Unit III

3 Linkers and Loaders

3.1 : Introduction

Q.1 What is loader ?

Ans. : Loader is a 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.

Q.2 Discuss four different functions of loader.

☞ [SPPU : April-18, Marks 4]

Ans. : 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**.

3.2 : Loader Schemes

Q.3 What are types of loaders ? Explain compile and go loader scheme with advantages and disadvantages. ☞ [SPPU : April-18, Marks 6]

Ans. : Types of Loaders :

Based on various functionalities of the loader, there are various types of loaders, which are known as loading schemes. Various loaders schemes are -

1. Compile and Go
2. General Loader Scheme
3. Absolute Loader
4. Subroutine Linkage
5. Relocating Loaders
6. Direct Linkage Loader

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 loader contains the instruction using which the location counter is set to the start of the newly assembled program.
- The typical example is WATFOR-77, its a FORTRAN compiler which uses such "load and go" scheme. This loading scheme is also called as **"assemble and go".**

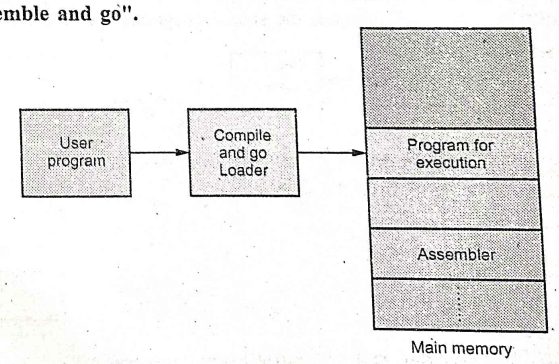


Fig. Q.3.1 Compile and go loading scheme

Advantage : It is simple to implement.

Disadvantages : 1) Assembler occupies memory. Hence memory space remains occupied.

2) There is no production of .obj file, the source code is directly converted to executable form.

Q.4 Explain general loading scheme (using suitable diagram) with advantages and disadvantages.

☞ [SPPU : Dec.-18, End Sem, March-20, In Sem, Marks 5, April-19, In Sem, Marks 6]

Ans. : 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

1. 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. If program is not modified, then loader can make use of this object program to convert it to executable form.
2. 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.
3. It is possible to write source program with multiple programs and multiple languages, because the source programs are first converted to

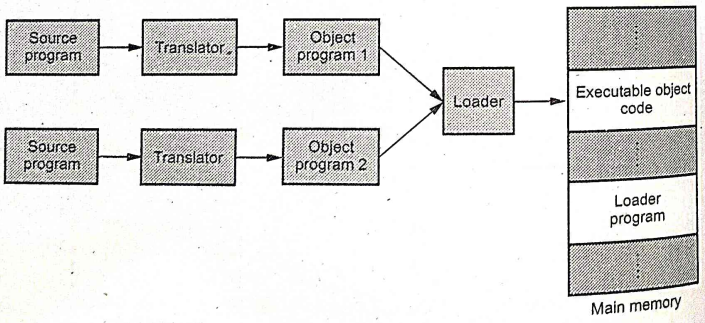


Fig. Q.4.1 General loader scheme

object programs always, and loader accepts these object modules to convert it to executable form.

Q.5 What are subroutine linkages ? What are the benefits of using it ? ☞ [SPPU : May-19, Marks 4]

Ans. : Subroutine Linkage : It is a loading scheme which makes it possible to call and return from subroutines.

In this mechanism the assembler pseudo-op **EXTRN** is followed by list of symbols. These are basically the symbols which are defined externally but referenced in the present program.

Similarly if a symbol is defined in one program and is referenced in others, then this type of symbol is preceded by pseudo-op **ENTRY** in the symbol table.

For example -

```
MAIN    START
        EXTRN   B  //subroutine B is defined externally
        -----
        -----
        L           15,A(B)
        BALR        14,15
        .
        .
        .
        END
```

```
B       START
        USING       *,15
        .
        .
        BR          14
        END
```

Assembler Linkage Pseudo-Ops

Suppose we have a main program named **A** and it uses variables A1,A2,A3.. which are defined in the same program then the pseudo-op ENTRY is used for their declaration

```
A       START
        ENTRY       A1,A2,A3

A1      -----
A2      -----
```

Subroutine Reference(Extrn Pseudo-Op)

Assembler symbols can be internal or external. External symbols are those symbols whose value is not known to the assembler but will be provided by loader. These symbols are referred by the assembler using **EXTRN**

For example -

EXTRN B1,B2

Here B1 and B2 are the extern symbols.

Q.6 Explain : Absolute loader. ☞ [SPPu : Dec.-19, Marks 2]

Ans. : • 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.

Q.7 What is the difference between Absolute loader and Compile and Go loader ?

Ans. : Difference between Absolute and Compile and Go loader :

Absolute loader	Compile and Go loader
The assembled code is placed in relocated object files. Then loader accepts these files and object code is place at some assigned memory location.	The assembled code is directly summed into the memory at some assigned memory location.
This scheme allows to load object modules of multiple programs written in different languages.	This scheme does not allow object modules of different source programs.
This is efficient scheme.	As in this type of loader the instruction is read line by line, its object code is obtained and it is then directly put into the memory, this scheme is less efficient but simple to implement.

The drawback of this method is that assembler and loader occupies large block of memory. Hence there is a wastage of memory.	In this scheme it is programmers duty to adjust all the inter segment addresses and manually do the linking activity. If at all any modification is done in some segment then starting address of next subsequent segment changes. The programmer has to take care of all these issues.

Q.8 What are the advantages and disadvantages of relocating loader ?

Ans. : Advantages :

1. Using branch instruction to corresponding subroutine the desired subroutine can be accessed. Thus the four functions of loader i.e. allocation, linking, relocation and loading are performed **automatically** by the loader.
2. Using **relocation bits** it can be identified that which instruction needs to be relocated and which instruction needs to be directly placed.
3. Using transfecter vector linking of external subroutines can be done with the main procedure. Thus transfer vector of relocating loader helps in solving the external references.
4. In this loader scheme, the assembler provides the additional information such as length of entire program and length of transfer vector to the loader. Using this information allocation problem gets solved.

Disadvantages :

1) The transfer vector links are useful for transferring the control to external subroutines but this vector table is not well-suited for loading or storing external data (i.e. data present in another program segment).
2) Due to transfer vector, the size of object program in the memory gets increased.
3) The relocating loader handles the shared procedure segments but it can not handle the shared data segments.

Q.9 Explain working of Binary Symbolic Subroutine(BSS) loader with example. [SPPU : Dec.-19, Marks 6]

Ans. : • When a single subroutine is changed then all the subroutines need to be reassembled. The task of allocation and linking must be carried out once again. To avoid this rework a new class of loaders is introduced which is called **relocating loader.**

- **Example :** The **Binary Symbolic Subroutine** (BSS) loader used in IBM 7094 machine is a relocating loader.
- In BSS loader there are **many procedure segments** and **one common data segment.**
- The assembler reads the source program, assembles each procedure segment independently. This information along with intersegment references is passed on to loader.
- The output of relocating loader is the object program and the program references. In addition to this the **relocation information** is given. Relocation information is the information about the locations that need to be changed. If it is to be loaded in an arbitrary place in core.

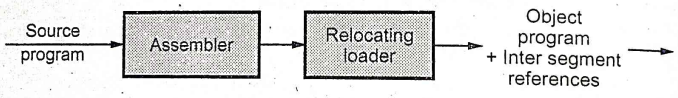


Fig. Q.9.1 Relocating loader

- The assembler takes the source program as input, this source program may call some external routines. Hence assembler produces the object code as an output along with a table. This table contains entries of the external references made in the program. This table is called **transfer vector.**

For example : If a program refers a routine for summing up the numbers (SUM) then transfer vector would contain the symbolic name **SUM.**

The statement calling **SUM** would be translated into a transfer instruction indicating the pointer to the location of the transfer vector associated with **SUM.**

- The assembler also provide some additional information to the loader which includes :
  - Length of entire program.
  - Length of transfer vector portion.
- The loader loads the object program and transfer vector into core. Then it would load each subroutine identified in the transfer vector.
- The loader then place the transfer instruction to the corresponding subroutine for each entry in transfer vector. Thus execution of the **Call for SUM** routine will result in branch to the first location in transfer vector. This location will provide transfer instruction to the location of **SUM.**
- The BSS loader scheme is normally used on the computers with a fixed length direct address instruction format.

Q.10 Explain the concept of transfer vector in BSS loader

Ans. : • The assembler takes the source program as input, this source program may call some external routines. Hence assembler produces the object code as an output along with a table. This table contains entries of the external references made in the program. This table is called **transfer vector.**

For example : If a program refers a routine for summing up the numbers (SUM) then transfer vector would contain the symbolic name **SUM.**

Q.11 Give the direct address instruction format for BSS loader scheme

Ans. : • The BSS loader scheme is normally used on the computers with a fixed length direct address instruction format. This format is as follows -

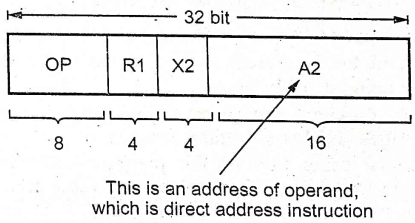


Fig. Q.11.1

Where A2 is 16 bit absolute address of operand. This is a direct address instruction format. This format is useful if there are less than $2^{16} = 65,356$ bytes of storage.

Q.12 Define the term - Relocation bit.

Ans. : Some instructions in the program source need to be relocated while other need not be relocated. Assembler associates a bit called **relocation bit** with each instruction or address. If this bit is equal to 1 then that means corresponding address field must be relocated, otherwise that address field need not be relocated.

Q.13 Which information does the assembler submit to the direct linking loader ?

Ans. : The source program is read by assembler and **assembler submits following information to loader -**

i) The length of program/segment.

ii) A list of symbols in the segment which may be referenced by other segment and relative address of these symbols within the segment.

iii) A list of all the symbols not defined in the segment but referred by other segments.

iv) The information about the locations of address constants, in the segment along with the description of how to revise them.

v) The translated machine code along with the assigned relative address.

3.3 : Overlay Structure

Q.14 Explain overlay structure. ☞ [SPPU : Dec.-19, Marks 2]

Ans. : Refer Q.15.

Q.15 Write short note on - Overlay structure

Ans. : • **Definition of Overlay :** Overlay is a part of the program that is currently required for the execution of program and it has the same load origin as other parts of the program.

• **Definition of Overlay Structure :** The program containing multiple overlays is called overlay structured program.

• There might be some part of the program that need to be loaded permanently in the memory for execution of other parts of the program. Such a permanent resident part of the program is called **root.** Other parts of the program can be loaded in the memory as per the requirements.

• In assembler, the overlay structure is used in assembler.

• Following Fig. Q.15.1 represents the concept -

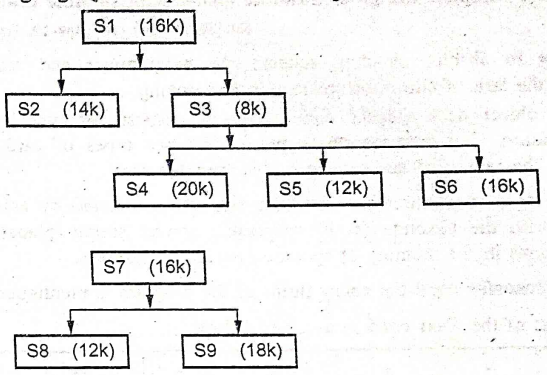


Fig. Q.15.1 Overlay structure

• It is clear from the figure that some parts of the program can reside in the memory simultaneously. Thus the total memory allocated is 78K with overlay structure whereas the memory requirement of entire program is 132K. Thus lot of space can be saved using overlay structuring.

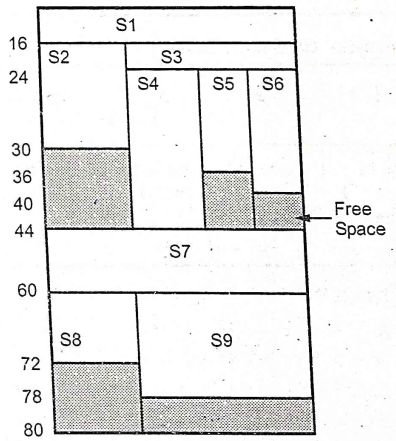


Fig. Q.15.2 Segments stored in the memory using overlay structure

3.4 : Design of Absolute Loader

Q.16 Give complete design of absolute loader with suitable example.

☞ [SPPU : Dec.-18, May-19, Marks 6]

Ans. : • In absolute loading scheme the programmer and assembler perform the task of allocation, relocation and linking.

- Every object deck (object code program) consists of two types of information. This information is present on two types of cards. First type is the **text card** and second one is **transfer card**.
- On the **text card**, there are machine instructions created by assembler along with the absolute (core) addresses. Loader simply places these instructions in the memory at specified absolute addresses.
- On the **transfer card** the **entry point** of the program is mentioned.

The format of the **Text card** is as given below

Column	1	2	3 to 5	6 - 7	8 - 72	73 - 80
Contents	Type of card (= 0 for text card)	Count of information (i.e. length of the code)	Addresses at which the translated code must be place.	Empty	Instructions and data to be loaded in memory	Card sequence number

The format of **Transfer card** is as follows -

Column Number	1	2	3 - 5	6 - 72	73 - 80
Contents	Type of card (= 1 for transfer card)	Count = 0	Address of entry point	Empty	Card sequence number

When the card is read it is stored in 80 continuous bytes in memory.

- Design of absolute loader can be illustrated by following flowchart -

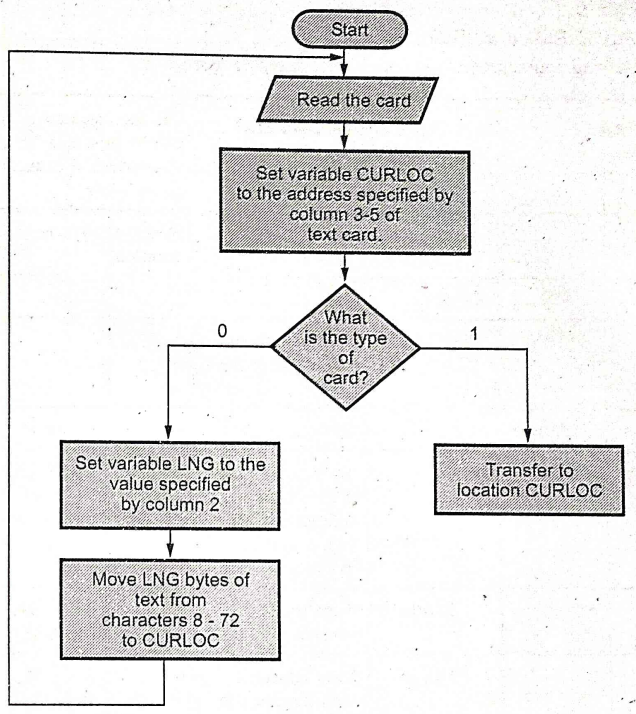


Fig. Q.16.1 Flowchart for design of absolute loader

3.5 : Design of Direct Linking Loader

Q.17 Explain the formats of ESD, RLD, TXT and END cards with respect to direct linking loader with suitable example.

☞ [SPPU : May-19, Marks 6]

Ans. : The four types of cards produced during direct linking loader are -

ESD,TXT,RLD and END

Following source code is considered as input to assembler.

Line No.			Comment	The translated code will be produced by assembler and is as given below -	
				Relative location	Object code
1.	MAIN	START	Start		
2.	ENTRY	RESULT	Locally defined subroutine (entry point)		
3.	EXTRN	SQRT	External subroutine		
4. 5.	BALR USING	12, 0 *, 12	Set register 12 as base address and instruct assembler to use it as base register.	0	BALR 12,0
6.	ST	14, TMPLOC	Store return address	2	ST 14, 54 (0, 12)
7.	L	1, PTRVAL	Load register 1 with the constant PTRVAL i.e. with table values	6	L 1, 46 (0,12)
8.	L	15, ASQRT	Load register 15 with the constant ASQRT	10	L 15, 58 (0, 12)
9.	BALR	14, 15	Call subroutine SQRT	14	BALR 14, 15

10.	ST	1, RESULT	Store return address in RESULT	16	ST 1, 50 (0, 12)
11.	L	14, TMPLOC	Get return address in register 14	20	L 14, 54 (0, 12)
12.	BR	14	Return to caller subroutine.	24	BCR 15, 14
				26	--
13.	TABLE	DC F' 11, 12, 13, 14, 15'	Table is defined by constants 11, 12, 13, 14, 15	28	11
				32	12
				36	13
				40	14
				44	15
14.	PTRVAL	DC A (TABLE)	Address of Table is stored in a constant which is labelled as PTRVAL	48	28
15.	RESULT	DS F		52	--
16.	TMPLOC	DS F		56	--
17.	ASQRT	DC A(SQRT)	Storing address of subroutine SQRT in a constant	60	?
18.		END		64	-

ESD : The External Symbol Dictionary (ESD) card contains information about all external symbols. These symbols are of two types -

i) The symbols are defined in this program but referred elsewhere

ii) The symbols referred in this program but defined elsewhere.

For above given source code, the ESD card will be -

Reference Number	Symbol	Type	Relative location	Length
1.	MAIN	SD	0	64
2.	RESULT	LD	52	-
3.	SORT	ER	-	-

The first entry is for the MAIN program, the type is Segment Definition (SD), the relative location of this program is 0, and total length is 64 bytes. The second entry is for locally defined symbol RESULT. The type is Local Definition (LD). The relative location is 52.

Then the entry for the symbol SQRT is made. This segment is an External Reference (ER). The location and length of this subroutine is not known. Hence kept blank. These entries can be obtained from RLD cards.

- **TXT :** This text card contains the translated code produced by assembler.
  For above discussed example the TXT card will be -

TXT card

Reference Number	Relative Location	Object Code
4	0	BALR 12, 0
6	2	ST 14, 54 (0, 12)
7	6	L 1, 46 (0, 12)
8	10	L 15, 58 (0, 12)
9	14	BALR 14, 15
10	16	ST 1, 50 (0, 12)
11	20	L 14, 54 (0, 12)
12	24	BCR 15, 14
13	26	-
13	28	11
13	32	12

13	36	13
13	40	14
13	44	15
14	48	28
17	60	0

- **RLD** The Relocation and Linkage Directory (RLD) contains following information :
  1) The relative location
  2) By what the constant must be changed to
  3) The operation to be performed.

The RLD Card for above example will be

RLD card

Reference Number	Symbol	Flag	Length	Relative location
14	MAIN	+	4	48
17	SQRT	+	4	60

The + sign denotes that something must be added to the constants. The two constant symbols MAIN and SQRT are relocated. The relative address of MAIN is 48 and that of SQRT is 60.

- **END** : The END card indicates end of object deck/program

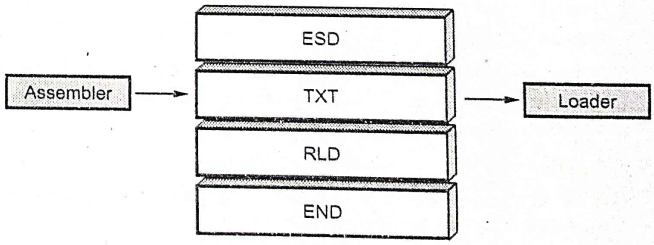


Fig. Q.17.1 Four types of cards

- The assembler submits the information in these four cards to the loader.

Q.18 Explain the working of direct linking loader by Pass1 algorithm and flowchart.

Ans. : Algorithm :

Pass 1 : Symbol definition and allocation of segments

- In this pass the location for each segment is defined. The values to external symbols are assigned. The amount of core storage required for total program must be minimized.

1) The first program segment must be loaded at the address Initial Program Load Address (IPLA). This address is normally specified by operating system.

2) An object card is read and a copy is prepared for pass 2 to read. The object card can be of ESD, TXT, RLD, END or LDT/EOF type.

   i) If the card is of **TXT** or **RLD** type then no processing is done. Read the next card.

   ii) If the card is of **ESD** type then the processing will be based on SD, LD or ER. If the type of the symbol is **SD** then **length** field is read in variable SLENFGTH. The value is in variable VALUE which is equal to Program Load Address (PLA).

   Then it is checked whether this symbol entry is made in **GEST** or not. If symbol entry is not there in GEST then store the symbol name and value. The symbol and value is printed as load map. Similar process is carried out for **LD**. The ER symbols do not require processing.

   iii) If the card is of type **END** then the load address of program is incremented by length of the segment. This length is stored in variable.

SLENGTH

   iv) If the card is of type LDT or EOF then pass 1 is supposed to be completed and go to pass 2.

