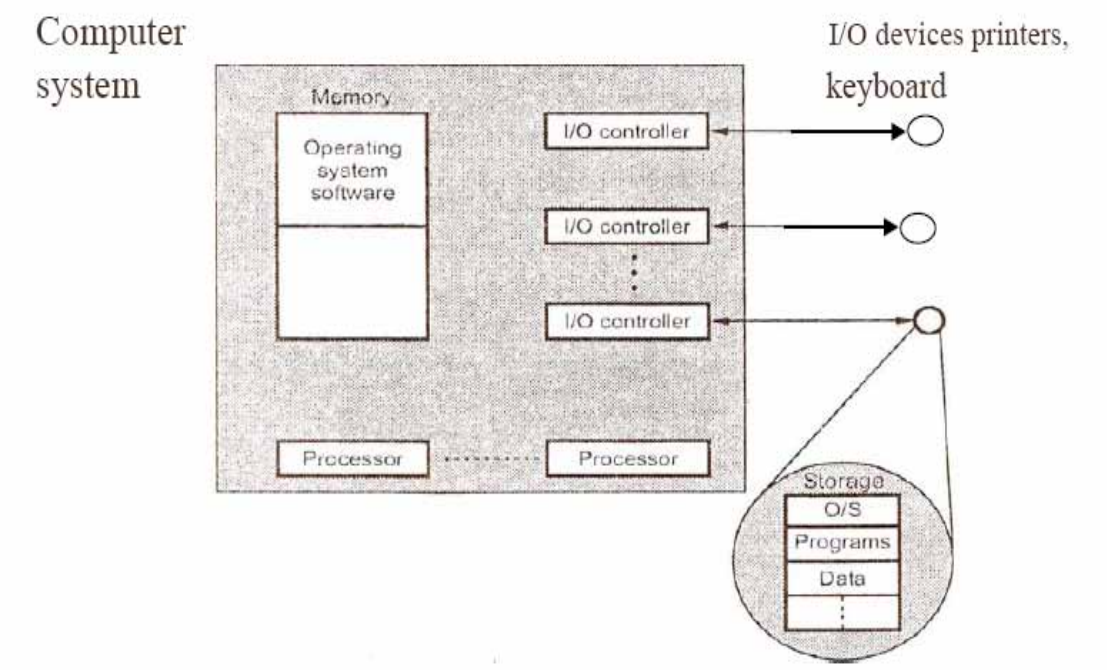
• An operating system is a lower level of software that user programs run on.

OS is built directly on the hardware interface and provides an interface between the hardware and the user program. It shares characteristics 'with both software and hardware.

• We can view an operating system as a resource allocator.
OS keeps track of the status of each resource and decides who gets a resource, for how long, and when. as makes sure that different programs and users running at the same time but do not interfere with each other. It is also responsible for security, ensuring that unauthorized users do not access the system.
• The primary objective of operating systems is to increase productivity of a processing resource, such as computer hardware or users.
• The operating system is the first program nm on a computer when the computer boots up. The services of the as are invoked with a system call instruction that is used just like any other hardware instruction.
• Name of the operating systems are: DOS, Windows 95, Windows NT/2000, Unix, Linux etc.

Operating System as Resource Manager

• A computer is a set of resources for the movement, storage and processing of data and for the control of these functions. The as is responsible for managing these resources.
• Figure below shows as a resource manager.



(OS as a resource manager)

• Main resources that are managed by the operating system. A portion of the operating system is in main memory. This includes the Kernel, which contains the most frequently used functions in the operating system and at a given time, other portions of the OS currently in use.
• The remainder of main memory contains other user programs and data. The allocation of main memory is controlled jointly by the OS and memory management hardware in the processor.
• The operating system decides when an I/O device can be used by a program in execution and controls access to and use of files. The processor itself is a resource, and the operating system must determine how much processor time is to be devoted to the execution of a particular user program.

History of Operating System

• Operating systems have been evolving through the years. Following table shows the history of OS.

Generation	Year	Electronic devices used	Types of OS and devices
First	1945-55	Vacuum tubes	Plug boards
Second	1955-1965	Transistors	Batch system
Third	1965-1980	Integrated circuit (IC)	Multiprogramming
Fourth	Since 1980	Large scale integration	PC

Mainframe System: An operating system may process its workload serially or concurrently. That is resources of the computer system may be dedicated to a single program until its completion, or they may be dynamically reassigned among a collection of active programs in different stages of execution.
• Several variations of both serial and multiprogrammed operating systems exist.

Characteristics of mainframe systems

1. The first computers used to tackle various applications and still found today in corporate data centers.
2. Room-sized, high I/O capacity, reliability, security, technical support.
3. Mainframes focus on I/O bound business data applications.

Mainframes provide three main functions:

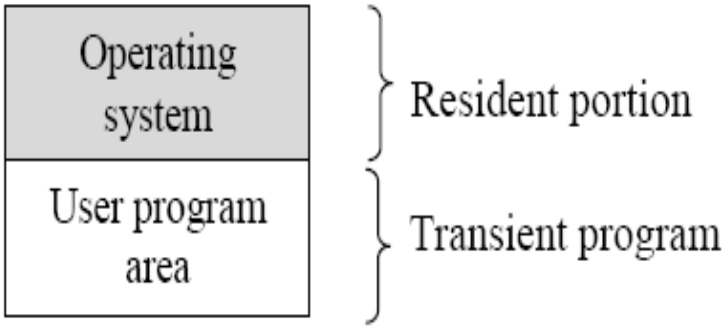
a. Batch processing: insurance claims, store sales reporting, etc.
b. Transaction processing: credit card, bank account, etc.
c. Time-sharing: multiple users querying a database.

Batch Systems

• Some computer systems only did one thing at a time. They had a list of instructions to carry out and these would be carried out one after the other. This is called a **serial system.** The mechanics of development and preparation of programs in such environments are quite slow and numerous manual operations involved in the process.

• Batch operating system is one where programs and data are collected together in a batch before processing starts. A job is predefined sequence of commands, programs and data that are combined into a single unit called **job.**

• Figure below shows the memory layout for a simple batch system.



(Memory layout for a simple batch system)

Memory management in batch system is very simple. Memory is usually divided into two areas: Operating system and user program area.

• Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e. first come first served fashion.

• When a job completes execution, its memory is released and the output for the job gets copied into an output **spool** for later printing.

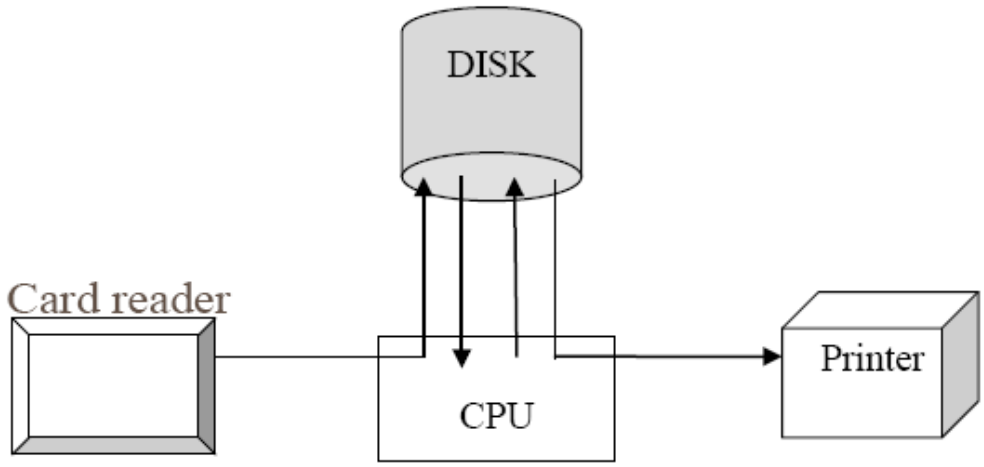
• Spooling an acronym for **simultaneous peripheral operation on line.** Spooling uses the disk as a large buffer for outputting data to printers and other devices. It can also be used for input, but is generally used for output. Its main use is to prevent two users from alternating printing lines to the line printer on the same page, getting their output completely mixed together. It also helps in reducing idle time and overlapped I/O and CPU.

• Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
• Batch systems are inconvenient for users because users can not interact with their jobs to fix problems. There may also be long turnaround times. Example of this system is generating monthly bank statement.

Spooling:

• Acronym for simultaneous peripheral operations on line. Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.
• Spooling is useful because device access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.

Figure below shows the spooling.



(Spooling)

• Computer can perform I/O in parallel with computation, it becomes possible to have the computer read a deck of cards to a tape, drum or disk and to write out to a tape printer while it was computing. This process is called **spooling.**
• The most common spooling application is print spooling. In print spooling, documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate.
• Spooling is also used for processing data at remote sites. The CPU sends the data via communications path to a remote printer. Spooling overlaps the I/O of one job with the computation of other jobs.

• One difficulty with simple batch systems is that the computer still needs to read the deck of cards before it can begin to execute the job. This means that the CPU is idle during these relatively slow operations.
• Spooling batch systems were the first and are the simplest of the multiprogramming systems.

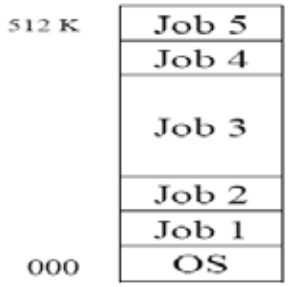
Advantages of Spooling:
1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

Advantages of Batch System:
1. Move much of the work of the operator to the computer.
2. Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of Bach System:
1. Turn around time can be large from user standpoint.
2. Difficult to debug program.
3. A job could enter an infinite loop.
4. A job could corrupt the monitor, thus affecting pending jobs.
5. Due to lack of protection scheme, one batch job can affect pending jobs.

Multiprogramming Operating System: When two or more programs are in memory at the same time, sharing the processor is referred to the multiprogramming operating system. Multiprogramming assumes a single processor that is being shared. It increases CPU utilization by organizing jobs so that the CPU always has one to execute.

Figure below shows the memory layout for a multiprogramming system.



(Memory layout for a multiprogramming system)

• The operating system keeps several jobs in memory at a time. This set of jobs is a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the job in the memory.
• Multiprogrammed systems provide an environment in which the various system resources are utilized effectively, but they do not provide for user interaction with the computer system.
• Jobs entering into the system are kept into the memory. Operating system picks the job and begins to execute one of the jobs in the memory. Having several programs in memory at the same time requires some form of memory management.
• Multiprogramming operating system monitors the state of all active programs and system resources. This ensures that the CPU is never idle unless there are no jobs.

Advantages
1. High CPU utilization.
2. It appears that many programs are allotted CPU almost simultaneously.

Disadvantages
1. CPU scheduling is required.
2. To accommodate many jobs in memory, memory management is required.

Time Sharing Systems:
• Time sharing system supports interactive users. Time sharing is also called **multitasking.** It is logical extension of multiprogramming. Time sharing system uses CPU scheduling and multiprogramming to provide an economical interactive system of two or more users.
• In time sharing, each user is given a time-slice for executing his job in round-robin fashion. Job continues until the time-slice ends.
• Time sharing systems are more complex than multiprogramming operating system. Memory management in time sharing system provides for isolation and protection of **co-resident** programs.
• Time sharing uses medium-term scheduling such as round-robin for the foreground. Background can use a different scheduling technique.
• Time sharing system can run several programs at the same time, so it is also a multiprogramming system. But multiprogramming operating system is not a time sharing system.
• Difference between both the systems is that, time sharing system allows more frequent context switches. This gives each user the impression that the entire computer is dedicated to his use. In multiprogramming system a context switch occurs only when the currently executing process stalls for some reason.

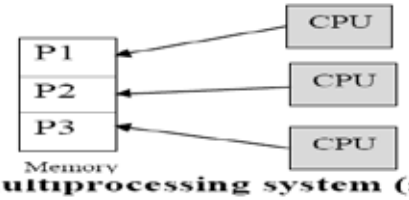
Desktop System: During the late 1970, computers had faster CPU, thus creating an even greater disparity between their rapid processing speed and slower I/O access time. Multiprogramming schemes to increase CPU use were limited by the physical capacity of the main memory, which was a limited resource and very expensive. These system includes PC running MS window and the Apple Macintosh. The Apple Macintosh OS support new advance hardware i.e. virtual memory and multitasking with virtual memory, the entire program did not need to reside in memory before execution could begin.

• Linux, a unix like OS available for PC, has also become popular recently. The microcomputer was developed for single users in the late 1970. Physical size was smaller than the minicomputers of that time, though larger than the microcomputers of today.
• Microcomputer grew to accommodate software with large capacity and greater speeds. The distinguishing characteristics of a microcomputer is its single user status. MS-DOS is an example of a microcomputer operating system.
• The most powerful microcomputers used by commercial; educational, government enterprises. Hardware cost for microcomputers are sufficiently low that a single user (individuals) have sole use of a computer. Networking capability has been integrated into almost every system.

Multiprocessor System:

• Multiprocessor system have more than one processor in close communication. They share the computer bus, system clock and input-output devices and sometimes memory. In multiprocessing system, it is possible for two processes to run in parallel.
• Multiprocessor systems are of two types: symmetric multiprocessing and asymmetric multiprocessing.
• In symmetric multiprocessing, each processor runs an identical copy of the operating system and they communicate with one another as needed. All the CPU shared the common memory.

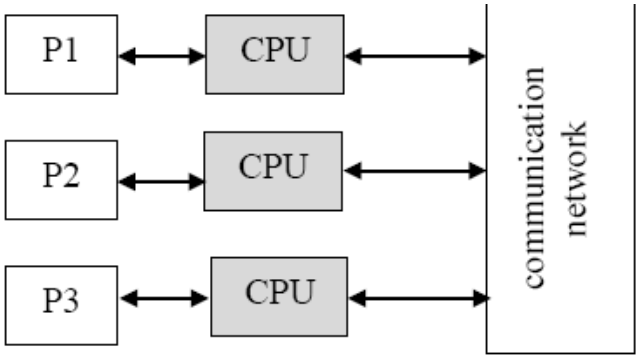
Figure below shows the symmetric multiprocessing system.



Symmetric multiprocessing system (shared memory)

• In asymmetric multiprocessing, each processor is assigned a specific task. It uses master-slave relationship. A master processor controls the system. The master processor schedules and allocates work to the slave processors.

Figure below shows the asymmetric multiprocessor.



