TYPICAL QUESTIONS & ANSWERS

PART - I

OBJECTIVE TYPE QUESTIONS

Each Question carries 2 marks.

Choose correct or the best alternative in the following:

Q.1 Translator for low level programming language were termed as
(A) Assembler **(B)** Compiler
(C) Linker **(D)** Loader

Ans: (A)

Q.2 Analysis which determines the meaning of a statement once its grammatical structure becomes known is termed as
(A) Semantic analysis **(B)** Syntax analysis
(C) Regular analysis **(D)** General analysis

Ans: (A)

Q.3 Load address for the first word of the program is called
(A) Linker address origin **(B)** load address origin
(C) Phase library **(D)** absolute library

Ans: (B)

Q.4 Symbolic names can be associated with
(A) Information **(B)** data or instruction
(C) operand **(D)** mnemonic operation

Ans: (B)

Q.5 The translator which perform macro expansion is called a
(A) Macro processor **(B)** Macro pre-processor
(C) Micro pre-processor **(D)** assembler

Ans: (B)

Q.6 Shell is the exclusive feature of
(A) UNIX **(B)** DOS
(C) System software **(D)** Application software

Ans: (A)

Q.7 A program in execution is called
(A) Process **(B)** Instruction
(C) Procedure **(D)** Function

Ans: (A)

Q.8 Interval between the time of submission and completion of the job is called

(A) Waiting time **(B)** Turnaround time
(C) Throughput **(D)** Response time

Ans: (B)

Q.9 A scheduler which selects processes from secondary storage device is called

(A) Short term scheduler. **(B)** Long term scheduler.
(C) Medium term scheduler**.** **(D)** Process scheduler.

Ans: (C)

Q.10 The scheduling in which CPU is allocated to the process with least CPU-burst time is called

(A) Priority Scheduling **(B)** Shortest job first Scheduling
(C) Round Robin Scheduling **(D)** Multilevel Queue Scheduling

Ans: (B)

Q.11 The term 'page traffic' describes

(A) number of pages in memory at a given instant.
(B) number of papers required to be brought in at a given page request.
(C) the movement of pages in and out of memory**.**
(D) number of pages of executing programs loaded in memory.

Ans: (C)

Q.12 The "turn-around" time of a user job is the

(A) time since its submission to the time its results become available.
(B) time duration for which the CPU is allotted to the job.
(C) total time taken to execute the job**.**
(D) time taken for the job to move from assembly phase to completion phase.

Ans: (C)

Q.13 Which of the following can be used as a criterion for classification of data structures used in language processing.

(A) nature of a data structure **(B)** purpose of a data structure
(C) lifetime of a data structure **(D)** all of the above**.**

Ans: (D)

Q.14 Memory utilization factor shall be computed as follows

(A) memory in use/allocated memory.
(B) memory in use/total memory connected**.**
(C) memory allocated/free existing memory.
(D) memory committed/total memory available.

Ans: (B)

Q.15 Program 'preemption' is
- **(A)** forced de allocation of the CPU from a program which is executing on the CPU.
- **(B)** release of CPU by the program after completing its task.
- **(C)** forced allotment of CPU by a program to itself.
- **(D)** a program terminating itself due to detection of an error.

Ans: (A)

Q.16 An assembler is
- **(A)** programming language dependent.
- **(B)** syntax dependant.
- **(C)** machine dependant.
- **(D)** data dependant.

Ans: (C)

Q.17 Which of the following is not a fundamental process state
- **(A)** ready
- **(B)** terminated
- **(C)** executing
- **(D)** blocked

Ans: (D)

Q.18 'LRU' page replacement policy is
- **(A)** Last Replaced Unit.
- **(B)** Last Restored Unit.
- **(C)** Least Recently Used**.**
- **(D)** Least Required Unit.

Ans: (C)

Q.19 Which of the following is true?
- **(A)** Block cipher technique is an encryption technique.
- **(B)** Steam cipher technique is an encryption technique.
- **(C)** Both (A) and (B).
- **(D)** Neither of (A) and (B).

Ans: (C)

Q.20 Which of the following approaches do not require knowledge of the system state?
- **(A)** deadlock detection.
- **(B)** deadlock prevention.
- **(C)** deadlock avoidance.
- **(D)** none of the above.

Ans: (D)

Q.21 Program generation activity aims at
- **(A)** Automatic generation of program
- **(B)** Organize execution of a program written in PL
- **(C)** Skips generation of program
- **(D)** Speedens generation of program

Ans: (A)

Q.22 Which amongst the following is not an advantage of Distributed systems?

(A) Reliability **(B)** Incremental growth
(C) Resource sharing **(D)** None of the above

Ans: (A)

Q.23 An imperative statement

(A) Reserves areas of memory and associates names with them
(B) Indicates an action to be performed during execution of assembled program
(C) Indicates an action to be performed during optimization
(D) None of the above

Ans: (B)

Q.24 Which of the following loader is executed when a system is first turned on or restarted

(A) Boot loader **(B)** Compile and Go loader
(C) Bootstrap loader **(D)** Relating loader

Ans: (C)

Q.25 Poor response time is usually caused by

(A) Process busy
(B) High I/O rates
(C) High paging rates
(D) Any of the above

Ans: (D)

Q.26 "Throughput" of a system is

(A) Number of programs processed by it per unit time
(B) Number of times the program is invoked by the system
(C) Number of requests made to a program by the system
(D) None of the above

Ans: (A)

Q.27 The "blocking factor" of a file is

(A) The number of blocks accessible to a file
(B) The number of blocks allocated to a file
(C) The number of logical records in one physical record
(D) None of the above

Ans: (C)

Q.28 Which of these is a component of a process precedence sequence?

(A) Process name **(B)** Sequence operator ';'
(C) Concurrency operator ',' **(D)** All of the above

Ans: (D)

Q.29 Which amongst the following is valid syntax of the **Fork** and **Join** Primitive?

(A) Fork <label>
Join <var>

(B) Fork <label>
Join <label>

(C) For <var>
Join <var>

(D) Fork <var>
join <var>

Ans: (A)

Q.30 Nested Macro calls are expanded using the

(A) FIFO rule (First in first out)
(B) LIFO (Last in First out)
(C) FILO rule (First in last out)
(D) None of the above

Ans: (B)

Q.31 A parser which is a variant of top-down parsing without backtracking is

(A) Recursive Descend.
(B) Operator Precedence.
(C) LL(1) parser.
(D) LALR Parser.

Ans: (A)

Q.32 The expansion of nested macro calls follows

(A) FIFO rule.
(B) LIFO rule.
(C) LILO rule.
(D) priority rule.

Ans: (B)

Q.33. In a two-pass assembler, the task of the Pass II is to

(A) separate the symbol, mnemonic opcode and operand fields.
(B) build the symbol table.
(C) construct intermediate code.
(D) synthesize the target program.

Ans: (D)

Q.34 A linker program

(A) places the program in the memory for the purpose of execution.
(B) relocates the program to execute from the specific memory area allocated to it.
(C) links the program with other programs needed for its execution.
(D) interfaces the program with the entities generating its input data.

Ans: (C)

Q.35 Which scheduling policy is most suitable for a time-shared operating system

(A) Shortest-job First.
(B) Elevator.
(C) Round-Robin.
(D) First-Come-First-Serve.

Ans: (C)

Q.36 A critical section is a program segment
- **(A)** which should run in a certain specified amount of time.
- **(B)** which avoids deadlocks.
- **(C)** where shared resources are accessed**.**
- **(D)** which must be enclosed by a pair of semaphore operations, P and V.

Ans: (C)

Q.37 An operating system contains 3 user processes each requiring 2 units of resource R .The minimum number of units of R such that no deadlocks will ever arise is
- **(A)** 4. **(B)** 3.
- **(C)** 5. **(D)** 6.

Ans: (A)

Q.38 Locality of reference implies that the page reference being made by a process
- **(A)** will always be to the page used in the previous page reference.
- **(B)** is likely to be the one of the pages used in the last few page references**.**
- **(C)** will always be to one of the pages existing in memory.
- **(D)**will always lead to a page fault.

Ans: (B)

Q.39 Which of these is not a part of Synthesis phase
- **(A)** Obtain machine code corresponding to the mnemonic from the Mnemonics table
- **(B)** Obtain address of a memory operand from the symbol table
- **(C)** Perform LC processing
- **(D)** Synthesize a machine instruction or the machine form of a constant

Ans: (C)

Q.40 The syntax of the assembler directive EQU is
- **(A)** EQU <address space> **(B)** **<**symbol>EQU<address space**>**
- **(C)** <symbol>EQU **(D)** None of the above

Ans: (B)

Q.41 The following features are needed to implement top down parsing
- **(A)** Source string marker
- **(B)** Prediction making mechanism
- **(C)** Matching and Backtracking mechanism
- **(D)** All of the above

Ans: (D)

Q.42 A macro definition consists of
- **(A)** A macro prototype statement **(B)** One or more model statements
- **(C)** Macro pre-processor statements **(D)** All of the above

Ans: (D)

Q.43 The main reason to encrypt a file is to _____.

(A) Reduce its size **(B) Secure it for transmission**
(C) Prepare it for backup **(D)** Include it in the start-up sequence

Ans: (B)

Q.44 Which of the following is not a key piece of information, stored in single page table entry, assuming pure paging and virtual memory

(A) Frame number
(B) A bit indicating whether the page is in physical memory or on the disk
(C) A reference for the disk block that stores the page
(D) None of the above

Ans: (C)

Q.45 A UNIX device driver is

(A) Structured into two halves called top half and bottom half
(B) Three equal partitions
(C) Unstructured
(D) None of the above

Ans: (A)

Q.46 The following is not a layer of IO management module

(A) PIOCS (Physical Input Output Control System)
(B) LIOCS (Logical Input Output Control System)
(C) FS (File System)
(D) MCS (Management Control System)

Ans: (D)

Q.47 Which amongst the following is not a valid page replacement policy?

(A) LRU policy (Least Recently Used)
(B) FIFO policy (First in first out)
(C) RU policy (Recurrently used)
(D) Optimal page replacement policy

Ans: (C)

Q.48 Consider a program with a linked origin of 5000. Let the memory area allocated to it have the start address of 70000. Which amongst the following will be the value to be loaded in relocation register?

(A) 20000 **(B)** 50000
(C) 70000 **(D)** 90000

Ans: (None of the above choice in correct**.)**

Q.49 An assembly language is a

(A) low level programming language

(B) Middle level programming language
(C) High level programming language
(D) Internet based programming language

Ans: (A)

Q.50 TII stands for
(A) Table of incomplete instructions
(B) table of information instructions
(C) translation of instructions information
(D) translation of information instruction

Ans: (A)

Q.51 An analysis, which determines the syntactic structure of the source statement, is called
(A) Sementic analysis **(B)** process analysis
(C) Syntax analysis **(D)** function analysis

Ans: (C)

Q.52 Action implementing instruction's meaning are a actually carried out by
(A) Instruction fetch
(B) Instruction decode
(C) instruction execution
(D) Instruction program

Ans: (C)

Q.53 The field that contains a segment index or an internal index is called
(A) target datum **(B)** target offset
(C) segment field **(D)** fix dat

Ans: (A)

Q.54 A program in execution is called
(A) process **(B)** function
(C) CPU **(D)** Memory

Ans: (A)

Q.55 Jobs which are admitted to the system for processing is called
(A) long-term scheduling **(B)** short-term scheduling
(C) medium-term scheduling **(D)** queuing

Ans: (A)

Q.56 A set of techniques that allow to execute a program which is not entirely in memory is called
(A) demand paging **(B)** virtual memory
(C) auxiliary memory **(D)** secondary memory

Ans: (B)

Q. 57 SSTF stands for
(A) Shortest-Seek-time-first scheduling **(B)** small – small-time-first
(C) simple-seek-time-first scheduling **(D)** small-simple-time-first

Ans: (A)

Q.58 Before proceeding with its execution, each process must acquire all the resources it needs is called
(A) hold and wait **(B)** No pre-emption
(C) circular wait **(D)** starvation

Ans: (A)

Q.59 Virtual memory is
(A) simple to implement
(B) used in all major commercial operating systems
(C) less efficient in utilization of memory
(D) useful when fast I/O devices are not available

Ans: (B)

Q.60 **R**elocation bits used by relocating loader are specified by
(A) Relocating loader itself **(B)** Assembler or Translator
(C) Macro processor **(D)** Both **(A)** and **(B)**

Ans: (B)

Q.61 Resolution of externally defined symbols is performed by
(A) Linker **(B)** Loader
(C) Compiler **(D)** Editor

Ans: (A)

Q.62 Relocatable programs
(A) cannot be used with fixed partitions
(B) can be loaded almost anywhere in memory
(C) do not need a linker
(D) can be loaded only at one specific location

Ans: (B)

Q.63 Page stealing
(A) is a sign of efficient system
(B) is taking page frames other working sets
(C) should be the tuning goal
(D) is taking larger disk spaces for pages paged out

Ans: (B)

Q.64 The total time to prepare a disk drive mechanism for a block of data to be read from is its

(A) latency
(B) latency plus transmission time
(C) latency plus seek time
(D) latency plus seek time plus transmission time

Ans: (C)

Q.65 To avoid race condition, the maximum number of processes that may be simultaneously inside the critical section is

(A) zero **(B)** one
(C) two **(D)** more than two

Ans: (B)

Q.66 The memory allocation scheme subject to "external" fragmentation is

(A) segmentation **(B)** swapping
(C) pure demand paging **(D)** multiple fixed contiguous partitions

Ans: (A)

Q.67 Page fault frequency in an operating system is reduced when the

(A) processes tend to the I/O-bound
(B) size of pages is reduced
(C) processes tend to be CPU-bound
(D) locality of reference is applicable to the process

Ans: (D)

Q.68 In which of the following page replacement policies Balady's anomaly occurs?

(A) FIFO **(B)** LRU
(C) LFU **(D)** NRU

Ans: (A)

Q.69 Which of the following are language processors?

(A) Assembler **(B)** Compiler
(C) Interpreter **(D)** All of the above

Ans: (D)

Q.70 Virtual memory can be implemented with

(A) Segmentation **(B)** Paging
(C) None **(D)** all of the above

Ans: (D)

Q.71 Recognition of basic syntactic constructs through reductions, this task is performed by

(A) Lexical analysis **(B)** Syntax analysis

(C) Semantic analysis **(D)** Structure analysis

Ans: (B)

Q.72 A grammar for a programming language is a formal description of

(A) Syntax **(B)** Semantics

(C) Structure **(D)** Code

Ans: (C)

Q.73 _____ is a technique of temporarily removing inactive programs from the memory of computer system

(A) Swapping **(B)** Spooling

(C) Semaphore **(D)** Scheduler

Ans: (A)

Q.74 _____ is a technique of improving the priority of process waiting in Queue for CPU allocation

(A) Starvation **(B)** Ageing

(C) Revocation **(D)** Relocation

Ans: (B)

Q.75 _____ is the time required by a sector to reach below read/write head.

(A) Seek Time **(B)** Latency Time

(C) Access time **(D)** None

Ans: (B)

Q.76 Which of the following is most general phase structured grammar?

(A) Context – Sensitive **(B)** Regular

(C) Context – Free **(D)** None of the above

Ans: (A)

Q.77 File record length

(A) Should always be fixed

(B) Should always be variable

(C) Depends upon the size of file

(D) Should be chosen to match the data characteristics.

Ans: (D)

Q.78 A public key encryption system

(A) Allows only the correct receiver to decode the data

(B) Allows only one to decode the transmission.

(C) Allows only the correct sender to decode the data.

(D) Does not encode the data before transmitting it.

Ans: (A)

PART – II

DESCRIPTIVES

Q.1. Discuss in detail Table management Techniques? **(7)**

Ans:
An Assembler uses the following tables:
OPTAB: Operation Code Table Contains mnemonic operation code and its machine language equivalent.
SYMTAB: Symbol Table maintains symbolic label, operand and their corresponding machine.
LITTAB is a table of literals used in the program
For efficiency reasons SYMTAB must remain in main memory throughout passes I and II of the assembler. LITTAB is not accessed as frequently as SYMTAB, however it may be accessed sufficiently frequently to justify its presence in the memory. If memory is at a premium, only a part of LITTAB can be kept in memory. OPTAB should be in memory during pass I

Q.2 Define the following:
(i) Formal language Grammars.
(ii) Terminal symbols.
(iii) Alphabet and String. **(9)**

Ans:
(i) A formal language grammar is a set of formation rules that describe which strings formed from the alphabet of a formal language are syntactically valid, within the language. A grammar only addresses the location and manipulation of the strings of the language. It does not describe anything else about a language, such as its semantics.
As proposed by Noam Chomsky, a grammar *G* consists of the following components:

- A finite set N of *non terminal symbols*.
- A finite set Σ of *terminal symbols* that is disjoint from N.
- A finite set P of *production rules*, each rule of the form

where * is the Kleene star operator and denotes set union. That is, each production rule maps from one string of symbols to another, where the first string contains at least one non terminal symbol.

- A distinguished non terminal symbol from set N that is the *start symbol*.

(ii)*Terminal* symbols are literal strings forming the input of a formal grammar and cannot be broken down into smaller units without losing their literal meaning. In simple words, terminal symbols cannot be changed using the rules of the grammar; that is, they're the end of the line, or terminal. For example, if the grammar rules are that *x* can become *x*a and *x* can become a*x*, then a is a terminal symbol because it cannot become something else. These are the symbols which can appear as it is in the programme.
(iii) A finite set of symbols is called alphabet. An alphabet is often denoted by sigma, yet can be given any name.
B = {0, 1} says B is an alphabet of two symbols, 0 and 1.
C = {a, b, c} says C is an alphabet of three symbols, a, b and c.

Sometimes space and comma are in an alphabet while other times they are meta symbols used for descriptions. A language is defined over an alphabet. For example binary language is defined over alphabet B.
A finite sequence of symbols from an alphabet is called string or word.
01110 and 111 are strings from the alphabet B above.
aaabccc and b are strings from the alphabet C above.
A null string is a string with no symbols, usually denoted by epsilon has zero length.