Flowchart for pass 1 :

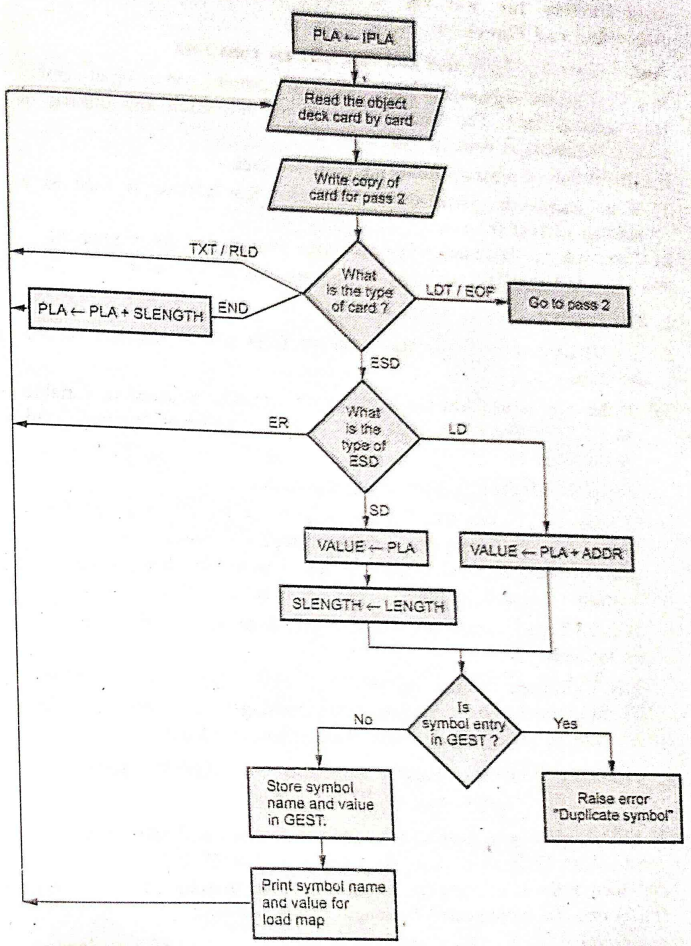


Fig. Q.18.1 Flowchart for Pass 1

Q.19 Explain the working of direct linking loader by Pass2 Algorithm and Flowchart

Ans. : Pass 2 : Load text and link address constants

In pass 1 all the segments get assigned with locations and external symbol table gets defined. The loading of text and relocation and linking of address constants is done in pass 2.

For Execution of program following rules are used -

1) If an address is specified on END card that address is used as a starting address for execution.
2) Otherwise the execution will begin from starting address of segment.

The variable EXADDR will store starting address of execution.

1. The object deck is used card by card.
2. If ESD card is read then based on the ESD type appropriate actions are taken -
   i) If the type is **SD** then the length of the segment is saved in variable SLENGTH. The LESA (ID) is set to current value of Program Load Address.
   ii) If type is **LD**, then no processing is required.
   iii) If type is ER, then GEST entry is searched for ER symbol. If the entry is not found the error message is issued. But if the corresponding external symbol is found in GEST then its value is extracted and corresponding LESA(ID) is set with this value.
3. When TXT card is read the text is copied from TXT card to relocated core location.
4. When RLD Card is read, the relocation and linking values of LESA (ID) are extracted and depending upon the flag values these relocation values can be added or subtracted from address constants

∴ **Actual address const = PLA + ADDR field specified on RLD Card.**

5. If END card is read, then check whether an execution start address is specified on END card; if so, then first add it to PLA (i.e. relocation) and then store it in variable EXADDR. The Program Load Address (PLA) must be incremented by length of the segment.
6. If LDT/EOF card is read, the loader transfers the control to loaded program.

Flowchart for Pass 2 :

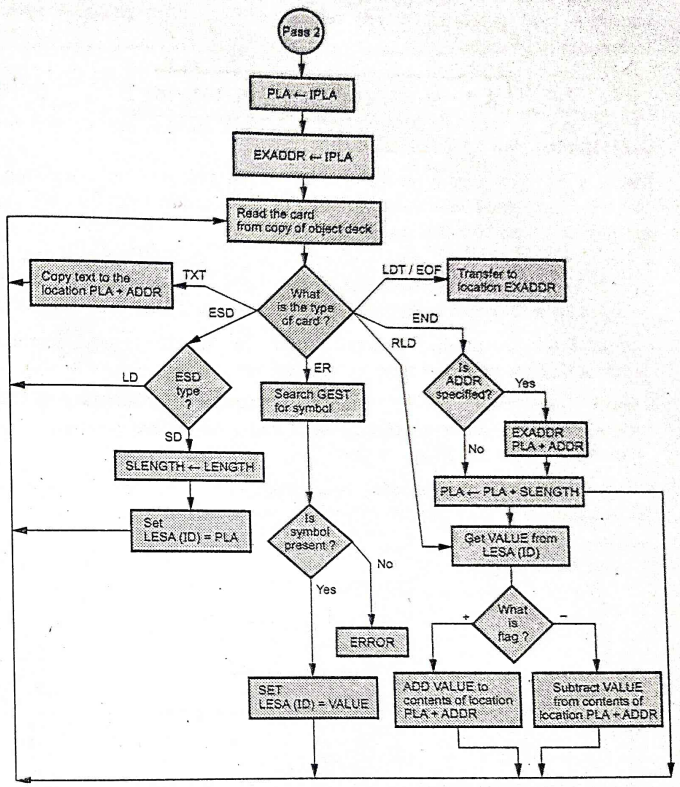


Fig. Q.19.1 Flowchart for Pass 2

3.6 : Self-relocating Programs

Q.20 Explain the term self relocating program.

Ans. : The **self relocatable program** is a program which itself performs relocation. It does not require linkage editor to perform relocation. The self relocating program cannot load itself but relocates itself. The self

relocating program does not bound to a specific memory area for its execution. For preparing a self relocating form of a program special techniques are needed.

3.7 : Static and Dynamic Linking

Q.21 Explain the need for DLL.

Ans. : • DLLs provide a mechanism for shared code and data, allowing a developer of shared code/data to upgrade functionality without requiring applications to be re-linked or re-compiled.

- From the application development point of view Windows can be thought of as a collection of DLLs that are upgraded, allowing applications for one version of the OS to work in a later one.
- kernel32.dll, the primary dynamic library for Windows' base functions such as file creation and memory management
- Many font resource files are also dynamic link libraries having extension .FON. They contain no code and data but have various fonts that any window program can use.
- Thus purpose of dynamic link library is to provide various functionalities and resources required by many different programs.

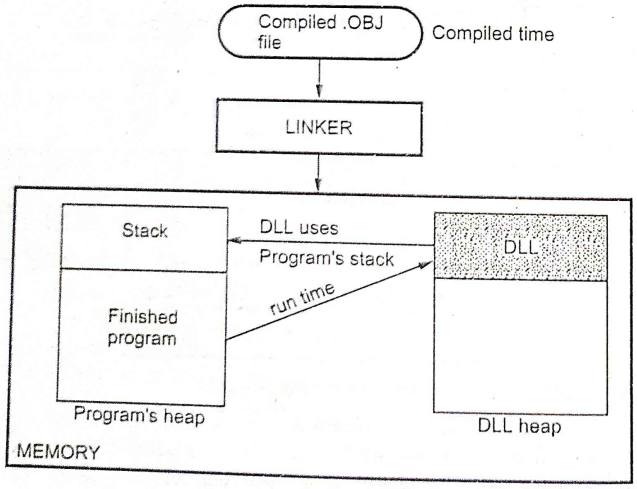


Fig. Q.21.1 Use of DLL

Q.22 What is the difference between DLL and EXE ?

Ans. :

Sr. No.	.DLL	.EXE
1.	It contains code and data.	It contains an executable code.
2.	The .DLL file is linked or referenced to the .EXE at run time.	.EXE runs on its own.
3.	The .DLL extension stands for dynamic link library file.	The .EXE extension stands for an executable file.
4.	It is a library file that can be used by many programs at the same time.	It is an executable file.

Q.23 Differentiate between static and dynamic link libraries.

☞ [May-19, March-20, Marks 4]

Ans. :

Sr. No.	Static link libraries	Static link libraries
1.	For static library, actual code is extracted from library by linker and final executable code is built at the compilation time.	For dynamic link library code need not be extracted and copied rather it is linked with executable code at run time.
2.	Compatibility issues do not arise.	Compatibility issues may arise.
3.	Size is larger.	It is compact in size.
4.	Examples - All the .lib files.	Examples - All the .dll files.

Q.24 What is the need of DLL ? Differentiate between static and dynamic linking.

☞ [SPPU : April-18, Marks 4]

Ans. : Refer Q.21 and Q.23.

END...

Unit IV

4 Operating System

4.1 : Introduction

Q.1 What is operating system ? Draw and explain layered approach with advantages and disadvantages. ☞[SPPU : May-19, End Sem, Marks 6]

Ans. : • Operating system is a program that controls the execution of application programs. It is an interface between applications and hardware.

• OS provides different types of view. For user, it is abstract view because it provides features which are important for users. OS is intermediary between user and the computer system.

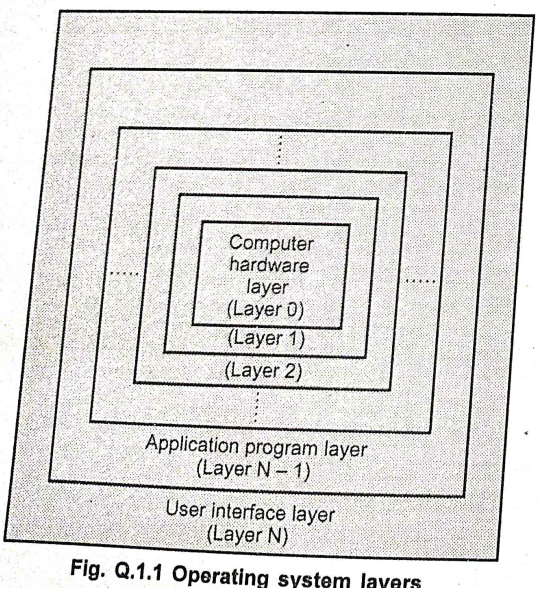


Fig. Q.1.1 Operating system layers

• Operating system is divided into number of layers. Each layer boundary is properly defined. Bottom layer is called layer 0 and top most layer is called layer N. Layer N provides user interface.

• A function of layer is also fixed. Fig. Q.1.1 shows the operating system layer. Each layer consists of data structure and a set of routines. Layer provides services to upper and lower layers. (See Fig. Q.1.1 on previous page.)

• First layer contains only basic hardware to implement function. So only first layer is debugged for checking the whole system. If error is not found then the system will works properly. If error is encounter while debugging second layer, then error is related to the second layer only.

• In this operating system is designed and implemented. It simplifies the design task of the operating system.

• Each layer uses system calls for performing their operation. Layer adds overhead to the system call.

Advantages of the layered approach to system design.

1. Modularity is the advantage of the layered system.
2. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections of the operating system.
3. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer.

Q.2 Why operating system is needed ? Justify "Operating system is the control program of the system."

Ans. : • Operating system is needed because it allows a program to run on multiple different types of hardware. An operating system provides a layer between the program and hardware to enable a program to use a standard interface no matter what hardware is used. The operating system translates the software calls to the hardware calls for that particular piece of hardware.

• The operating system is responsible for managing the all resources. A portion of the OS is in main memory. This portion of the OS is called kernel.

• User program and data is also stored in remaining parts of the memory. Allocation of main memory is controlled by operating system with the help of memory management hardware.

- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Modern OS allows multiple programs to run at the same time. If multiple users are using computer then there is need of managing and protecting the memory, I/O devices and other resources.

Q.3 What are the major functions of operating system ?

Ans. : The major functions of an operating system are :

1. Efficient use of a computer system 2. User convenience
3. Ability to evolve

1. Efficient use : • For efficient use of resources, it must be monitored by operating system. Proper scheduling of resources is also required.

- Computer contains different types of resources like CPU, memory and I/O device etc. Proper monitoring is required on these resources to avoid the overhead. As per the resource, scheduling is required.

2. User convenience : • User convenience is affected by computing environment of the computer system. The computing environment is comprised of computer system, its interfaces with other systems and nature of computations performed by its users.

- Computer architecture and use change the computing environment of the system.

3. Ability to evolve : • An OS 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 interfering with service.

Q.4 Explain operating system as a resource manager.

Ans. : • A computer is a set of resources. These resource provides various functions to the user. Functions like data movement, storing of data and program, operation on data are control by an operating system.

- Fig. Q.4.1 (See Fig. Q.4.1 on next page.) shows OS as a resource manager.
- The operating system is responsible for managing the all resources. A portion of the OS is in main memory. This portion of the OS is called kernel.
- User program and data is also stored in remaining parts of the memory. Allocation of main memory is controlled by operating system with the help of memory management hardware.

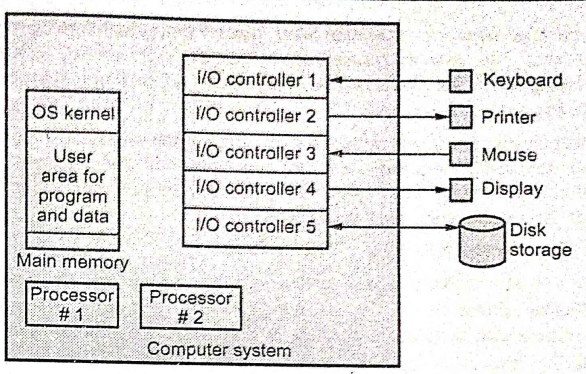


Fig. Q.4.1 OS as a resource manager

- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Modern OS allows multiple programs to run at the same time. If multiple users are using computer then there is need of managing and protecting the memory, I/O devices and other resources.
- Resource management includes sharing resources in different ways. Time and space are the two concept for resource sharing.
  1. **Time :** Time slot is allocated to each program first one gets to use the resource then another and so on.
  2. **Space :** Consider the example of main memory. Main memory is normally divided up among several running programs, so each one can be resident at the same time.

4.2 : Evolution of OS

Q.5 Explain following term clearly : Multiprogramming.

Ans. : • 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.

- Resource management is the main aim of multiprogramming operating system. File system, command processor, I/O control system and transient area are the essential components of a single user operating system.
- Multiprogramming operating system divides the transient area to store the multiple programs and provides resource management to the operating system. The concurrent execution of programs improves the utilization of system resources.

Q.6 Explain the following : Multitasking.

Ans. : • Multitasking is also called as time sharing operating system. It is logical extension of the multiprogramming operating systems. User interaction with program is possible in time sharing operating system.

- During execution of the program, user interacts directly with the program, supplying information to the program.
- Multi-tasking means that the computer can work with more than one program at a time. For example, user could be working with information from one database on the screen analyzing data, while the computer is sorting information from another database, while a excel sheet is performing calculations on a separate worksheet.
- Many users share the computer system simultaneously in time sharing operating system. Time sharing system uses multiprogramming and CPU scheduling. Each user has at least one separate program in memory.
- Concept of virtual machine is used in time sharing system. It creates virtual machine one per user. User interaction with system by using virtual machine. User enters the command for virtual machine and result will received back to user.
- Time sharing system is more complex than multiprogramming operating system. It also takes help of file system. File system is stored on the disk so disk management is also required.

Q.7 Write down advantages of batch processing system.

Ans. :

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.
3. Batch system often provides simple forms of file management.

4. Batch systems do not require any time critical device management.
5. Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e. first come first served fashion.

Q.8 What do you mean spooling? Explain in detail.

Ans. : • When a job completes execution, its memory is released and the output for the job gets copied into an outputs pool 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.
- 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.
- 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. Spooling batch system were the first and are the simplest of the multiprogramming systems.
- In spooling, I/O of one job is overlapped with the computation of another job. For example, a spooler at a time may read input of one job, and at the same time, it may also print the output of another job.
- Spooling can also process data at the remote sites. The spooler only has to notify when a process gets completed at the remote site so that spooler can spool next process to the remote side device.
- Spooling increases the performance of the system by increasing the working rate of the devices. It naturally leads to multiprogramming.

Advantages of spooling

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

Q.9 Explain features of RTOS and time sharing operating system.

Ans. : Features of Real Time Operating System :

1. Real time system fails if it does not give result within the time limits.
2. Real time task cannot keep waiting for longer time without allocating kernel.
3. Real time operating system uses priority scheduling algorithm.

Features of Time Sharing Operating System :

1. User interaction with program is possible in time sharing operating system.
2. Time sharing system uses medium term scheduling such as round robin for the foreground.
3. In time sharing system, each user is given a time slice for executing his/her job in round robin fashion.

Q.10 What is real time OS ? Explain its types with suitable example.

☞ [SPPU : May-19, End Sem, Marks 4]

Ans. : • Time constraints is the key parameter is real time systems. It controls autonomous system such as robots, satellites, air traffic control and hydroelectric dams.

- When user gives an input to the system, it must process within the time limit and result is sent back. Real time system fails if it does not give result within the time limits.
- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period.
- Real time systems are of two types : Hard and soft.

1. Hard Real Time Systems :

- A hard real-time system is one where the response time is specified as an absolute value. This time is normally dictated by the environment.

- A system is called a hard real-time if tasks always must finish execution before their deadlines or if message always can be delivered within a specified time interval.
- Hard real-time is often associated with safety critical applications. A failure (e.g. missing a deadline) in a safety-critical application can lead to loss of human life or severe economical damage.

2. Soft Real Time System :

- A soft real-time system is one where the response time is normally specified as an average value. This time is normally dictated by the business or market.
- A single computation arriving late is not significant to the operation of the system, though many late arrivals might be.
- Soft real time means that only the precedence and sequence for the task-operations are defined, interrupt latencies and context switching latencies are small but there can be few deviations between expected latencies of the tasks and observed time constraints and a few deadline misses are accepted.

Q.11 What is distributed system ? Explain its advantages.

Ans. : • **Definition :** A distributed system is a collection of autonomous hosts that are connected through a computer network.

- A distributed system is a collection of independent computers that appears to its users as a single coherent system. Each host executes components and operates a distribution middleware.
- Middleware enables the components to coordinate their activities. Users perceive the system as a single, integrated computing facility.
- A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network.
- A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers and so on.
- Distributed operating systems depend on networking for their operation. Distributed OS 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 OS.

- Distributing OS owns the whole network and makes it look like a virtual uni-processor or may be a virtual multiprocessor.
- Definition : A distributed operating system is one that looks to its users like an ordinary operating system but runs on multiple, independent CPU.
- Distributed systems depend on networking for their functionality. Fig. Q.11.1 shows the distributed system.

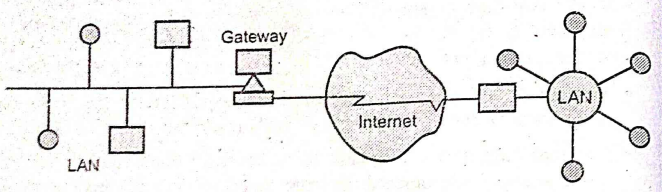


Fig. Q.11.1 Distributed system

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 aspects of reliability. Availability refers to the fraction of time for which a system is available for use.
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.

Q.12 What is Real Time Operating System ? What is the difference between Hard real time and Soft real time operating system ?

Ans. : Real time system is one that must react to inputs and responds to them quickly. A real time system has well defined, fixed time constraints.

Difference Between Hard and Soft Real Time System

Sr. No	Hard real time	Soft real time
1.	Hard response time is required.	Soft response time is required.
2.	Data integrity is short term.	Data integrity is long term.
3.	Size of data file is small or medium.	Size of data file is large.
4.	Peak load performance is predictable.	Peak load performance is degraded.
5.	Hard real-time systems have little laxity and generally provide full deadline compliance.	Soft real-time systems are more flexible. They have greater laxity and can tolerate certain amounts of deadline misses.
6.	Safety critical systems are typically hard real time systems.	Linux and many OS provide soft real time system.

4.3 : Operating System Services

Q.13 What are the different services provided by the operating system ?

Ans. : 1. Program execution : Before executing the program, it is loaded into the memory by operating system. Once program loads into memory, its start execution.

2. **Input :** output operation : Program is combination of input and output statement. While executing the program, it requires I/O device.
3. **Error detection :** Error is related to the memory, CPU, I/O device and in the user program. Memory is full, stack overflow, file not found, directory not exist, printer is not ready, attempt to access illegal memory are the example of error detection.
4. **File and directory operation :** User wants to read and writes the file and directory.
5. **Communication :** Communication may be inter-process communication and any other type of communication. Data transfer is one type of communication.

4.4 : Process Management : Process, Process States : 5 and 7 State Model, Process Control Block

Q.14 Explain the following terms clearly : Process.

Ans. : • Process is an active entity that requires a set of resources, including a processor, program counter, registers to perform its function. Multiple processes may be associated with one program.

- Process means a program in execution. Process execution must progress in sequential order. It also called task. Task is a single instance of an executable program.
- Each process has its own address space. Address space is divided into regions :

1. **Text region :** It stores the code that the processor executes.
2. **Data region :** It stores variables and dynamically allocated memory that the process uses during execution.
3. **Stack region :** It stores instructions and local variables for active procedure calls.

- Process is a dynamic entity that executes a program on a particular set of data using resources allocated by the operating system.
- A process can run to completion only when all requested hardware and software resources have been allocated to the process.

Q.15 Explain process control block in detail.

☞ [SPPU : May-19, End Sem, Marks 6]

OR Explain process control block with suitable diagram.