Asymmetric multiprocessors (NO shared memory)

Features of multiprocessor systems

1. If one processor fails, then another processors should retrive the interrupted process state so that executation of the process can continue.
2. The processors should support efficient context switching operation.
3. Multiprocessor system supports large physical address space & large virtual address space.
4. The IPC mechanism should be provided & implemented in hardware as it becomes efficient & easy.

Distributed System: Distributed operating systems depend on networking for their operation. Distributed as runs on and controls the resources of multiple machines. It provides resource sharing across the boundaries of a single computer system. It looks to users like a single machine as. Distributing as owns the whole network and makes it look like a virtual uniprocessor or may be a virtual multiprocessor.

- A distributed operating system is one that looks to its users like an ordinary operating system but runs on multiple, independent CPU.

Advantages of distributed OS:

1. Resource sharing: Sharing of software resources such as software libraries, database and hardware resources such as hard disks, printers and CDROM can also be done in a very effective way among all the computers and the users.
2. Higher reliability: Reliability refers to the degree of tolerance against errors and component failures. Availability is one of the important aspect

of reliability. Availability refers to the fraction of time for which a system is available for use. Availability of a hard disk can be increased by having multiple hard disks located at different sites. If one hard disk fails or is unavailable, the program can use some other hard disk.
3. Better price performance ratio. Reduction in the price of microprocessor and increasing computing power gives good price-performance ratio.
4. Shorter responses times and higher throughput.
5. Incremental growth: To extend power and functionality of a system by simply adding additional resources to the system.

Difficulties in distributed OS are:
1. There are no current commercially successful examples.
2. Protocol overhead can dominate computation costs.
3. Hard to build well.
4. Probably impossible to build at the scale of the Internet.

Cluster System:
• It is a group of computer system connected with a high speed communication link. Each computer system has its own memory and peripheral devices. Clustering is usually performed to provide high availability. Clustered systems are integrated with hardware cluster and software cluster. Hardware cluster means sharing of high performance disks. Software cluster is in the form of unified control of the computer system in a cluster.
• A layer of software cluster runs on the cluster nodes. Each node can monitor one or more of the others. If the monitoring machine fails, the monitoring machine can take ownership of its storage and restart the application that were running on the failed machine.
• Clustered system can be categorized into two groups: asymmetric clustering and symmetric clustering.
• In asymmetric clustering, one machine is in hot standy mode while the other is running the applications. Hot standy mode monitors the active server and sometimes becomes the active server when the original server fails.
• In symmetric clustering mode, two or more than two hosts are running applications and they are monitoring each other.
• Parallel clusters and clustering over a WAN is also available in clustering.
Parallel clusters allow multiple hosts to access the same data on the shared storage. A cluster provides all the key advantages of distributed systems. A cluster provides better reliability than the symmetrical multiprocessor system.

• Cluster technology is rapidly changing. Clustered system use and features should expand greatly as storage area networks. Storage area network allows easy attachment of multiple hosts to multiple storage units.

Real Time System:

• Real time systems which were originally used to control autonomous systems such as satellites, robots and hydroelectric dams. A real time operating system is one that must react to inputs and responds to them quickly. A real time system cannot afford to be late with a response to an event.

• A real time system has well defined, fixed time constraints.

Deterministic scheduling algorithms are used in real time systems. Real time systems are divided into two groups :

Hard real time system and **soft real time system.**

• A hard real time system guarantees that the critical tasks be completed on time. This goal requires that all delay in the system be bounded. Soft real time system is a less restrictive type. In this, a critical r.eal time task gets priority over other tasks, and retains that priority until it completes.

• Real time operating system uses priority scheduling algorithm to meet the response requirement of a real time application.

• General real time applications with some examples are listed below.

Sr. No.	Real Time Application	Examples
1.	Detection	Radar system Burglar alarm
2.	Process monitoring and control	Petroleum, Paper mill.
3.	Communication	Telephone switching system
4.	Flight simulation and control	Auto pilot shuttle mission simulator
5.	transportation	Traffic light system, Air traffic control

• Memory management in real time system is comparatively less demanding than in other types of multiprogramming systems. Time-critical device management is one of the main characteristics of real time systems. The primary objective of file management in real time system is usually speed of access, rather than efficient utilization of secondary storage.

Comparison between Hard and Soft Real Time System

• Hard real time system guarantees that critical tasks complete on time. To achieve this, all delays in the system must be bounded i.e. the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Soft real time system are less restrictive than the hard real time system. In soft real time, a critical real time task gets priority over other tasks and retains that priority until it complete.

• Time constraints are the main properties for the hard real time systems. Since none of the operating system support hard real time system, Kernal delays need to be bounded in soft real time system. Soft real time systems are useful in the area of multimedia, virtual reality and advance scientific projects. Soft real time systems can not be used in -robotics and industrial control because of their lack of deadline support. Soft real time system requires two conditions to implement. CPU scheduling must be priority based and dispatch latency must be small.

Handheld System:

• Personal Digital Assistants (PDA) is one type of handheld systems. Developing such device is the complex job and many challenges will face by developers. Size of these system is small i.e. height is 5 inches and width is 3 inches.

• Due to the limited size, most handheld devices have a small amount of memory, include slow processors and small display screen. Memory of handheld system is in the range of 512 kB to 8 MB. Operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer needed. Developers are working only on confines of limited physical memory because any handheld devices not using virtual memory.

• Speed of the handheld system is major factor. Faster processors require for handheld systems. Processors for most handheld devices often run at a fraction of the speed of a processor in a Pc. Faster processors require more power. Larger battery requires for faster processors.

• For mimimum size of handheld devices, smaller, slower processors which consumes less power are used. Typically small display screen is available in these devices. Display size of handheld device is not more than 3 inches square.

• At the same time, display size of monitor is up to 21 inches. But these handheld device provides the facility for reading email, browsing web pages on smaller display. Web clipping is used for displaying web page on the handheld devices.

• Wireless technology is also used in handheld devices. Bluetooth protocol is used for remote access to email and web browsing. Cellular telephones with connectivity to the Internet fall into this category.

Computing Environments:

• Different types of computing environments are:

a.Traditional computing
b.Web based computing
c. Embedded computing

• Typical office environment uses traditional computing. Normal PC is used in traditional computing.
• Web technology also uses traditional computing environment. Network computers are essentially terminals that understand web based computing. In domastic application, most of user had a single computer with Internet connection. Cost of the accessing Internet is high.
• Web based computing has increased the emphasis on networking. Web based computing uses PC, handheld PDA and cell phones. One of the features of this type is load balancing. In load balancing, network connection is distributed among a pool of similar servers.
• Embedded computing uses realtime operating systems. Application of embedded computing is car engines, manufacturing robots to VCR and microwave ovens. This type of system provides limited features.

Essential Properties of the Operating System

1. **Batch:** Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off line operation, spooling and multiprogramming. A Batch system is good for executing large jobs that need little interaction, it can be submitted and picked up latter.

2. **Time sharing:** Uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another i.e. the CPU is shared between a number of interactive users. Instead of having a job defined by spooled card images, each program reads its next control instructions from the terminal and output is normally printed immediately on the screen.

3. **Interactive:** User is on line with computer system and interacts with it via an interface. It is typically composed of many short transactions where the result of the next transaction may be unpredictable. Response time needs to be short since the user submits and waits for the result.

4.**Real time system**: Real time systems are usually dedicated, embedded systems. They typically read from and react to sensor data. The system must guarantee response to events within fixed periods of time to ensure correct performance.