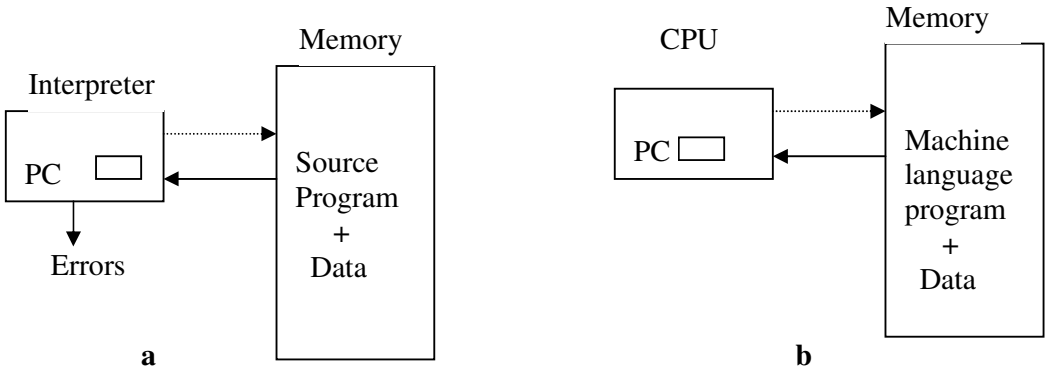
Q.3. What is parsing? Write down the drawback of top down parsing of backtracking. **(7)**

Ans:
Parsing is the process of analyzing a text, made of a sequence of tokens, to determine its grammatical structure with respect to a given formal grammar. Parsing is also known as syntactic analysis and parser is used for analyzing a text. The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. The input is a valid input with respect to a given formal grammar if it can be derived from the start symbol of the grammar.
Following are drawbacks of top down parsing of backtracking:

(i) Semantic actions cannot be performed while making a prediction. The actions must be delayed until the prediction is known to be a part of a successful parse.
(ii) Precise error reporting is not possible. A mismatch merely triggers backtracking. A source string is known to be erroneous only after all predictions have failed.

Q.4. Give the Schematic of Interpretation of HLL program and execution of a machine language program by the CPU. **(8)**

Ans:



The CPU uses a program counter (PC) to note the address of next instruction to be executed. This instruction is subjected to the instruction execution cycle consisting of the following steps:

1. Fetch the instruction.
2. Decode the instruction to determine the operation to be performed, and also its operands.
3. Execute the instruction.

At the end of the cycle, the instruction address in PC is updated and the cycle is repeated for the next instruction. Program interpretation can proceed in a similar manner. The PC can indicate which statement of the source program is to be

interpreted next. This statement would be subjected to the interpretation cycle, which consists of the following steps:

1. Fetch the statement.
2. Analyse the statement and determine its meaning, viz . the computation to be performed and its operands.
3. Execute the meaning of the statement.

Q.5. Give the difference between multiprogramming and multiprocessing. **(5)**

Ans:

A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit. The term multiprocessing is generally used to refer to large computer hardware complexes found in major scientific or commercial applications. The multiprocessor system is characterized by-increased system throughput and application speedup-parallel processing. The main feature of this architecture is to provide high speed at low cost in comparison to uni- processor.

A multiprogramming operating system is system that allows more than one active user program (or part of user program) to be stored in main memory simultaneously. Multi programmed operating systems are fairly sophisticated. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on mass storage awaiting allocation of main memory. If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. A time-sharing system is a multiprogramming system.

Q.6. Write down different system calls for performing different kinds of tasks. **(4)**

Ans:

A **system call** is a request made by any program to the operating system for performing tasks -- picked from a predefined set -- which the said program does not have required permissions to execute in its own flow of execution. System calls provide the interface between a process and the operating system. Most operations interacting with the system require permissions not available to a user level process, e.g. I/O performed with a device present on the system or any form of communication with other processes requires the use of system calls.

The main types of system calls are as follows:

• **Process Control**: These types of system calls are used to control the processes. Some examples are end, abort, load, execute, create process, terminate process etc.

• **File Management**: These types of system calls are used to manage files. Some examples are Create file, delete file, open, close, read, write etc.

• **Device Management**: These types of system calls are used to manage devices. Some examples are Request device, release device, read, write, get device attributes etc.

Q.7. Differentiate between pre-emptive and non-pre-emptive scheduling. **(4)**

Ans:

In a **pre-emptive scheduling** approach, CPU can be taken away from a process if there is a need while in a non-pre-emptive approach if once a process has been given the CPU, the CPU cannot be taken away from that process, unless the process completes or leaves the CPU for performing an Input Output.

Pre-emptive scheduling is more useful in high priority process which requires immediate response, for example in real time system. While in **nonpreemptive systems,** jobs are made to wait by longer jobs, but treatment of all processes is fairer.

Q.8. CPU burst time indicates the time, the process needs the CPU. The following are the set of processes with their respective CPU burst time (in milliseconds).

Processes	CPU-burst time
P1	10
P2	5
P3	5

Calculate the average waiting time if the process arrived in the following order:

(i) P1, P2 & P3 (ii) P2, P3 & P1 **(6)**

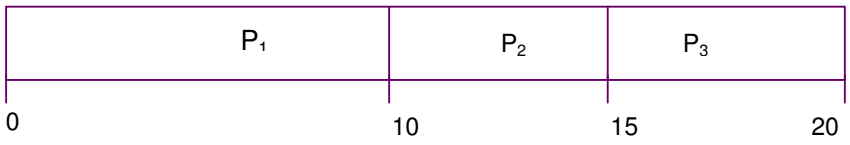
Ans:

Considering FCFS scheduling

Process	Burst Time
P_1	10
P_2	5
P_3	5

(i) Suppose that the processes arrive in the order: $P1$, $P2$, $P3$

The Gantt Chart for the schedule is:

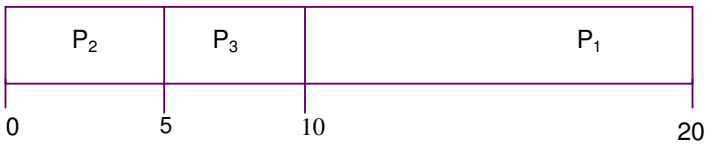


Waiting time for $P1 = 0$; $P2 = 10$; $P3 = 15$

Average waiting time: $(0 + 10 + 15)/3 = 8.33$ unit of time

(ii)Suppose that the processes arrive in the order $P2$, $P3$, $P1$.

The Gantt chart for the schedule is:



Waiting time for $P1 = 10$; $P2 = 0$; $P3 = 5$

Average waiting time: $(10 + 0 + 5)/3 = 5$ unit of time.

Q.9. What is a semaphore? Explain busy waiting semaphores. **(6)**

Ans:

A **semaphore** is a protected variable or abstract data type which constitutes the classic method for restricting access to shared resources such as shared memory in a parallel programming environment.

Weak, Busy-wait Semaphores:

- The simplest way to implement semaphores.
- Useful when critical sections last for a short time, or we have lots of CPUs.
- S initialized to positive value (to allow someone in at the beginning).
- S is an integer variable that, apart from initialization, can only be accessed through 2 *atomic and mutually exclusive* operations:

```
wait(s):
      while (s.value != 0);
      s.value--;
signal(s):
      s.value++;
```

All happens atomically i.e. wrap pre and post protocols.

Q.10. What are the four necessary conditions of deadlock prevention? **(4)**

Ans:

Four necessary conditions for deadlock prevention:

1. Removing the mutual exclusion condition means that no process may have exclusive access to a resource. This proves impossible for resources that cannot be spooled, and even with spooled resources deadlock could still occur. Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms.

2. The "hold and wait" conditions may be removed by requiring processes to request all the resources they will need before starting up. Another way is to require processes to release all their resources before requesting all the resources they will need.

3. A "no preemption" (lockout) condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent or thrashing may occur. However, inability to enforce preemption may interfere with a priority algorithm. Algorithms that allow preemption include lock-free and wait-free algorithms and optimistic concurrency control.

4. The circular wait condition: Algorithms that avoid circular waits include "disable interrupts during critical sections", and "use a hierarchy to determine a partial ordering of resources" and Dijkstra's solution.

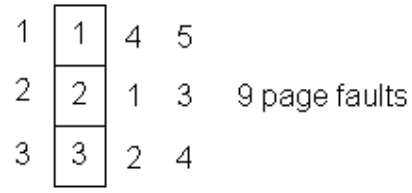
Q.11. Define the following:

(i) FIFO Page replacement algorithm.

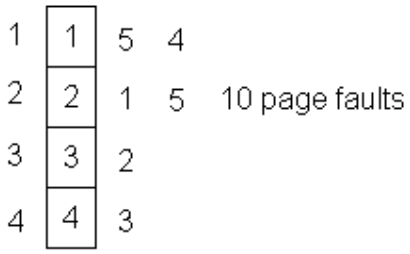
(ii) LRU Page replacement algorithm. **(6)**

Ans:

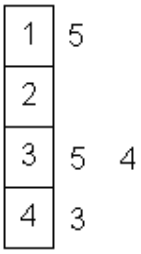
(i) FIFO policy: This policy simply removes pages in the order they arrived in the main memory. Using this policy we simply remove a page based on the time of its arrival in the memory. For example if we have the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and 3 frames (3 pages can be in memory at a time per process) then we have 9 page faults as shown



If frames are increased say to 4, then number of page faults also increases, to 10 in this case.



(ii) LRU policy: LRU expands to least recently used. This policy suggests that we remove a page whose last usage is farthest from current time. For reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, we have the following page faults



Q.12. List the properties which a hashing function should possess to ensure a good search performance. What approaches are adopted to handle collision? **(8)**

Ans:

A hashing function h should possess the following properties to ensure good search performance:

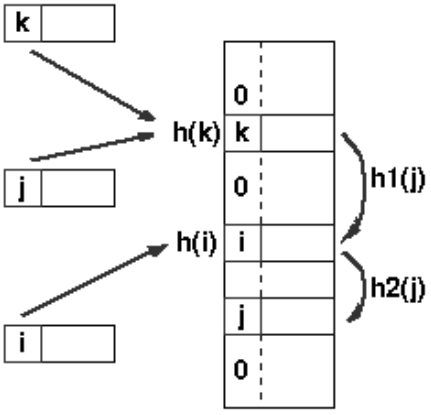
1.The hashing function should not be sensitive to the symbols used in some source program. That is it should perform equally well for different source programs.

2.The hashing function h should execute reasonably fast.

The following approaches are adopted to handle collision are:

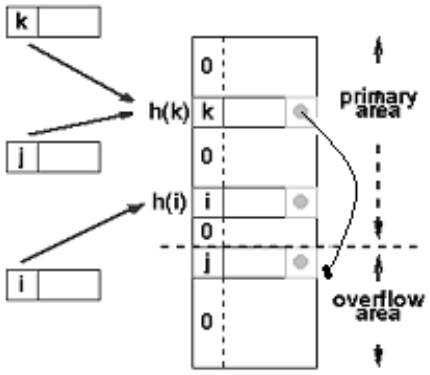
Chaining: One simple scheme is to chain all collisions in lists attached to the appropriate slot. This allows an unlimited number of collisions to be handled and doesn't require *a priori* knowledge of how many elements are contained in the collection. The tradeoff is the same as with linked lists versus array implementations of collections: linked list overhead in space and, to a lesser extent, in time.

Rehashing: Re-hashing schemes use a second hashing operation when there is a collision. If there is a further collision, we *re-hash* until an empty "slot" in the table is found. The re-hashing function can either be a new function or a re-application of the original one. As long as the functions are applied to a key in the same order, then a sought key can always be located.



h(j)=h(k), so the next hash function, **h1** is used. A second collision occurs, so **h2** is used.

Overflow chaining: Another scheme will divide the pre-allocated table into two sections: the *primary area* to which keys are mapped and an area for collisions, normally termed the *overflow area*. When a collision occurs, a slot in the overflow area is used for the new element and a link from the primary slot established as in a chained system. This is essentially the same as chaining, except that the overflow area is pre-allocated and thus possibly faster to access. As with re-hashing, the maximum number of elements must be known in advance, but in this case, two parameters must be estimated: the optimum size of the primary and overflow areas.



Q.13. What is assembly language? What kinds of statements are present in an assembly language program? Discuss. Also highlight the advantages of assembly language. **(8)**

Ans:

Assembly language is a family of low-level language for programming computers, microprocessors, microcontrollers etc. They implement a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonic) that help the programmer remember

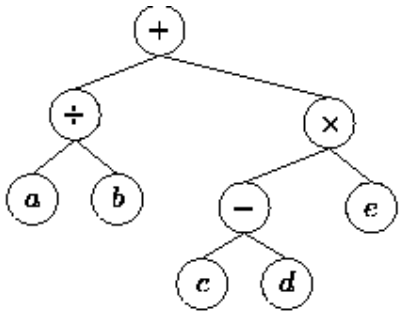
individual instruction, register etc. Assembly language programming is writing machine instructions in mnemonic form, using an assembler to convert these mnemonics into actual processor instructions and associated data.
An assembly program contains following three kinds of statements:
1. **Imperative statements**: These indicate an action to be performed during execution of the assembled program. Each imperative statement typically translates into one machine instruction.
2. **Declaration statements**: The syntax of declaration statements is as follows:
[Label] DS<constant>
[Label] DC '<value>'
The DS statement reserves areas of memory and associates names with them.
The DC statement constructs memory words containing constants.
3. **Assembler directives**: These instruct the assembler to perform certain actions during the assembly of a program. For example
START <constant> directive indicates that the first word of the target program generated by the assembler should be placed in the memory word with address <constant>.
The advantages of assembly language program would be
- reduced errors
- faster translation times
- changes could be made easier and faster

Q.14. What is an expression tree? How an expression is evaluated using an expression tree? Discuss its advantages over the other evaluation techniques. **(8)**

Ans:
Algebraic expressions such as

```
a/b+(c-d)e
```

have an inherent tree-like structure. For example, following figure is a representation of the expression in above equation. This kind of tree is called an *expression tree*.



The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (*a*, *b*, *c*, *d*, and *e*). The non-terminal nodes of an expression tree are the operators (+, -, ×, and ÷)
The expression tree is evaluated using a post-order traversal of the expression tree as follows:
1. If this node has no children, it should return the value of the node
2. Evaluate the left hand child
3. Evaluate the right hand child
4. Then evaluate the operation indicated by the node and return this value

An expression tree is advantageous for:

- Understanding the order of operation. Operations that must be done sooner are further to the right in the tree.
- Counting the number of terms or factors in an expression. Each term or factor is a child node. For example the expression (a+b)/c+2*d contains two terms.

Q.15. Draw an expression tree for the string.

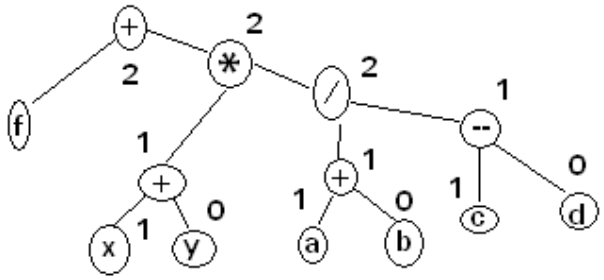
f + (x+y) *((a+b)/(c-d))

Indicate the register requirement for each node and list out the evaluation order for the expression tree. **(8)**

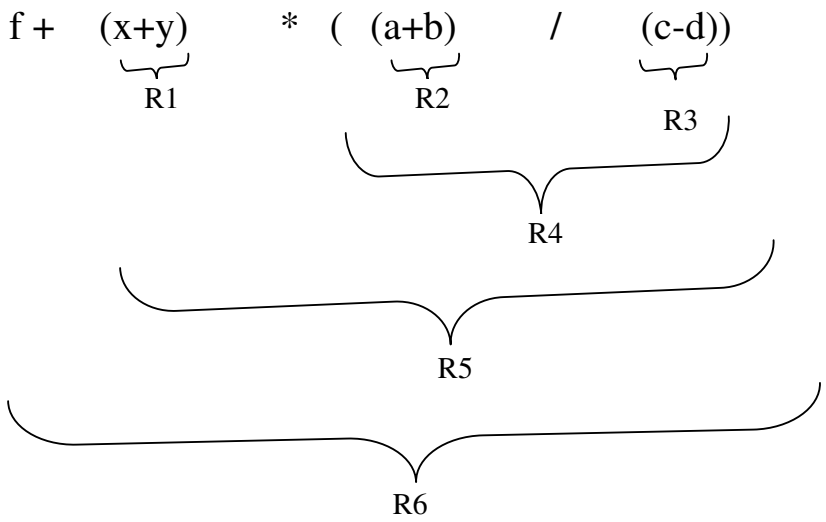
Ans:

An expression tree for the string "f + (x+y) *((a+b)/(c-d))" is given below:

Maximun register requirement is 2.



The expression will be evaluated in the following order: resister R1 first, then register R2, and so on.



Q.16. Explain the following:-

(i) Elimination of common sub expressions during code optimisation.
(ii) Pure and impure interpreters.
(iii) Lexical substitution during macro expansion.
(iv) Overlay structured program.
(v) Facilities of a debug monitor.
(vi) Actions of an interrupt processing routine.
(vii) Real time operating system.
(viii) Fork-join. **(16)**

Ans:

(i) Elimination of common sub expression during code optimization

An optimizing transformation is a rule for rewriting a segment of a program to improve its execution efficiency without affecting its meaning. One of the techniques is "Common sub expression elimination"

In the expression "(a+b)-(a+b)/4", "common subexpression" refers to the duplicated "(a+b)". Compilers implementing this technique realize that "(a+b)" won't change, and as such, only calculate its value once.