☞ [SPPU : May-18, End Sem, Marks 6]

Ans. : Process control block : When a process is created by operating system, it allocates a PCB for it. OS initializes PCB and puts PCB on the correct queue. Fig. Q.15.1 shows PCB.

Following information is stored in process control block.

1. **Process identification :** Each process is uniquely identified by the user's identification and a pointer connecting it to its descriptor.
2. **Priority number :** Operating system allocates the priority number to each process. According to the priority number it allocates the resources.

3. **Program counter :** The PC indicates the address of the next instruction to be executed for this current process.
4. **Memory allocation :** It contains the value of the base registers, limit registers and the page tables depending on the memory system used by the operating system.
5. **I/O status information :** It maintains information about the open files, list of I/O devices allocated to the process etc.
6. **List of open files :** Process uses number of files for operation.
7. **Process state :** Process may be in any one of the state : New, ready, running, and waiting, terminate.

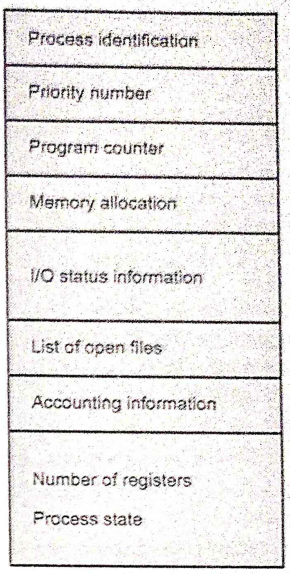


Fig. Q.15.1 Process control block

Q.16 Draw and explain process state transition diagram.

☞ [SPPU : May-18, End Sem, Marks 6]

Ans. :

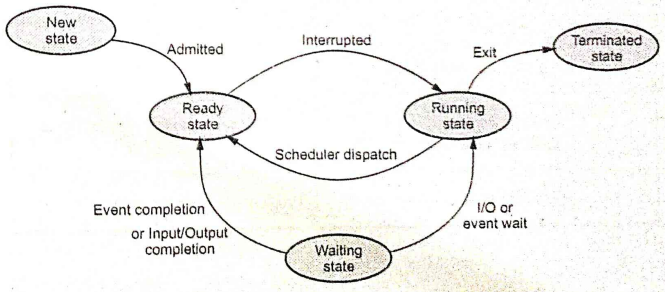


Fig. Q.16.1 Process state diagram

- Fig. Q.16.1 shows a process state diagram. A state diagram is composed of a set of states and transitions between states.
- State diagram is used by process manager to determine the type of service to provide to the process.
- The process states are as follows :

  1. New 2. Ready 3. Running 4. Waiting 5. Terminated
- **New** : Operating system creates new process by using fork( ) system call. These process are newly created process and resources are not allocated.
- **Ready** : The process is competing for the CPU. Process reaches to the head of the list (queue).
- **Running** : The process that is currently being executed. Operating system allocates all the hardware and software resources to the process for execution.
- **Waiting** : A process is waiting until some event occurs such as the completion of an input-output operation.
- **Terminated** : A process completes its operations and releases all resources.

Q.17 Differentiate between the following : Process and program.

Ans. : Difference between process and program

Sr. No.	Process	Program
1.	Process is active entity.	Program is passive entity.
2.	Process is a sequence of instruction executions.	Program contains the instructions.
3.	Process exists in a limited span of time.	A program exists at single place and continues to exist.
4.	Process is a dynamic entity.	Program is a static entity.

4.5 : Threads

Q.18 What is process ? What is thread ? List down benefits of using thread.

[SPPU : May-18, End Sem, Marks 6]

Ans. : Process : Refer Q.14.

Thread :

- Thread is a dispatchable unit of work. It consists of thread ID, program counter, stack and register set. Thread is also called a Light Weight Process (LWP). Because they take fewer resources than a process. A thread is easy to create and destroy.
- It shares many attributes of a process. Threads are scheduled on a processor. Each thread can execute a set of instructions independent of other threads and processes. Fig. Q.18.1 shows a thread.

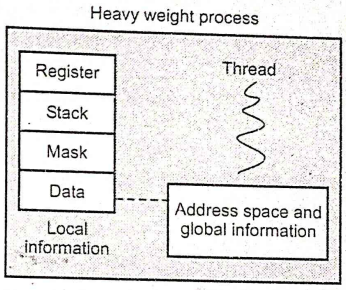


Fig. Q.18.1 Thread

- Every program has at least one thread. Programs without multithreading executes sequentially. That is, after executing one instruction the next instruction in sequence is executed.
- Thread is a basic processing unit to which an operating system allocates processor time and more than one thread can be executing code inside a process.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Threads are of two types : User level thread and Kernel level thread.

Benefit of using thread :

1. Context switching time is minimized.
2. Thread support for efficient communication.
3. Resource sharing is possible using threads.

4. A thread provides concurrency within a process.

5. It is more economical to create and context switch threads.

Q.19 How thread is different from process?

Ans. :

Sr. No.	Thread	Process
1.	Thread is also called lightweight process.	Process is also called heavyweight process.
2.	Operating system is not required for thread switching.	Operating system interface is required for process switching.
3.	One thread can read, write or even completely clean another threads stack.	Each process operates independently of the other process.
4.	All threads can share same set of open files and child processes.	In multiple processing, each process executes the same code but has its own memory and file resources.
5.	If one thread is blocked and waiting then second thread in the same task can run.	If one server process is blocked then other server process cannot execute until the first process is unblocked.
6.	Uses fewer resources.	Uses more resources.

Q.20 Explain thread lifecycle.

Ans. : • Fig. Q.20.1 shows a thread lifecycle. A thread has one of the following states. (See Fig. Q.20.1 on next page.)

1. **New :** Thread is just created.
2. **Ready :** Thread's start ( ) method invoked and now it is executing. OS put thread into ready queue.
3. **Running :** Highest priority ready thread enters the running state. Thread is assigned a processor and now is running.
4. **Blocked :** This is the state when a thread is waiting for a lock to access an object.

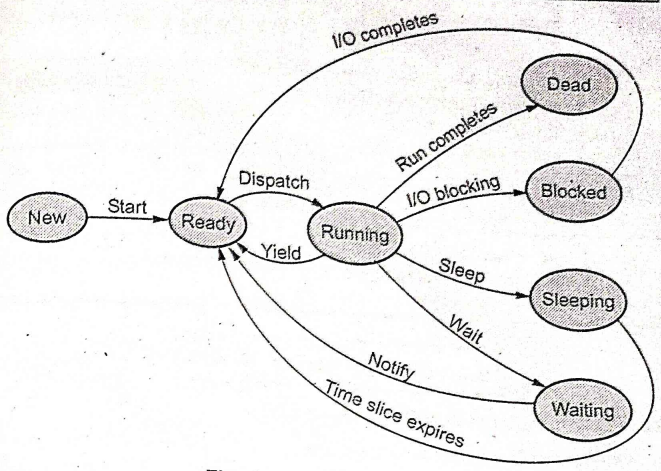


Fig. Q.20.1 Thread lifecycle

5. **Waiting :** Here thread is waiting indefinitely for another thread to perform an action.
6. **Sleeping :** Thread sleep for a specified time of period. When sleeping time expires, it enters to ready state. CPU is not used by sleeping thread.
7. **Dead :** When thread completes task or operation.

Q.21 What is difference between user level thread and kernel level thread ?

Ans. :

Sr. No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to Kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.

4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called Kernel level threads.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.
6.	Example : POSIX P threads and Mach C-threads	Example : Windows 95/98/NT, Sun Solaris and Digital UNIX
7.	User level threads are also called many to one mapping thread.	Kernel level thread support one to one thread mapping.

4.6 : Multithreading Model

Q.22 Explain various multithreading model.

Ans. : • Operating system uses user level thread and kernel level thread. User level threads are managed without kernel support. Operating system support and manage the kernel level threads.

- Modern operating system support kernel level threads. Kernel performs multiple simultaneous tasks in operating system. In most of the application, user threads are mapped with kernel level threads.
- Different methods of mapping is used in operating system are as follows :

1. One to one 2. Many to one 3. Many to many

1. One to one

- In this model, one user level thread maps with one kernel level thread. If there are three user threads then system creates three separate kernel thread. This model provides more concurrency because of separate thread. Fig. Q.22.1 shows one to one mapping.
- In this method, the operating system allocates data structure that represents kernel threads. Here multiple threads are run in parallel on multiprocessor system.
- Number of threads in the system increases, the amount of memory required is also increases.
- Windows 95/XP and Linux operating system uses this one to one thread mapping.

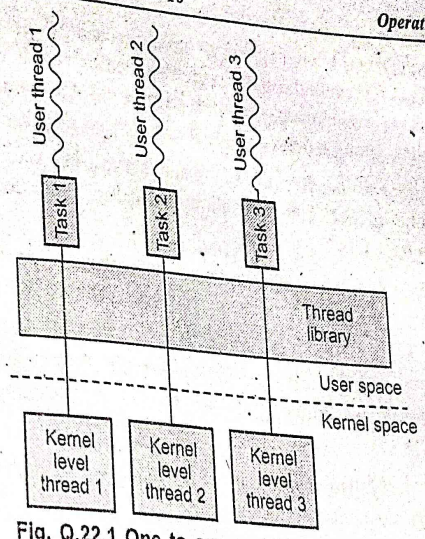


Fig. Q.22.1 One to one multithreading model

- Only overhead in this method is creation of kernel level thread for user level thread. Because of this overhead, system performance is slowdown.

2. Many to one

- Many to one mapping means many user thread maps with one kernel thread. Fig. Q.22.2 shows many to one mapping.

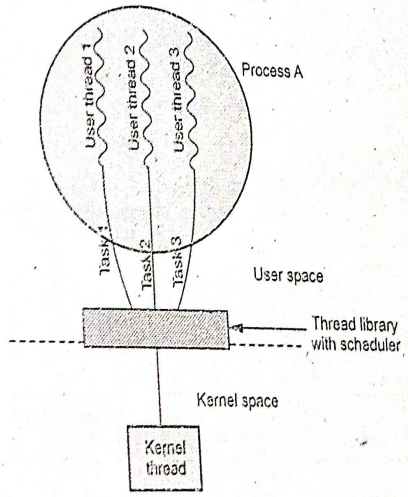


Fig. Q.22.2 Many to one multithreading model

- Operating system blocks the entire multi-threaded process when a single thread blocks because the entire multi-threaded process is a single thread of control. So when operating system receive any a blocking I/O request, it blocks the entire process.
- Thread library handle thread management in user space. The many-to-one model does not allow individual processes to be split across multiple CPUs.
- This type of relationship provides an effective context-switching environment, easily implementable even on simple kernels with no thread support.
- Green threads for Solaris and GNU portable threads implement the many-to-one model in the past, but few systems continue to do so today.
- **Advantage :** System performance is improved by customizing the thread library scheduling algorithm.

3. Many to many

- In the many to many mapping, many user-level threads maps with equal number of kernel threads. Fig. Q.22.3 shows many to many thread mapping.

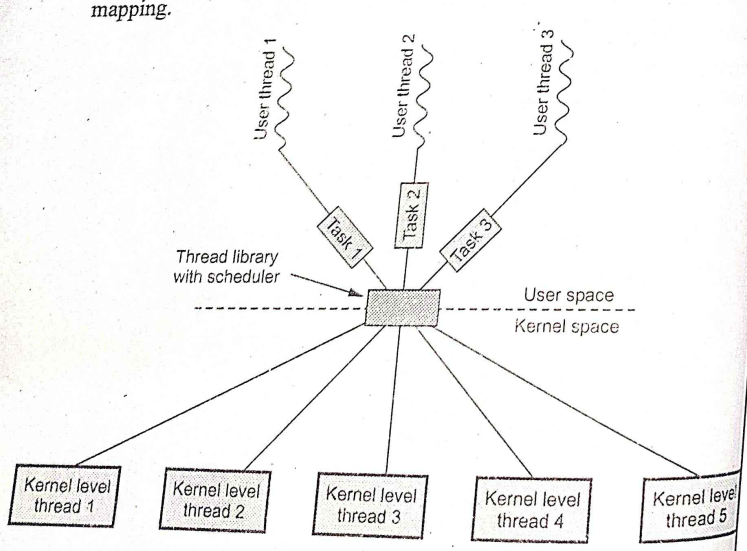


Fig. Q.22.3 Many to many multithreading model

- Many to many mapping is also called M-to-N thread mapping. Thread pooling is used to implement this method.
- Users can create required number of thread and there is no limitation for user. Again here blocking Kernel system calls do not block the entire process.
- This model support for splitting processes across multiple processors.
- **Limitations :** Operating system design becomes complicated.

4.7 : Process Control System Calls

Q.23 How process are created by using system call ? Explain.

Ans. : Operating system creates the process in following situations :

1. Starting of new batch job.
2. User request for creating new process.
3. To provide new services by OS.
4. System call from currently running process.

- Operating system creates a new process with the specified or default attributes and identifier. A process may create several new sub-process.
- Parent process is creating process and the new processes are called the **children** of the process. When operating system creates process, it builds the data structure for managing process and allocates address space in primary main memory.
- Operating system creates foreground and background process. Process is identified by unique Process Identifier (PID) in UNIX and windows operating system. PID value is an integer number.
- All processes in UNIX are created using the fork ( ) system call. The forking process is called the parent process. The new process is called the **child** process.
- Both the parent and child process have their own and private memory. Open files are shared between parent and child.
- If the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork ( ) calls will not be affected even though they have identical variable names.

- When a process is created, OS assigns some attributes. These are priority, privilege level, requirement of memory, access right, memory protection, PID etc. To perform operation, process needs software and hardware resources. It includes CPU time, files, memory, I/O device.
- Relation between parent process and child process is as follows :
  1. Parent process continues to execute concurrently with its child process.
  2. Parent process waits until some or all of its children have terminated.

```
void main( )
    {
        printf("Operating System\n");
        fork( );
            printf("Technical Publications\n");
    return 0;
    }
```

- In above program Operating System is printed only once and Technical Publications is printed two times.

4.8 : Process Scheduling

Q.24 Explain scheduling queue, ready queue and device queue with respect to process scheduling.

Ans. : • CPU utilization is maximizing by using multiprogramming concept. Processor is not idle, it is executing a process. Processor scheduler selects one process for execution from the ready queue.

Scheduling queue : Scheduling queue is queue of processes or input-output devices. When the user process enters into the system, they put into the job queue. Job queue consists of all processes of the system.

- Operating system maintains different types of queues for different purposes. The queue are ready queue, device queue etc. Fig. Q.24.1 shows ready queue and device queue.
- **Ready queue :** The processes which are ready and waiting to execute are kept in the ready queue. Ready queue is stored in main memory. Linked list is used for representing ready queue. Pointer field of PCB is used for this.

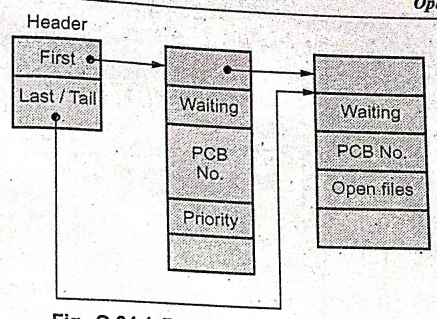


Fig. Q.24.1 Ready and device queue

- **Device queue :** Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has maintained its own device queue.
- Process scheduling is represented by queueing diagram. Newly created process is kept into the ready queue and waits for processor. When CPU select the process for executing, following things happens :
  1. Process may require I/O device to perform operation. So it is put into the I/O queue.
  2. Process may create child process and wait for child process termination.
  3. Because of interrupt, process preempted from CPU and put into the ready queue.

4.9 : Scheduling

Q.25 What are the types of schedulers ? Explain them with suitable diagram. ☞ [SPPU : May-19, End Sem, Marks 8]

OR Explain following types of schedulers :
i) Short term ii) Long term iii) Medium Term. ☞ [SPPU : May-18, End Sem, Marks 6]

Ans. : Schedulers are used to handles process scheduling. It is one type of system software and selects the jobs from the system and decide which process to run. Schedulers are of three types -

1. Long term scheduler 2. Short term scheduler
3. Medium term scheduler

Fig. Q.25.1 shows queueing diagram for process scheduling.

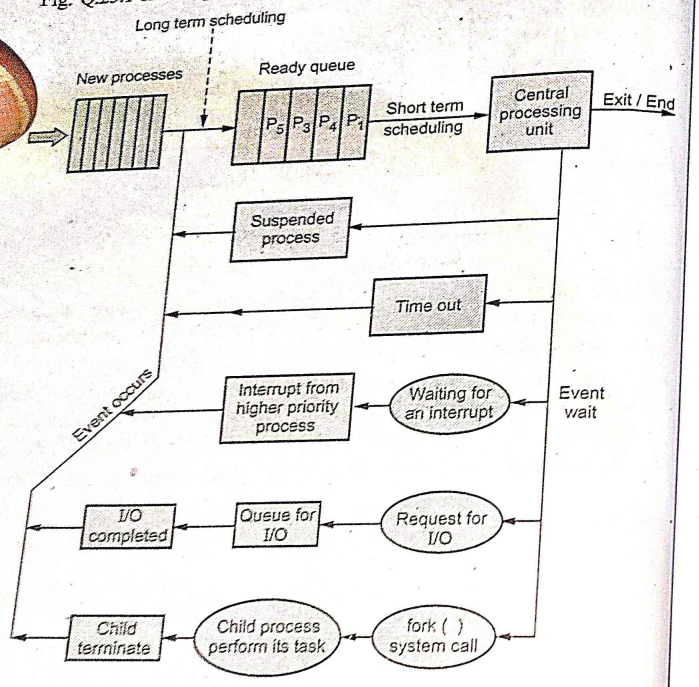


Fig. Q.25.1 Process scheduling queueing diagram

Long term scheduler

- *Long term scheduler is also called* **job scheduler**. *It determines which process are admitted to the system for processing. Processes are selected from the queue and loads into the main memory for execution.*
- *Long term scheduling controls the degree of multiprogramming in multitasking systems. It provides a balanced mix of jobs, such as I/O bound and CPU bound.*
- *Long term scheduling is used in real time operating system. Time sharing operating system has no long term scheduler.*

Medium term scheduler

- Medium term scheduler is part of swapping function. Sometimes it removes the process from memory. It also reduces the degree of multiprogramming.
- If process makes an I/O request and it is in memory then operating system takes this process into suspended states. Once the process becomes suspended, it cannot make any progress towards completion.
- In this situation, the process is removed from memory and makes free space for other process.
- The suspended process is stored in the secondary storage device i.e. hard disk. This process is called swapping.

Short term scheduler

- Short term scheduler is also called CPU scheduler. It selects the process from queue which are ready to execute and allocate the CPU for execution.
- Short term scheduler is faster that long term scheduler. This scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers.
- A scheduling decision will at a minimum have to be made after every time slice, and these are very short.
- It is also known as dispatcher.

Q.26 Discuss the difference among short term, medium term and long term scheduling.

Ans. :

Sr. No.	Long term	Short term	Medium term
1.	It is job scheduler.	It is CPU scheduler.	It is swapping.
2.	Speed is less than short term scheduler.	Speed is very fast.	Speed is in between both.
3.	It controls the degree of multiprogramming.	Less control over degree of multiprogramming.	Reduce the degree of multiprogramming.
4.	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system use medium term scheduler.

5.	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can be reintroduced into memory and its execution can be continued.
6.	Process state is (New to Ready).	Process state is (Ready to Running)	–
7.	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	–

Q.27 Define and explain context switching.

Ans. : • A context switch is the switching of the CPU from one process or thread to another. A context is the contents of a CPU's registers and program counter at any point in time.

- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**.
- A context switch can mean a register context switch, a task context switch, a thread context switch or a process context switch.
- A register is a small amount of very fast memory inside of a CPU that is used to speed the execution of computer programs by providing quick access to commonly used values.
- A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system.
- Context switching can be described in more detail as the Kernel performing the following activities with regard to processes (including threads) on the CPU :

1. Suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory.
2. Retrieving the context of the next process from memory and restoring it in the CPU's registers and
3. Returning to the location indicated by the program counter in order to resume the process.

- Context switches can occur only in Kernel mode (system mode). Kernel mode is a privileged mode of the CPU in which only the Kernel runs and which provides access to all memory locations and all other system resources.
- Other programs, including applications, initially operate in user mode, but they can run portions of the Kernel code via system calls. Software context switching can be used on all CPUs and can be used to save and reload only the state that needs to be changed.
- To use the hardware context switch you need to tell the CPU where to save the existing CPU state and where to load the new CPU state from. The CPU state is always stored in a special data structure called a TSS (Task State Segment).
- Context switch times are highly dependent on hardware support. Context switching represents a substantial cost to the system in terms of CPU time and it can be the most costly operation on an operating system.
- There are three situations where a context switch needs to occur. They are multitasking, interrupt handling, user and Kernel mode switching.

4.10 : Scheduling Algorithms : FCFS, SJF, RR and Priority

Q.28 Explain preemptive and non-preemptive scheduling.

Ans. : • The scheduling policy determines when it is time for process to be removed from the CPU and which ready process should be allocated the CPU next. The scheduling mechanism is composed of several different parts, depending on exactly how it is implemented in any particular operating system.