5. **Distributed:** Distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines.

System Components: Modern operating systems share the goal of supporting the system components. The system components are:
1. Process management
2. Main memory management
3. File management
4. Secondary storage management
5. I/O system management
6. Networking
7. Protection system
8. Command interpreter system.

Process Management

• Process refers to a program in execution. The process abstraction is a fundamental operating system mechanism for management of concurrent program execution. The operating system responds by creating a process.
• A process needs certain resources, such as CPU time, memory, files and I/O devices. These resources are either given to the process when it is created or allocated to it while it is running.
• When the process terminates, the operating system will reclaim any reusable resources.
• The term process refers to an executing set of machine instructions. Program by itself is not a process. A program is a passive entity.
• The operating system is responsible for the following activities of the process management.
1. Creating and destroying the user and system processes.
2. Allocating hardware resources among the processes.
3. Controlling the progress of processes.

4. Providing mechanisms for process communications.
5. Also provides mechanisms for deadlock handling.

Main Memory Management

• The memory management modules of an operating system are concerned with the management of the primary (main memory) memory. Memory management is concerned with following functions:
1. Keeping track of the status of each location of main memory. i.e. each memory location is either free or allocated.
2. Determining allocation policy for memory.
3. Allocation technique i.e. the specific. location must be selected and allocation information updated.
4. Deallocation technique and policy. After deallocation, status information must be updated.

• Memory management is primarily concerned with allocation of physical memory of finite capacity to requesting processes. The overall resource utilization and other performance criteria of a computer system are affected by performance of the memory management module. Many memory management schemes are available and the effectiveness of the different algorithms depends on the particular situation.

File Management
• Logically related data items on the secondary storage are usually organized into named collections called files. In short, file is a logical collection of information. Computer uses physical media for storing the different information.
• A file may contain a report, an executable program or a set of commands to the operating system. A file consists of a sequence of bits, bytes, lines or records whose meanings are defined by their creators. For storing the files, physical media (secondary storage device) is used.
• Physical media are of different types. These are magnetic disk, magnetic tape and optical disk. All the media has its own characteristics and physical organization. Each medium is controlled by a device.
• The operating system is responsible for the following in connection with file management.

1. Creating and deleting of files.
2. Mapping files onto secondary storage.
3. Creating and deleting directories.
4. Backing up files on stable storage media.

5. Supporting primitives for manipulating files and directories.
6. Transmission of file elements between main and secondary storage.
• The file management subsystem can be implemented as one or more layers of the operating system.

Secondary Storage Management

• A storage device is a mechanism by which the computer may store information in such a way that this information may be retrieved at a later time. Secondary storage device is used for storing all the data and programs. These programs and data access by computer system must be kept in main memory. Size of main memory is small to accommodate all data and programs. It also lost the data when power is lost. For this reason secondary storage device is used. Therefore the proper management of disk storage is of central importance to a computer system.
• The operating system is responsible for the following activities in connection with the· disk management.

1. Free space management
2. Storage allocation
3. Disk scheduling

• The entire speed and performance of a computer may hinge on the speed of the disk subsystem.

I/O System Management :

The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that resides in a separate process associated with that device.
• The I/O subsystem consists of
1. A memory management component that includes buffering, caching and spooling.
2. A general device driver interface.
3. Drivers for specific hardware devices.

Networking: Networking enables computer users to share resources and speed up computations. the processors communicate with one another through various communication lines. For example, a distributed system. A distributed system is a collection of processors. Each processor has its own local memory and clock. The processors in the system are connected through a communication network, which can be configured in a number of different ways.

• Following parameter are considered while designing the networks.
1. Topology of network
2. Type of network
3. Physical media
4. Communication protocols
5. Routing algorithm.

Protection System:

• Modern computer systems support many users and allow the concurrent execution of multiple processes. Organizations rely on computers to store information. It is necessary that the information and devices must be protected from unauthorised users or processors. The protoction is any mechanism for controlling the access of programs, processes or users to the resources defined by a computer system.

• Protection mechanisms are implemented in operating systems to support various security policies. The goal of the security system is to authenticate subjects and to authorise their access to any object.

• Protection can improve reliability by detecting latent errors at the interfaces between compoent subsystems. Protection domains are extensions of the hardware supervisor mode ability

Command Interpreter System:

• Command interpreter is the interface between user and the operating system. It is system programs for an operating system.
Command interpreter is a special program in Unix and MS-DOS operating system.

• When users login first time or when a job is initiated, the command interpreter is initially some operating system is included in the Kernel. A control statement is processed by the command interpreter. Command interpreter reads the control statement, analyses it and carries out the required action.

Operating System Services:

• An operating system provides services to programs and to the users of those programs. It provides an environment for the execution of programs. The services provided by one operating system is different than other operating system. Operating system makes the programming task easier.

• The common services provided by the operating system is listed below.
1. Program execution

2. I/O operation
3. File system manipulation
4. Communications
5. Error detection.

1. **Program execution**: Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.

2. **I/O Operation**: I/O means any file or any specific I/O device. Program may require any I/O device while running. So operating system must provide the required I/O.

3. **File system manipulation**: Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.

4. **Communication**: Data transfer between two processes is required for some time. The both processes are on the one computer or on different computer but connected through computer network. Communication may be implemented by two methods : shared memory and message passing.

5. **Error detection:** Error may occur in CPU, in I/O devices or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing.

• Operating system with multiple users provides following services.
1. Resource allocation
2. Accounting
3. Protection

A) Resource allocation:
If there are more than one user or jobs running at the same time, then resources must be allocated to each of them. Operating system manages different types of resources. Some resources require special allocation code, i.e., main memory, CPU cycles and file storage.
• There are some resources which require only general request and release code. For allocating CPU, CPU scheduling algorithms are used for better utilization of CPU. CPU scheduling routines consider the speed of the CPU, number of available registers and other required factors.

B) Accounting:

• Logs of each user must be kept. It is also necessary to keep record of which user uses how much and what kinds of computer resources. This log is used for accounting purposes.

• The accounting data may be used for statistics or for the billing. It also used to improve system efficiency.

C) Protection:

• Protection involves ensuring that all access to system resources is controlled.

Security starts with each user having to authenticate to the system, usually by means of a password. External I/O devices must be also protected from invalid access attempts.

• In protection, all the access to the resources is controlled. In multiprocess environment, it is possible that, one process to interface with the other, or with the operating system, so protection is required.

System Calls:

• Modern processors provide instructions that can be used as system calls.

System calls provide the interface between a process and the operating system. A system call instruction is an instruction that generates an interrupt that cause the operating system to gain control of the processor.

• System call works in following ways :

1. First the program executes the system call instructions.
2. The hardware saves the current (instruction) and PSW register in the ii and iPSW register.
3. 0 value is loaded into PSW register by hardware. It keeps the machine in system mode with interrupt disabled.
4. The hardware loads the i register from the system call interrupt vector location. This completes the execution of the system call instruction by the hardware.
5. Instruction execution continues at the beginning of the system call interrupt handler.
6. The system call handler completes and executes a return from interrupt (rti) instructions. This restores the i and PSW from the ii and iPSW.
7. The process that executed the system call instruction continues at the instruction after the system call.

Types of System Call: A system call is made using the system call machine language instruction. System calls can be grouped into five major categories.

1. File management
2. Interprocess communication
3. Process management
4. I/O device management
5. Information maintenance.