(ii) Pure and impure interpreters

In a pure interpreter, the source program is retained in the source form all through its interpretation. This arrangement incurs substantial analysis overheads while interpreting a statement.An impure interpreter performs some preliminary processing of the source program to reduce the analysis overheads during interpretation. The preprocessor converts the program to an intermediate representation (IR), which is used during interpretation. This speeds up interpretation as the code component of the IR i.e the IC, can be analyzed more efficiently than the source form of the program.

(iii)Lexical substitution during macro expansion: Lexical substitution is used to generate an assembly statement from a model statement. A model statement consists of 3 types of strings:

1. An ordinary string, which stands for itself.
2. The name of a formal parameter which is preceded by the character '&'.
3. The name of a preprocessor variable, which is also preceded by the character '&'.

During lexical expansion, strings of type 1 are retained without substitution. Strings of types 2 and 3 are replaced by the 'values' of the formal parameters or preprocessor variables. The value of a formal parameter is the corresponding actual parameter string.

(iv) Overlay structured program: A program containing overlays is referred as overlay structured program where an overlay is a part of program which has the same load origin as some other part(s) of the program. Such a program consists of

1. A permanently resident portion, called the root
2. A set of overlays

The overlay structure of a program is designed by identifying mutually exclusive modules-that is, modules that do not call each other. The basic idea is that such modules do not need to reside simultaneously in memory. Hence they are located in different overlays with the same load origin.

(v) Facilities of a debug monitor are as follows:

a. Setting breakpoints in the program

b. Initiating a debug conversation when control reaches a breakpoint

c. Displaying values of variables

d. Assigning new values to variables

e. Testing user defined assertions and predicates involving program variables.

(vi) Action of an interrupt processing routine are as follows:

1. Save contents of CPU registers. This action is not necessary if the CPU registers are saved by the interrupt action itself.

2. Process the interrupt and take appropriate actions. The interrupt code field of saved PSW information unit corresponding to this interrupt contains useful information for this purpose.

3. Return from interrupt processing.

(vii) Real time operating System

A real-time operating system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. A real time system is considered to function correctly only if it returns the correct result within any time constraints. So the following features are desirable in a real-time operating system:

1. Multi-tasking within an application
2. Ability to define the priorities of tasks
3. Priority driven or deadline oriented scheduling
4. Programmer defined interrupts.

(viii) fork-join are primitives in a higher level programming language for implementing interacting processes. The syntax is as follows:

fork <label>;

join <var>;

where <label> is a label associated with some program statement, and <var> is a variable. A statement fork label1 causes creation of a new process that starts executing at the statement with the label label1. This process is concurrent with the process which executed the statement fork label1.A join statement synchronizes the birth of a process with the termination of one or more processes.

Fork-Join provide a functionally complete facility for control synchronization.

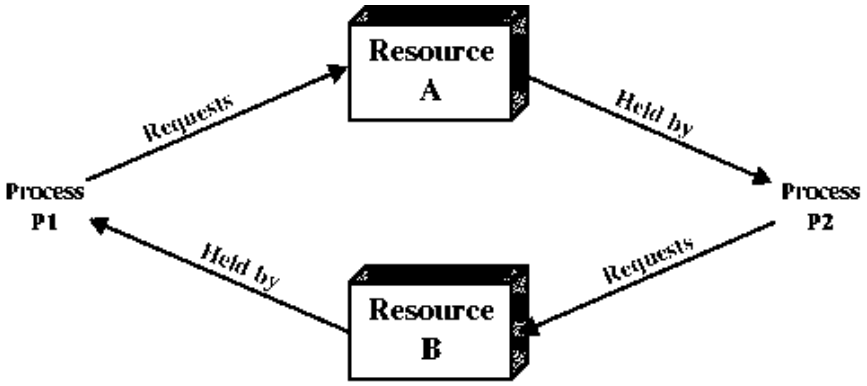
Q.17. List and explain the three events concerning resource allocation. Define the following:

(i) Deadlock.
(ii) Resource request and allocation graph (RRAG)
(iii)Wait for graph (WFG) **(6)**

Ans:

(i) **Deadlock**: Each process in a set of processes is waiting for an event that only a process in the set can cause.

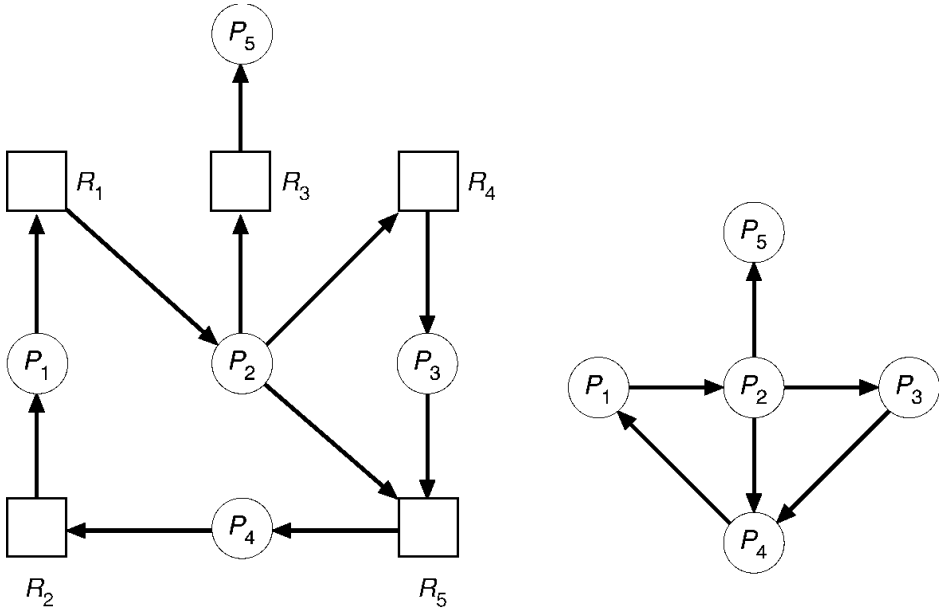
(ii) Deadlocks can be described by a directed bipartite graph called a **Resource-Request-Allocation graph (RRAG)**.A graph G = (V,E) is called bipartite if V can be decomposed into two disjoint sets V_1 and V_2 such that every edge in E joins a vertex in V_1 to a vertex in V_2.Let V_1 be a set of processes and V_2 be a set of resources. Since the graph is directed we will consider:



- an edge (R_j,P_i) (an assignment edge) to mean that resource R_j has been allocated to process P_i
- an edge (P_i,R_j) (called a request edge) to mean that process Pi has requested resource R_j

(iii)

1. Use a resource allocation graph to derive a **wait-for graph**.
2. Wait-for graph obtained by making an edge from p1 to p2 iff p1 is waiting for a resource that is allocated to p2.
3. Deadlock exists iff a cycle exists in resulting wait-for graph.



Q.18. A system contains 10 units of resource class Ru. The resource requirements of three user processes P1, P2 and P3 are as follows

	P1	P2	P3
Maximum requirements	8	7	5
Current allocation	3	1	3
Balance requirements	5	6	2
New request made	1	0	0

Using Banker's algorithm, determine if the projected allocation state is safe and whether the request of P1 will be granted or not. **(6)**

Ans:

From the given data:

Total_alloc=[7]

Total_exist=[10]

The projected allocation state is feasible since the total allocation in it does not exceed the number of resource units of Ru. Since P3 is two units short of its maximum requirements and two unallocated units exits in the system, hence P3 can complete. This will release the resources allocated to it, that is, 5 resources. Now P1 can complete since the number of unallocated units of Ru exceeds the units needed to satisfy its maximum requirement then P2 can be completed. Thus the

processes can finish in the sequence P3, P1, and P2. Hence projected allocation state is safe so algorithm will grant the request made by P1.

Q.19. What is a race condition? Explain how does a critical section avoid this condition. What are the properties which a data item should possess to implement a critical section? **(6)**

Ans:

Race condition: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last. To prevent race conditions, concurrent processes must be **synchronized**.

Data consistency requires that only one processes should update the value of a data item at any time. This is ensured through the notion of a critical section. A critical section for data item d is a section of code, which cannot be executed concurrently with itself or with other critical sections for d. Consider a system of n processes (P_0, P_1,…, P_{n-1}), each process has a segment of code called a critical section, in which the proceses may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus the execution of critical sections by the processes is mutually exclusive in time.

repeat

Entry section

critical section

Exit section

remainder section

until FALSE

Solution to the Critical Section Problem must satisfy the following three conditions:

1. **Mutual Exclusion**. If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting**. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

—Assume that each process executes at a nonzero speed

—No assumption concerning relative speed of the *n* processes.

Q.20. Describe a solution to the Dining philosopher problem so that no races arise. **(4)**

Ans:

A solution to the dining philosopher problem**:**

```
monitor DP
     {
```

```
    enum { THINKING; HUNGRY, EATING) state [5] ;
    condition self [5];
    void pickup (int i) {
           state[i] = HUNGRY;
           test(i);
           if (state[i] != EATING) self [i].wait;
    }
           void putdown (int i) {
           state[i] = THINKING;
                   // test left and right neighbors
            test((i + 4) % 5);
            test((i + 1) % 5);
        }
void test (int i) {
            if ( (state[(i + 4) % 5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i + 1) % 5] != EATING) ) {
                 state[i] = EATING ;
              self[i].signal () ;
             }
     }
        initialization_code() {
           for (int i = 0; i < 5; i++)
           state[i] = THINKING;
    }
}
```

Each philosopher *I* invokes the operations pickup() and putdown() in the following sequence:

```
       dp.pickup (i)
            EAT
        dp.putdown (i)
```

Q.21. Discuss two main approaches to identify and reuse free memory area in a heap. **(6)**

Ans:

Two popular techniques to identify free memory areas as a result of allocation and de-allocations in a heap are:

1. Reference count: the system associates a reference count with each memory area to indicate the number of its active users. This number is incremented when a user accesses that area and decrements when user stops using that. The area is free if the reference counts drops to zero. This scheme is very simple to implement however incurs incremental overheads.

2. Garbage collection: In this technique two passes are made over the memory to identify unused areas. In the first pass it traverses all pointers pointing to allocated areas and marks the memory areas that are in use. The second pass finds all unmarked areas and declares them to be free. The garbage collection overheads are not incremental. They are incurred every time the system runs out of free memory to allocate to fresh requests.

Two main approaches to reuse free memory area in a heap are:

First-fit: Allocate the *first* hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. Searching is stopped as soon as a free hole is found that is large enough

Best-fit: Allocate the *smallest* hole that is big enough; Entire list is searched, unless ordered by size. This strategy produces the smallest leftover hole.

Q.22. List the steps needed to perform page replacement. Explain the different page replacement policies. Also list out the main requirements, which should be satisfied by a page replacement policy. **(8)**

Ans:

The steps needed to perform page replacement are:

1.Determine which page is to be removed from the memory.
2.Perform a page-out operation.
3.Perform a page-in operation.

Different page replacement algorithms are briefly described below:

1. First-in, first-out

The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm. Here the operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the earliest arrival in front. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected.

Advantage: FIFO is cheap and intuitive.

Disadvantage: 1. Performs poorly in practical application.
2. Suffers from Belady's anomaly.

2. Not recently used

The not recently used (NRU) page replacement algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set. At a certain fixed time interval, the clock interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current clock interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:

- Class 0: not referenced, not modified
- Class 1: not referenced, modified
- Class 2: referenced, not modified
- Class 3: referenced, modified.

The NRU algorithm picks a random page from the lowest category for removal.

3. Optimal page replacement algorithm

The optimal page replacement algorithm (also known as OPT)is an algorithm that works as follows: when a page needs to be swapped in, the operating system swaps out the page whose next use will occur farthest in the future. For example, a page that is not going to be used for the next 6 seconds will be swapped out over a page that is going to be used within the next 0.4 seconds.

Disadvantage: This algorithm cannot be implemented in the general purpose operating system because it is impossible to compute reliably how long it will be before a page is going to be used.

The main requirements, which should be satisfied by a page replacement policy, are:

1. Non-interference with the program's locality of reference: The page replacement policy must not remove a page that may be referenced in the immediate future.
2. The page fault rate must not increase with an increase in the memory allocation for a program.

Q.23. What is an I/O buffer? What is the advantage of buffering? Is buffering always effective? Justify your answer with help of an example. **(8)**

Ans:

One kind of I/O requirement arises from devices that have a very high character density such as tapes and disks. With these characteristics, it is not possible to regulate communication with devices on a character-by-character basis. The information transfer, therefore, is regulated in blocks of information. Additionally, sometimes this may require some kind of format control to structure the information to suit the device and/or data characteristics. For instance, a disk drive differs from a line printer or an image scanner. For each of these devices, the format and structure of information is different. It should be observed that the rate at which a device may provide data and the rates at which an end application may consume it might be considerably different. In spite of these differences, the OS should provide uniform and easy to use I/O mechanisms. Usually, this is done by providing a I/O buffer. The OS manages this buffer so as to be able to comply with the requirements of both the producer and consumer of data. Basically, the buffers absorb mismatch in the data transfer rates of processor or memory on one side and device on the other.

Q.24. Discuss the different techniques with which a file can be shared among different users. **(8)**

Ans:

Some popular techniques with which a file can be shared among different users are:

1. Sequential sharing: In this sharing technique, a file can be shared by only one program at a time, that is, file accesses by P1 and P2 are spaced out over time. A lock field can be used to implement this. Setting and resetting of the lock at file open and close ensures that only one program can use the file at any time.

2. Concurrent sharing: Here a number of programs may share a file concurrently. When this is the case, it is essential to avoid mutual interference between them. There are three categories of concurrent sharing:

a. **Immutable files**: If a file is shared in immutable mode, none of the sharing programs can modify it. This mode has the advantage that sharing programs are independent of one another.

b. **Single image immutable files**: Here the changes made by one program are immediately visible to other programs. The Unix file system uses this file-sharing mode.

c. **Multiple image mutable files**: Here many programs can concurrently update the shared file. Each updating program creates a new version of the file, which is different from the version created by concurrent programs. This sharing mode can only be used in applications where concurrent updates and the existence of multiple versions are meaningful.

Q.25. Differentiate between protection and security. Explain the techniques used for protection of user files. **(8)**

Ans:

Operating system consists of a collection of objects, hardware or software. Each object has a unique name and can be accessed through a well-defined set of operations. Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Security must consider external environment of the system, and protect it from:

- Unauthorized access.
- malicious modification or destruction
- Accidental introduction of inconsistency.

It is easier to protect against accidental than malicious misuse.
Protection of user files means that file owner/creator should be able to control: what can be done and by whom. Various categories of access to files are:

- Read
- Write
- Execute
- Append
- Delete
- List

Q.26. What is parsing? Explain any three parsing techniques. **(8)**

Ans

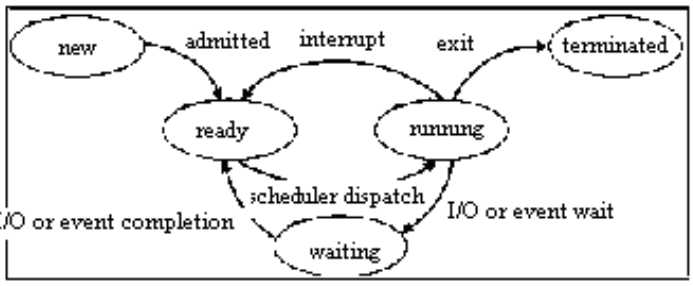
Parsing is the process of analyzing a text, made of a sequence of tokens, to determine its grammatical structure with respect to a given formal grammer. Parsing is also known as syntactic analysis and parser is used for analyzing a text. The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar.
Following are three parsing techniques:
Top-down parsing - Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules.
Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing.
Recursive descent parsing- It is a top down parsing without backtracking. This parsing technique uses a set of recursive procedures to perform parsing. Salient advantages of recursive descent parsing are its simplicity and generality. It can be implemented in any language supporting recursive procedures.

Q.27. Draw the state diagram of a process from its creation to termination, including all transitions, and briefly elaborate **every state** and **every transition**. **(8)**

Ans:

As a process executes, it changes *state*

- **new**: The process is being created.
- **running**: Instructions are being executed.
- **waiting**: The process is waiting for some event to occur.
- **ready**: The process is waiting to be assigned to a processor.
- **terminated**: The process has finished execution.



Q.28. Consider the following system snapshot using data structures in the Banker's algorithm, with resources A, B, C, and D, and process P0 to P4:

	Max				Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	6	0	1	2	4	0	0	1								
P1	1	7	5	0	1	1	0	0								
P2	2	3	5	6	1	2	5	4								
P3	1	6	5	3	0	6	3	3								
P4	1	6	5	6	0	2	1	2								
													3	2	1	1

Using Banker's algorithm, answer the following questions.

(a) How many resources of type A, B, C, and D are there? **(2)**

(b) What are the contents of the Need matrix? **(3)**

(c) Is the system in a safe state? Why **(4)**

(d) If a request from process P4 arrives for additional resources of (1,2,0,0,), can the Banker's algorithm grant the request immediately? Show the new system state and other criteria. **(7)**

Ans:

(a) A-9; B-13;C-10;D-11

(b) Need[i, j]=Max[i,j]-Allocation[i,j] so content of Need matrix is

	A	B	C	D
P0	2	0	1	1
P1	0	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

(c) The system is in a safe state as the processes can be finished in the sequence P0, P2, P4, P1 and P3.

(d) If a request from process P4 arrives for additional resources of (1,2,0,0,), and if this request is granted, the new system state would be tabulated as follows.

	Max				Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	6	0	1	2	4	0	0	1	2	0	1	1				
P1	1	7	5	0	1	1	0	0	0	6	5	0				
P2	2	3	5	6	1	2	5	4	1	1	0	2				
P3	1	6	5	3	0	6	3	3	1	0	2	0				
P4	1	6	5	6	1	4	1	2	0	2	4	4				
													2	0	1	1

After P_O completes P_3 can be allocated. 1020 from released 6012 and available 2011(Total 80 23) and <P_o, P_3, P_4, P_2, P_1> is a safe sequence.