- CPU scheduling is divided into two types : Preemptive scheduling and non-preemptive scheduling.
- **Preemptive scheduling :** A scheduling method that interrupts the processing of a process and transfers the CPU to another process is called a preemptive CPU scheduling. The process switches from running state to the ready state and waiting state to the ready state.

- **Non-preemptive scheduling** : Non-preemptive operation usually proceeds towards completion uninterrupted. Once the system has assigned a processor to a process, the system cannot remove that processor from the process. The process switches from running state to the waiting state and termination of process.
- Preemptive scheduling increases the cost and it has higher overheads. This scheduling method is useful only in the high priority processes require rapid response.
- Non-preemptive scheduling is simple to implement. It requires some hardware platform. This method is attractive because of its simplicity. Windows 3.1 and Apple Macintosh operating system uses this scheduling method.

Q.29 Explain difference between preemptive and Non-preemptive scheduling.

Ans. :

Sr. No.	Preemptive scheduling	Non-preemptive scheduling
1.	Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.	Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
2.	Preemptive scheduling incurs a cost associated with access shared data.	Non-preemptive scheduling does not increase the cost.
3.	It also affects the design of the operating system Kernel.	It does not affects the design of OS Kernel.
4.	Preemptive scheduling is more complex.	Simple, but very inefficient.
5.	Example : Round robin method.	Example : First come first serve method.

Q.30 Explain CPU scheduling criteria.

Ans. : • Depending on the system, CPU scheduling criteria will change.

1. **Throughput** : CPU scheduling should attempt to service the maximum number of processes per unit time. The higher is the number, the more work is done by the system.
2. **Waiting time** : The average period of time a process spends waiting. Process is normally in the ready queue in waiting time.
3. **Turnaround time** : Turnaround time start from process submission to completion of process.

   Turnaround time = Burst time + Waiting time
4. **Response time** : It is the time from the submission of a request until the first response is produced.
5. **CPU utilization** : It is average function of time during which the processor is busy.
6. **Fairness** : Avoid the process from the starvation. All the processes must be given equal opportunity to execute.
7. **Priority** : If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher priority processes.
8. **Predictability** : A given process always should run in about the same amount of time under similar system loads.

- Depending upon the nature of operations the scheduling policy may differ. The CPU utilization and throughput are the system centered parameters. Fairness is affect by user and system.

Q.31 What is CPU scheduling ? Explain 2 different scheduling algorithms with examples.

Ans. : CPU scheduling : • In a multiprogramming environment, usually more programs to be executed than could possibly be run at one time. In CPU scheduling, it switches from one process to another process. CPU resource management is commonly known as scheduling.

- Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundamental operating system functions.
- User program contains combination of CPU burst cycle and I/O burst cycles. Process starts with CPU burst cycle then I/O burst cycle again

CPU burst cycle again I/O burst cycle. In this way the process completes its executions. Process will terminates after final CPU burst cycle completions.

- Scheduling mechanism is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Scheduler is responsible for multiplexing processes on the CPU.

1. First In First Out (FIFO) scheduling : • FIFO is also called first come first served scheduling method. The processes are dispatched according to their arrival time at the ready queue. FIFO is non-preemptive CPU scheduling algorithm.

- FIFO is simple to implement because it uses a FIFO queue. This algorithm is fine for most of the batch operating systems. FIFO is not useful in scheduling interactive processes because it cannot guarantee short response time.
- When new process enters into the system, its process control block is linked to the end of the ready queue and it is removed from the front of the queue.
- Turnaround time is unpredictable with the FIFO algorithm. Average waiting time of the FCFS is often quite long. Real life analogy is buying tickets.
- Convey effect : To reduce I/O device utilization, all I/O bound processes will be waiting excessively long for processor bound ones. This is called as convey effect.
- If one process monopolizes the system, the extent of its overall effect of system performance depends on the scheduling policy and whether the process is processor bound or I/O bound.

2. Priority scheduling : • Priority scheduling is preemptive or non-preemptive. Priority of the process can be defined either internally or externally. Internally defined priority considers the time limits, number of open files, use of memory and use of I/O devices.

- External priorities are set by using external parameter of the process, like importance of a process, cost of process etc.
- In this scheduling algorithm, CPU select higher priority process first. If the priority of two process is same then FCFS scheduling algorithm is applied for solving the problem.

- A non-preemptive priority algorithm will simply put the new process at the head of the ready queue. In non-preemptive scheduling algorithm, currently executing process will not change state.
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

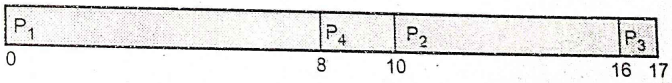
Drawback of priority scheduling :

- Waiting time is more for lower priority process even if there required CPU burst time is less. Priority scheduling algorithm faces the starvation problem.
- Starvation problem can be solved by using **aging techniques.**
- In aging, the priority of the process will increase which is waiting for long time in the ready queue.

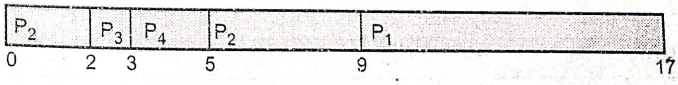
Q.32 Draw Gantt chart and calculate average turnaround time, avarage waiting time for the following processes using priority based (no-preemptive) scheduling and SJF (preemptive) scehduling policies. (Consider low no. as high priority) ☞ [SPPU : May-19, End Sem, Marks 6]

Processes	*Arrival time*	*Burst time*	Priority
P_1	0	8	1
P_2	0	6	2
P_3	2	1	3
P_4	3	2	0

Ans. : Priority based (Non-preemptive)



SJF preemptive :



Process	Waiting time		Turnaround time	
	Priority non - preemptive	SJF preemptive	Priority non - preemptive	SJF preemptive
P_1	0	9	8	17
P_2	10	3	16	9
P_3	14	0	15	1
P_4	5	0	7	2

Average waiting time priority non - preemptive = (0 + 10 + 14 + 5) / 4
= 7.25

Average waiting time SJF preemptive = (9 + 3 + 0 + 0) / 4 = 3

Average turnaround time priority non - preemptive = (8 + 16 + 15 + 7) / 4
= 11.5

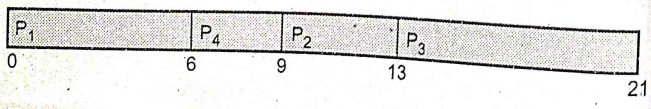
Average turnaround time SJF preemptive = (17 + 9 + 1 + 2) / 4 = 7.25

Q.33 Draw Gantt chart and calculate average turnaround time, average waiting time for the following processes using SJF non preemptive and round robin with time quantum 2.

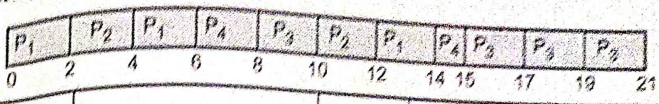
☞ [SPPU : Dec.-18, End Sem, Marks 6]

Process	*Arrival time*	*Burst time*
P_1	0	6
P_2	1	4
P_3	4	8
P_4	3	3

Ans. : SJF Non - premeptive : Gantt chart :



RR (Time quantum = 2)



Process	Waiting time		Process	Turnaround time	
	SJF Non - preemptive	RR		SJF Non - preemptive	RR
P_1	0	8	P_1	6	14
P_2	9 – 1 = 8	7	P_2	12	11
P_3	13 – 4 = 9	9	P_3	17	17
P_4	6 – 3 = 3	9	P_4	6	12

Average waiting time for SJF non - preemptive = (0 + 8 + 9 + 3) / 4
= 20 / 4 = 5

Average waiting time for RR = (8 + 7 + 9 + 9) / 4 = 33 / 4 = 8.25

Average turnaround time for SJF non - preemptive = (6 + 12 + 17 + 6) / 4
= 41 / 4 = 10.25

Average turnaround time for RR = (14 + 11 + 17 + 12) / 4
= 54 / 4 = 13.5

Q.34 What is pre-emptive scheduling ? Consider following example and supply round robin scheduling policy and calculate average Turnaround (TT) Time. (consider time slice = 1).

☞ [SPPU : Dec.-19, End Sem, Marks 6]

Process ID	*Arrival time*	*Serve time*
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Ans. : Gantt chart for Round Robin (Time slice = 1)

A	A	B	A	B	C	B	C	D	B	C	E	D	B	C	E	D	B	D	D

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 2

Process ID	Waiting time	Turnaround time
A	1	4
B	10	16
C	7	11
D	9	14
E	6	8

$$\text{Average turnaround time} = \frac{4+16+11+14+8}{5} = 10.6$$

END...

Unit V

5 Synchronization and Concurrency Control

5.1 : Concurrency

Q.1 Explain the principle of concurrency.

Ans. : • Concurrent access to shared data may result in data inconsistency. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

- Concurrency arises in the same way at different levels of execution streams. Following are the example of concurrency in different types of operating systems :

  1. **Concurrenrrency in multiprogramming :** An interaction between multiple processes running on one CPU.
  2. **Concucy in multithreading :** An interaction between multiple threads running in one process.
  3. **Concurrency in multiprocessors :** An interaction between multiple CPUs running multiple processes or threads.
  4. **Concurrency in multi-computers :** An interaction between multiple computers running distributed processes or threads.

- Java is a concurrent programming language.
- Process synchronization is required in uni-processor system, multiprocessor system and network. If more than one thread exists in a system at the same time, then the threads are said to be concurrent. Synchronization problems can occur whenever two or more concurrent processes use any shared resource.
- Cooperating process share the logical address space.

Is concurrency is possible without parallelism ?

- **Yes, concurrency is possible without parallelism.**

- Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

Q.2 Explain the following terms : i) Critical section ii) Race condition

Ans. : i) Critical Section :

- A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process may be in its critical section. The critical section problem is to ensure that **only one process at a time is allowed** to be operating in its critical section.

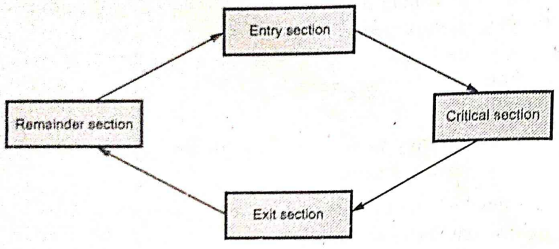


Fig. Q.2.1 Critical section

- Critical section means, process may change some common variable, writing files, updating memory location, updating a process table etc. When process is accessing shared modifiable data, it is said to be in a critical section.
- Each process takes permission from operating system to enter into the critical section. Structure of process is as follows :
  1. Entry section
  2. Remainder section
  3. Exit section.
- **Entry section :** It is block of code executed in preparation for entering critical section.
- **Exit section :** The code executed upon leaving the critical section.
- **Remainder section :** Rest of the code is remainder section.

- Each process cycles through remainder, entry, critical, exit sections in this order.

ii) Race Condition :

- **Race condition** occurs when two or more operations occur in an undefined manner. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called **race condition.**
- There is a "race condition" if the outcome depends on the order of the execution. The outcome depends on the CPU scheduling or "interleaving" of the threads.
- **Race condition** should be avoided because they can cause fine errors in applications and are difficult to debug.
- In banking system **balance** is a shared variable. After depositing the amount, it update by bank employee
  Balance = Balance + Amount.
  At the same time, some amount is transfer then it becomes
  Balance = Balance – Amount.
  These two tasks are in a race to write variable **balance.** In this the process that updates last determines the final value of balance.
- The concurrent execution of two processes is not guaranteed to be determinate, since different executions of the same program on the same data may not produce the same result.

Q.3 Explain the following terms :
i) Mutual exclusion ii) Synchronization

Ans. : i) Mutual exclusion : • Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called **critical sections.**

- Requirement of mutual exclusion is that, when process P1 is accessing a shared resource R1 then on other process should be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers and shared data structures.

ii) Synchronization : Synchronization means sharing system resources by processes in such way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.

Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes. Process synchronization was introduced to handle problems that arose while multiple process executions.

Q.4 List the requirements of mutual exclusion.

☞ [SPPU : May-16, Dec.-17, End Sem, Marks 5]

Ans. : Requirements of mutual exclusion

1. At any time, only one process is allowed to enter in its critical section.
2. Solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about relative speeds of asynchronous concurrent processes.
5. A process cannot prevent any other process for entering into critical section.
6. A process must not be indefinitely postponed from entering its critical section.

5.2 : Mutual Exclusion : Hardware and Software Approach

Q.5 How livelock situation is avaoided by using Dekker's solution ?

Ans. : • Dekker's solution avoids a livelock situation.

- Dekker's algorithm was the first correct solution to the critical section problem for two processes/threads. It uses an array of Boolean values and an integer variable.
- Dekker's algorithm is purely software solution with no special purpose hardware instruction. It uses flag to indicate a process desire to enter its critical section
- If two processes try to enter their critical sections at the same time, variable "turn" indicates which of the processes should be allowed to enter.
- It allows two threads to share a single-use resource without conflict, using only shared memory for communication.
- Processes are loops indefinitely, repeating entering and re-entering its critical section. Here flag and turn are the two global variables.

- Variable flag indicates the position of the process with respect to mutual exclusion. The turn variable is used for resolving the conflicts.
- Algorithm is as follows :

```
int c1=1;
int c2=1;
int turn=1;
```

```
process P1
while (true) {
 nonCriticalSection1;
 c1=0;
 while (c2!=1)
   if (turn==2){
     c1=1;
     while (turn!=1) {}
     c1=0;
   }
 criticalSection1;
 c1=1; turn=2;
}
end P1;
```

```
process P2
while (true) {
 nonCriticalSection2;
 c2=0;
 while (c1!=1)
   if (turn==1){
     c2=1;
     while (turn!=2) {}
     c2=0;
   }
 criticalSection2;
 c2=1; turn=1;
}
end P2;
```

- Here, if P1 finds c2 = 0, it defers to P2 only if turn = 2; otherwise, it simply waits for c2 to become 1 before entering its critical section.
- Process P2, which is also trying to enter its critical section at the same time, is forced to defer to P1 only if turn = 1.
  - Dekker's algorithm is correct and it satisfies mutual exclusion and it is free from deadlock and starvation
  - **Limitation of Dekker's Algorithm :**

1. Dekker's algorithm does not provide strict alternation.
2. Dekker's algorithm will not work with many modern CPUs.
3. This algorithm won't work on Symmetric Multi-Processors (SMP) CPUs.
4. Additionally, Dekker's algorithm can fail regardless of platform due to many optimizing compilers. Compiler may remove writes to flag since never accessed in loop. Creating an infinite loop.

Q.6 Write short note on : Peterson's solution.

Ans. : • Peterson's solution is software based solution for critical section problem. This algorithm will not work correctly on the modern computer architecture. It is simpler algorithm for two process mutual exclusion with busy waiting.

- This algorithm is less complicated than Dekker's algorithm.
- Peterson's algorithm solves critical section problem of two processes only. It does not require any special hardware. Two processes alternates execution between their critical section and remainder sections.
- Deadlock and indefinite postponement are impossible in Peterson's algorithm as long as no process or thread terminates unexpectedly.
- For indefinite postponement to occur, one process would have to be able to continually complete and reenter its critical section while the other process busy waited.

Q.7 Explain interrupt disabling concept of mutual exclusion.

Ans. : • On uniprocessor system, parallel execution of process is not possible. But they can only be interleaved. To guarantee mutual exclusion, it is sufficient to prevent a process from being interrupted.

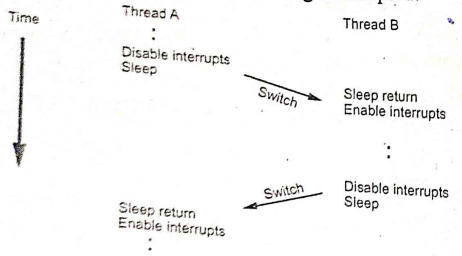


Fig. Q.7.1

- A process can prevent the operating system from executing by not issuing any supervisor call instructions and by disabling interrupts. Access to a critical section can be controlled by calling those primitives.
- Use interrupts,
  a) Implement preemptive CPU scheduling
  b) Internal events to relinquish the CPU
  c) External events to reschedule the CPU.
- Disable interrupts,
  a) Introduce uninterruptible code regions
  b) Think sequentially most of the time
  c) Delay handling of external events.

- Mutual exclusion is guaranteed because the critical section cannot be interrupted

Disadvantages

1. It works only in a single processor environment.
2. Interrupts can be lost if not serviced promptly.
3. A process waiting to enter its critical section could suffer from starvation.

Q.8 What are the advantages and disadvantages of test and set operations ?

Ans. : Advantages :

1. Simple to implement.
2. It works well for a small number of processes.
3. It can be used to support multiple critical sections.

Drawbacks :

1. If suffers from starvation.
2. There is possibility of busy waiting.
3. There may be deadlock.

5.3 : Semaphore

Q.9 Explain with definition, the concept of general and binary semaphore.

Ans. : Concept of general semaphore :

- Semaphore is described by Dijkstra. Semaphore is a nonnegative integer variable that is used as a flag. Semaphore is an operating system abstract data type. It takes only integer value. Its used to solve critical section problem.
- Dijkstra introduced two operations (P and V) to operate on semaphore to solve process synchronization problem. A process calls the P operation when it wants to enter its critical section and calls V operation when it wants to exit its critical section. The P operation is called as wait operation and V operation is called as signal operation.
- A wait operation on a semaphore decrease its value by one.

```
waits : while S < 0
        do loops;
```

- Monitor is implemented in programming languages like Pascal, java and C—. Java makes extensive use of monitors to implement mutual exclusion.
- Monitor is an abstract data type for which only one process may be executing a procedure at any given time. Monitor is a collection of procedure, variables and data structure. Data inside the monitor may be either global to all routines within the monitor or local to a specific routine.
- Monitor data is accessible only within the monitor. A shared data structure can be protected by placing it in a monitor. If the data in a monitor represents some resource, then the monitor provides a mutual exclusion facility for accessing the resource. A procedure defined within a monitor can access only those variable declared locally within the monitor and its formal parameters.
- When a process calls a monitor procedure, the first few instruction of the procedure will check to see if any other process is currently active within the monitor. If process is active then calling process will be suspended until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.
- Monitor supports synchronization by the use of condition variables that are contained within the monitor and accessible only within the monitor. Every conditional variable has an associated queue. Condition variables operates on two functions :

  cwait(condition variable)

  csignal (condition variable)

- **cwait** : It suspend execution of the calling process on condition.
- **csignal** : Resume execution of some process blocked after a cwait on the same condition.
- The cwait must come before the csignal.

Q.14 List the drawback of monitor.

Ans. : 1. Major weakness of monitors is the absence of concurrency if a monitor encapsulates the resource, since only one process can be active within a monitor at a time.

2. There is the possibility of deadlocks in the case of nested monitors calls.

3. Monitor concept is its lack of implementation most commonly used programming languages.
4. Monitors cannot easily be added if they are not natively supported by the language.

Q.15 Compare and contrast the use of monitors and semaphores operations.

Ans. : Difference between Monitors and Semaphore

- A semaphore is an operating system abstract data type whereas monitors are based on abstract data types.
- Semaphore-level synchronization primitives are difficult to use for complex synchronization situations. Monitors were derived to simplify the complexity of synchronization problems by abstracting away details.
- Semaphores provide a general-purpose mechanism for controlling access to critical sections. Their use does not guarantee that access will be mutually exclusive or deadlock will be avoided. A monitor is a programming language construct that guarantees appropriate access to critical sections.
- Monitor uses condition variables. A condition variable is like a semaphore, with two differences,
  1) A semaphore counts the number of excess up operations, but a signal operation on a condition variable has no effect unless some process is waiting. A wait on a condition variable always blocks the calling process.
  2) A wait on a condition variable automatically does an up on the monitor and blocks the caller.

5.5 : Reader Writer Problem

Q.16 Explain reader writer problem.

Ans. : • When two types of processes need to access a shared resource such as a file or database. They called these **processes reader and writers.**

- Multiple readers can concurrently read from the data base. But when updating the database, there can only be one writer (i.e., no other writers and no readers either)
- More than one reader may read shared data (no writers). When a writer uses shared data, all other writers and readers must be excluded.

- Fig. Q.16.1 shows reader-writer problem.

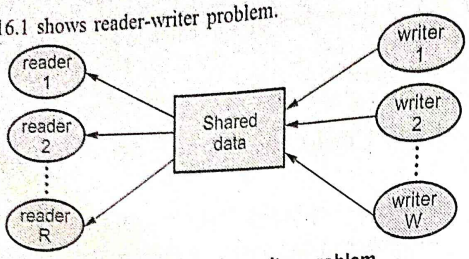


Fig. Q.16.1 Reader-writer problem

- For example in railway reservation system, readers are those who want train schedule information. They are called reader because readers do not change the content of the database. Many readers may access the database at once. There is no need to enforce mutual exclusion among readers.
- The writers are those who making reservations on a particular train. Writer can modify the database, so it must have exclusive access.
- When writer is active, no other readers or writers may be active. Therefore, it must enforce mutual exclusion if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set of processes. If a process want only to read the file, then it may share the file with any other process that also wants to read the file.
- We apply rules for access order :
  1. If a writer is waiting for readers to be finished, do not allow any more readers.
  2. If a reader is waiting for a writer to finish, give reader priority.
- If the writer wants to append the file, then no other process should have access to the file when the writer has access to it.
- If reader having higher priority than the writer, then there will be starvation with writers. For writer having higher priority than reader then starvation with readers.