Hardware Protection:

• For single-user programmer operating systems, programmer has the complete control over the system. They operate the system from the console. When new operating systems developed with some additional features, the system control transfers from programmer to the operating system.

•Early operating systems were called resident monitors, and starting with the resident monitor, the operating system began to perform many of the functions, like input-output operation.

•Before the operating system, programmer is responsible for the controls of input-output device operations. As the requirements of programmers from computer systems go on increasing and development in the field of communication helps to the operating system.

•Sharing of resource among different programmers is possible without increasing cost. It improves the system utilization but problems increase. If single system was used without share, an error occurs, that could cause problems for only the one program which was running on that machine.

•In sharing, other programs also affected by single program. For example , batch operating system faces the problem of infinite loop. This loop could prevent the correct operation of many jobs. In multiprogramming system, one erroreous program affects the other program or data of that program.

• For proper operation and error free result, protection of error is required . Without protection, only single process will execute one at a time otherwise the output of each program is separated. While designing the operating system, this type of care must be taken into consideration.

• Many programming errors are detected by the computer hardware. Operating system handled this type of errors. Execution of illegal instruction or access of memory that is not in the user's address space, this type of operation found by the hardware and will trap to the operating system.

• The trap transfers control through the interrupt vector to the operating system. Operating system must abnormally terminate the program when program error occurs. To handle this type of situation, different types of hardware protection is used.

CP/M

Control Program/Microcomputer. An operating system created by Gary Kildall, the founder of Digital Research. Created for the old 8-bit microcomputers that used the 8080, 8085, and Z-80 microprocessors. Was the dominant operating system in the late 1970s and early 1980s for small computers used in a business environment.

DOS

Disk Operating System. A collection of programs stored on the DOS disk that contain routines enabling the system and user to manage information and the hardware resources of the computer. DOS must be loaded into the computer before other programs can be started.

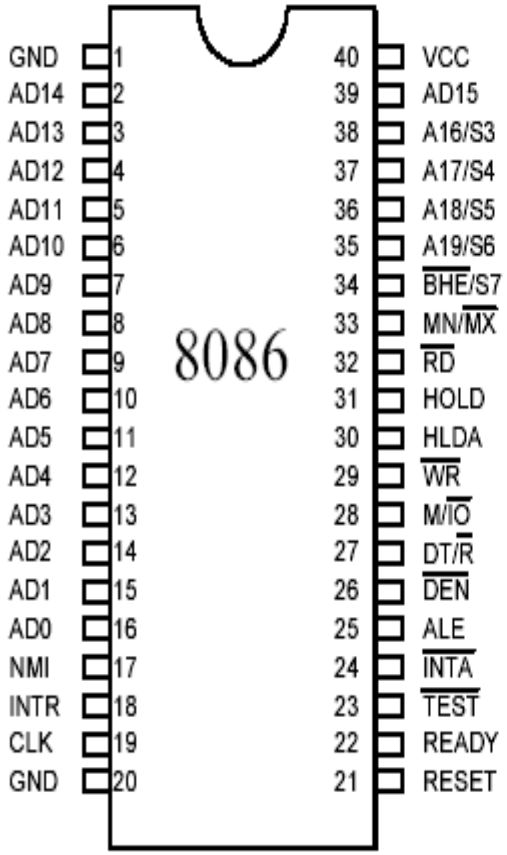
operating system (OS)

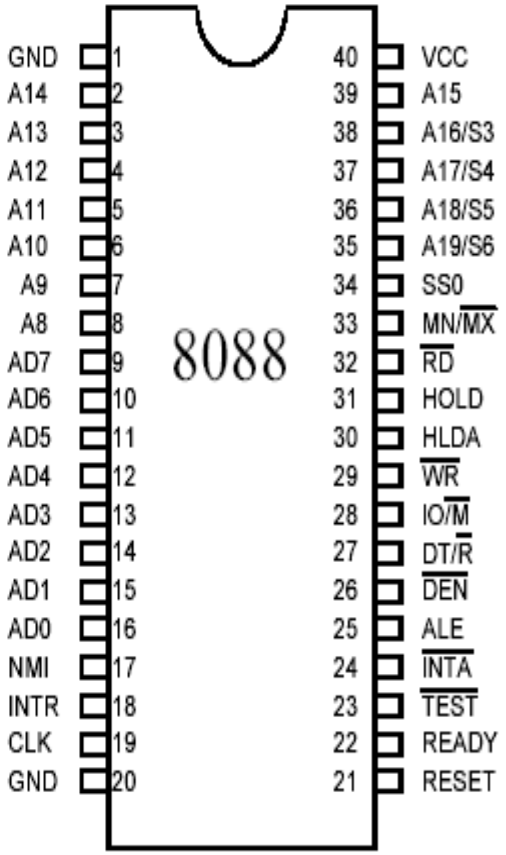
A collection of programs for operating the computer. Operating systems perform housekeeping tasks such as input and output between the computer and peripherals as well as accepting and interpreting information from the keyboard. DOS and OS/2 are examples of popular 0S's.

0S/2

A universal operating system developed through a joint effort by IBM and Microsoft Corporation. The latest operating system from IBM for microcomputers using the Intel 386 or better microprocessors. OS/2 uses the protected mode operation of the processor to expand memory from 1M to 4G and to support fast, efficient multitasking. The 0512 Workplace Shell, an integral part of the system, is a graphical interface similar to Microsoft Windows and the Apple Macintosh system. The latest version runs DOS, Windows, and OS/2-specific software.

8086/8088 Pinout Diagrams





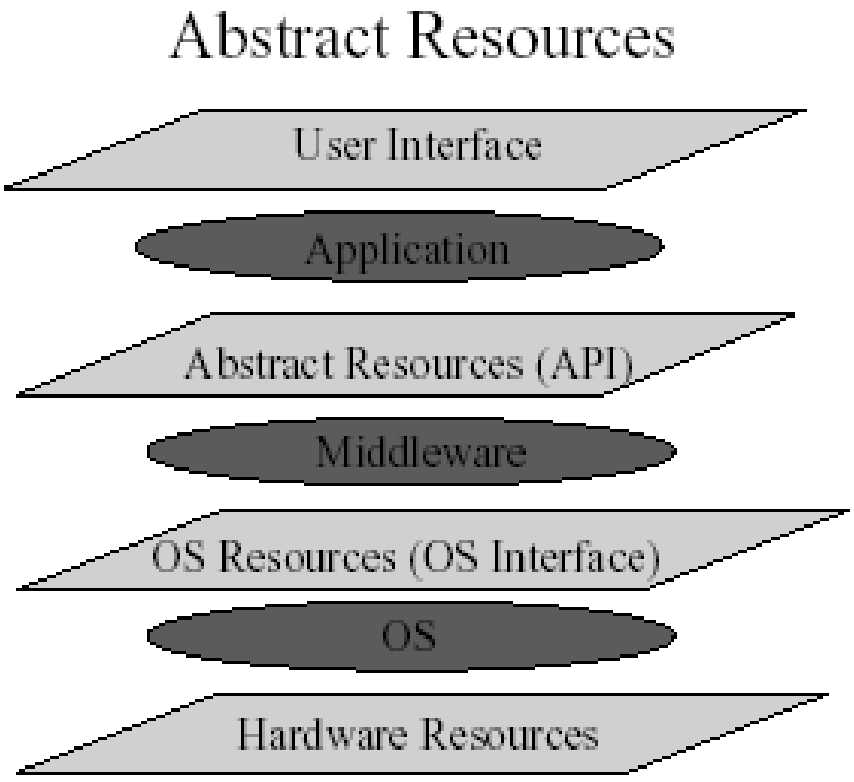
8086			8088		
Pin	Mode		Pin	Mode	
	Minimum	Maximum		Minimum	Maximum
31	HOLD	RQ/GT0	31	HOLD	RQ/GT0
30	HLDA	RQ/GT1	30	HLDA	RQ/GT1
29	WR	LOCK	29	WR	LOCK
28	M/IO	S2	28	IO/M	S2
27	DT/R	S1	27	DT/R	S1
26	DEN	S0	26	DEN	S0
25	ALE	QS0	25	ALE	QS0
24	INTA	QS1	24	INTA	QS1
			34	SS0	High State