Q.29. Define the following
(i) Process;
(ii) Process Control Block; (PCB)
(iii) Multi programming;
(iv)Time sharing. **(8)**

Ans:
(i) Process: Process is a program in execution; process execution must progress in sequential fashion. A process includes:
- program counter
- stack
- data section

(ii) Process Control Block (PCB): Information associated with each process is stored in Process control Block.
Process state
Program counter
CPU registers
CPU scheduling information
Memory-management information
Accounting information
I/O status information

(iii) Multiprogramming: A multiprogramming operating system is system that allows more than one active user program (or part of user program) to be stored in main memory simultaneously. Multi programmed operating systems are fairly sophisticated. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on mass storage awaiting allocation of main memory. If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. A time-sharing system is a multiprogramming system.

(iv) Time Sharing: Sharing of a computing resource among many users by means of multiprogramming and multi-tasking is known as timesharing. By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability, made it possible for individuals and organizations to use a computer without owning one, and promoted the interactive use of computers and the development of new interactive applications.

Q.30. Why are Translation Look-aside Buffers (TLBs) important? In a simple paging system, what information is stored in a typical TLB table entry? **(8)**

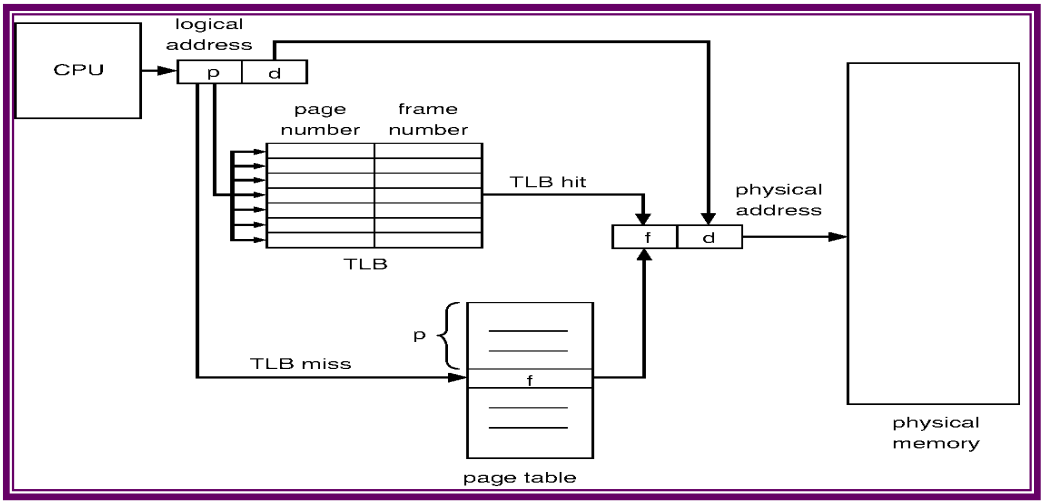
Ans:

The implementation of page-table is done in the following way:

- Page table is kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register* (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses.

One for the page table and one for the data/instruction.

The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative memory* or *translation look-aside buffers (TLBs)*. A set of associative registers is built of high-speed memory where each register consists of two parts: a key and a value. When the associative registers are presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is the output.

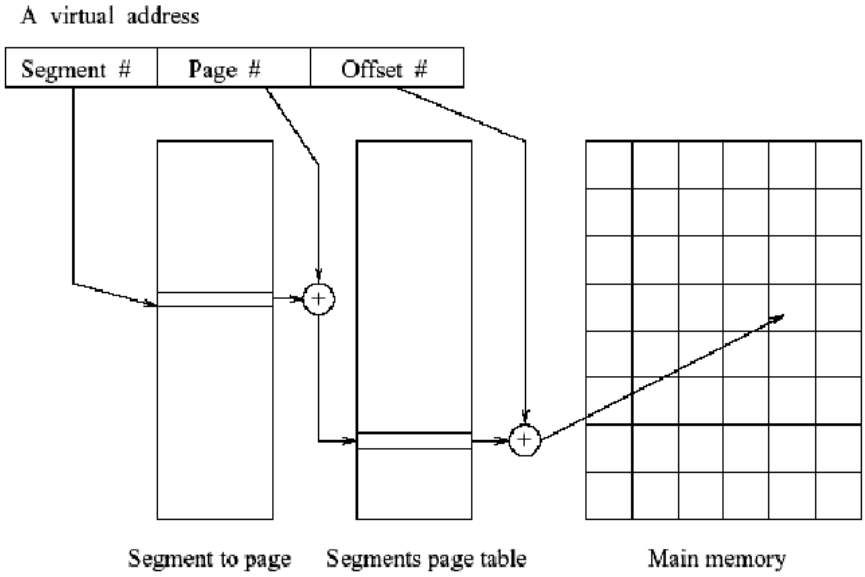


A typical TLB table entry consists of page# and frame#, when a logical address is generated by the CPU, its page number is presented to a set of associative registers that contain page number along with their corresponding frame numbers. If the page number is found in the associative registers, its frame number is available and is used to access memory. If the page number is not in the associated registers, a memory reference to the page table must be made. When the frame number is obtained, it can be used to access memory and the page number along with its frame number is added to the associated registers.

Q.31. Why is segmented paging important (as compared to a paging system)? What are the different pieces of the virtual address in a segmented paging? **(6)**

Ans:

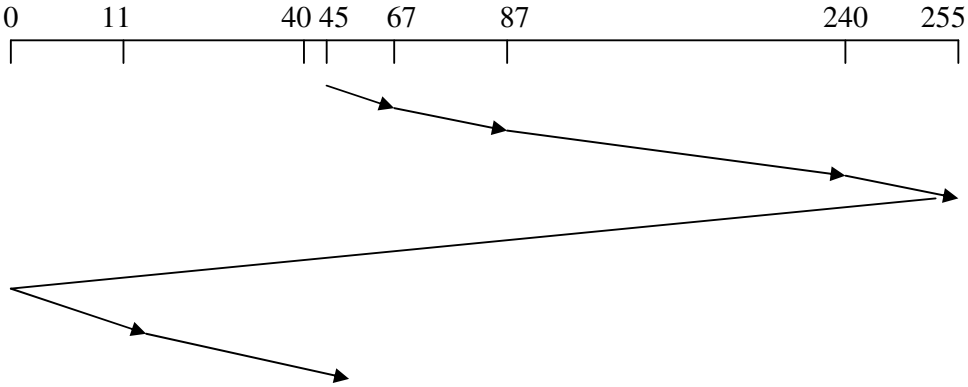
Paging can be superimposed on a segment oriented addressing mechanism to obtain efficient utilization of the memory. This is a clever scheme with advantages of both paging as well as segmentation. In such a scheme each segment would have a descriptor with its pages identified. So we have to now use three sets of offsets. First, a segment offset helps to identify the set of pages. Next, within the corresponding page table (for the segment), we need to identify the exact page table. This is done by using the page table part of the virtual address. Once the exact page has been identified, the offset is used to obtain main memory address reference. The final address resolution is exactly same as in paging. The different pieces of virtual address in a segmented paging is as shown below:



Q.32. Consider the situation in which the disk read/write head is currently located at track 45 (of tracks 0-255) and moving in the positive direction. Assume that the following track requests have been made in this order: 40, 67, 11, 240, 87. What is the order in which optimised C-SCAN would service these requests and what is the total seek distance? **(6)**

Ans:

Disk queue: 40, 67, 11, 240, 87 and disk is currently located at track 45.The order in which optimised C-SCAN would service these requests is shown by the following diagram.



Total seek distance=(67-45)+(87-67)+(240-87)+(255-240)+255+(11-0)+(40-11)
=22+20+153+15+255+11+29
=505

Q.33. Explain any three policies for process scheduling that uses resource consumption information. What is response ratio? **(8)**

Ans:

Three policies for process scheduling are described below in brief:

1. First-come First-served (FCFS) (FIFO)

– Jobs are scheduled in order of arrival
– Non-preemptive

• **Problem:**

– Average waiting time can be large if small jobs wait behind long ones
– May lead to poor overlap of I/O and CPU and convoy effects

2. Shortest Job First (SJF)

– Choose the job with the shortest next CPU burst
– Provably optimal for minimizing average waiting time

• **Problem:**

– Impossible to know the length of the next CPU burst

3. Round Robin(RR)

– Often used for timesharing
– Ready queue is treated as a circular queue (FIFO)
– Each process is given a time slice called a *quantum*
– It is run for the quantum or until it blocks
– RR allocates the CPU uniformly (fairly) across all participants. If average queue length is n, each participant gets 1/n
– As the time quantum grows, RR becomes FCFS
– Smaller quanta are generally desirable, because they improve response time

• **Problem:**

– Context switch overhead of frequent context switch

Highest Response Ratio Next (HRRN) scheduling is a non-preemptive discipline, similar to Shortest Job First (SJF) in which the priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait which prevents indefinite postponement . In fact, the jobs that have spent a long time waiting compete against those which are estimated to have short run times.

$$Priority = \frac{waiting\ time + estimated\ run\ time}{estimated\ run\ time} = 1 + \frac{waiting\ time}{estimated\ run\ time}$$

Q.34. What is a semaphore? Explain a binary semaphore with the help of an example? **(4)**

Ans:

A **semaphore** is a synchronization tool that provides a general-purpose solution to controlling access to critical sections.

A semaphore is an abstract data type (ADT) that defines a nonnegative integer variable which, apart from initialization, is accessed only through two standard operations: wait and signal. The classical definition of wait in pseudo code is

```
wait(S){
      while(S<=0)
            ; // do nothing
      S--;
}
The classical definitions of signal in pseudocode is
signal(S){
      S++;
}
```

A binary semaphore is one that only takes the values 0 and 1. These semaphores are used to implement mutual exclusion.

Q.35. Consider the following page reference and reference time strings for a program:
Page reference string: 5,4,3,2,1,4,3,5,4,3,2,1,5,…..
Show how pages will be allocated using the FIFO page replacement policy. Also calculate the total number of page faults when allocated page blocks are 3 and 4 respectively. **(8)**

Ans:
Page reference string is: 5,4,3,2,1,4,3,5,4,3,2,1,5,…..
For allocated page blocks 3, we have following FIFO allocation. Page reference marked with '+' cause page fault and result in page replacement which is performed by replacing the earliest loaded page existing in memory:

		3	3	3	4	4	4	4	4	2	2	2
	4	4	4	1	1	1	5	5	5	5	5	5
5	5	5	2	2	2	3	3	3	3	3	1	1

5^+	4^+	3^+	2^+	1^+	4^+	3^+	5^+	4	3	2^+	1^+	5

Page Reference

For allocated page blocks 4, we have following FIFO allocation. Page reference marked with '+' cause page fault and result in page replacement.

			2	2	2	2	2	2	3	3	3	3
		3	3	3	3	3	3	4	4	4	4	5
	4	4	4	4	4	4	5	5	5	5	1	1
5	5	5	5	1	1	1	1	1	1	2	2	2

5^+	4^+	3^+	2^+	1^+	4	3	5^+	4^+	3^+	2^+	1^+	5^+

Total number of page faults =10 when allocated page blocks=3
Total number of page faults =11 when allocated page blocks=4

Q.36. What are the different parameter passing mechanisms to a function? Explain with the help of example? **(8)**

Ans:

The various parameter-passing mechanisms are:
1. Call by value
2. Call by value-result
3. Call by reference
4. Call by name

In **call by value** mechanism, the values of actual parameters are passed to the called function. These values are assigned to the corresponding formal parameters. If a function changes the value of a formal parameter, the change is not reflected on the corresponding actual parameter. This is commonly used for built-in functions of the

language. Its main advantage is its simplicity. The compiler can treat formal parameter as a local variable. This simplifies compilation considerably.

Call by value-result: This mechanism extends the capabilities of the call by value mechanism by copying the values of formal parameters back into corresponding ctual parameters at return. This mechanism inherits the simplicity of the call by value mechanism but incurs higher overheads.

Call by reference: Here the address of an actual parameter is passed to the called function. If the parameter is an expression, its value is computed and stored in a temporary location and the address of the temporary location is passed to the called function. If the parameter is an array element, its address is similarly computed at the time of call. This mechanism is very popular because it has 'cleaner' semantics than call by value-result.

Call by name: This parameter transmission mechanism has the same effect as if every occurrence of a formal parameter in the body of the called function is replaced by the name of the corresponding actual parameter. The actual parameter corresponding to a formal parameter can change dynamically during the execution of a function. This makes the call by name mechanism immensely powerful. However the high overheads make it less attractive in practice.

Q.37. What is meant by inter process communication? Explain the two fundamental models of inter process communication. **(8)**

Ans:

Inter process Communication: The OS provides the means for cooperating processes to communicate with each other via an inter process communication (IPC) facility.

IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network.

IPC is best implemented by message passing system where communication among the user processes is accomplished through the passing of messages. An IPC facility provides at least the two operations:

send(message) and receive(message).

Two types of message passing system are as follows:

(a) **Direct Communication:** With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send and receive primitives are defined as:

- send(P, message)- Send a message to process P.
- receive(Q, message)- Receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly two processes.
- Exactly one link exists between each pair of processes.

(b)With indirect communication, the messages are sent to and received from mailboxes, or ports. Each mailbox has a unique identification. In this scheme, a process can communicate with some other process via a number of different

mailboxes. Two processes can communicate only if they share a mailbox. The send and receive primitives are defined as follows:

- send (A, message)- Send a message to mailbox A
- receive (A, message)- Receive a message from mailbox A.

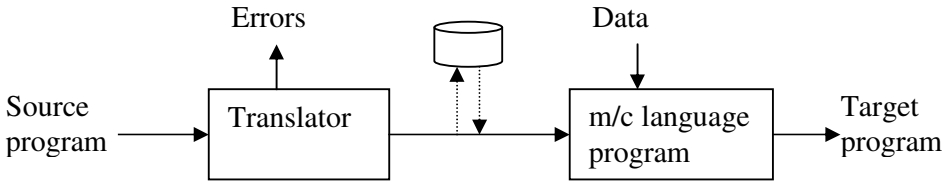
In this scheme, a communication link has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox.

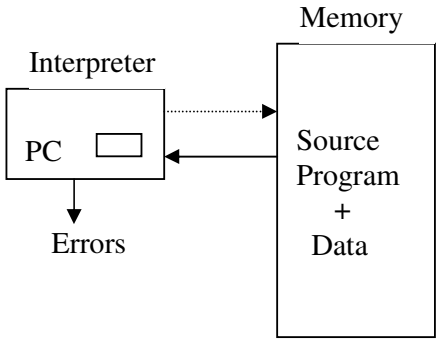
Q.38. Differentiate between program translation and program interpretation. **(6)**

Ans:

The **program translation model** bridges the execution gap by translating a program written in a programming language, called the source program (SP), into an equivalent program in the machine or assembly language of the computer system, called the target program (TP). Following diagram depicts the program translation model.



In a program interpretation process, the interpreter reads the source program and stores it in its memory. It bridges an execution gap without generating a machine language program so we can say that the interpreter is a language translator. However, it takes one statement of higher-level language at a time, translates it into machine language and executes it immediately. Translation and execution are carried out for each statement. The absence of a target program implies the absence of an outer interface of the interpreter. Thus language-processing activity of an interpreter cannot be separated from its program execution activities. Hence we can say that interpreter executes a program written in a programming language. In essence, the execution gap vanishes. Following figure depicts the program interpretation model.



Characteristics of the program translation model are:

- A program must be translated before it can be executed.
- The translated program may be saved in a file. The saved program may be executed repeatedly.
- A program must be retranslated following modifications.

Characteristics of the program interpretation model:

- The source program is retained in the source form itself i.e. no target program form exists.
- A statement is analyzed during its interpretation.

Q.39. Explain the differences between macros and subroutines. **(4)**

Ans:

Macros Vs Subroutines

(i) Macros are pre processor directives i.e. it is processed before the source program is passed to the compiler.

Subroutines are blocks of codes with a specific task, to be performed and are directly passed to the compiler.

(ii) In a macro call the pre processor replaces the macro template with its macro expansion, in a literal way.

As against this, in a function call the control is passed to a function along with certain arguments, some calculations are performed in the function and a useful value is returned back from the function.

(iii) Macro increases the program size. For example, if we use a macro hundred times in a program, the macro expansion goes into our source code at hundred different places. Whereas, functions make the program smaller and compact. For example, if a function is used, the even if it is called from hundred different places in the program, it would take the same amount of space in the program.

(iv) Macros make the program run faster as they have already been expanded and placed in the source code before compilation. Whereas, passing arguments to a function and getting back the returned values does take time and would therefore slow down the program.

(v) Example of macro

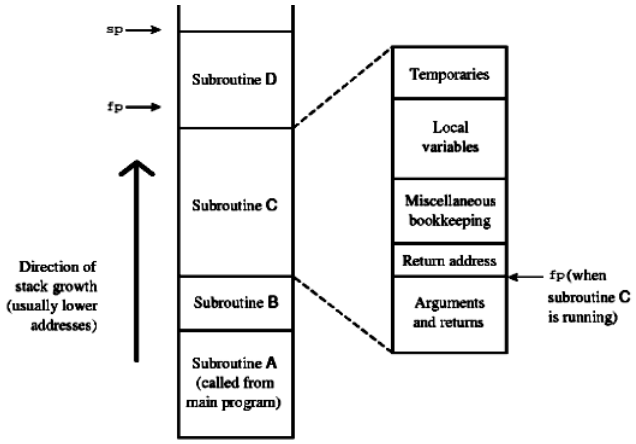
```
#define AREA(x) (3.14*x*x) // macro definition
main(){
  float r1=6.25, r2=2.5, a;
  a=AREA(r1); // expanded to (3.14 * r1 * r1)
  printf("\n Area of circle =%f",a);
  a=AREA(r2); // // expanded to (3.14 * r2 * r2)
 printf("\n Area of circle= %f",a);}
Example of subroutine
main(){
  float r1=6.25, r2=2.5, a;

  a=AREA(r1); // calls AREA()
  printf("\n Area of circle =%f",a);
  a=AREA(r2); // calls AREA()
 printf("\n Area of circle= %f",a);}
float AREA(float r) // subroutine{
  return 3.14*r*r;}
```

Q.40. Explain the stack storage allocation model. **(6)**

Ans:

In **stack-based allocation**, objects are allocated in last-in, first-out data structure, a *stack*.–*E.g.* Recursive subroutine parameters. The stack storage allocation model

- Grow and shrink on procedure calls and returns.
- Register allocation works best for stack-allocated objects.
- Memory allocation and freeing are partially predictable.
- Restricted but simple and efficient.
- Allocation is hierarchical: Memory freed in opposite order of allocation. That is If alloc (A) then alloc (B) then alloc (C), then it must be free(C) then free(B) then free(A).