Q.17 Write a semaphore solution for readers - writer's problem.

☞ [SPPU : May-18, Dec.-18, End Sem, Marks 5]

Ans. : • When two types of processes need to access a shared resource such as a file or database. They called these **processes reader** and **writers.**

- The writers are those who making reservations on a particular train. Writer can modify the database, so it must have exclusive access. When writer is active, no other readers or writers may be active. Therefore, it must enforce mutual exclusion if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set of processes. If a process want only to read the file, then it may share the file with any other process that also wants to read the file. If the writer wants to append the file, then no other process should have access to the file when the writer has access to it.

```
int readcount = 0;
semaphore wsem = 1;
semaphore x = 1;
void main( )
        {
            int p = fork( );
            if(p) reader;                    // assume multiple instances
            else writer;                     // assume multiple instances
        }
```

```
void reader( )
        {
            while(1)
                {
                    wait(x);
                    readcount ++;
            if (readcount = = 1)
                    wait (wsem);
                    signal(x);
                doReading( );
            wait(x);
                readcount--;
            it (readcount==0)
                signal(wsem);
        signal(x);
        }
    }
```

```
void writer( ) {
    while(1) {
        wait(wsem)
        doWriting( );
        signal(wsem)
    }
}
```

5.6 : Producer Consumer Problem

Q.18 Write a pseudo code for producer-consumer problem using semaphore. [SPPU : Aug-17, In Sem, Marks 6]

Ans. : • Solution to producer consumer problem by using binary semaphore :

```
void main ( )
{
    count = 0;
}
int i;
binarysemaphore s = 1;
int producer ( )
{
   While (true)
{
        produce_data_item ( );
       semwaitB(s);
   append ( );
   count = count + 1;
   if ( count == 1)
   semSignalB(delay);
   semSignalB(s);
   }
}
int consumer ( )
{
   int p;
semWaitB(delay);
while (true)
{
  semWaitB(s);
consume( );
count = count - 1;
p = count;
semSignalB(s);
consumedata( );
if ( p == 0)
semWaitB(delay);
}
}
```


Q.19 What is critical section ? Give semaphore solution for producer - consumer problem.

Ans. : • A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process may be in its critical section. The critical section problem is to ensure that only one process at a time is allowed to be operating in its critical section.

- Producer - consumer problem is example classic problems of synchronization. Producer process produce data item that consumer process consumes later.
- Buffer is used between producer and consumer. Buffer size may be fixed or variable. The producer portion of the application generates data and stores it in a buffer and the consumer reads data from the buffer.
- In order to synchronize these processes, both procedure and consumer are blocked on some condition. The producer is blocked when the buffer is full and the consumer is blocked when the buffer is empty.

1. Code for producer	2. Code for consumer process

producer (void)
{
int item;
while (TRUE)
{
produce item (& item);
produce_if (counter==N)
sleep ( );
enter_item (item);
counter = counter + 1;
if (counter = = 1)
wakeup (consumer);
}
}
	consumer (void)
	{
	int item;
	while (TRUE)
	{
	if (count = = 0)
	sleep ( );
	remove_item (& item);
	counter = counter – 1;
	if (count = = N – 1)
	wakeup (producer);
	consume_item (item);
	}
	}

5.7 : Dining Philosopher Problem

Q.20 Write semaphore solution for dinning Philosophers problem. ☞ [SPPU : Dec.-17,19, End Sem, Marks 5]

Ans. : • There is one semaphore for each chopstick. In addition, a local two-phase prioritization scheme is used, under which philosophers defer to their neighbors who have declared themselves "hungry." All arithmetic is modulo 5.

```
system DINING_PHILOSOPHERS
VAR
me:   semaphore, initially 1;        /* for mutual exclusion */
s[5]: semaphore s[5], initially 0;   /* for synchronization */
pflag[5]: {THINK, HUNGRY, EAT}, initially THINK;
                                     /* philosopher flag */

// As before, each philosopher is an endless cycle of thinking and
eating.
procedure philosopher(i)
 {
  while TRUE do
   {
    THINKING;
    take_chopsticks(i);
    EATING;
    drop_chopsticks(i);
   }
}
// The take_chopsticks procedure involves checking the status of
neighboring philosophers and then declaring one's own intention to
eat. This is a two-phase protocol; first declaring the status HUNGRY,
then going on to EAT.
procedure take_chopsticks(i)
 {
  DOWN(me);          /* critical section */
  pflag[i] := HUNGRY;
```



```
  test[i];
  UP(me);            /* end critical section */
  DOWN(s[i])         /* Eat if enabled */
 }
void test(i)         /* Let phil[i] eat, if waiting */
 {
  if ( pflag[i] == HUNGRY
   && pflag[i-1] != EAT
   && pflag[i+1] != EAT)
    then
     {
      pflag[i] := EAT;
      UP(s[i])
     }
 }
// Once a philosopher finishes eating, all that remains is to relinquish
the resources, its two chopsticks and thereby release waiting
neighbors.
void drop_chopsticks(int i)
 {
  DOWN(me);          /* critical section */
  test(i-1);         /* Let phil. on left eat if possible */
  test(i+1);         /* Let phil. on rght eat if possible */
  UP(me);            /* up critical section */
 }
```

Q.21 Explain classical problem of synchronization in terms of Dinning Philosopher problem. ☞ [SPPU : May-17, End Sem, Marks 5]

Ans. : • Dinning philosophers problem is one of classical process synchronization problem. Here five philosophers are seated around a circular table. They spend their lives in thinking and eating. Five plates with five forks are kept on the circular table.

- While philosophers think, they ignore the eating and do not require fork. When a philosopher decides to eat, then he/she must obtain a two fork, one from left side fork and another from right side fork. Philosophers can pick up only a single fork at one time.

- After consuming a food, the philosophers replace the fork and resume thinking. A philosophers to the left or right of a dinning philosopher cannot eat while the dinning philosophers is eating, since forks are a shared resource.
- Fig. Q.21.1 shows seating arrangement of five philosophers.

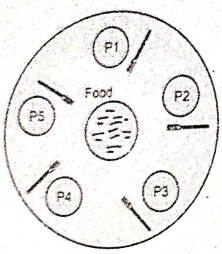


Fig. Q.21.1 Dinning philosophers

- Consider the following code :

```
void dinning_Philosopher ( )
  {
        while (true)
  {
  thinking ( );
  eating ( );
        }

  }
void eating ( )
{
takeleftfork ( );
        takerightfork ( );
        eatfood(delay);
        putrighttfork ( );
putleftfork ( );
}
```


- Here philosophers operates asynchronously and concurrently, it is possible for each philosophers to enter into eating mode. The procedure takeleftfork ( ) waits until the specified fork is available and then seizes it. This is not possible because of the following reasons.

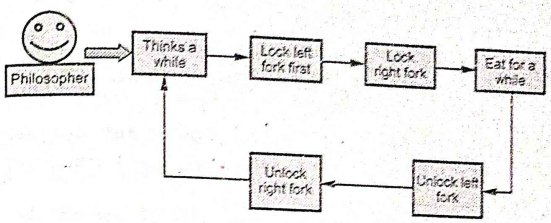


Fig. Q.21.2 Philosopher states

- Each philosopher will hold exactly one fork, and no forks will remain available on the table. The philosophers will all deadlock.
- To solve this problem, write some extra code in takerightfork ( ) procedure. User can specify the philosophers to put down the left fork if that philosophers cannot obtain the right fork.

Problem analysis :

- **Shared resource :** Here each fork is shared by two philosophers so we called it is a shared resource.
- **Race condition :** We do not want a philosopher to pick up a fork that has already been picked up by his neighbor. This is a race condition.
- **Solution :** We consider each fork as a shared item and protected by a mutex lock. Before eating, each philosopher first lock left fork and then right fork. If philosopher acquires both locks successful, then philosophers have two locks (two fork) so he can eat a food. After finishes easting, this philosopher releases both fork, and thinks.

Some other alternatives :

1. Pick up the left fork, if the right fork is not available for a given time, put the left fork down, wait and try again. Even if each philosopher waits a different random time, an unlucky philosopher may starve.
2. Require all philosophers to acquire a binary semaphore before picking up any forks. This guarantees that no philosopher starves but limits parallelism dramatically.

- Because we need to lock and unlock a chopstick, each chopstick is associated with a mutex lock. Since we have five philosophers who think and eat simultaneously, we need to create five threads, one for each philosopher. Since each philosopher must have access to the two mutex locks that are associated with its left and right chopsticks, these mutex locks are global variables.

5.8 : Deadlocks : Principle of Deadlock

Q.22 What is deadlock ? State and explain the conditions for deadlock.

☞ [SPPU : May-18, End Sem, Marks 8]

OR What are necessary conditions for deadlock ? Explain them with example.

☞ [SPPU : Dec.-18, End Sem, Marks 4]

Ans. : Deadlock can be defined as the **permanent blocking** of a set of processes that either complete for system resources. Deadlock can occur on sharable resources.

- A process is in deadlock state if it was waiting for a particular event that will not occur. In a system deadlock, one or more processes are deadlocked.
- Following four conditions are necessary for deadlock to exist.

1. Mutual exclusion 2. Hold and wait
3. No preemption 4. Circular wait

1. **Mutual exclusion :** A resource may be acquired exclusively by only one process at a time.
2. **Hold and wait :** Processes currently holding resources that were granted earlier can request new resources.
3. **No preemption :** Once a process has obtained a resource, the system cannot remove it from the process control until the process has finished using the resource.
4. **Circular wait :** A circular chain of hold and wait condition exists in the system.

- All four of these conditions must be present for a resource deadlock to occur. If one of them is absent, no resource deadlock is possible.

Q.23 Differentiate between the following : Deadlock and starvation.

Ans. : • **Deadlocks :** A group of processes are waiting for resources held by others in the group. None of them will ever make progress.

- **Starvation :** A process may wait indefinitely because other process keep coming in and getting the requested resources before this process does.
- In a deadlock situation, none of the involved processes can possibly make progress. In a starvation situation, a process is ready to execute, but it is not being allowed to execute.

Q.24 Write short notes on : Resource allocation graph and resource allocation graph algorithm.

Ans. : Resource allocation graph :

- Resource allocation graph is introduced by Holt. It is a directed graph that depicts a state of the system of resources and processes.
- Process and resource are represented by node in directed graph. Graph consists of a set of vertices (V) and set of edges (E).
- A process node is graphically represented by circle.
- A resource node is graphically represented by a rectangle and represents one type of resources.
- The number of bullet symbols in a resource node indicates how many units of that resource class exist in the system.

Fig. Q.24.1 Process

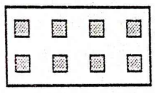


Fig. Q.24.2 Resource with 8 instances

- An arrow from the process to resource indicates the process is requesting the resource. An arrow from resource to process shows an instance of the resource has been allocated to the process.
- Claim edge P → R indicated that process P may request resource R in the future ; represented by a dashed line. Claim edge converts to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process. When a resource is released by a process, assignment edge reconverts to a claim edge.

(a) Request edge (b) Claim edge

Fig. Q.24.3 Edges

- Fig. Q.24.4 shows a resource allocation graph. System consists of process and resources.

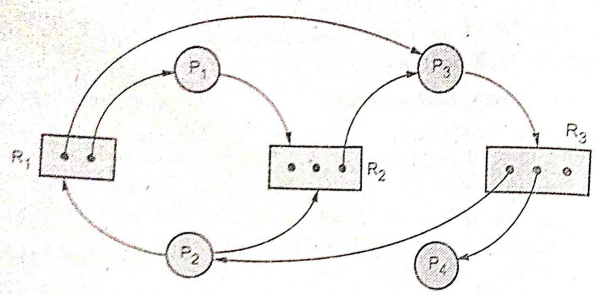


Fig. Q.24.4 Resource allocation graph

Process : P_1, P_2

Resource : R_1, R_2

Resource allocation graph algorithm : In resource-allocation graph, normally request edge and assignment edge is used.

In addition to the request and assignment edge, new edge called **claim edge** is added. For example, a calim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. This claim is shown by dotted lines in the figure. Fig. Q.24.5 shown this deadlock avoidance with resource allocation graph.

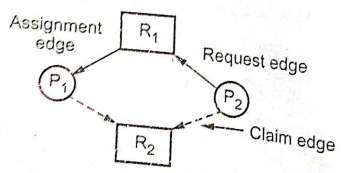


Fig. Q.24.5

- When process P_i request resources R_j, the claim edge is converted to a request edge.
- When a resource R_j is released by P_i, the assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge $P_i \rightarrow R_j$.
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.

5.9 : Deadlock Prevention

Q.25 Explain deadlock prevention techniques with example.

☞ [SPPU : May-16, End Sem, Marks 5]

Ans. : • To prevent a deadlock, the OS must eliminate one of the four necessary conditions.

1. Mutual exclusion 2. Hold and wait
3. No preemption 4. Circular wait

1. **Mutual exclusion :** It is necessary in any computer system because some resources (memory, CPU) must be exclusively allocated to one user at a time. No other process can use a resource while it is allocated to a process.
2. **Hold and wait :** If a process holding certain resources is denied a further request, it must release its original resources and if required request them again.
3. **No preemption :** It could be bypassed by allowing the operating system to deallocate resources from process.
4. **Circular wait :** Circular wait can be bypassed if the operating system prevents the formation of a circle.

- A deadlock is possible only if all four of these conditions simultaneously hold in the system.
- Prevention strategies ensure that at least one of the conditions is always false.

Q.26 How can the hold-and wait condition be prevented ?

Ans. : To prevent hold-and wait condition, processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

- There are several possibilities for this :
  1. Require that all processes request all resources at one time. This can be wasteful of system resources if a process needs one resource early in its execution and doesn't need some other resource until much later.
  2. Require that processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request. This can be a problem if a process has partially completed an operation using a resource and then fails to get it re-allocated after releasing it.
- Either of the methods described above can lead to starvation if a process requires one or more popular resources.

5.10 : Deadlock Avoidance

Q.27 Explain deadlock avoidance algorithm with suitable example.

☞ [SPPU : Dec.-19, End Sem, Marks 7]

OR Write Banker's algorithm and explain with suitable example.

☞ [SPPU : Dec.-18, End Sem, Marks 6]

OR Write and explain Bankers algorithm for avoidance of deadlock.

Ans. : • Banker's algorithm is the deadlock avoidance algorithm. Banker's is named because the algorithm is modeled after a banker who makes loads from a pool of capital and receives payments that are returned to that pool.

- Algorithm is check to see if granting the request leads to an unsafe state. If it does, the request is denied. If granting the request leads to a safe state, it is carried out.
- The Dijkstra proposed an algorithm to regulate resource allocation to avoid deadlocks. The banker's algorithm is the best known of the avoidance method.
- By using avoidance method, the system is always kept in a safe state.
- It is easy to check if a deadlock is created by granting a request, deadlock analysis method is used for this.
- Deadlock avoidance uses the worst-case analysis method to check for future deadlocks.

- **Safe state :** There is at least one sequence of resource allocations to processes that does not result in a deadlock.
- System is in a safe state only if there exist a safe sequence. A safe state is not a deadlocked state.
- Deadlocked state is an unsafe state. It does mean the system is in a deadlock.
- As long as the state is safe, the resource manager can be guaranteed to avoid a deadlock.
- Initially the system is in a safe state. When process requests a resource and that resource is available then the system must decide whether the resources can be allocated immediately or process must wait.
- If system remains in safe state after allocating resource then only OS allocates resources to process.
- Banker algorithm uses following data structures.

1. Allocation : Allocation is a table in which row represents process and column represents resources (R).

alloc [i, j] = Number of unit of resource R_j held by process P_i.

2. Max : Max be the maximum number of resources that process requires during its execution.

- **Need (claim) :** It is current claim of a process, where a process's claim is equal to its maximum need minus its current allocation.

$$\text{Need} = \text{Max} - \text{Allocation}$$

- **Available :** There will be number of resources still available for allocation. This is equivalent to the total number of resources minus the sum of the allocation to all processes in the system.

$$\text{Available} = \text{Number of resources} - \text{Sum of the allocation to all process}$$

$$= \text{Number of resources} - \sum_{i=1}^{n} \text{Allocation}(P_i)$$

- Each process cannot request more that the total number of resources in the system. Each process must also guarantee that once allocated a resource, the process will return that resource to the system within a finite time.

Weakness of Banker's algorithm

1. It requires that there be a fixed number of resources to allocate.
2. The algorithm requires that users state their maximum needs (requests) in advance.
3. Number of users must remain fixed.
4. The algorithm requires that the bankers grant all requests within a finite time.
5. Algorithm requires that process returns all resource within a finite time.

Q.28 Consider the following snapshot of a system.

Processes	Allocation				Max.				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	0	1	4	2	1	2	3	3	2	1
P_1	3	1	2	1	5	2	5	2				
P_2	2	1	0	3	2	3	1	6				
P_3	1	3	1	2	1	4	2	4				
P_4	1	4	3	2	3	6	6	5				

Answer the following question using Bankers algorithm.

i) Illustrate that the system is in safe state by demonstrating an order in which the processes may complete ?

ii) If a request from a process P_1 arrives for (1, 1, 0, 0), can the request be granted immediately ?

iii) If the request from P_4 arrives for (0, 0, 2, 0), can the request be granted immediately ?

Ans. : Need Matrix :

	A	B	C	D
P_0	2	2	1	1
P_1	2	1	3	1
P_2	0	2	1	3
P_3	0	1	1	2
P_4	2	2	3	3

i) Safe State :

Process	Need		Available	T/F	Available
P_0	(2, 2, 1, 1)	<	(3, 3, 2, 1)	T	(2, 0, 0, 1) + (3, 3, 2, 1) = (5, 3, 2, 2)
P_1	(2, 1, 3, 1)	>	(5, 3, 2, 2)	F	(5, 3, 2, 2)
P_2	(0, 2, 1, 3)	>	(5, 3, 2, 2)	F	(5, 3, 2, 2)
P_3	(0, 1, 1, 2)	<	(5, 3, 2, 2)	T	(1, 3, 1, 2) + (5, 3, 2, 2) = (6, 6, 3, 4)
P_4	(2, 2, 3, 3)	<	(6, 6, 3, 4)	T	(1, 4, 3, 2) + (6, 6, 3, 4) = (7, 10, 6, 6)

Second iteration :

P_1	(2, 1, 3, 1)	<	(7, 10, 6, 6)	T	(3, 1, 2, 1) + (7, 10, 6, 6) = (10, 11, 8, 7)
P_2	(0, 2, 1, 3)	>	(10, 11, 8, 7)	T	(2, 1, 0, 3) + (10, 11, 8, 7) = (12, 12, 8, 10)

Safe sequence : P_0, P_3, P_4, P_1, P_2

ii) Request from process P_1 (1, 1, 0, 0)

We assume that, requested is granted immediately. So calculate need matrix :

	A	B	C	D
P_0	2	2	1	1
P_1	2 – 1 = 1	1 – 1 = 0	3 – 0 = 3	1 – 0 = 1
P_2	0	2	1	3
P_3	0	1	1	2
P_4	2	2	3	3

Available = (2, 2, 2, 1)

Calculate safe sequence :

Process	Need		Available	T/F	Available
P_0	(2, 2, 1, 1)	=	(2, 2, 2, 1)		unsafe state
P_1	(1, 0, 3, 1)	>	(2, 2, 2, 1)	F	(2, 2, 2, 1)
P_2	(0, 2, 1, 3)	>	(2, 2, 2, 1)	F	(2, 2, 2, 1)
P_3	(0, 1, 1, 2)	<	(2, 2, 2, 1)	F	(2, 2, 2, 1)
P_4	(2, 2, 3, 3)	<	(2, 2, 2, 1)	F	(2, 2, 2, 1)

There is no safe sequence, so request is not granted immediately.

iii) Request from process P_4 (0, 0, 2, 0)

We assume that, request is granted immediately. So calculate need matrix :

	A	B	C	D
P_0	2	2	1	1
P_1	2	1	3	1
P_2	0	2	1	3
P_3	0	1	1	2
P_4	2 – 0 = 2	2 – 0 = 2	3 – 2 = 1	3 – 0 = 3

Available = (3, 3, 0, 1)

There is no safe sequence, so request is not granted immediately.

Q.29 Consider following status of system, compute need of each process and check given state is safe or unsafe using Bankers algorithm. Also check, if P3 generates new request (0, 1, 0, 0) system is safe or unsafe and request is granted or not ?