**Minimum/Maximum Mode Refers to the Bus Handshaking*

Resources

• **Process**: An executing program

• **Resource**: Anything that is needed for a process to run
  - Memory
  - Space on a disk
  - The CPU
• An **OS** creates resource abstractions
• An **OS** manages resource sharing



The First OS

• **Resident monitors** were the first, rudimentary, operating systems
– monitor is similar to OS kernel that must be resident in memory
– control-card interpreters eventually become command processors or shells
• There were still problems with computer utilization. Most of these problems revolved around I/O operations

Operating System Classification

OS	single-user	multi-user
single-tasking	MS-DOS, CP/M	Intellec S-IV (MS-DOS station in Novell)
multi-tasking	Windows, MacOS	Unix, Windows-NT server

Single-tasking system: only one process can be run simultaneously

Multi-tasking system: can run arbitrary number of processes simultaneously (yes, limited by the size of memory, etc.)
More precise classification:

• **multiprogrammed systems** - several tasks can be started and left un- finished; the CPU is assigned to the individual tasks by rotation, task waiting to the completion of the I/O operation (or other event) are blocked to save CPU time

• **time-sharing systems** - the CPU switching is so frequent that several users can interact with the computer simultaneously - interactive processing

Classification
From the hardware point of view :
- **software** = set of instructions
  - either always in memory (resident)
  - either loaded on request (non-resident or transient)

From the user point of view : Classification from the functionality
- **System software** :
  - Operating systems (including monitor, supervisor, …)
  - Loaders
  - Libraries and utility programs
- **Support software** (developpers) :
  - Assemblers
  - Compilers and interpreters
  - Editors
  - Debuggers
- **Application software**

The supervisor (monitor or kernel)

Memory Resident

- The utility programs are stored on the secondary storage device
- **Loaded** into memory at power-on (bootstrapping)
- On **request** (user or automatically) : tasks execution
- User interface : Job Control Language task(JCL)

(Macro processors)

Introduction:

A macro represents a commonly used group of statements in the source pro ramming language. The macro processor replaces each macro instruction with the corresponding group of source language statements. This is called expanding of macros.

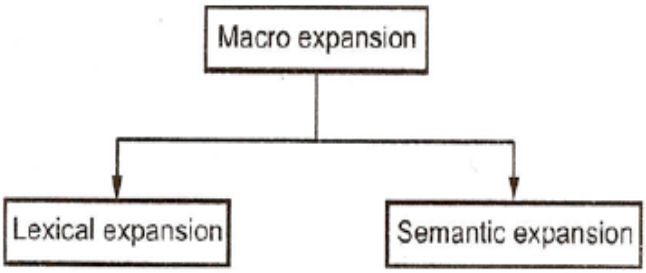
Basic Macro Processor Functions:

We study the fundamental functions that are commonly used by all macro

•**Macro Definition and Expansion**

• Macro is a unit of specification for program generation through expansion.

• A macro consists of a name, a set of formal parameters and body of code.



• **Macro expansion** is a macro name with a set of formal parameters is replaced by some code.

• **Lexical expansion** is replacement of a character string by another character string during program generation.

• **Semantic expansion** is generation of instructions tailored to the requirements of specific usage.

• Macro definition consists of

i) macro prototype ii) one of more model iii) macro preprocessor

• Macro definition is in between a macro header statement and a macro e statement.

• Syntax of macro prototype statement

< macro name> [< formal parameter spec> [;
...]] where

< macro name> = It is in the mnemonic field of an assembly statement < formal parameter spec > is in the following form

< parameter name > [< parameter kind >]

• Syntax of the macro call is

< macro name> [< actual parameter spec> [; ...]]

Macro Expansion

• Macro call leads to macro expansion.

• Macro call statement is replaced by a sequence of assembly statement in macro expansion.

• Two key notions are used in macro expansion

a) Expansion time control flow

b) Lexical substitution

• Algorithm for macro expansion

1. Initialize the macro expansion counter (MEC)
2. Check the statement which is pointed by MEC is not a MEND

```
statement a) if the statement is model statement
        then
                expand the statement
                and increment the MEC
                by 1
    else
    MEC := new value specified in the
statement;
```

3. Exit from the macro expansion

• RDBUFF and WRBUFF are the two macro instruction used in the Fig. SIC/XE program. MACRO and MEND are the two new assembler directives also used. RDBUFF is the name of macro which is in the label field. The entry in the operand field identify the parameters of the macro instruction.

• Each parameter begins with the character & which facilities the substitution of parameters during macro expansion.

• The macro name and parameters define a pattern or prototype for the macro instruction used by the programmer. Body of the macro definition is defined by the MACRO' directive statements. Macro expansion generate these statements. End of the macro definition is defined by the MEND assembler directive.
• In the main program . Macro invocation statement that gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro .
• Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from the macro invocation substituted for the parameters in the macro prototype. Parameters and arguments, both are associated with one another according to their positions.
• The argument F1 is substituted for the parameter &INDEV whenever it occurs in the body of the macro. BUFFER is substituted for &BUFADR and &LENGTH is substituted for &RECLTH.
After macro processing, the expanded file can be used as input to the assembler.

Macro Processor Algorithm and Data Structures:

• For designing two pass macro processor, all macro definitions are processed during the first pass and all macro invocation statements are expanded during the second pass. This type of two pass macro processor would not allow the body of one macro instruction to contain definitions of other macros. The reason behind is that, all macros defined during the first pass before any macro invocation were expanded.
• Following example shows the definition of macros within a macro body.

```
1    MACROS   MACRO    //define SIC standard version macros
2    RDBUFF   MACRO
                .
                .
3                MEND     //END OF RDBUFF
4    WRBUFF   MACRO
                .
5                MEND     //END OF WRBUFF
                .
6                MEND     //END OF MACROS
```

figure (A)

```
1       MACROX   MACRO
2       RDBUFF   MACRO
                 .
                 .
3                MEND      //END OF RDBUFF
4       WRBUFF   MACRO
                 .
                 .
5                MEND      //END OF WRBUFF
                 .
                 .
6                MEND      //END OF MACROX        figure (B)
```

• The body of first macro (MACROS) contains statements that define RDBUFF WRBUFF and other macro instructions for a SIC system. Second macro instruction body (MACROX) defines these same macros for a SIC\XE system. The same program could run on either a standard SIC machine or a SIC\XE machine. Invocation of MACROS or MACROX is only changed for use. Defining MACROS or MACROX does not define RDBUFF and other macro instructions. These definitions are processed only when an invocation 0MACROS or MACROX is expanded.