Q.41. Give an account of the issue pertaining to compilation of if statement in C language (6)

Ans

Control structures like if cause significant gap between the PL domain and the execution domain because the control transfers are implicit rather than explicit. The semantic gap is bridged in two steps as follows:

1. Control structure is mapped into an equivalent program containing explicit goto's. The compiler generates its own labels and put them against the appropriate statements. For example, the equivalent of (a) given below is (b) where int1, int2 are labels generated by compiler for its own purposes.

```
if (e1) then                    if(e1) then goto int1;
    S1;                             S2;
else                                goto int2;
    S2;                         int1:S1;
S3;                             int2:S3;
```

(a) **(b)**

2. These programs are translated into assembly programs.

Q.42. Differentiate between non-relocatable, relocatable and self relocatable programs. (6)

Ans:

A **non- relocatable program** is one that cannot be executed in any memory area other than the area starting on its translated origin. For example a hand coded machine language program.

A relocatable program is one that can be processed to relocate it to a desired area of memory. For example an object module. The difference between a relocatable and a non-relocatable program is the availability of information concerning the address sensitive instructions in it. A self-relocating program is the one that can perform the relocation of its own address sensitive instructions. **A self-relocating program can** execute in any area of memory. This is very important in time-sharing operating system where load address of a program is likely to be different for different executions.

Q.43. Explain briefly any three of the commonly used code optimisation techniques. (6)

Ans:

1.Common sub expression elimination:

In the expression "(a+b)-(a+b)/4", "common sub expression" refers to the duplicated "(a+b)". Compilers implementing this technique realize that "(a+b)" won't change, and as such, only calculate its value once and use the same value next time.

2.Dead code Elimination:

Code that is unreachable or that does not affect the program (e.g. dead stores) can be eliminated. In the example below, the value assigned to i is never used, and the dead store can be eliminated. The first assignment to global is dead, and the third assignment to global is unreachable; both can be eliminated.

```
int global;
void f (){
  int i;
  i = 1;          /* dead store */
  global = 1;     /* dead store */
  global = 2;
  return;
  global = 3;     /* unreachable */}
       Below is the code fragment after dead code
elimination.
int global;
void f (){
  global = 2;
  return;}
```

3. Loop-invariant code motion

If a quantity is computed inside a loop during every iteration, and its value is the same for each iteration, it can vastly improve efficiency to hoist it outside the loop and compute its value just once before the loop begins. This is particularly important with the address-calculation expressions generated by loops over arrays. For correct implementation, this technique must be used with loop inversion, because not all code is safe to be hoisted outside the loop.

for example:

```
for (i=1; i≤10, i++){
x=y*2-(1)
.
.
.
}
```

statement 1 can be hoisted outside the loop assuming value of x and y does not change in the loop.

Q.44. Write short notes on:

i. YACC.

ii. Debug monitors. **(8)**

Ans:

(i)YACC stands for "Yet another Compiler-Compiler" : Computer program input generally has some structure; in fact, every computer program that does input can be thought of as defining an "input language" which it accepts. An input language may be as complex as a programming language, or as simple as a sequence of numbers. Unfortunately, usual input facilities are limited, difficult to use, and often are lax about checking their inputs for validity.

YACC provides a general tool for describing the input to a computer program. The YACC user specifies the structures of his input, together with code to be invoked as each such structure is recognized. YACC turns such a specification into a subroutine that handles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine. The output from YACC is LALR parser for the input programming laughing

(ii)Debug monitors provide debugging support for a program. A debug monitor executes the program being debugged under its own control thereby providing execution efficiency during debugging. There are debug monitors that are language independent and can handle programs written in many languages. For example-DEC-10. Debug monitor provide the following facilities for dynamic debugging:

1. Setting breakpoints in the program
2. Initiating a debug conversation when control reaches a breakpoint.
3. Displaying values of variables
4. Assigning new values to variables.
5. Testing user defined assertions and predicates involving program variables.

Q.45. What is an operating system? List the typical functions of operating systems. (4)

Ans:

An **operating system** is system software that provides interface between user and hardware. The operating system provides the means for the proper use of resources (CPU, memory, I/O devices, data and so on) in the operation of the computer system.

An operating system provides an environment within which other programs can do useful work.

Typical functions of operating system are as follows:

(1)Process management: A process is a program in execution. It is the job, which is currently being executed by the processor. During its execution a process would require certain system resources such as processor, time, main memory, files etc. OS supports multiple processes simultaneously. The process management module of

the OS takes care of the creation and termination of the processes, assigning resources to the processes, scheduling processor time to different processes and communication among processes.

(2)Memory management module: It takes care of the allocation and deallocation of the main memory to the various processes. It allocates main and secondary memory to the system/user program and data. To execute a program, its binary image must be loaded into the main memory.

Operating System decides.

(a) Which part of memory are being currently used and by whom.

(b) which process to be allocated memory.

(c) Allocation and de allocation of memory space.

(3)I/O management: This module of the OS co-ordinates and assigns different I/O devices namely terminals, printers, disk drives, tape drives etc. It controls all I/O devices, keeps track of I/O request, issues command to these devices.

I/O subsystem consists of

(i) Memory management component that includes buffering, caching and spooling.

(ii) Device driver interface

(iii) Device drivers specific to hardware devices.

(4)File management: Data is stored in a computer system as files. The file management module of the OS would manage files held on various storage devices and transfer of files from one device to another. This module takes care of creation, organization, storage, naming, sharing, backup and protection of different files.

(5)Scheduling: The OS also establishes and enforces process priority. That is, it determines and maintains the order in which the jobs are to be executed by the computer system. This is so because the most important job must be executed first followed by less important jobs.

(6)Security management: This module of the OS ensures data security and integrity. That is, it protects data and program from destruction and unauthorized access. It keeps different programs and data which are executing concurrently in the memory in such a manner that they do not interfere with each other.

(7)Processor management: OS assigns processor to the different task that must be performed by the computer system. If the computer has more than one processor idle, one of the processes waiting to be executed is assigned to the idle processor.

OS maintains internal time clock and log of system usage for all the users. It also creates error message and their debugging and error detecting codes for correcting programs.

Q.46. What are interrupts? How are they handled by the operating system? (5)

Ans:

Interrupt: An interrupt is a hardware mechanism that enables an external device, typically I/O devices, to send a signal to the CPU. An interrupt signal requests the CPU to interrupt its current activities and attend to the interrupting device's needs. A CPU will check interrupts only after it has completed the processing of one instruction and before it fetches a subsequent one. The basic interrupt mechanism works as follows:

The CPU hardware has wire called the interrupt-request line that the CPU senses after executing instruction. The device controller raises an interrupt by asserting a signal on the interrupt request line. CPU detects that a controller has asserted a signal on the interrupt request line.

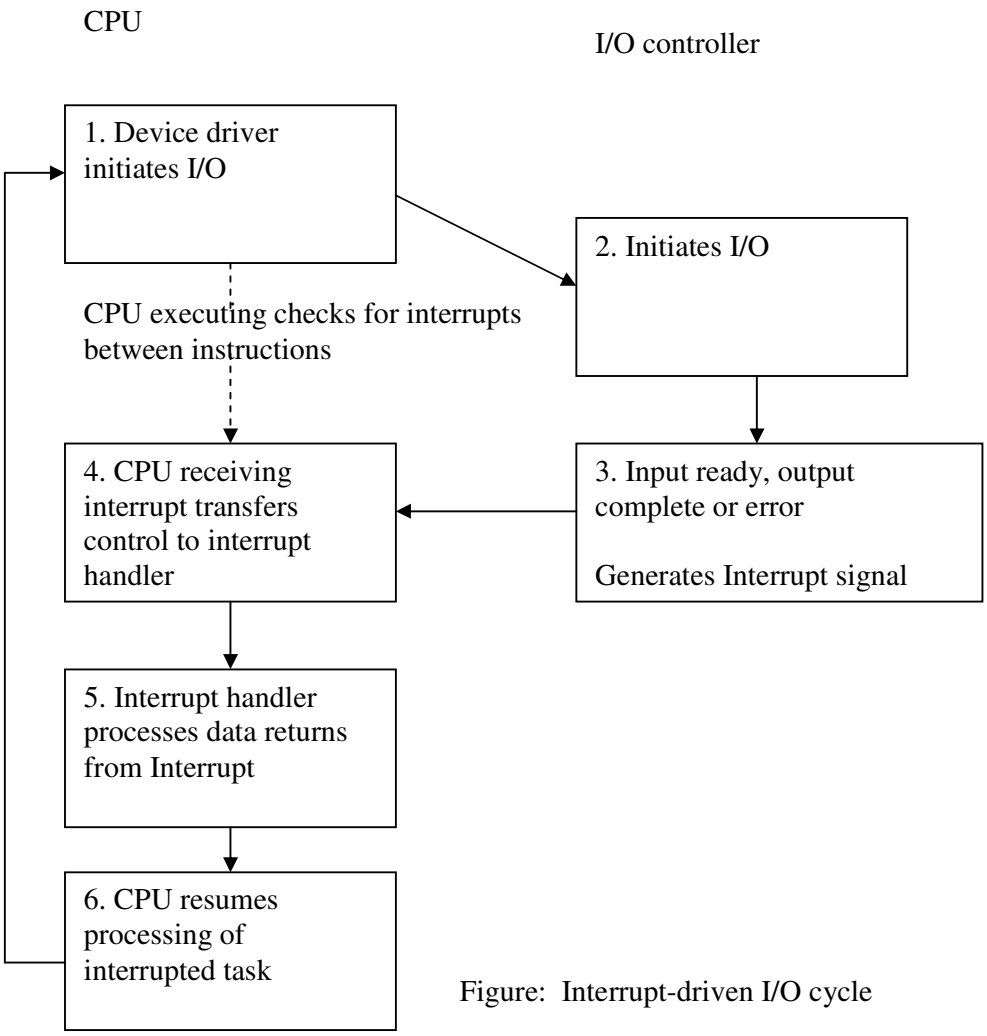


Figure: Interrupt-driven I/O cycle

The CPU saves small amount of state, such as the current value of the instruction pointer i.e. return address, and jumps to the interrupt handler routine at fixed address in memory. The interrupt handler determines the cause of the interrupt, performs the necessary processing, and executes a return from interrupt instruction, thereby clearing the interrupt. The CPU resumes to the execution state prior to the interrupt.

Q.47. Define process. Describe the contents of a Process Control Block (PCB). (5)

Ans:

Process: A process is a program in execution.

A process is an active entity, represented by the value of the program counter and the contents of the processor's registers. A process generally includes the process stack, which contains temporary data (such as method parameters, return addresses, and local variables. Two processes (may or may not be associated with the same program) are two separate execution sequences with its own text and data sections. A process may spawn many processes as it runs. Process Control Block (PCB): Each process is represented in the operating system by a process control block or task control block.

It contains many pieces of information associated with a specific process such as:

(1) Process state: The state may be new, ready, running, waiting, halted and so on.

(2) Program counter: The counter indicates the address of the next instruction to be executed for this process.

(3) CPU registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when and interrupt occurs, to allow the process to be continued correctly afterward.

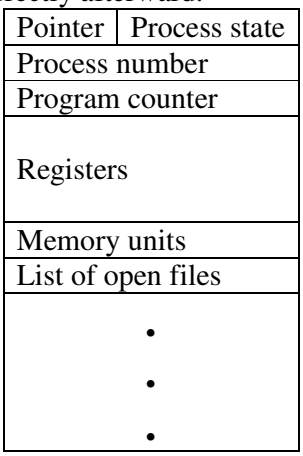


Figure: Process Control Block(PCB)

(4) CPU-scheduling information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

(5) Memory-management information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the OS.

(6) Accounting information: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers and so on.

(7) I/O status information: The information includes the list of I/o devices allocated to this process, a list of open files, and so on.

The PCB simply serves as the repository for any information that may vary from process to process

Q48. What are interacting processes? Explain any two methods of implementing interacting processes. (8)

Ans:

Interacting processes: The concurrent processes executing in the operating system are interacting or cooperating processes if they can be affected by each other.

Any process that shares data with other processes is an interacting process.

Two methods of implementing interacting process are as follows:

(i) **Shared memory solution:** This scheme requires that these processes share a common buffer pool and the code for implementing the buffer be written by the application programmer.

For example, a shared-memory solution can be provided to the bounded-buffer problem. The producer and consumer processes share the following variables:

```
#define BUFFER_SIZE 10
Typedef struct{
   ……. .
```

```
}item;
Item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

The shared buffer is implemented as a circular array with two logical pointers: in and out. The variable in points to the next free position in the buffer; out points to the first full position in the buffer. The buffer is empty when in==out; the buffer is full when ((in + 1)%BUFFER_SIZE)==out.

The producer process has a local variable nextProduced in which the new item to be produced is stored:

```
while(1){
    /* produce and item in nextProduced */
    While(((in + 1)%BUFFER_SIZE)==out)
     ; // do nothing
     Buffer[in]=nextProduced;
     in =(in+1)% BUFFER_SIZE;}
```

The consumer process has a local variable nextConsumed in which the item to be consumed is stored:

```
while(1){
  while(in==out)
  ; //do nothing
 nextConsumed = buffer[out];
 out=(out +1)% BUFFER_SIZE;
/* consume the item in nextConsumed */}
```

(ii) **Inter process Communication:** The OS provides the means for cooperating processes to communicate with each other via an interprocess communication (IPC) facility. IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network. IPC is best implemented by message passing system where communication among the user processes is accomplished through the passing of messages. An IPC facility provides at least the two operations: send(message) and receive(message).

Some types of message passing system are as follows:

Direct or Indirect Communication: With direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send and receive primitives are defined as:

- send(P, message)- Send a message to process P.
- receive(Q, message)- Receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly two processes.
- Exactly one link exists between each pair of processes.

With indirect communication, the messages are sent to and received from mailboxes, or ports. Each mailbox has a unique identification. In this scheme, a process can communicate with some other process via a number of different

mailboxes. Two processes can communicate only if they share a mailbox. The send and receive primitives are defined as follows:

- send (A, message)- Send a message to mailbox A
- receive (A, message)- Receive a message from mailbox A.

In this scheme, a communication link has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox.

Q.49. Consider the following set of jobs with their arrival times, execution time (in minutes), and deadlines.

Job Ids	Arrival Time	Execution time	Deadline
1	0	5	5
2	1	15	25
3	3	12	10
4	7	25	50
5	10	5	12

Calculate the mean turn-around time, the mean weighted turn-around time and the throughput for FCFS, SJN and deadline scheduling algorithms. (6)

Ans:

Chart for First Come First Served scheduling

1	2	3	4	5
0 5	20	32	57	62

Turnaround time = Terminated time – Arrival time

i.e, T = Tr - Ta

So, turnaround time for various jobs are

For job 1, T1 = 5-0=5 unit time

For job 2, T2 = 20-1=19 unit time

For job 3, T3 = 32-3=29 unit time

For job 4, T4 = 57-7=50 unit time

For job 5, T5 = 62-10=52 unit time

Mean turnaround time, Tm = (T1+T2+T3+T4+T5)/5= 155/5=31 unit time/job

Throughput= no of process completed per unit time= 5/62= 0.081 jobs/unit time

Chart for Shortest Job Next scheduling

1	3	5	2	4
0 5	17	22	37	62

Turnaround time for various jobs are

For job 1, T1 = 5-0=5 unit time

For job 2, T2 = 37-1=36 unit time

For job 3, T3 = 17-3=14 unit time
For job 4, T4 = 62-7=55 unit time
For job 5, T5 = 22-10=12 unit time
Mean turnaround time, Tm = (T1+T2+T3+T4+T5)/5= 122/5= 24.4unit time/job
Throughput= no of process completed per unit time= 5/62= 0.081 jobs/unit time

Q.50. What are the differences between user level threads and kernel supported threads? (4)

Ans:

A **thread,** sometimes called a lightweight process(LWP), is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set and a stack.
A thread shares with other threads belonging to the same process its code section, data section and other operating-system resources, such as open files and signals.
If the process has multiple threads of control, it can do more than one task at a time.
User Level Threads Vs Kernel Supported Threads

i. User threads are supported above the kernel and are implemented by a thread library at the user level.
Whereas, kernel threads are supported directly by the operating system.

ii. For user threads, the thread library provides support for thread creation, scheduling and management in user space with no support from the kernel as the kernel is unaware of user-level threads. In case of kernel threads, the kernel performs thread creation, scheduling and management in kernel space.

iii. As there is no need of kernel intervention, user-level threads are generally fast to create and manage. As thread management is done by the operating system, kernel threads are generally slower to create and manage that are user threads.

iv. If the kernel is single-threaded, then any user-level thread performing blocking system call, will cause the entire process to block, even if other threads are available to run within the application.
However, since the kernel is managing the kernel threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.

v. User-thread libraries include POSIX P threads, Mach C-threads and Solaris 2 UI-threads.

Some of the cotemporary operating systems that support kernel threads are Windows NT, Windows 2000, Solaris 2, BeOS and Tru64 UNIX(formerly Digital UNIX).

Q.51. Define deadlock? Explain the necessary conditions for deadlock to occur. (5)

Ans:

Deadlock is a situation, in which processes never finish executing and system resources are tied up, preventing other jobs from starting. A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes, thereby causing deadlock.Necessary conditions for deadlock to occur are:

i. **Mutual exclusion:** At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource. If another process requests

that resource, the requesting process must be delayed until the resource has been released.

ii. **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

iii. **No pre-emption:** Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after the process holding it has completed its task.

iv. **Circular wait:** A set{P0, P1,……, Pn) of waiting processes must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ……., Pn-1 is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P0.

All four conditions must hold simultaneously for a deadlock to occur and conditions are not completely independent. For example, the circular-wait implies the hold-and-wait condition.

Q52. An operating system contains 3 resource classes. The number of resource units in these classes is 7, 7 and 10. The current resource allocation state is shown below:

Processes	Allocated resources			Maximum requirements		
	R1	R2	R3	R1	R2	R3
P1	2	2	3	3	6	8
P2	2	0	3	4	3	3
P3	1	2	4	3	4	4

(i) Is the current allocation state safe?

(ii) Can the request made by process P1 (1, 1, 0) be granted? (5)

Ans:

(i) In the given question,

Available matrix for resources [R1 R2 R3] = No of resource unit -Total Allocation = [7 7 10]-[5 4 10]= [2 3 0]

Need matrix is defined as (Max – Allocation),

Processes	Need of resources		
	R1	R2	R3
P1	1	4	5
P2	2	3	0
P3	2	2	0

Using Safety Algorithm, we get sequence:

Processes	Available resources after satisfying need		
	R1	R2	R3
	2	3	0
P2	4	3	3
P3	5	5	7
P1	7	7	10

The sequence <P2, P3, P1> satisfies the safety criteria. So current allocation state is safe.

(ii) Request made by process P1, Request(P1)= [1 1 0]

Here, Request(P1)< Need(P1) < Available

i.e. [1 1 0]< [1 4 5] < [2 3 0]

Pretending that request can be fulfilled, we get new state:

Processes	Allocation			Need			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	3	3	0	3	5	1	2	0
P2	2	0	3	2	3	0			
P3	1	2	4	2	2	0			

As Need > Available for all process, no need can be fulfilled

So allocation is not thread safe i.e. request made by Process P1 can't be granted.

Q.53. What are semaphores? How do they implement mutual exclusion? (6)

Ans:

Semaphore: A semaphore is a synchronization tool that provides a general-purpose solution to controlling access to critical sections. A semaphore is an abstract data type (ADT) that defines a nonnegative integer variable which, apart from initialization, is accessed only through two standard operations: wait and signal. The classical definition of wait in pseudo code is