[SPPU : Dec.-18, End Sem, Marks 8]

	MAX_CLAIM					ALLOCATED			
	R1	R2	R3	R4		R1	R2	R3	R4
P1	0	0	1	2		0	0	1	2
P2	2	7	5	0		2	0	0	0
P3	6	6	5	6		0	0	3	4
P4	4	3	5	6		2	3	5	4
P5	0	6	5	2		0	3	3	2

Currently Available Resources →

R1	R2	R3	R4
2	1	0	0

Ans. : Content of need matrix :

Processes	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	0
P3	6	6	2	2
P4	2	0	1	1
P5	0	3	2	0

System is safe or not :

Available = (2 1 0 0)

Process	Need	Available	Condition	New Available
P1	(0 0 0 0)	(2 1 0 0)	Need < Available	(2 1 1 2)
P2	(0 7 5 0)	(2 1 1 2)	Need > Available	(2 1 1 2)
P3	(6 6 2 2)	(2 1 1 2)	Need > Available	(2 1 1 2)
P4	(2 0 1 1)	(2 1 1 2)	Need < Available	(4 4 6 6)
P5	(0 3 2 0)	(4 4 6 6)	Need < Available	(4 7 9 8)

Again process for process P2 and P3

P2	(0 7 5 0)	(4 7 9 8)	Need < Available	(6 7 9 8)
P3	(6 6 2 2)	(6 7 9 8)	Need < Available	(6 7 12 12)

Safe sequence = P1, P4, P5, P2, P3

- If P3 generates new request (0, 1, 0, 0)

We assume that, request is granted. Then new need matrix will be,

Processes	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	0
P3	6	5	2	2
P4	2	0	1	1
P5	0	3	2	0

Available = (2 0 0 0)

Again calculate safe sequence :

Process	Need	Available	Condition	New Available
P1	(0 0 0 0)	(2 0 0 0)	Need < Available	(2 0 1 2)
P2	(0 7 5 0)	(2 0 1 2)	Need > Available	(2 0 1 2)
P3	(6 5 2 2)	(2 0 1 2)	Need > Available	(2 0 1 2)
P4	(2 0 1 1)	(2 0 1 2)	Need < Available	(4 3 6 6)
P5	(0 3 2 0)	(4 3 6 6)	Need < Available	(4 6 9 8)

Again process for process P2 and P3

P2	(0 7 5 0)	(4 6 9 8)	Need > Available	(4 6 9 8)
P3	(6 6 2 2)	(6 7 9 8)	Need > Available	(4 6 9 8)

So there is no **Safe sequence**, request is **not granted.**

5.11 : Deadlock Detection

Q.30 What is deadlock detection?

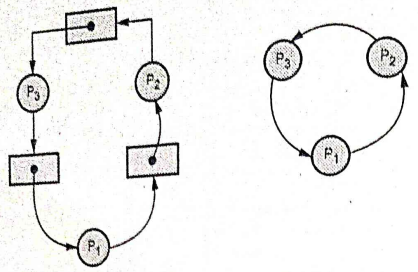
Ans. : • Deadlock detection is the process of determining that a deadlock exists and identifying the processes and resources involved in the deadlock.

- If processes are blocked on resources for an inordinately long time, the detection algorithm is executed to determine whether the current state is a deadlock.
- Deadlock detection algorithms can incur significant runtime overhead.

Q.31 What is wait for graph ?

Ans. : • Any resource allocation graph with a single copy of resources can be transferred to a **wait for graph.**

- Fig. Q.31.1 shows resource allocation graph with corresponding wait for graph. The state of the system can be modeled by directed graph, called a wait for graph.



(a) Resource allocation graph (b) Wait for graph

Fig. Q.31.1

- Wait for graph is a graph where each node represents a process. An edge $P_i \rightarrow P_j$ means that process P_i is blocked and waiting for process P_j to release a resource.
- The wait for graph of a system is always smaller than the resource allocation graph of that same system. There is a deadlock in a system if and only if there is a loop in the wait for graph of that system.
- Deadlock detection invloves two issues :
  1. Maintenance of the wait for graph.
  2. Searching of the wait for graph for the presence of cycles.
- Deadlock detection requires overhead for run time cost of maintaining necessary information and executing the detection algorithm.
- For multiple instances of a resource type use an algorithm similar to banker's algorithm.

5.12 : Deadlock Recovery

Q.32 What is deadlock ? Explain deadlock recovery methods ?

☞ [SPPU : May-19, End Sem, Marks 6]

Ans. : • Once deadlock has been detected in the system, the deadlock must be broken by removing one or more of the four necessary conditions.

- Here one or more processes will have to be preempted, thus releasing their resources so that the other deadlocked processes can become unblocked.

1. Process termination

- Deadlock is removed by aborting a process. But aborting process is not easy. All deadlocked processes are aborted.
- Circular wait is eliminated by aborting one by one process. There will be lot of overhead. Deadlock detection algorithm must rerun after each process kill.
- Selection of process for aborting is difficult. Following parameters are considered :

1. Priority of the process.
2. What percentage the process finished its execution ?
3. Resource used by process.
4. Need of resources to complete process remaining operation.
5. How many processes will need to be terminated ?
6. Process type : Batch or interactive.

2. Resource preemption

- Some times, resource temporarily take away from its current process and allocate it to another process.
- For selecting victim, following factors are considered,
  1. Priority of the process. Higher priority process are usally not selected.
  2. CPU time used by process. The process which is close to completion are usually not selected.
  3. The number of other process that would be affected if this process were selected as the victim.

3. Recovery through rollback

- When a process in a system terminates, the system performs a rollback by undoing every operation related to the terminated process.
- Checkpointing a process means that its state is written to a file so that it can be restarted later.

- Risk in this method is that the original deadlock may recover but the nondeterminancy of concurrent processing may ensure that this does not happen.

4. Starvation

- Starvation is one type situation in which a process waits for an event that might never occur in the system.
- Select the victim only for finite number of times. Use rollback method for selecting victim process.

END...

Unit VI

6 Memory Management

6.1 : Memory Management Concepts

Q.1 List the Memory Management Function.

Ans. : Functions are list below :

1. Allocate primary memory space to processes.
2. Minimize access time.
3. Determining allocation policy for memory.
4. Deallocation technique and policy.

Q.2 Explain concept of dynamic relocation.

Ans. : • Base register is sometimes called as a relocation register. The value of the relocation register is added to every address generated by a user process at the time it is sent to main memory.

- User can load a process with only absolute addresses for instructions and data, only when those specific addresses are free in main memory. Program's instruction, data and any other data structure required by the process can be accessed easily if the addresses are relative.
- Fig. Q.2.1 shows dynamic relocation.

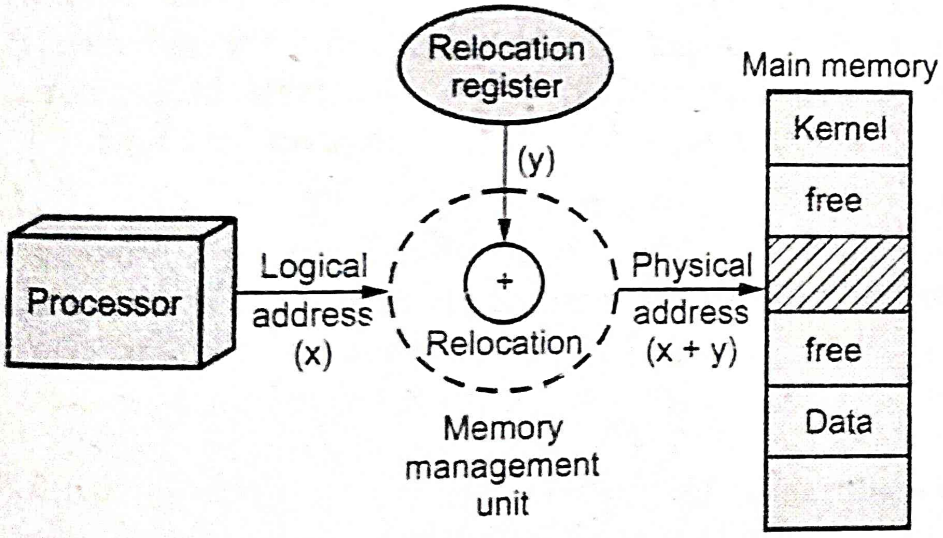


Fig. Q.2.1 Dynamic relocation

- User programs never reads the main memory physical address. Dynamic relocation requires extra hardware. It is mapping of the virtual address space to the physical address space at run time.
- Dynamic relocation makes it possible to move a partially executed process from one area of main memory into another without affecting other process.
- Problem with relocation is that, it is necessary to perform an addition and a comparison on every memory reference. For good memory management, logical address space is bound with a separate physical address space.

Q.3 How the user program is processed ? Also explain binding of data and instruction to memory.

Ans. : Address Space Mapping

- Secondary storage device stores program in binary executable format. Before executing, the program is loaded into the main memory.
- Most of the operating systems allow a user process to store in any section of the main memory. Source program uses symbolic addresses.
- Binding of instruction and data to main memory address is following ways : 1. Compile time 2. Load time 3. Execution time
- **Compile time :** Source program is translated at compile time to produce a relocatable object module. At compile time, the translator generates code to allocate storage for the variable. This storage address is used for code reference. Target address is unknown at compile time, it cannot be bound at compile time. Example of compile time binding is MS DOS.com programs.
- **Load time :** Compiler generates relocatable code if compile time binding is not performed. The loader modifies the addresses in the load module at load time to produce the executable image stored in main memory. Final binding is delayed until program load time.
- **Execution time :** Memory address of the program is changed at execution time, then execution time binding is used. Binding is delayed until the run time of the program. Normally all operating system uses execution time binding. Special hardware is used for execution time binding.

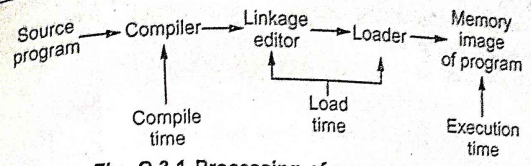


Fig. Q.3.1 Processing of user program

- Memory allocation and deallocation is done using run-time support of the programming language in which a program is coded. Allocation and deallocation requests are made by calling appropriate routines of the run time library.
- Kernel is not involved in this kind of memory management.

6.2 : Memory Partitioning

Q.4 Explain contiguous and non-contiguous memory allocation policies with suitable example. ☞ [SPPU : May-18, End Sem, Marks 8]

Ans. : 1. Contiguous memory allocation :

- In a fixed size partitioning of the main memory all partitions are of the same size. The operating system divides main memory into a number of fixed size partition. Each partition holds a single program.
- Fixed sized partitions are relatively simple to implement. Degree of multiprogramming depends on the number of partitions.
- Normally main memory is divided into two partitions :

  1. For resident program 2. For user processes

- Batch operating system uses the fixed size partition scheme. The operating system keeps the record of memory allocation and deallocation in the form of table. Initially all the memory is available for user processes.
- When the process arrives in the system and needs memory, operating system search for large hole for this process. Hole is one large block of free available memory. If any free hole is found, process is allocated to the free hole of memory as is needed.
- After allocating number of holes for the processes, a set of various size holes is scattered throughout memory at any given time. When a process arrives and searches for (memory) set of holes. But the holes must be large enough to accomodate the process.

- Fixed sized partitions are simple to implement. Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- Fig. Q.4.1 shows an example of fixed partitioning of a memory.
- There are two difficulties with the use of equal size fixed partitions :
  1. A program may be too big to fit into a partition.
  2. Memory utilization is extremely inefficient.

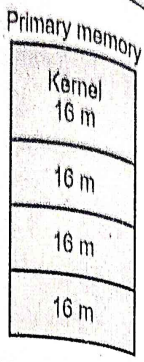


Fig. Q.4.1 Fixed size partition

- First difficulty is solved by using overlays. Overlays involves moving data and program segments in and out of memory.

2. Non-contiguous memory allocation :

- The use of unequal size partitions provides a degree of flexibility to fixed partitioning. In dynamic partitioning, the partitions are of variable length and number.
- In noncontiguous memory allocation, a program is divided into blocks that the system may place in nonadjacent slots in main memory.
- This allocation method do not suffer from internal fragmentation, because a process partition is exactly the size of the process.
- Fig. Q.4.2 shows noncontiguous memory allocation method. Operating system maintain the table which contains the memory areas allocated to process and free memory. Memory management unit use this information for allocating processes.

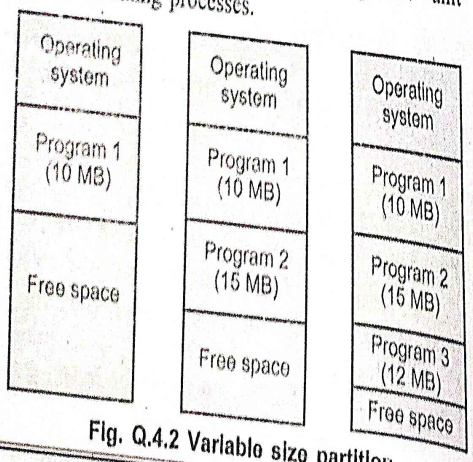


Fig. Q.4.2 Variable size partition

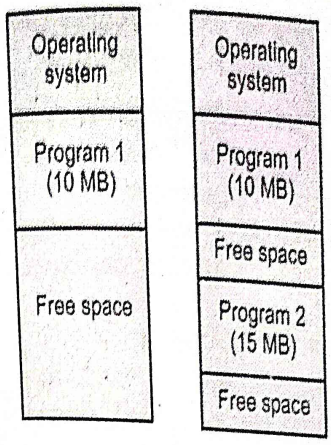


Fig. Q.4.3 Noncontiguous memory allocation

- Table contains information about memory starting address for process/program and their size.
- CPU sends the logical address of the process to the MMU and the MMU uses the memory allocation information stored in the table for calculating logical address. We called this address as effective memory address of the data/instruction.

Q.5 Differentiate internal and external fragmentation.

☞ [SPPU : May-18, End Sem, Marks 4]

Ans. :

Internal Fragmentation	External Fragmentation
Internal fragmentation is the area occupied by a process but cannot be used by the process.	External fragmentation exists when total free memory is enough for the new process but it's not contiguous and can't satisfy the request.
First fit and best fit memory allocation does not suffer from internal fragmentation.	First fit and best fit memory allocation suffers from external fragmentation.
In fixed partitioning, inefficient use of memory due to internal fragmentation.	In dynamic partitioning, inefficient use of processor due to need for compaction to counter external fragmentation.

Paging's suffer from internal fragmentation.	Paging does not suffer from external fragmentation
Segmentation does not suffer from internal fragmentation.	Segmentation suffer from external fragmentation

Q.6 Compare contiguous and non-contiguous memory allocation.

☞ [SPPU : Dec.-18, End Sem, Marks 3]

OR Compare fixed and variable sized partitioning.

☞ [SPPU : May-19, End Sem, Marks 3]

Ans. :

Sr. No.	Contiguous allocation	Noncontiguous allocation
1.	Program execution take place without overhead.	Address translation is overhead.
2.	Swapped-in processes are placed in the original area.	Swapped-in processes can be placed any where in memory.
3.	Suffer from internal fragmentation.	Only paging, suffers from internal fragmentation.
4.	Allocates single area of memory for process.	Allocates more than one block of memory for process.
5.	Wastage of memory.	No wastage of memory.

Q.7 What is fragmentation ? Explain types of fragmentation with suitable diagram/example.

☞ [SPPU : Dec.-18, End Sem, Marks 5, May-19, End Sem, Marks 3]

Ans. : • As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

- Fragmentation are of two types :

  1. Internal fragmentation. 2. External fragmentation.

- In fragmentation, OS cannot use certain area of available memory.

Internal fragmentation

- There is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition is called as internal fragmentation.
- Fig. Q.7.1 shows an internal fragmentation.

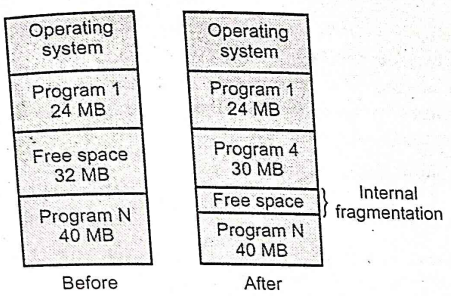


Fig. Q.7.1 Internal fragmentation

- To keep track of this free hole is overhead for system.

External fragmentation

- It occurs when enough total main memory space exists to satisfy a request, but it is not contiguous, storage is fragmentated into a large number of small holes.
- Fig. Q.7.2 shows external fragmentation.

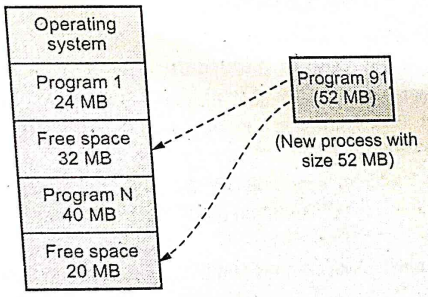


Fig. Q.7.2 External fragmentation

- Following are the solution for external fragmentation :
  1. Compaction
  2. Logical address space of a process/program

Q.8 How memory is protected by using relocation and limit register ?

Ans. : • Memory allocated to the process need to be protected against interference by other processes. Memory protection provides this facility.

- Relocation register and limit register are used for memory protection. Physical address is stored in the relocation register and range of virtual address is stored in the limit register.
- Fig. Q.8.1 shows logical address space.

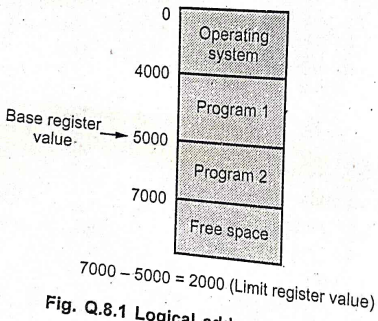


Fig. Q.8.1 Logical address space

- Fig. Q.8.2 shows memory protection by using relocation and limit registers. (See Fig. Q.8.2 on next page.)
- Use of registers method for memory protection requires proper arrangement. This method has to be used individually for each memory area allocated to a process.
- Every logical address generated by the CPU is checked against these registers. A protection violation interrupt is raised if any of these checks fail.
- Operating system load a base register and limit register. OS uses privileged instruction for this purpose. Operating system executes in monitor mode and privilege instruction also executes in privilege mode.

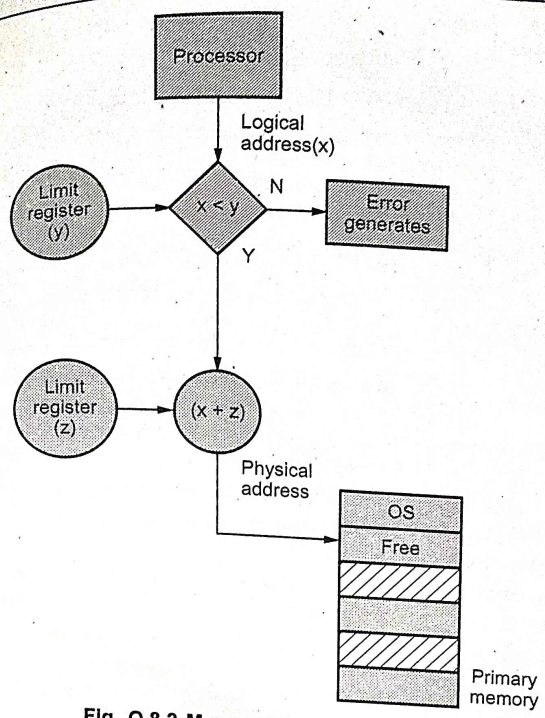


Fig. Q.8.2 Memory protection hardware

- Operating system prevents the user programs from changing the content of the registers. While executing in the monitor mode, operating system is given unrestricted access to both monitor and user's memory.
- In a multiprogramming environment, protection of main memory is essential. Paging or segmentation or both in combination provides an effective means of managing main memory.

Q.9 Explain the term in brief : Compaction.

☞ **[SPPU : May-18, End Sem, Marks 2]**

Ans. : • Compaction solves problem of external fragmentation. Fig. Q.9.1 shows compaction.

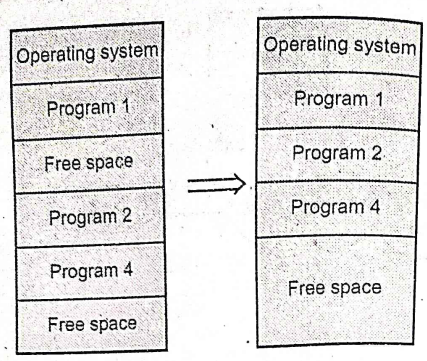


Fig. Q.9.1 Compaction

- Operating system moves all the free holes to one side of main memory and creates large block of free size.
- It must be performed on each new allocation of process to memory or completion of process for memory. System must also maintain relocation information.
- All free blocks are brought together as one large block of free space. Compaction requires dynamic relocation.
- Compaction has a cost and selection of an optimal compaction strategy is difficult. One method for compaction is swapping out those processes that are to be moved within the memory and swapping them into different memory locations.

6.3 : Buddy System

Q.10 Describe with example Buddy system. What are the advantages and disadvantages of Buddy system ?

Ans. : • The buddy system is a simple dynamic storage allocation method. The buddy system is a memory allocation and management algorithm that manages memory in power of two increments. Linux operating system manages memory using buddy algorithm.

- Fig Q.10.1 shows buddy system.