• Macro processor uses three main data structure.

1. Definition table (DEFT AB)
2. Name table (NAMTAB)
3. Argument table (ARGT AB)

• Definition table stores macro definitions. It contains the macro prototype and the statements that make up the macro body. The macro names are also entered into NAMT AB, which serves as an index to DEFT AB.

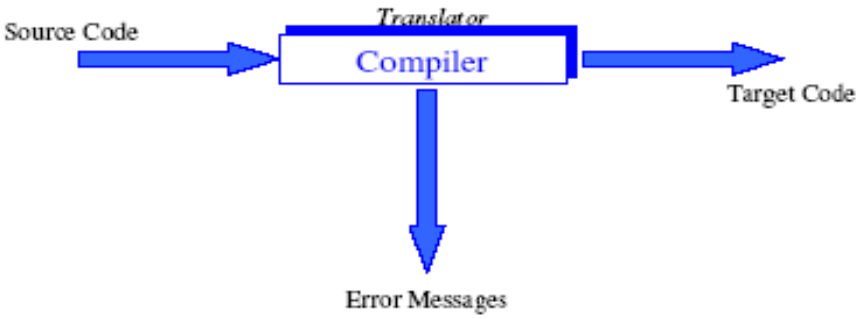
• When macro instruction defined, NAMTAB contains pointers to the beginning and end of the definition in DEFTAB.

• Argument table (ARGT AB) is used during the expansion of macro invocations. When a macro invocation statement is recognized the argument are stored in ARGTAB according to their position in the argument list.

Compilers

Introduction

Compiler is tool: which translate notations from one system to another, usually from source code (high level code) to machine code (object code, target code, low level code).



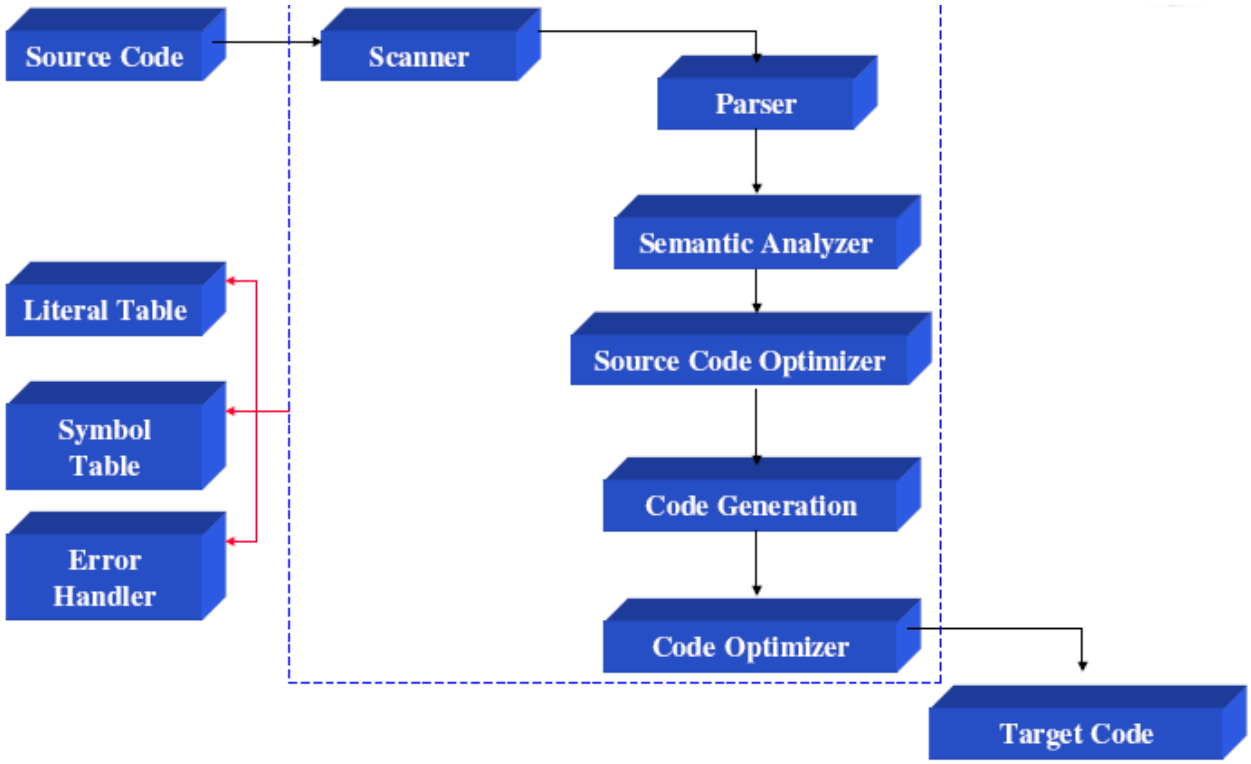
- A compiler is a **program** that translates a **sentence**
  a. from a **source** language (e.g. Java, Scheme, LATEX)
  b. into a **target** language (e.g. JVM, Intel x86, PDF)
  c. while preserving its **meaning** in the process
- Compiler design has a **long** history (FORTRAN 1958)
  a. lots of **experience** on how to structure compilers
  b. lots of **existing** designs to study (many freely available)
  c. take CS 152: Compiler Design for **some** of the details. . .

- We use natural languages to communicate
- We use programming languages to speak with computers

Components of a Compiler

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis

- Intermediate Code Generation
- Code Optimization
- Code Generation



The Scanner

a. Performs **lexical analysis**

b. Collects character sequences into **tokens**

c. Example: **a[index] = 4 + 2**

Tokens:		
	a	identifier
	[	left bracket
	index	identifier
	]	right bracket
	=	assignment
	4	number
	+	plus sign
	2	number

The scanner *may* also:

1. Enter identifiers into the symbol table
2. Enter literals (numeric constants and strings) into the literal table

Lexical Analysis (LA)

Maradona kicks the ball

Token Generation:

- Maradona
- kicks
- the
- ball

Who performs this ?

Lexical Analysis

```
X = Y + 30
```

Token Generation:

- `X` id_1
- `=` operator
- `Y` id_2
- `+` operator
- `30` literal/constant

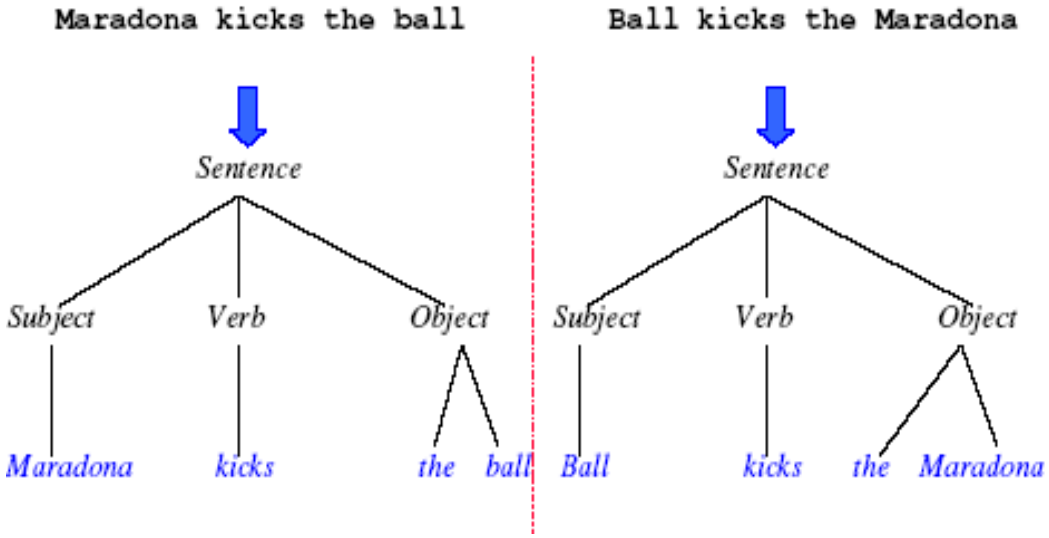
What other functions Scanner can perform ?