```
wait(S){
     while(S<=0)
          ; // do nothing
     S--; }
```

The classical definitions of signal in pseudo code is

```
signal(S){
     S++; }
```

When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. In addition, in the case of the wait(S), the testing of the integer value of S(S<=0), and its possible modification(S--), must also be executed without interruption.

Mutual-exclusion implementation with semaphores:

Let there are n-processes and they share a semaphore, mutex (standing for mutual exclusion), initialized to 1.Each process Pi is organized as shown below:

```
do{
     wait(mutex);
     critical section
     signal(mutex);
          remainder section   }while(1);
```

Disadvantage: Mutual-exclusion solutions given by semaphores require busy waiting. That is, while a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. Hence, busy waiting wastes CPU cycles that some other process might able to use productively.

Advantage: This type of semaphore is also called spinlock because the process "spins" while waiting for the lock. Spinlocks are useful in multiprocessor systems as no context switch is required when a process must wait on a lock. Thus, when locks are expected to be held for short times, spinlocks are useful.

Q.54. Give a solution for readers-writers problem using conditional critical regions. (8)

Ans:

Readers-writers problem: Let a data object(such as a file or record) is to be shared among several concurrent processes. Readers are the processes that are interested in only reading the content of shared data object. Writers are the processes that may want

to update (that is, to read and write) the shared data object. If two readers access the shared data object simultaneously, no adverse effects will result. However if a writer and some other process (either a reader or writer) access the shared object simultaneously, anomaly may arise. To ensure that these difficulties do not arise, writers are required to have exclusive access to the shared object. This synchronization problem is referred to as the readers-writers problem.

Solution for readers-writers problem using conditional critical regions. Conditional critical region is a high level synchronization construct. We assume that a process consists of some local data, and a sequential program that can operate on the data. The local data can be accessed by only the sequential program that is encapsulated within same process. One process cannot directly access the local data of another process. Processes can, however, share global data.

Conditional critical region synchronization construct requires that a variable v of type T, which is to be shared among many processes, be declared as

v: shared T;

The variable v can be accessed only inside a region statement of the following form:

region v when B do S;

This construct means that, while statement S is being executed, no other process can access the variable v. When a process tries to enter the critical-section region, the Boolean expression B is evaluated. If the expression is true, statement S is executed. If it is false, the process releases the mutual exclusion and is delayed until B becomes true and no other process is in the region associated with v.

Now, let A is the shared data object.

Let readcount is the variable that keeps track of how many processes are currently reading the object A.

Let writecount is the variable that keeps track of how many processes are currently writing the object A. Only one writer can update object A, at a given time.

Variables readcount and writecount are initialized to 0.

A writer can update the shared object A when no reader is reading the object A.

```
region A when( readcount = = 0 AND writecount = = 0){
                  ……
        writing is performed
           ……    }
```

A reader can read the shared object A unless a writer has obtained permission to update the object A.