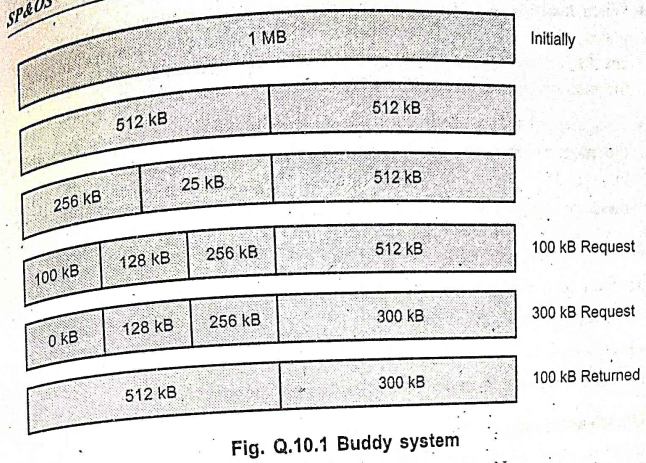


Fig. Q.10.1 Buddy system

- The buddy system assumes that memory is of size 2^N for some integer N. Both free and reserved blocks will always be of size 2^k for $k<N$. At any given time, there might be both free and reserved blocks of various sizes.
- The buddy system keeps a separate list for free blocks of each size. There can be at most N such lists, because there can only be N distinct block sizes.
- Single allocation block to be split, to form two blocks half the size of the parent block. These two blocks are known as 'buddies'.
- Initially memory consists of a single contiguous area of 128 pages. When user send request for memory, it is first rounded up to a power of 2.
- All the lists are initially empty, except for the 1 MB list, which has one hole listed. All allocations are rounded up to a power of two. Suppose the request for memory is 75 K then allocations rounded up to 128 K, 13 K allocations rounded up to 16 K, etc.
- If a block of that size is free, it is taken; otherwise, the smallest free block larger than the desired size is found and split in two halves. This splitting continues until the appropriate size is reached.

- When memory is deallocated, the buddy system groups contiguous free pages. When process free a block of memory, the memory manager checks the bitmap for checking block size. If adjacent block is also free, the memory manager combines the two blocks into a larger block.
- This method makes deallocation fast. If a block of size 2^k is free then the memory manager checks only the list of 2^k holes to merge them into a 2^{k+1} sized partition. Internal fragmentation is caused since memory requests are fitted in 2^k sized partitions.

Advantages :

a) Easy to implement a buddy system

b) Allocates block of correct size

c) It is easy to merge adjacent holes

d) Fast to allocate memory and de-allocating memory

Disadvantages :

a) It requires all allocation unit to be powers of two

b) It leads to internal fragmentation.

c) Merging of holes is recursive, so poor worst-case behavior.

6.4 : Paging

Q.11 What is paging ? Explain address translation in paging.

Ans. : • In paging, operating system divides each incoming programs into pages of equal size. The sections of a disk are called block or sectors. The sections of main memory are called page frames. One sector will hold one page of job instructions and fit into one page frame of memory.

- In paging, logical address space of a program can be noncontiguous. It solves external fragmentation problem.
- The relation between virtual addresses and physical memory addresses given by page table.

- Fixed sized blocks are called frames and breaking of logical memory into blocks of same size called pages.

- Memory manager prepares following things before executing a program :

1. Find out the number of pages in the program.
2. Free space in the main memory.
3. Loading of all the programs pages into memory.

- Pages are not loaded continuosly in main memory. Each page can be stored in any available page frame anywhere in main memory. Memory manager keeps the track of pages of program. Paging avoids external fragmentation and need for compaction.
- Memory manager uses three tables to keep track of process and memory.

1. **Job table :** It stores the size of the active job and memory location where its page table is stored.
2. **Page map table :** It contains vital information about each page. PMT contains a page number and corresponding page frame memory address. It includes only one entry per page. Page numbers are in sequential order.
3. **Memory map table :** MMT is used to store page frame location and its status.

- Page size depends upon underlying hardware. Operating system maintains a page table for each process. The page table shows the frame location for each page of a process. Fig. Q.11.1 shows the virtual address format for paging system.

Page number (P)	Displacement or offset (D)

Fig. Q.11.1 Virtual address format

- Processor hardware performs the logical to physical address translation. Logical address contains page number and offset. The physical address contains frame number and offset. Offset is a relative factor. It is used to locate that line within its page frame.

Q.12 What is difference between page and a frame ?

Ans. : • In paging, operating system divides each incoming programs into pages of equal size. The sections of a disk are called block or sectors. The sections of main memory are called page frames.

- Fixed sized blocks are called frames and breaking of logical memory into blocks of same size called pages.
- Pages are not loaded continuously in main memory. Each page can be stored in any available page frame anywhere in main memory.
- Page frames on main memory are required in paging. No frames are required in segmentation.

Q.13 Explain design issues in paging. Explain with example.

Ans. : Design issues for paging systems :

a) Local versus Global allocation policies b) Load control c) Page size d) Shared pages e) Cleaning policy

- Local versus Global allocation policies

1. Local replacement policy : Choose among resident set of process that generated page fault
2. Global replacement policy : Choose among all pages eligible for replacement in main memory
3. Types of allocations :
   a. Fixed allocation, Local scope. Allocations cannot be too large or too small as it would be irreparable.
   b. Variable allocation, Global scope
   c. Difficult to obtain a good replacement policy.
   d. Variable allocation, Local scope
   e. Re-evaluate allocation from time to time

- Load control : Trashing can be expected :

a. System will spend a lot of time copying data from disk to memory;
b. If the wrong pages are evicted from memory they will have to be loaded again very soon. This results in a lot of wasteful copying.

- Page size : Determining best size requires balancing several competing factors. Choose a page size too big : Possibility for wasted memory. Choose a page size too small : OS needs to maintain larger page table.
- Shared pages : Sometimes it is useful to share pages among processes. For example : User may be running several programs that use the same library; more efficient to share the pages, avoiding having multiples copies of the same page.

- Problem is that not all pages are sharable : Pages that are read-only, such as program text, can be shared; for data pages sharing is more complicated. When two or more processes share some code; special care needs to be employed with the shared pages.
- Cleaning policy : Paging works best when, there is an abundant supply of free page frames; such frames can be claimed as page faults occur. If every page frame is full and modified before a new page can be brought in, old page must first be written to disk.

Q.14 What elements are typically found in a page table entry ? Briefly define each element.

Ans. : • The region table entry contains a pointer to a table of physical page numbers called a page table.

- Page table entries may also contain machine dependent information such as permission bits to allow reading or writing of the page. The Kernel stores page tables in memory and accesses them like all other Kernel data structures.
- Fig. Q.14.1 shows Page table entry.

Page frame number	Age	Copy on write	Modify	Reference	Protect

Q.14.1

1. Page frame number is the real number in the main memory.
2. Age indicates total time period of frame without being referred in the memory.
3. Copy on write : Set when more than one process shares a page.
4. Modify field indicates page has been modified.
5. Reference : Page is referred or not.
6. Valid : Page is valid if it is in main memory.
7. Protect : Whether write operation is allowed or not.

Q.15 How TLB is used with paging?

Ans. : Paging with TLB : • Set of registers are used for implementing page table. Registers are suitable only for small page table. If page table size increases then special purpose hardware cache is used for page table.

- Special cache memory called a Translation Lookaside Buffer (TLB) is used with the address translation hardware. The full page table is kept in primary memory.
- TLB is very effective in improving the performance of page frame access. The cache buffer is implemented in a technology which is faster than the primary memory technology.
- Fig. Q.15.1 shows paging with TLB.

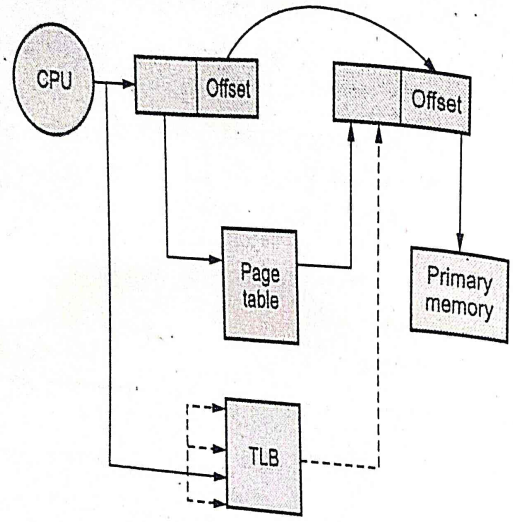


Fig. Q.15.1 Paging with TLB

- The TLB contents may be controlled by the operating system or by hardware, depending on the architecture.
- When a page is first translated to a page frame, the map is read from primary memory into the TLB. The TLB entry contains the page number, page frame's physical address etc.
- The page mapping mechanism first tries to find page number in the TLB. If the TLB contains the page number, then it is called as **page hit** or **TLB hit.**
- If the TLB does not contain an entry for page p, then it called as page miss or TLB miss. In case of TLB miss, the operating system locates the page table entry in the primary memory, which increases execution time.

- The TLB is associative, high-speed memory. A translation buffer is used to store a few of the translation table entries. It is very fast, but only for a small number of entries. On each memory reference.

1. First ask TLB if it knows about the page. If so, the reference proceeds fast.
2. If TLB has no information for page, must go through page and segment table to get information. Reference takes a long time, but gives the info for this page to TLB so it will know it for next reference.

- The greatest performance improvement is achieved when the associative memory is significantly faster than the normal page table lookup and the hit ratio is high. The hit ratio is the ratio between accesses that find a match in the associative memory and those that do not.
- **Disadvantage of TLB is that :** If two pages use the same entry of the memory, only one of them can be remembered at once. If process is referencing both pages at same time, TLB does not work very well.
- Each entry in the TLB must include the page number as well as the complete page table entry. The processor is equipped with hardware that allows it to simultaneously interrogate a number of TLB entries to determine if there is a match on page number. This technique is referred to as *associative mapping*.

Q.16 Explain working of hashed page table with diagram.

Ans. : • Hashed page table support address space of 32 bits and more. Hash value is used as virtual page number.

- Hash table contains a link list of elements. Each elements consists of following fields :
  1. Virtual page number
  2. Mapped page frame value.
  3. Pointer to the next element.
- Hash table improves search speed.
- Fig. Q.16.1 shows block diagram of hashed page table.

Working

1. Virtual page number is taken from virtual address.
2. Virtual page number is hashed into the hash table.
3. Virtual page number is compared with the first element of linked list.

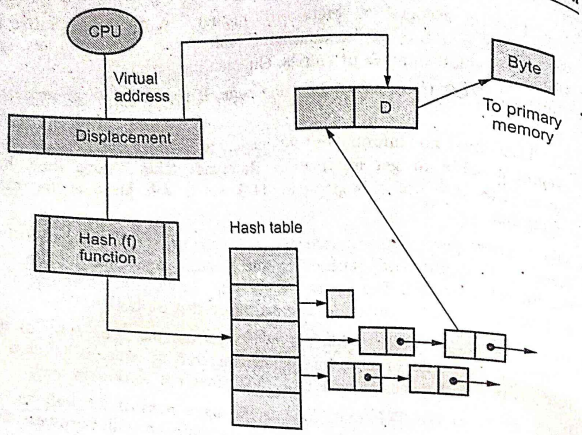


Fig. Q.16.1 Hashed page table

4. Both the value is matched, that value is (i.e. page frame) used for calculating physical address.
5. If the both value is not matched, the entire linked list is searched for a matching.

- Clustered page table is same as hashed page table but only difference is that each entry in the hash table refers to several pages.

6.5 : Segmentation

Q.17 Explain segmentation using suitable example in brief.

☞ [SPPU : Dec.-18, End Sem, Marks 3]

Ans. : Segmentation

- In segmentation, a program's data and instructions are divided into blocks called **segments.** A segment is a logical entity in a program.
- **Logical view :** A process consists of a set of segments.

- **Physical view :** It consists of non-adjacent areas of memory allocated to segments.
- All segment size may be equal or may not be equal. Segmentation support user view of memory. Fig. Q.17.1 shows programmer's view of a program.

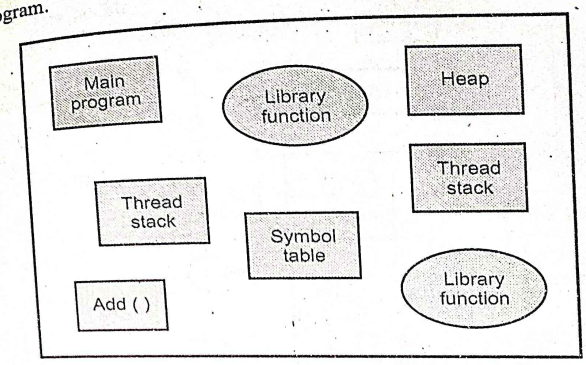


Fig. Q.17.1 Programmer's view

- Segmentation can be implemented with or without paging.
- Collection of segment is called as logical address space. Each segment is identified by its name.
- Processor generates logical addresses. These addresses consists of a segment number and an offset into the segment. Segment number is used as an index to segment page table.

Segment number	Offset

Fig. Q.17.2 Logical address

- Segment names are normally symbolic names.
- Operating system maintains a segment table for each process. It is usually stored in main memory as a segment that is not to be loaded as long as the process can run.
- Each entry in the segment table has a segment base and a segment limit. The base field contains the segment relocation register for the target segment. The segment limit field contains the length of the segment.

- Fig. Q.17.3 shows an address translation in segmentation.

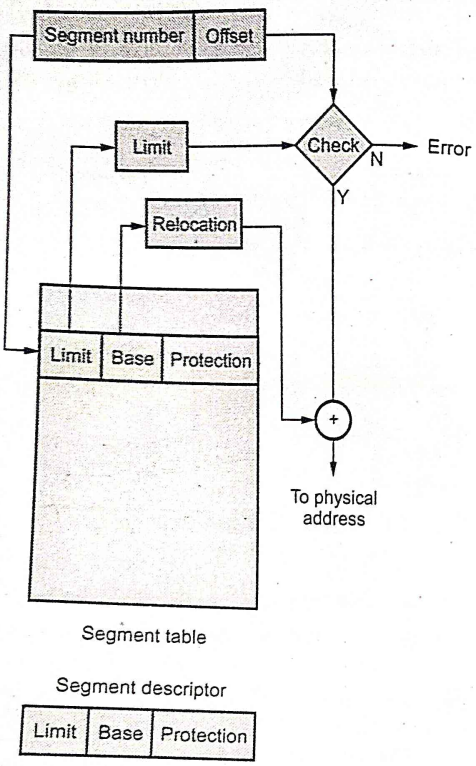


Fig. Q.17.3 Address translation in segmentation

- Segment table contains the physical address of the start of the segment. Then add the offset to the base and generate the physical address.
- If the required reference is not found in one of the segment registers, then error is generated. At the same time, operating system does lookup in segment table and loads new segment descriptor into the register.

Advantages

1. It provides virtual memory.
2. Allows dynamic segment growth.
3. Segmentation assists dynamic linking.
4. Segmentation is visible.

Q.18 Write difference between segmentation and paging.

Ans. : Difference between Segmentation and Paging

Sr. No.	Segmentation	Paging
1.	Program is divided into variable size segments.	Program is divided into fixed size pages.
2.	User (or compiler) is responsible for dividing the program into segments.	Division into pages is performed by the operating system.
3.	Segmentation is slower than paging.	Paging is faster than segmentation.
4.	Segmentation is visible to the user.	Paging is invisible to the user.
5.	Segmentation eliminates internal fragmentation.	Paging suffers from internal fragmentation.
6.	Segmentation suffers from external fragmentation.	There is no external fragmentation.
7.	Processor uses page number, offset to calculate absolute address.	Processor uses segment number, offset to calculate absolute address.
8.	OS maintain a list of free holes in main memory.	OS must maintain a free frame list.

Q.19 Write short note on : Segmentation with paging.

Ans. : • Most of the architecture support paging and segmentation. All the pages of segment need not be in main memory.

- It simplify the memory allocation and speed increases. It requires a high speed register to store the base address of the segment map table.
- For each segment, page table is created by OS. A pointer to the page table is kept in the segment's entry in the segment table.

- Segments are typically larger than pages.
- The base address get from the segment descriptor table is concatenated with the offset. This new address is referred to as a **linear address**. Linear address is generated by paging hardware.

Advantages

Combines all advantages of paging and segmentation.

Disadvantages

1. It increases hardware cost. 2. It increases processor overheads.
3. Dangers of thrashing.

6.6 : Placement Strategies

Q.20 Explain in brief the memory allocation algorithms with examples.

Ans. : Best fit : The allocator places a process in the smallest block of unallocated memory in which it will fit. For example, suppose a process requests 12 kB of memory and the memory manager currently has a list of unallocated blocks of 6 kB, 14 kB, 19 kB, 11 kB and 13 kB blocks. The best-fit strategy will allocate 12 kB of the 13 kB block to the process. Best fit chooses the block that is closest in the size to the request.

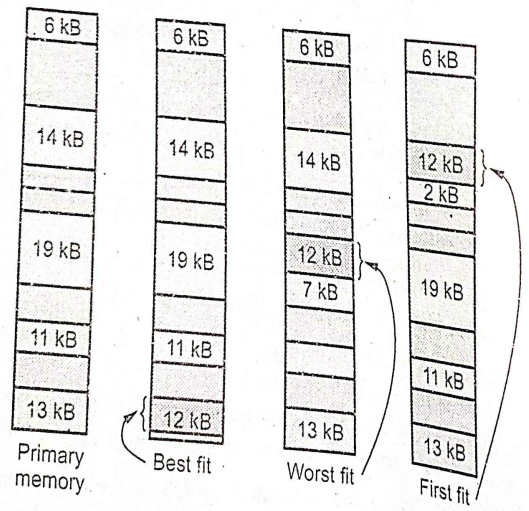


Fig. Q.20.1

Worst fit : The memory manager places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that compared to best fit ; another process can use the remaining space. Using the same example as above, worst fit will allocate 12 kB of the 19 kB block to the process, leaving a 7 kB block for future use.

First fit : There may be many holes in the memory, so the operating system, to reduce the amount of time it spends analyzing the available spaces, begins at the start of primary memory and allocates memory from the first hole it encounters large enough to satisfy the request. Using the same example as above, first fit will allocate 12 kB of the 14 kB block to the process.

Q.21 Given a memory partitions of 100 K, 500 K, 200 K, 300 K and 600 K (in order) how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?

☞ [SPPU : May-19, End Sem, Marks 6]

Ans. : First-fit :

212 K is put in 500 K partition

417 K is put in 600 K partition

112 K is put in 288 K partition (new partition 288 K = 500 K – 212 K)

426 K must wait.

Best-fit :

212 K is put in 300 K partition

417 K is put in 500 K partition

112 K is put in 200 K partition

426 K is put in 600 K partition

Worst-fit :

212 K is put in 600 K partition

417 K is put in 500 K partition

112 K is put in 388 K partition (600 K – 212 K)

426 K must wait

In this example, Best-fit turns out to be the best.

6.7 : Virtual Memory

Q.22 What is virtual memory ? Explain demand paging with example. [SPPU : Dec.-18, End Sem, Marks 6]

Ans. : • Virtual memory is a method of using hard disk space to provide extra memory. It simulates additional main memory.

- Virtual memory is stored in the secondary storage device. It helps to extend additional memory capacity and work with primary memory to load applications.
- The virtual memory will reduce the cost of expanding the capacity of physical memory. The implementations of virtual memory will different for operating systems to operating system.
- The logical addressable space is referred to as virtual memory. The virtual address space is much larger than the physical primary memory in a computer system. The virtual memory works with the help of secondary storage device and its speed is low compared to the physical storage location.
- Virtual memory uses two types of addresses : Virtual address and physical address.
- Virtual address is referred by process and physical address is actual address of the main memory.

Demand paging :

- In demand paging, a page is loaded into primary memory only when the process references it. It is the simplest fetch policy implemented in virtual memory. Fig. Q.22.1 shows demand paging.
- Demand paging guarantees that the system brings into primary memory only those pages that processes actually need. It involves interaction between hardware and software component of the virtual memory.

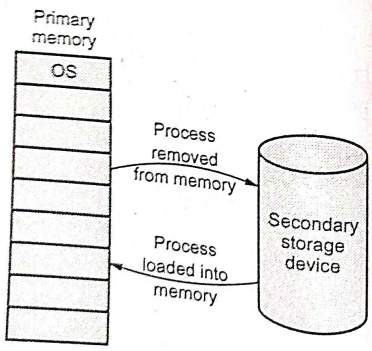


Fig. Q.22.1 Demand paging

- Paging with swapping method will become as demand paging concept. Process's logical address space is stored on the secondary storage device.
- Demand paging requires that the page map table for each process keep track of each page as it is loaded or removed from primary memory.
- When user want to execute a process, user swap a process into a memory. Lazy swapper is used for swapping the process into memory. Lazy swapper swaps the pages whenever needed

Q.23 What is structure of typical page table entry? What is significance of modified bit, reference bits, protection bits and present/absent bit in page table entry?

Ans. : • Fig. Q.23.1 shows page table entry.