Syntax Analysis (SA)

The Parser
a. Performs **syntax analysis**
b. Builds a **parse tree** or **syntax tree**

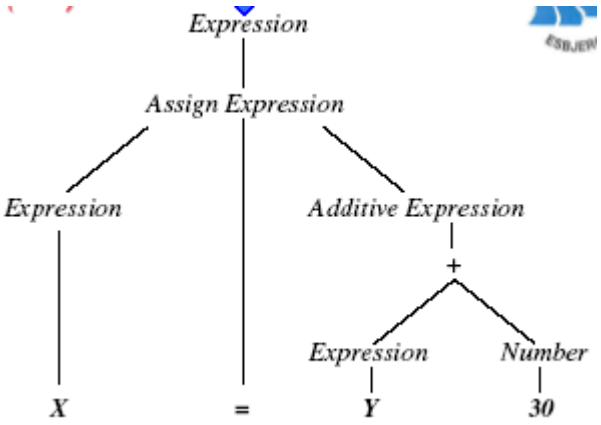
Parse tree for **a[index] = 4 + 2**

Structure of the program is determined by SA. Some thing similar to grammatical analysis.

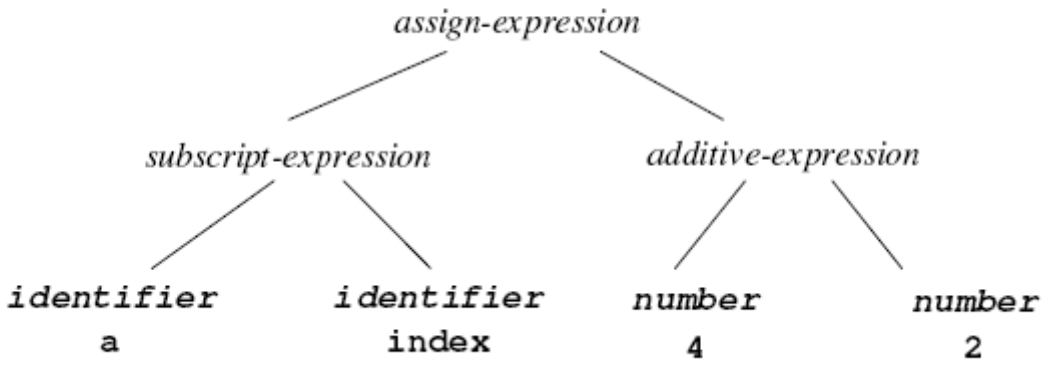


An **abstract syntax tree** is a more concise version of a parse tree.

```
X = Y + 30
```



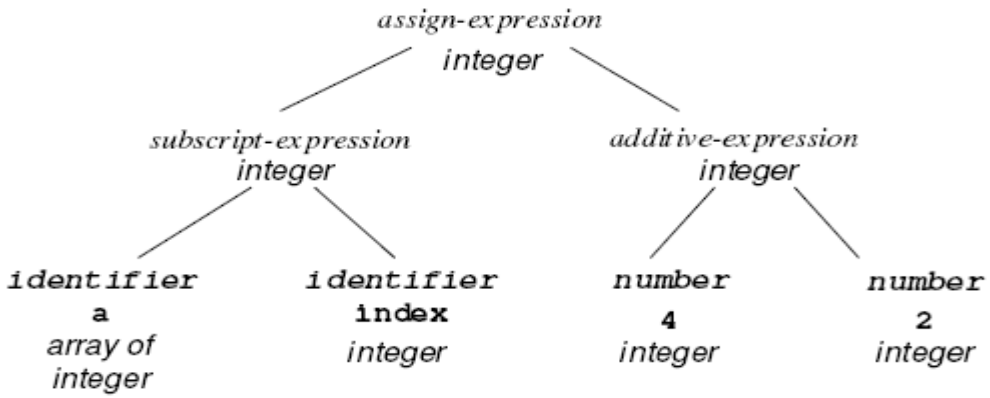
Some time syntax tree is also called as Abstract Syntax tree and could be a "trimmed" version of the parse tree with only essential information:
For Example: a[index] = 4 + 2



Semantic Analyzer

This attach meaning of tokens; For example to the same expression

a[index] = 4 + 2



Intermediate Code Generation

Same example: X = Y + 30

Temp1 = 30

Temp2 = Y

Temp3 = Temp2 + Temp1

X = Temp3

Code Optimization

Same example: X = Y + 30

Temp1 = 30

Temp2 = Y

X = Temp2 + Temp1

Code Generation

Same example: X = Y + 30

Movei Y, r1

Addi 30, r1

Movei r1, X

Algorithmic Tools

- **Token:**
  - Using Regular Expressions.
- **Scanner:**
  - Implementation of finite state machine to recognize tokens.
- **Parser:**
  - An Automaton (i.e. uses a stack), based on grammar rules in a standard format (BNF -- Backus Naur Form).
- **Semantic Analyzer and Code Generator:**
  - Recursive evaluators based on semantic rules for attributes (properties of language constructs).

Error handling

- One of the difficult part of a compiler to design.
- Must handle a wide range of errors
- Must handle multiple errors.
- Must not get stuck.
- Must not get into an infinite loop.

Kinds of errors

- **Syntax**:

```
if (x == 0) y + = z + r; }
```

- **Semantic**:

```
int x = "Hello, world!";
```

- **Runtime**:

```
int x = 2;
...
double y = 3.14159 / (x - 2);
```

Error Handling Requirements

- A compiler must handle syntax and semantic errors, but not runtime errors (whether a runtime error will occur is million dollar question).

- Sometimes a compiler is required to generate code to catch runtime errors and handle them in some graceful way (either with or without exception handling). This, too, is often difficult.

Major Compiler Data Structures

a. Tokens
   1. Represented as an enumerated type
   2. May require other information:
      a. Spelling of identifier
      b. Numeric value
   3. Scanner needs generate only one token at a time (single symbol lookahead)

b. Syntax Tree
   1. A linked structure built by the parser, with information added by the semantic analyzer
   2. Each node is a record whose fields contain information about the syntactic construct which the node represents
   3. Node may be represented using a variant record

c. Symbol Table
   1. Keeps information about identifiers
   2. Efficient insertion and lookup is required → hash table or tree structure may be used
   3. Several tables may be maintained in a list or stack

d. Literal Table

1. Stores constants and strings used in a program
2. Data in literal table applies globally to a program → deletions are not necessary

e. Intermediate code
   1. Could be kept in an array, temporary file, or linked list
   2. Representations include P-code and 3-address code

f. Temporary files
   1. May not be used if memory constraints are not a problem
   2. **Backpatching** of addresses necessary during translation