```
region A when(readcount >=0 AND writecount = = 0){
              ……
      reading is performed                    ……    }
```

Q.55. Given memory partitions of 100k, 500k, 200k, 300k, and 600k (in order), apply first fit and best fit algorithms to place processes with the space requirement of 212k, 417k, 112k and 426k (in order)? Which algorithm makes the most effective use of memory? (3)

Ans:

Given memory partitions of 100k, 500k, 200k, 300k, and 600k (in order), applying first fit algorithms to place processes with the space requirement of 212k, 417k, 112k and 426k (in order), we have the following status:

Memory → / Request ↓	100K	500K	200K	300K	600K
212K	100K	288K	200K	300K	600K
417K	100K	288K	200K	300K	183K
112K	100K	176K	200K	300K	183K
426K	Can't fulfil request of 426K,so memory status will remain same				

And applying best fit algorithm the status is as follows:

Memory → / Request ↓	100K	500K	200K	300K	600K
212K	100K	500K	200K	88K	600K
417K	100K	83K	200K	88K	600K
112K	100K	83K	88K	88K	600K
426K	100K	83K	88K	88K	174K

Best fit makes the most efficient use of memory.

Q.56. Differentiate between
(i) Problem-oriented and procedure-oriented language
(ii) Dynamic and static binding
(iii) Scanning and parsing (9)

Ans:
(i) **Problem-oriented and procedure-oriented language:** The programming languages that can be used for specific applications are called problem oriented

languages. Such languages have large execution gaps and this gap is bridged by the translator or interpreter and does not concern the software designer.
A procedure-oriented language provides general purpose facilities required in most application domains. Such a language is independent is independent of specific application domains and results in a large specification gap which has to be bridged by an application designer.
(ii) Dynamic and static binding: A dynamic binding is a binding performed after the execution of a program has just begun while static binding is a binding performed before the execution of a program begins.
Static bindings lead to more efficient execution of a program than dynamic bindings.
(iii) Scanning and parsing: Scanning is the process of recognizing the lexical components in a source string while parsing is the process of checking the validity of a source string, and to determine its syntactic structure. The reason for separating scanning from parsing is that the lexical features of a language can be specified using Type-3 grammars. Each Type-3 production specifying lexical components is also a Type-2 production. However, a recognizer for Type-3 productions is simple, easier to build and more efficient during execution than a recognizer for Type-2 productions. Hence it is better to handle the lexical and syntactic components of a source language separately.

Q.57. Define Grammar of a language. Identify the different classes of grammar. Explain their characteristics and limitations. (10)

Ans:A **formal language grammar** is a set of formation rules that describe which strings formed from the alphabet of a formal language are syntactically valid, within the language. A grammar only addresses the location and manipulation of the strings of the language. It does not describe anything else about a language, such as its semantics.

As proposed by Noam Chomsky, a grammar *G* consists of the following components:

- A finite set N of *non terminal symbols*.
- A finite set Σ of *terminal symbols* that is disjoint from N.
- A finite set P of *production rules*, each rule of the form

where * is the Kleene star operator and denotes set union. That is, each production rule maps from one string of symbols to another, where the first string contains at least one non terminal symbol.
A distinguished symbol that is the *start symbol*. The Chomsky hierarchy consists of the following levels:

- **Type-0 grammars** (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. The language that is recognized by a Turing machine is defined as all the strings on which it halts. These languages are also known as the recursively enumerable languages.
- **Type-1 grammars** (context-sensitive grammars) generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with A a non terminal and α, β and γ strings of terminals and non terminals. The strings α and β may be empty, but γ must be nonempty. The rule $S \rightarrow \varepsilon$ is allowed if S does not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a non-deterministic

- **Type-2 grammars** (context-free grammars) generate the context-free languages. These are defined by rules of the form $A \rightarrow \gamma$ with A a non terminal and γ a string of terminals and non terminals. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context free languages are the theoretical basis for the syntax of most programming languages.
- **Type-3 grammars** (regular grammars) generate the regular languages. Such a grammar restricts its rules to a single non terminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed by a single non terminal. The rule $S \rightarrow \varepsilon$ is also here allowed if S does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

Q.58. Enumerate the data structures used during the first pass of the assembler. Indicate the fields of these data structures and their purpose/usage. (8)

Ans:

Three major data structures used during the first pass of the assembler are:

- LOCCTR (Location counter)
- OPTAB (operation code table)
- SYMTAB (Symbol table)

LOCCTR: Location Counter keeps track machine addresses of symbolic tables.

- Initialized by START.
- Increased for each instruction
  - Pseudo Instruction: BYTE, WORD,RESB, RESW.
  - Machine Instruction: Fixed length (3 bytes) for SIC, variable length for SIC/XE (looking up OPTAB).
  - Assign the value (address) of LOCCTR to corresponding symbol table.

OPTAB: operation table contains mnemonic operation code and its machine language equivalent.

ADD	18
LDS	6C
. . . . .	. . . . .

OPTAB can be implemented using hashing function for fast access.

SYMTAB: Symbol table maintain symbolic label, operand and their corresponding machine.

Addresses

First	1000
CLOOP	1003
RDREC	2039
⋮	⋮

•SYMTAB is dynamic, constructed during Pass 1:
- for symbolic label, fill in symbol and address (according to current LOCCTR).
- for symbolic operand, fill in symbol and mark it as undefined. Address field will be defied later.

•SYMTAB can be implemented using hashing function for fast access.

Q.59. What is macro-expansion? List the key notions concerning macro expansion. Write an algorithm to outline the macro-expansion using macro-expansion counter. (8)

Ans:

macro call leads to macro expansion. During macro expansion, the macro call statement is replaced by a sequence of assembly statements. Two key notions concerning macro expansion are:

1.**Expansion time control flow-** this determines the order in which model statements are visited during macro expansion.

2.**Lexical substitution:** Lexical substitution is used to generate an assembly statement from a modal statement.

The flow of control during macro expansion can be implemented using a macro-expansion counter (MEC). The outline of algorithm is as follows:

1. MEC:=statement number of first statement following the prototype statement;
2. While statement pointed by MEC is not a MEND statement
   (a) If a model statement then
      (i) expand the statement.
      (ii) MEC:=MEC+1;
   (b) Else (i.e. a pre processor statement)
      (i) MEC:=new value specified in the statement;
3. Exit from macro expansion.

Q.60 What is a heap? Name and explain the popular techniques to identify free memory areas as a result of allocation and de-allocations in a heap. **(8)**

Ans:

The **heap** is an area of memory, which is dynamically allocated. Like a stack, it may grow and shrink during runtime. Unlike a stack, a heap is not LIFO implies more complicated to manage. Two popular techniques to identify free memory areas as a result of allocation and de-allocations in a heap are:

1. Reference count: the system associates a reference count with each memory area to indicate the number of its active users. This number is incremented when a

user accesses that area and decrements when user stops using that. The area is free if the reference counts drops to zero. This scheme is very simple to implement however incurs incremental overheads.

2. Garbage collection: In this technique two passes are made over the memory to identify unused areas. In the first pass it traverses all pointers pointing to allocated areas and marks the memory areas that are in use. The second pass finds all unmarked areas and declares them to be free. The garbage collection overheads are not incremental. They are incurred every time the system runs out of free memory to allocate to fresh requests.

Q.61. What are threads? Why are they required? Discuss the differentiate between Kernel level and user level threads? (8)

Ans:

A thread, sometimes called a lightweight process(LWP), is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set and a stack.

A thread shares with other threads belonging to the same process its code section, data section and other operating-system resources, such as open files and signals.

If the process has multiple threads of control, it can do more than one task at a time.

User Level Threads Vs Kernel Supported Threads

i. User threads are supported above the kernel and are implemented by a thread library at the user level.
   Whereas, kernel threads are supported directly by the operating system.

ii. For user threads, the thread library provides support for thread creation, scheduling and management in user space with no support from the kernel as the kernel is unaware of user-level threads.
   In case of kernel threads, the kernel performs thread creation, scheduling and management in kernel space.

iii. As there is no need of kernel intervention, user-level threads are generally fast to create and manage.
   As thread management is done by the operating system, kernel threads are generally slower to create and manage that are user threads.

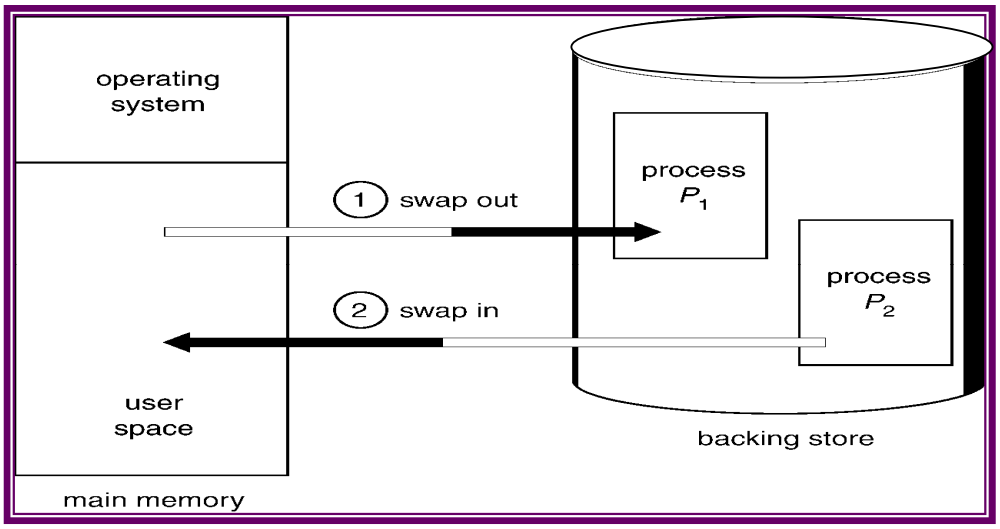
iv. If the kernel is single-threaded, then any user-level thread performing blocking system call, will cause the entire process to block, even if other threads are available to run within the application.
   However, since the kernel is managing the kernel threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.

v. User-thread libraries include POSIX P threads, Mach C-threads and Solaris 2 UI-threads.
   Some of the cotemporary operating systems that support kernel threads are Windows NT, Windows 2000, Solaris 2, BeOS and Tru64 UNIX(formerly Digital UNIX).

Q.62. What is swapping? Does swapping increase the Operating Systems' overheads? Justify your answer. (8)

Ans:

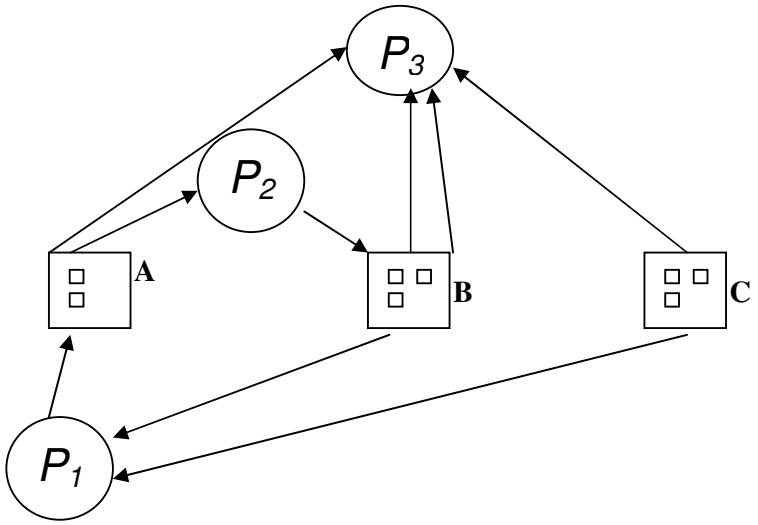
A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.



Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped. Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.

Q.63. Suppose there are 2 copies of resource A, 3 copies of resource B, and 3 copies of resource C. Suppose further that process 1 holds one unit of resources B and C and is waiting for a unit of A; that process 2 is holding a unit of A and waiting on a unit of B; and that process 3 is holding one unit of A, two units of B, and one unit of C. Draw the resource allocation graph. Is the system in a deadlocked state? Why or why not? (8)

Ans:
Resource allocation graph is given below:



There is a cycle in the resource allocation graph implies system is not in a safe state. System is in deadlock as required resources are 1 unit of resource A and 1 unit of resource B while available resource is 1 unit of resource C. None of the process can be completed.

Q.64 what is system programming? Explain the evolution of system software. (6)

Ans:

System software is collection of system programs that perform a variety of functions, viz file editing, recourse accounting, IO management, storage management etc.

System programming is the activity of designing and implementing SPs.

System programs which are the standard component of the s/w of most computer systems; The two fold motivation mentioned above arises out of single primary goal viz of making the entire program execution process more effective.

Q.65 Give difference between assembler, compiler and interpreter. (6)

Ans:

An **assembler** is the translator for an assembly language of a computer. An assembly language is a low-level programming language which is peculiar to a certain computer.

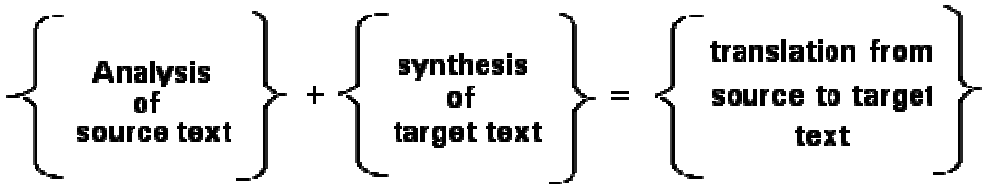
A **compiler** is a translator for machine independent HLL like say FORTRAN, COBOL etc.

An **interpreter** analysis the source program statement by statement and it self carries out the actions implied by each statement.

Q.66 Write down the general model for the translation process. (4)

Ans:

General model for the translation process can be represented as follows:



Q.67 Pass I of the assembler must also generate the intermediate code for the processed statements. Justify your answer. (8)

Ans: Criteria for selection of an appropriate intermediate code form are;

(i) Ease of use: It should be easy to construct the intermediate code form and also easy to analyze and interpret it during pass II, i.e. the amount of processing required to be done during its construction and analysis should be minimal.

(ii) Economy of storage: It should be compact as the target code itself .This will reduce the overall storage requirements of assembler.

Q.68 what are the advantages and disadvantages of macro pre-processor? (8)

Ans: The **advantage** of macro pre-processor is that any existing conventional assembler can be enhanced in this manner to incorporate macro processing. It would reduce the programming cost involved in making a macro facility available.

The **disadvantage** is that this scheme is probably not very efficient because of the time spent in generating assembly language statement and processing them again for the purpose of translation to the target language.

Q.69. What is parsing? Give difference between top down parsing and bottom up parsing. (6)

Ans:

The goal of **parsing** is to determine the syntactic validity of a source string. If the string is valid, a tree is built for use by subsequent phase of compiler.

Top down parsing: Given an input string, top down parsing attempts to derive a string identical to it by successive application of grammar rules to the grammar's distinguished symbol. When such a string is obtained , a tree representing its derivation would be the syntax tree for an input string. Thus if ά is input-string, a top down parse determines a derivation sequence.

Bottom up parsing: A bottom up parse attempts to develop syntax tree for an input string through a sequence of reduction. If the input string can be reduced to the distinguished symbol, the string is valid. If not, error would be detected and indicated during the process of reduction itself.

Q.70 How non-relocatable programs are different from relocatable programs? (4)

Ans: A **non relocatable** program is one which cannot be made to execute in any area of storage other than the one designated for it at the time of its coding or translation. A **relocatable** program form is one which consists of a program and relevant information for its relocation. Using this information it is possible to relocate the program to execute from a storage area then the one designated for it at the time of its coding or translation.

Q.71 what are the fundamental steps in program development? Discuss program testing and debugging in detail. (6)

Ans:

The fundamental steps in program development are:

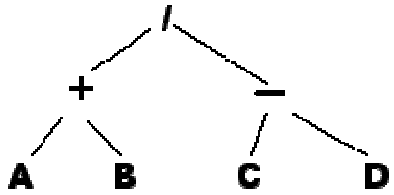
(i) Program design, coding and documentation.
(ii) Preparation of the program in machine readable form, and initial editing to adopt two required formats.
(iii) Program translation and linking/ loading
(iv) Program testing and debugging.
(v) Program modification for performance enhancement.
(vi) Reformatting programs data and/or results to suite other programs which process them.

In program testing and debugging important steps are as follows:

(i) Construction of test data for the program
(ii) Analysis of test results to detect program errors.
(iii) Localization of errors and modification of the program to eliminate them, i.e. debugging.

Q.72 Give LOAD-STORE optimization based on expression trees for the expression (A+B)/(C-D) (8)

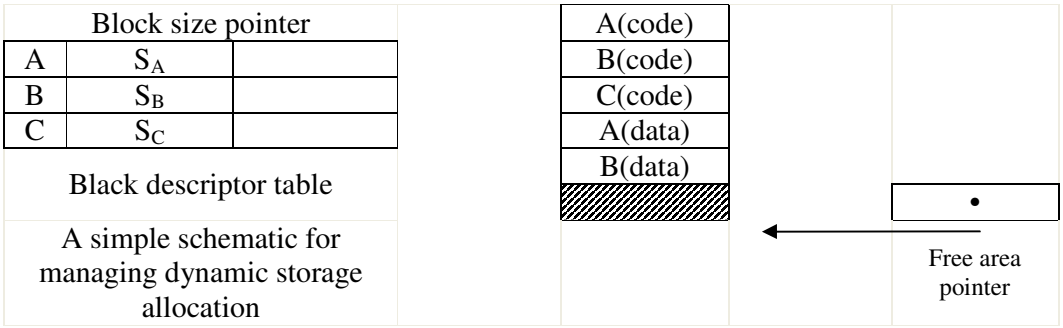
Ans:



Load	A		LOAD	C
ADD	B		SUB	D
STORE	TEMP		STORE	TEMP1
LOAD	C		LOAD	A
SUB	D		ADD	B
STORE	TEMP2		DIU	TEMP1
LOAD	TEMP1			
DIU	TEMP2			

Q.73 Draw a simple schematic for managing dynamic storage allocation. (8)

Ans:



A simple schematic for managing dynamic storage allocation

Q.74. Differentiate between synchronous and asynchronous input / output with the help of an example. (8)

Ans: The I/O operation is asynchronous input output operation because after the start of input/output, control is returned to the user program without waiting for the input/output to complete. The input/output continues and on its completion, an interrupt is generated by the controller to attain CPU's attention.
CPU execution waits, while I/O proceeds, in which case, there is a possibility that at most one request I/O request is outstanding at a time. This is known as synchronous I/O.

Q.75. List the major activities of an operating system with respect to memory management, **secondary storage management and process management.** (8)

Ans:
Operating system is responsible for following activities in connection with management of memory;
(i) Allocation and de allocation of memory as and when needed

(ii) Keeping track of used and unused memory space.
(iii) Deciding what process to be loaded into memory in case space becomes available.

For secondary space management:
(i) Swap space and free space management
(ii) Disk scheduling
(iii) Allocating space to the data and programs onto the secondary storage device.

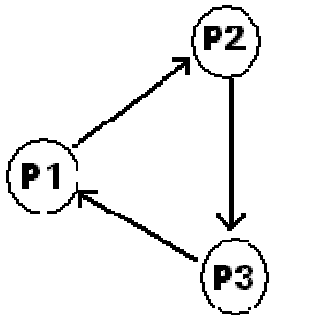
For process management:
(i) Creation, deletion of both user and system process.
(ii) Handling process synchronization.
(iii) Deadlock handling.

Q.76. What are the disadvantages of FCFS scheduling algorithm as compared to shortest job first (SJF) scheduling? (8)

Ans:
Disadvantages:
(i) Waiting time can be large if short requests wait behind the long ones.
(ii) It is not suitable for time sharing systems where it is important that each user should get the CPU for an equal amount of time interval.
(iii) A proper mix of jobs is needed to achieve good results from FCFS scheduling.

Q.77 Explain deadlock detection algorithm for single instance of each resource type. (8)

Ans:
(i) Maintain a wait: nodes in a graph represent process. If process i is waiting for resource hold by process j. Then there is an edge from I to j.



WAIT- FOR- GRAPHS

(ii) Periodically invokes an algorithm that searches for cycles in the graph. If there is a cycle in the wait-for-graph a dead lock is said to be exist in the system.

Q.78 Discuss the concept of segmentation? What is the main problem with segmentation? (8)

Ans:
Segmentation is techniques for the non contiguous storage allocation. It is different from paging as it supports user's view of his program.
Problem with segmentation
(i) Is with paging, this mapping requires two memory references per logical address, which slows down the computer system by a factor of two. Caching is the method used to solve this problem.

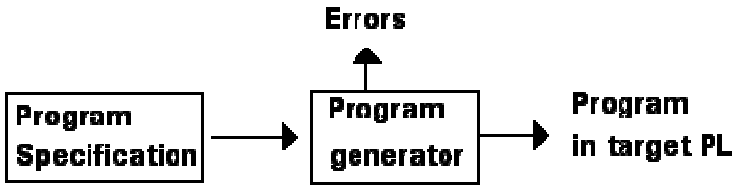
(ii) Problem of external fragmentation.

Q79. What is the difference between absolute and relative path name of a file? (8)

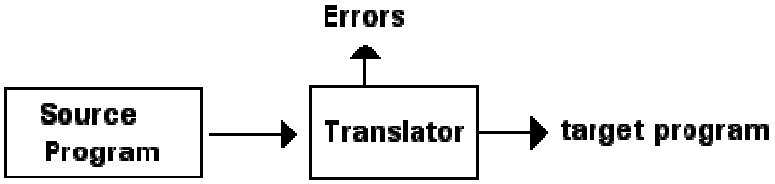
Ans:
Absolute path name:
It is listing of the directories and files from the root directory to the intended file.
Relative path name :
A user can specify a path particular directory as his current working directory and all the path names instead of being specified from the root directory are specified relative to the working directory.

Q.80. Describe language processing activities? **(8)**

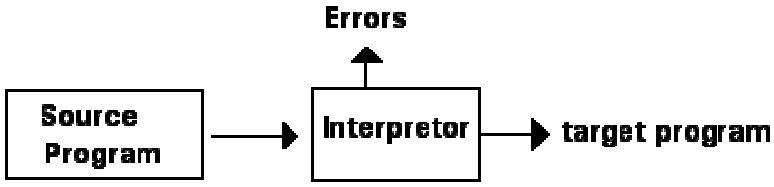
Ans:
There are two different types of language processing activities:
1. Program generation activities
2. Program execution activities
Program generation activities: A program generation activity aims at automatic generation of a program. The source language is a specification language of an application domain and the target language is typically a procedure oriented programming language. the following figure shows program generation activity



The program generator is a software system which accepts the specification of a program to be generated and generates a program in the target. PL. The program generator introduces a new domain between the application and PL domains. We call this the program generator domain. The specification gab is now gab between the application domain and the program generator domain. This gab is smaller than the gab between the application domain and PL domain.
Program execution activities: A program execution activity organizes system. Two model program executions are:
1. Translation 2. Interpretation
Translation: The program translation models bridges execution gap by translating a program written in a PL, called the source program into an equivalent program in the machine or assembly language of the computer system.



Interpretation: The interpreter reads the source program and stores it in its memory. During interpretation it takes a statement, determines its meaning and performs actions which implement it.



Q.81 Explain the criteria to classify data structures used for language processors? **(8)**

Ans:
The data structures used in language processing can be classified on the basis of the following criteria:
1. Nature of data structure (whether a linear or non-linear data structure)
2. Purpose of a data structure (whether a search data structure or an allocation data structure)
3. Life time of a data structure (whether used during language processing or during target program execution)
A linear data structure consists of a linear arrangement of elements in the memory. A linear data structure requires a contiguous area of memory for its elements. This poses a problem in situations where the size of a data structure is difficult to predict. the elements of non linear data structures are accessed using pointers. Hence the elements need not occupy contiguous area of memory.
Search Data structures are used during language processing to maintain attribute information concerning different entities in the source program. In this the entry for an entity is created only once, but may be searched for large number of times. Allocation data structures are characterized by the fact that the address of memory area allocated to an entity is known to the users. So no search operations are conducted.

Q.82 Explain macro definition, macro call and macro expansion? **(6)**

Ans :
A unit of specification for a program generation is called a macro. It consists of name, set of formal parameters and body of code. When a macro name is used with a set of actual parameters it is replaced by a code generated from its body. This code is called macro expansion. There are two types of expansions:
1. lexical expansion
2. Semantic expansion
Lexical expansion: It means a replacement of character string by another string during program generation. It is generally used to replace occurrences of formal parameters by corresponding actual ones.
Semantic Expansion: It implies generation of instructions build to the requirements of specific usage. It is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.
The macro definition is located at the beginning of the program is enclosed between a macro header and macro end statement.
A macro statement contains macro name and parameters

< macro name > { < parameters>}

A macro call: A macro is called by writing the macro name in the mnemonic field of an assembly statement.

Q.83. What are the advantages of code optimization? Explain optimizing transformations? **(10)**

Ans:

Code optimization aims at improving the execution efficiency of a program. This is achieved in two ways. Redundancies in a program are eliminated and computations in a program are rearranged to make it execute efficiently. The optimized program occupies 25 percent less storage and execute three times as fast as the unoptimized program.

Optimizing transformations:

An optimizing transformation is a rule for rewriting a segment of a program to improve its execution efficiency without affecting its meaning.

Commonly used optimizing transformations are

(i) Compile time evaluation: Execution efficiency can be improved by performing certain action specified in a program during compilation itself. Constant folding is the main optimization of this kind. When all operands in an operation are constants, the operation can be performed at compilation time.