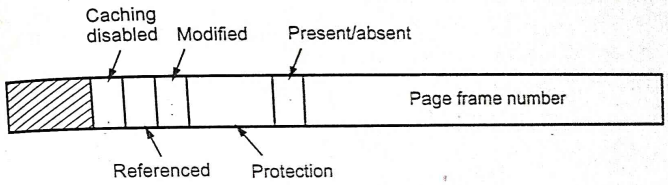


Fig. Q.23.1 Page table entry

- Page frame number : Most important field, the goal of the page mapping;
- Present /absent : If this bit is 1, the entry is valid and can be used;
- Protection bit is used for provide protection to the memory. Page table adds one more column for providing protection bit.
- Modified : Useable in reclaiming a page frame. If dirty, write back to disk.
- Referenced : The bit is set whenever a page is referenced. Help the OS choose a page to evict when a page fault occurs.
- Caching disabled : Useful for pages that map onto device register rather than memory

Q.24 Describe in detail the actions taken by the operating system when a page fault occurs.

Ans. : Steps in handling a page fault

- When page fault occurs, the system performs following steps :

1. Check the process control block table for reference which bit is set i.e. valid bit or invalid bit for memory access.
2. If invalid bit is set, user will terminate the process. If valid bit is set, then the page is loaded into memory.
3. Check for free frame for new page loading.
4. Start reading secondary storage device for desired page for allocated frame.
5. Required page is loaded into the memory after completion of read operation. PCB table is updated.
6. System restart the instruction that was interrupted by the error. Page is loaded in memory so process can use it.

- Fig. Q.24.1 shows page fault handling steps in demand paging.

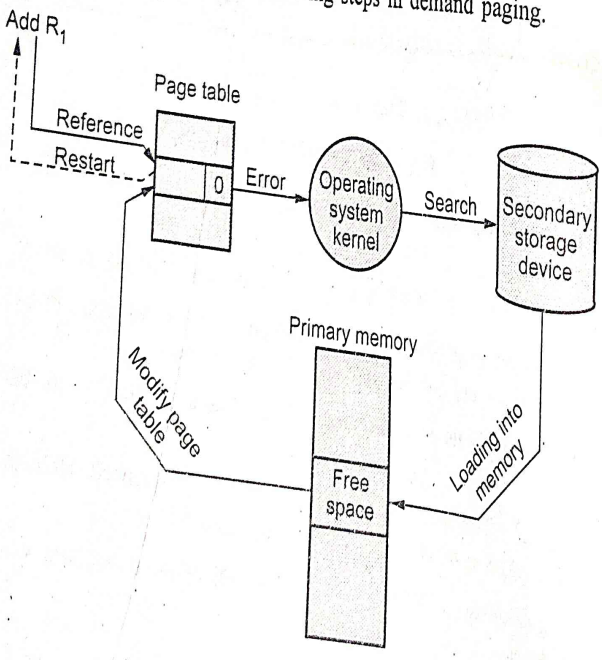


Fig. Q.24.1 Handling of page fault

6.8 : Swapping

Q.25 Write short note on swapping.

[SPPU : Dec-18,19, End Sem, Marks 5]

Ans. : Keeping all processes in memory all the time requires a large amount of main memory. For execution of the process, it must be swapped.

- **Swapping** is method of temporarily removing inactive process/programs from the physical memory.
- Inactive program means, which is neither executing on the CPU, nor performing an I/O operation.
- Fig. Q.25.1 shows swapping.

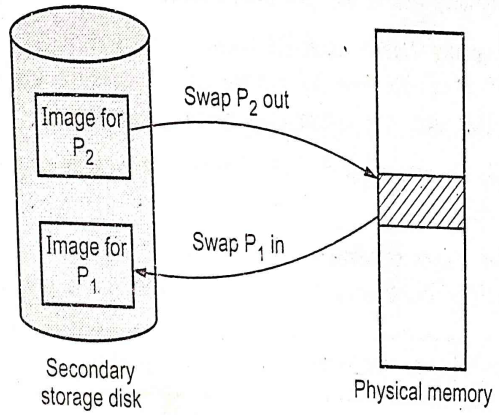


Fig. Q.25.1 Swapping

- When process P_2 send requests for an I/O operation, OS blocked P_2 process and will not return to the ready state for long period of time. Process P_2 is in blocked state. Process manager place the process P_2 in blocked state and inform to the memory manager regarding this action.
- Process manager moves a blocked process P_1 into the ready state and inform to memory manager. Process P_1 is loaded into main memory by memory manager when there is space available.
- Swapped out process is stored on the secondary storage disk. It contains executable image with code, data and stack.
- Relocation hardware supports for swapping. Because of address binding problem, swapping is difficult without relocation hardware.

- Round robin and priority based CPU scheduling algorithm uses the swapping concept.
- Time sharing system uses swapping. In time sharing, kernel can estimate when a program is likely to be scheduled next.
- A swapped out process will occupy same memory location when process swap in by OS. But this totally depends on address binding. If address binding is load time or assembly time, then swapped out process uses same memory location when swap-in. But execution-time binding uses different memory location.
- Secondary storage is required to implement the swapping technique.
- Content switching time increases when swapping increases. Major part of the swap time is transfer time. Swapping increases the operating system overhead due to the disk I/O involved.
- When swapping creates multiple holes in memory, it is possible to combine them all into one by moving all the processes to one side of memory. This technique is known as **memory compaction.**
- Compaction requires a lot of CPU time, so most operating systems not used this technique.
- Compaction means movement of processes in the memory during their execution. It is sometimes called dynamic relocation of a program.

6.9 : Page Replacement Policies

Q.26 Discuss page replacement algorithm.

Ans. : • When a page fault occurs, page replacement algorithms are used for loading the page in memory and no free page frame exist in memory.

- Page fault occurs if a running process references a nonresident page.
- The goal of a replacement strategy is to minimize the fault rate. To evaluate a replacement algorithm, following parameters are used :

1. The size of a page
2. A set of reference strings
3. The number of page frames.

- Given these inputs, we can determine the number of page faults and hence the page-fault rate. A reference string is a list of pages in order of their reference by the processor.
- When we find that a page frame reference is in main memory then we have a page hit and when page fault occurs we say it is page miss. A poor choice of page replacement algorithm may result in lot of page misses.
- Page replacement algorithm deals with how the kernel decides which page reclaim. Operating systems selects the local or global page replacement policy. Local replacement policy allocates a certain number of pages to each process. If a process needs new page, it must replace one of its own pages. If the OS uses a global policy, it can take a page from any process, using global selection criteria.
- UNIX OS uses global page replacement policy.

Q.27 List the page replacement algorithms. Explain LRU in detail.

☞ [SPPU : May-16, End Sem, Marks 6]

Ans. : List of page replacement algorithms :

1. First-In-First-Out 2. LRU Page Replacement Algorithm
3. Optimal Page Replacement

The LRU page replacement policy :

- LRU stands for least recently used. This replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes the recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.
- System must maintain a time stamp for every page that indicates the time of the last access. When deciding which page to remove, the operating system must scan all the pages of memory for finding the oldest time stamp. Instead of keeping an extra time stamp bits, LRU is implement by maintaining pages in an ordered list.
- System can manage LRU with a list called the LRU stack or the paging stack. In the LRU stack, the first entry describes the page referenced least recently, the last entry describes to the last page referenced. If a page is referenced, move it to the end of the list.
- LRU algorithm does not interfere with the locality of the process.

- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	4	4	4	3	3	3	6	6	6	6	9	9	9	2	2	2	9
Page frame 2	-	2	2	2	2	2	2	4	4	4	7	7	7	7	7	7	7	5	5	5
Page frame 3	-	-	3	3	3	5	5	5	2	2	2	8	8	8	8	8	8	8	4	4
Page fault	PF	PF	PF	PF		PF	PF	PF	PF	PF	PF	PF		PF			PF	PF	PF	PF

Total number of page fault = 16

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	6	6	6	6	6	6	6	2	2	2	2
Page frame 2	-	2	2	2	2	2	2	2	2	2	2	2	2	9	9	9	9	5	5	5
Page frame 3	-	-	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	4	4
Page frame 4	-	-	-	4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	9
Page fault	PF	PF	PF	PF		PF				PF	PF	PF		PF			PF	PF	PF	PF

Total number of page fault = 13

- LRU is a stack algorithm. Increase in memory size will never cause an increase in the number of page interrupts.

Q.28 Consider page sequence 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 and discuss working of following page replacement policies. Also count page faults. (use no. of frames = 3).

i) FIFO ii) LRU iii) Optimal ☞ [SPPU : May-18, End Sem, Marks 9]

Ans. :

i) FIFO

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	5	5	5	5	3	3	3	3
1		3	3	3	3	2	2	2	2	2	5	5
2				1	1	1	4	4	4	4	4	2
Page fault	*	*		*	*	*	*		*		*	*

Number of page fault = 9.

ii) LRU

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	2	2	3	3	3	3
1		3	3	3	5	5	5	5	5	5	5	5
2				1	1	1	4	4	4	2	2	2
Page fault	*	*		*	*		*		*	*		

Number of page fault = 7.

iii) Optimal

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	4	4	4	2	2	2
1		3	3	3	3	3	3	3	3	3	3	3
2				1	5	5	5	5	5	5	5	5
	*	*		*	*		*			*		

Number of page fault = 6.

Q.29 Consider given page sequence a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame is 4. Show the output of LRU and optimal policies, also count page faults.

[SPPU : Dec.-18, End Sem, Marks 6]

Ans. : LRU (Page frame = 4)

	a	b	c	d	c	a	d	b	e	b	a	b	c	d
1	a	a	a	a	a	a	a	a	a	a	a	a	a	d
2	-	b	b	b	b	b	b	b	b	b	b	b	b	b
3	-	-	c	c	c	c	c	c	e	e	e	e	e	b
4	-	-	-	d	d	d	d	d	d	d	d	d	c	c
Page fault	PF	PF	PF	PF					PF				PF	PF

Total number of page fault = 7

Optimal (Page frame = 4)

	a	b	c	d	c	a	d	b	e	b	a	b	c	d
1	a	a	a	a	a	a	a	a	a	a	a	a	a	d
2	-	b	b	b	b	b	b	b	b	b	b	b	b	b
3	-	-	c	c	c	c	c	c	c	c	c	c	c	c
4	-	-	-	d	d	d	d	d	e	e	e	e	e	e
Page fault	PF	PF	PF	PF										PF

Total number of page fault = 5

Q.30 Consider the following page reference string :
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 1, 5, 6
How many page faults would occur for the following replacement algorithms, assuming three frames.
(1) LRU page replacement (2) FIFO page replacement
(3) Optimal page replacement

Ans. : (1) LRU page replacement :

	1	2	3	4	2	1	5	6	2	1	2	1	5	6
Page Frame 0	1	1	1	4	4	4	5	5	5	1	1	1	1	1
Page Frame 1	-	2	2	2	2	2	2	6	6	6	6	6	5	5
Page Frame 2	-	-	3	3	3	1	1	1	2	2	2	2	2	6
Page fault	PF	PF	PF	PF		PF	PF	PF	PF	PF			PF	PF

Total number of page fault = 11

(2) FIFO page replacement :

	1	2	3	4	2	1	5	6	2	1	2	1	5	6
Page Frame 0	1	1	1	4	4	4	4	6	6	6	6	6	5	5
Page Frame 1	-	2	2	2	2	1	1	1	2	2	2	2	2	6
Page Frame 2	-	-	3	3	3	3	5	5	5	1	1	1	1	1
Page fault	PF	PF	PF	PF		PF	PF	PF	PF	PF			PF	PF

Total number of page fault = 11

(3) Optimal page replacement

	1	2	3	4	2	1	5	6	2	1	2	1	5	6
Page Frame 0	1	1	1	1	1	1	1	1	1	1	1	1	5	5
Page Frame 1	-	2	2	2	2	2	2	2	2	2	2	2	2	2
Page Frame 2	-	-	3	4	4	4	5	6	6	6	6	6	6	6
Page fault	PF	PF	PF	PF			PF	PF					PF	

Total number of page fault = 7

Q.31 Explain with example Belady's anomaly.

Ans. : • Using FIFO algorithm for some reference strings actually generate more page faults when more page frames are allotted. This truly unexpected result was first demonstrated by Belady in 1970 and is known as Belady's anomaly.

- FIFO does not satisfy the inclusion property and is not a stack algorithm.
- A page replacement algorithm that satisfies the inclusion property is free from Belady anomaly.
- Consider the reference string : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Page frame size = 3

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Page frame 2	-	2	2	2	1	1	1	1	1	3	3	3
Page frame 3	-	-	3	3	3	2	2	2	2	2	4	4
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	

Total number of page fault = 9

Page frame size = 4

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Page frame 2	-	2	2	2	2	2	2	1	1	1	1	5
Page frame 3	-	-	3	3	3	3	3	3	2	2	2	2
Page frame 4	-	-	-	4	4	4	4	4	4	3	3	3
Page fault	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

Total number of page fault = 10

- Paging algorithm has worse performance when the amount of primary memory allocated to the process is increased. The problem creates because the set of pages loaded with a memory allocation of 3 is not same as the with a memory allocation of 4.
- There are a set of demand paging algorithms whereby the set of pages loaded with an allocation of m frames is always a subset of the set of pages loaded with an allocation of m + 1 frame. This property is called the inclusion property.
- FIFO does not satisfy the inclusion property and is not a stack algorithm. LRU can be a stack algorithm.
- FIFO algorithm uses only the modified and status bits when swapping is used.

6.10 : Thrashing

Q.32 What is thrashing ? [May-18, Dec.-18, End Sem, Marks 4]

Ans. : • Thrashing occurs when a process does not have "enough" frames allocated to store the pages it uses repeatedly, the page fault rate will be very high.

- A process is thrashing if it is spending more time for paging in/out than executing. Since thrashing leads to low CPU utilization, the operating system's medium or long term scheduler may step in, detect this and increase the degree of multiprogramming.
- Local replacement algorithms can limit the effects of thrashing. If the degree of multiprogramming is increased over a limit, processor utilization falls down considerably because of thrashing. Fig. Q.32.1 shows thrashing.

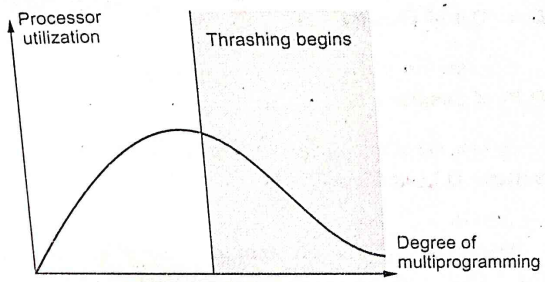


Fig. Q.32.1 Thrashing

- The selection of a replacement policy to implement virtual memory plays an important part in the elimination of the potential for thrashing. A policy based on the local mode will tend to limit the effect of thrashing. Replacement policy based on the global mode is more likely to cause thrashing. Since all pages of memory are available to all transactions, a memory-intensive transaction may occupy a large portion of memory, making other transactions susceptible to page faults and resulting in a system that thrashes.
- Thrashing is solved by using working set model and page fault frequency.

END...

SOLVED MODEL QUESTION PAPER (In Sem)
Systems Programming and Operating System
T.E. (Computer) Semester - V [As Per 2019 Pattern]

Time : 1 Hour] **[Maximum Marks : 30**

N.B. : ***i) Attempt Q.1 or Q.2, Q.3 or Q.4.***

ii) Neat diagrams must be drawn wherever necessary.

iii) Figures to the right side indicate full marks.

iv) Assume suitable data, if necessary.

Q.1 a) *What is system software ? Explain any four system software's in brief.* **(Refer Q.4 of Chapter - 1)** **[4]**

b) *Compare one pass and two pass assembler.* **(Refer Q.29 of Chapter - 1)** **[3]**

c) *Explain the data structures required for TWO PASS assembler in details.* **(Refer Q.22 of Chapter - 1)** **[8]**

OR

Q.2 a) *Define following system software components with suitable diagram, i) Compiler ii) Loader iii) Editor iv) Linker v) Debugger.* **(Refer Q.2 of Chapter - 1)** **[5]**

b) *Explain two pass translation of assembler.* **(Refer Q.19 of Chapter - 1)** **[4]**

c) *Explain in brief imperative statements, declaration statements and assembler directives with examples for assembly language programming.* **(Refer Q.13 of Chapter - 1)** **[6]**

Q.3 a) *Explain expansion time variable with example.* **(Refer Q.10 of Chapter - 2)** **[3]**

b) *What are the types of arguments used with macros ? Explain with suitable example.* **(Refer Q.8 of Chapter - 2)** **[4]**

c) *Explain design process task in macro expansion while designing two pass macro processor.* **(Refer Q.18 of Chapter - 2)** **[8]**

OR

Q.4 a) *Define macro. What are the advantages of macro facility ? how are macros different from functions.* **(Refer Q.4 of Chapter - 2)** **[5]**

b) *What is interpreter ? Explain components of interpreter.* **(Refer Q.27 of Chapter - 2)** **[4]**

c) *What are the data structure used in design of macro processor ?* **(Refer Q.14 of Chapter - 2)** **[6]**

SOLVED MODEL QUESTION PAPER (End Sem)
Systems Programming and Operating System
T.E. (Computer) Semester - V [As Per 2019 Pattern]

Time : $2\frac{1}{2}$ Hours] **[Maximum Marks : 70**

N.B. : ***i) Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***

ii) Neat diagrams must be drawn wherever necessary.

iii) Figures to the right side indicate full marks.

iv) Assume suitable data, if necessary.

Q.1 a) *What are types of loaders ? Explain compile and go loader scheme with advantages and disadvantages.* **(Refer Q.3 of Chapter - 3)** **[8]**

b) *Explain the formats of ESD, RLD, TXT and END cards with respect to direct linking loader with suitable example.* **(Refer Q.17 of Chapter - 3)** **[10]**

OR

Q.2 a) *Explain the working of direct linking loader by Pass2 Algorithm and Flowchart.* **(Refer Q.19 of Chapter - 3)** [8]

b) *Write short note on - Overlay structure* **(Refer Q.15 of Chapter - 3)** [5]

c) *Explain the concept of transfer vector in BSS loader.* **(Refer Q.10 of Chapter - 3)** [5]

Q.3 a) *Draw and explain process state transition diagram.* **(Refer Q.16 of Chapter - 4)** [6]

b) *Draw Gantt chart and calculate average turnaround time, avarage waiting time for the following processes using priority based (no-preemptive) scheduling and SJF (preemptive) scehduling policies. (Consider low no. as high priority)* **(Refer Q.32 of Chapter - 4)** [6]

c) *Explain CPU scheduling criteria.* **(Refer Q.30 of Chapter - 4)** [5]

OR

Q.4 a) *Draw Gantt chart and calculate average turnaround time, average waiting time for the following processes using SJF non preemptive and round robin with time quantum 2.* **(Refer Q.33 of Chapter - 4)** [5]

b) *Explain scheduling queue, ready queue and device queue with respect to process scheduling.* **(Refer Q.24 of Chapter - 4)** [5]

c) *Explain various multithreading model.* **(Refer Q.22 of Chapter - 4)** [7]

Q.5 a) *What is critical section ? Give semaphore solution for producer - consumer problem.* **(Refer Q.19 of Chapter - 5)** [8]

b) *How livelock situation is avaoided by using Dekker's solution ?* **(Refer Q.5 of Chapter - 5)** [6]

c) *Explain monitor in brief.* **(Refer Q.13 of Chapter - 5)** [4]

OR

Q.6 a) *Consider the following snapshot of a system.*

Processes	Allocation				Max.				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	0	1	4	2	1	2	3	3	2	1
P_1	3	1	2	1	5	2	5	2				
P_2	2	1	0	3	2	3	1	6				
P_3	1	3	1	2	1	4	2	4				
P_4	1	4	3	2	3	6	6	5				

Answer the following question using Bankers algorithm.

i) Illustrate that the system is in safe state by demonstrating an order in which the processes may complete ?

ii) If a request from a process P_1 arrives for (1, 1, 0, 0), can the request be granted immediately ?

iii) If the request from P_4 arrives for (0, 0, 2, 0), can the request be granted immediately ? **(Refer Q.28 of Chapter - 5)** [8]

b) *Write a semaphore solution for readers - writer's problem.* **(Refer Q.17 of Chapter - 5)** [6]

c) *Explain the following terms :*

i) Critical section ii) Race condition **(Refer Q.2 of Chapter - 5)** [4]

Q.7 a) *What is paging ? Explain address translation in paging.* **(Refer Q.11 of Chapter - 6)** **[6]**

b) *Given a memory partitions of 100 K, 500 K, 200 K, 300 K and 600 K (in order) how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?* **(Refer Q.21 of Chapter - 6)** **[6]**

c) *How memory is protected by using relocation and limit register ?* **(Refer Q.8 of Chapter - 6)** **[5]**

OR

Q.8 a) *What is virtual memory ? Explain demand paging with example.* **(Refer Q.22 of Chapter - 6)** **[6]**

b) *Consider page sequence 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 and discuss working of following page replacement policies. Also count page faults. (use no. of frames = 3).*
i) FIFO ii) LRU iii) Optimal **(Refer Q.28 of Chapter - 6)** **[8]**

c) *How the user program is processed ? Also explain binding of data and instruction to memory.* **(Refer Q.3 of Chapter - 6)** **[3]**

END...