For Example: a: =3.141557/2 can be replaced by a:=1.570785 eliminating a division operation.

(ii) Elimination of common subexpressions:

Common subexpressions are occurrences of expressions yielding the same value.

For Example: a:=b*c t: = b*c

λ: = b*c + 5.2 ⟹ a: = t

λ: = t + 5.2

(iii) **Dead code elimination:** Code which can be omitted from a program without affecting its results is called dead code. For example An assignment statement x: = < exp > constitutes dead code if the value assigned to x is not used in the program.

(iv)Frequency and strength reduction:

Execution time of a program can be reduced by moving code from a part of a programs which is executed very frequently to another part of the program which is executed fewer times.

The strength reductions replaces the occurrance of a time consuming operation by an occurrance of a faster operation.

Q.84 Explain the following terms

(i) Translated address
(ii) Linked address
(iii) Load address

Explain the relationship amongst these. **(6)**

Ans:

(i)Translated address: Address assigned by the translator

(ii)Linked address: Address assigned by the linker

(iii)Load time address: Address assigned by the loader.

While compiling a program P, a translator is given an original specification for P. This is called the translated origin of P. The translator uses the value of the translated origin to perform memory allocation for the symbols declared in P. This results in the assignment of a translated time address to each symbol in the program. The origin of a program may have to be changed by the linker or loader for one of the following reasons:

1. Object modules of library routines often have the same translated origin so memory allocation to such programs would conflict unless their origins are changed.

2. Operating system requires a program to be executed from a specific location so this may require a change in its origin.

Q.85 What are the functions of passes used in two-pass assembler? Explain pass-1 algorithm? **(10)**

Ans :
Two pass translation of an assembly language program can handle forward references early.
The following tasks are performed by the passes of a two pass assembler are as follows:
Pass I: (i) Separate the symbol, mnemonic opcode and operand fields
(ii) Build the symbol table
(iii) Perform LC processing
(iv) Construct intermediate representation.
Pass II: Synthesize the target program
Pass I uses the following data structures:
OPTAB : A table of mnemonic opcodes and related information
SYMTAB: symbol table
LITTAB: A table literally used in the program
OPTAB contains the fields mnemonic opcode, class and mnemonic information. The class field indicated whether the opcode corresponds to an imperative statement (IS), a declaration statement (DL) or an assembler directive (AD).
(SYMTAB entry contains the fields address and length. A LITTAB entry contains literals and address.)

Q.86 What data structure is used by an operating system to keep track of process information? Explain **(4)**

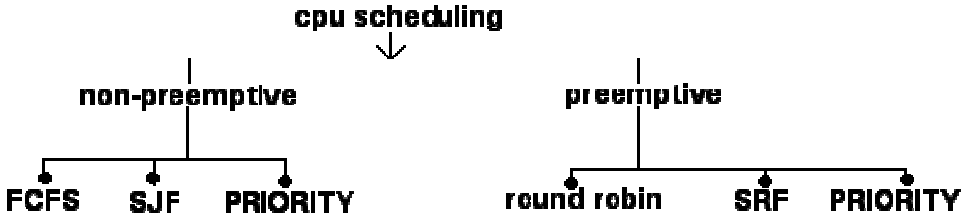
Ans:
A process is a program in execution. An operating system considers a process to be the fundamental unit for resource allocation. Following resources could be allocated to a process
(i) Memory (ii) Secondary memory (iii) I/O Devices (iv) files opened by the process (v) CPU time consumed by process
A data structure called process control block (PCB) is used by an OS to keep track of all information concerning a process. The PCB of a process contains the following information.

Process ID
Priority
Process state
PSR
Registers
Event information
Memory allocation
Resources held
PCB pointer

(i) Process scheduling information: This information consists of three fields process ID, priority, process state.
(ii) PSR and machine registers: These fields hold contents of the processor states register (PSR) and the machine registers when the execution of the process was last suspended.
(iii) Event information: when a process is in blocked state, this field contains information concerning the event for which the process is waiting.
(iv) Memory and resource information: This information is useful for the deallocating memory and resources when the process terminates.
(v) PCB pointer: It is a pointer to the next PCB in the process scheduling list.

Q.87. Categorize the CPU scheduling algorithms? Explain non-pre-emptive algorithms? **(4)**

Ans
The various **CPU scheduling algorithms** are classified as follows:



Non preemptive algorithms: In this method a job is given to CPU for execution as long as the job is non completed the CPU cannot be given to other processes.
There are three types of non preemptive algorithms.
(i) First-come-first-serve (FCFS):
This is simplest CPU scheduling algorithm . With this scheme, the process that requests the CPU at first is given to the CPU at first. The implementation of FCFS is easily managed by with a FIFO queue.
(ii) Shortest-job-first (SJF): This is also called SPN (shortest process next). In this the burst times of all the jobs which are waiting in the queue are compared. The job which is having the least CPU execution time will be given to the processor at first. In this turnaround time and waiting times are least. This also suffers with starvation. Indefinite waiting time is called as starvation. It is complex than FCFS.
(iii) Priority: In this algorithm every job is associated with CPU execution time, arrival time and the priority. Here the job which is having the higher priority will be given to the execution at first. This also suffers with starvation. And by using aging technique starvation effect may be reduced.

Q.88. Differentiate between Batch Operating System and Time Sharing Operating System? **(6)**

Ans
Batch operating systems: A batch is a sequence of jobs. This batch is submitted to batch processing operating systems, and output would appear some later time in the form of a program or as program error. To speed up processing similar jobs are batched together. The major task of batch operating systems is to transfer control automatically from one job to next. Here the operating is always in the memory.
(i) It is lack of interaction between user and job while executing
(ii) Turnaround time is more.
(iii) CPU is often idle, because of 1/0 devices are very slow.

Time sharing: Time sharing or multi tasking is a logical execution of multiprogramming. Multiple jobs are executed by the CPU switching between them. Here the computer system provides on line communication between the user and the system.

Here the CPU is never idle. Time shared operating system allows many users to share the computer simultaneously.

Time sharing systems requires some sort of memory management and protection.

Q.89. What is a Deadlock? Write an algorithm for deadlock detection. **(8)**

Ans.

Deadlock is a situation, in which processes never finish executing and system resources are tied up, preventing other jobs form starting.

A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes, thereby causing deadlock.

An algorithm for deadlock detection:

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if Allocation $_i$ $^1\neq$ 0, then
Finish[i] = false; otherwise, Finish[i] = true.

2. Find an index i such that both:

(a) Finish[i] == false

i. (b) Request$_i$ ≤ Work

If no such i exists, go to step 4.

3. Work = Work + Allocation$_i$
Finish[i] = true
go to step 2.

4. If Finish[i] == false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state. Moreover, if Finish[i] == false, then P$_i$ is deadlocked.

Q 90 Develop a regular expression for

(i) Integer
(ii) Real number
(iii) Real number with optional fraction
(iv) Identifier **(8)**

Ans :

A regular expression for

(i) integer is $[+ \mid -] (d)^+$
(ii) real number is $[+ \mid -] (d)^+ . (d)^+$
(iii) real number with optional fraction is $[+ \mid -] (d)^+ . (d)^*$
(iv) identifier is $l(l \mid d)^*$

Q.91. What is critical section problem? Give two solutions for critical section problem? **(8)**

Ans :

A race condition on a data item arises when many processes concurrently update its value data consistency, requires that only one process should update the value of a data item at any time. This ensured through the notion of a critical section.

A critical section for a dataitem d is a section of code, which cannot be executed concurrently with itself or with other critical section(s) for d.
Consider a system of n processes (P_0, P, .P_{n-1}).
Each process has a segment of code called a critical section, in which the process may be changing common variables, updating a table, waiting a file and so on. Important feature of the system is, when one process is executing in its critical section, no other process is to be allowed to execute its critical section. Thus the execution of critical sections by the processes is mutually exclusive in time.
A solution to the critical section problem must specify the following requirements.
(i) Mutual exclusion (ii) Progress (iii) Bounded waiting
One solution for critical section problem is provided by semaphores.
Another solution for critical section problems is by Moniters

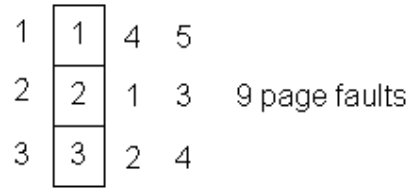
Q.92. Explain difference between Security and Protection? Describe the scheme of capability lists to implement protection? **(8)**

Ans :
Protection mechanism: The following mechanisms are commonly used for protecting files containing programs and data.
(i) Access controls lists (ACL's)
(ii) Capability lists (C- lists)
These lists are used to ensure that uses only access files which are explicitly authorized access. These files include
(i) files created by a user himself/herself
(ii) files owned by others, for which a user process explicit access privileged granted by other owners.
Security mechanisms: Authentication is the primary security mechanism. Authentication is the act of verifying the identity of a user. Authentication is typically performed through passwords at login time. The system stores the password information in a system as set as pair of the form (user id, password info)
The password information is protected by encryption
C-list: A capability is a file access privilege concerning capabilities possessed by a user is stored in a capability list. A C-list is a set of pairs{ (file.id, access privileges),…..}
C-lists are usually small in size. This limits the space and time over heads in using them to control file accesses. A c-list is a token representing certain access privileges for an object. An object is any hardware or software entity in the system. A capability possessed by a process. A process possessing a capability for an object can access the object in a manner consistent with the access privileges described in the capability. Thus maintaining C-lists provide:
1. A uniform addressing mechanism for long and short life objects
2. It does not explicitly associate memory with processes. It associates C-lists with processes.
3. A process may access objects existing anywhere in the system.

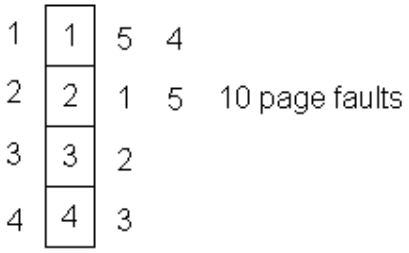
Q.93. Explain with the help of examples FIFO and LRU page replacement algorithms? **(10)**

Ans :
FIFO policy: This policy simply removes pages in the order they arrived in the main memory. Using this policy we simply remove a page based on the time of its arrival in the memory.

For example if we have the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and 3 frames (3 pages can be in memory at a time per process) then we have 9 page faults as shown



If frames are increased say to 4, then number of page faults also increases, to 10 in this case.



LRU policy: LRU expands to least recently use. This policy suggests that we remove a page whose last usage is farthest from current time.

Q.94. Describe the necessary conditions for Deadlock. **(8)**

Ans:

Necessary conditions for deadlock
1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

Mutual exclusion: The mutual exclusion condition must hold for non sharable resources. For example, a printer cannot be simultaneously shared by several processes. Shared resources on the other hand, do not require mutually exclusive access, and thus cannot be involved in the deadlock. In general, however it is not possible to prevent deadlocks by denying the mutual exclusion condition.

Hold and wait: To ensure that the hold and wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources. A process may request some resources and use them before it can request any other resources, however it must release all the resources that it is currently allocated.

No preemption: The third necessary condition is that there be no preemption of resources that have already been allocated. If a process that is holding some resources requests another resource that it cannot be immediately allocated to it. Then all resources currently being held are preempted.

Circular Wait: One way to ensure that the circular wait condition never holds is to impose a total ordering of all resources types, and to ensure that each process requests resources in an increasing order of enumeration.

Q.95 What is a Process Scheduling? Explain the different sub-functions of Process Scheduling. **(8)**

Ans:

Scheduling is a key part of the workload management software which usually perform some or all of:

- Queuing
- Scheduling
- Monitoring
- Resource management
- Accounting

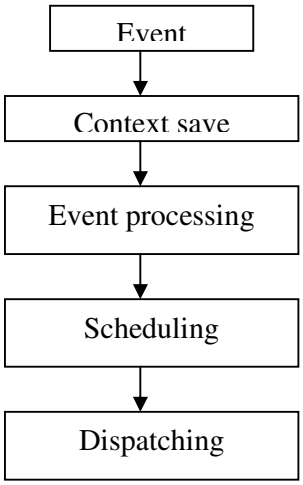
The difficult part of scheduling is to balance policy enforcement with resource optimization in order to pick the *best* job to run. Essentially one can think of the scheduler performing the following loop:

- Select the *best* job to run, according to policy and available resources.
- Start the job.
- Stop the job and/or clean up after completion.
- repeat.

Process scheduling consists of the following sub-functions:

1. Scheduling: Selects the process to be executed next on the CPU. The scheduling function uses information from the PCB's and selects a process based on the scheduling policy in force.
2. Dispatching: Sets up execution of the selected process on the CPU. This function involves setting up the execution environment of the selected process, and loading information from the PSR and registers fields of the PCB into the CPU.
3. Context save: Saves the status of a running process when its execution is to be suspended. This function performs housekeeping whenever a process releases the CPU or is pre-empted.

The following diagram illustrates the use of scheduling sub functions: Occurrence of an event invokes the context save function. The kernel now processes the event that has occurred. The scheduling function is now invoked to select a process for execution on the CPU. The dispatching function arranges execution of the selected function on the CPU.



Q.96. Describe the essential properties of the following operating systems

Real Time and Distributed Operating System **(8)**

Ans:

Real time operating system:

1. Time constraint result
2. Priority driven or deadline oriented scheduling
3. Programmer defined interrupts
4. Hard Real time and soft real time system
5. Multitasking
6. Event driven
7. Embedded systems
8. Robotics

Distributed operating systems:

1. Resource sharing
2. Computation speed up
3. Reliability
4. Communication

Q.97. Describe Data structures used during passes of assembler and their use. **(10)**

Ans:

Data structure during passes of assembler and their use.

Pass 1 data base

1. Input source program
2. A location counter (LC)
3. A table, the machine-operation table (MOT), that indicates the symbolic mnemonic for each instruction and its length.
4. Pseudo- operation table
5. Symbol table
6. Literal table
7. Copy of the input to be used later by pass 2

Pass 2

1. Copy of source program input to pass 1
2. Location counter (LC)
3. MOT
4. POT
5. ST
6. Base table that indicates which registers are currently specified as base register.
7. A work space, INST, that's used to hold instruction as its various parts are being assembled together
8. Punch line, used to produce a printed listing.
9. Punch card for converting assembled instructions into the format needed by the loader.

Q.98. what is parsing and specify the goals of parsing **(6)**

Ans:

Source programmed statements are regarded as tokens, building block of language the task of scanning the source statement, recognizing and classifying the various tokens is known as lexical analysis. The part of the compiler that performs this task is commonly called a scanner.

After the token scan, each statement in the program must be recognized as some language constructs, such as declaration or an assignment statement described by the grammar.

This process is called Syntactic analysis or parsing is performed by the part of compiler called parser.

Goals:

1. to check the validity of source string
2. to determine the syntactic structure of a source string.

For invalid string it reports error, for a valid string it builds a parse tree to reflect the sequences of derivations or reductions performed during parsing.

Q.99. Write short note on code optimization **(8)**

Ans:

Code optimization is the optional phase designed to improve the intermediate code so that the

Ultimate object program runs faster or takes less space. Code optimization in compilers aims at improving the execution efficiency of a program by eliminating redundancies and by rearranging the computations in the program without affecting the real meaning of the program.

Scope – First optimization seeks to improve a program rather than the algorithm used in the program. Thus replacement of algorithm by a more efficient algorithm is beyond the scope of optimization. Also efficient code generation for a specific target machine also lies outside its scope.

The structure of program and the manner in which data is defined and used in it provide vital clues for optimization.

Optimization transformations are classified into local and global transformations.

Q.100. Explain the Features of Major scheduling algorithms **(8)**

Ans:

1. FCFS – First come first served scheduling
2. Shortest job – First scheduling
3. Priority scheduling
4. Round robin scheduling

FCFS-

1. Process that request the CPU first is allocated CPU first
2. Managed by FIFO queue.
3. Average waiting time is generally long
4. Non preemptive
5. Troublesome for time sharing systems

Shortest job first

1. The process which has the smallest CPU burst time gets first
2. It increases the waiting time of long processes.
3. Problem – difficult to predict the length of next CPU request
4. May be preemptive or non preemptive

Priority

1. Highest priority process gets CPU allocation first
2. The larger the CPU burst, lower the priority
3. Priority can be defined internally or externally
4. Can be preemptive or non-preemptive
5. Problem-starvation, blocking of process
6. Solution – aging – increases the priority

Round robin

1. Time quantum from 10 100 millisecond is defined
2. Processes are considered in circular queue
3. CPU allocation is divided among processes accordingly
4. Performance depends heavily on time quantum
5. Preemptive

Q.101. List the criteria on the basis of which data structures used in language processing can be classified. **(3)**

Ans:
The data structures used in language processing can be classified on the following criterion:

1. **Nature of a data structure-**whether a linear or nonlinear data structure. A linear data structure consists of a linear arrangement of elements in memory and elements require a contiguous area of memory. The elements of a non-linear data structure are accessed using pointers and hence the elements need not occupy contiguous areas of memory.
2. **Purpose of a data structure-**whether a search data structure or an allocation data structure. Search data structures are used during language processing to maintain attribute information concerning different entities in the source program. Allocation data structures are characterized by the fact that the user of that entity knows the address of the memory area allocated to an entity thus no search operations are conducted on them.
3. **Lifetime of a data structure-**whether used during language processing or during target program execution.

Q.102. Differentiate between logical address and physical address. **(4)**

Ans:
A logical address is the address of the instruction or data word as used by a program (this includes the use of index, base or segment register).
A physical address is the effective memory address of an instruction or data word. The set of physical addresses generated during operation of system constitute the **physical address** space of the system. The compile time and the load time address binding schemes result in environment where the logical and physical address are same, whereas during execution time addresses differ.

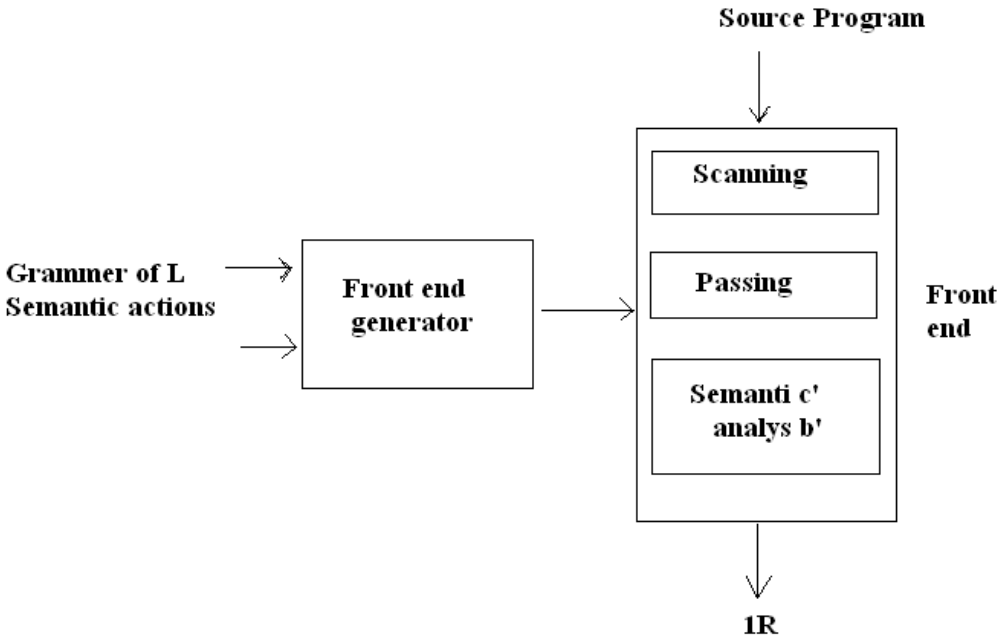
Q103. What are Language Processor Development Tools (LPDTs)? Explain through schematic diagram. Name the widely used LPDTs. **(4)**

Ans:
Language processor development tools (LPDTs) focuses on generation of the analysis phase of language processors. The LPDT requires the following two inputs:

1. Specification of a grammar of language L
2. Specification of semantic actions to be performed in the analysis phase.

It generates programs that perform lexical, syntax and semantic analysis of the source program and construct the IR. These programs collectively form the analysis phase of the language

processor.



Widely used LPDT's are:

Lex - A Lexical Analyzer Generator

Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. It is well suited for editor-script type transformations and for segmenting input in preparation for a parsing routine.

Lex source is a table of regular expressions and corresponding program fragments. The table is translated to a program, which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions. As each such string is recognized the corresponding program fragment is executed. The recognition of the expressions is performed by a deterministic finite automaton generated by Lex. The program fragments written by the user are executed in the order in which the corresponding regular expressions occur in the input stream.

YACC: Yet another Compiler-Compiler

Computer program input generally has some structure; in fact, every computer program that does input can be thought of as defining an ``input language" which it accepts. An input language may be as complex as a programming language, or as simple as a sequence of numbers. Unfortunately, usual input facilities are limited, difficult to use, and often are lax about checking their inputs for validity.

YACC provides a general tool for describing the input to a computer program. The YACC user specifies the structures of his input, together with code to be invoked as each such structure is recognized. YACC turns such a specification into a subroutine that han- dles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine.

Q.104. Explain the following term

(i) Overlays **(4)**

(ii) Macro definition and call **(4)**

Ans:

(i) An overlay is a part of program which has the same load origin as some other part of the program. Overlays are used to reduce the main memory requirement of a program. When the problem arises where the size of program exceeds the memory size, then overlays are used. Structure of such programs consists of a **permanently resident portion, called the root.**

A set of overlays

To start with, the root is loaded in memory and given control for the purpose of execution. Other overlays are overwrites a previously loaded overlay with same load origin, this reduces memory requirement of a program. It is also possible to execute programs whose size exceeds the memory size.

(ii) Macro: The assembly language programming often finds it necessary to repeat certain piece of code many times during the course of program. In such situations we find macro facility useful. Whole process consists of three steps:

1. Macro definition instruction
2. macro call
3. macro expansion

A macro definition is enclosed between a macro header statement and a macro end statement. Macro definitions are typically located at the start of a program. It consists of

1. A macro prototype statement
2. One or more model statements
3. Macro preprocessor

A macro call is called by writing the macro name in the mnemonics field of an assembly statement. Syntax for the same is

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

A macro call leads to macro expansion, during macro call expansion the macro call statement is replaced by the sequence of assembly statements.

Q.105. Can the operand expression in an ORG statement contain forward references? If so, outline how the statement can be processed in a two-pass assembly scheme. (9)

Ans: (ORG (origin) is an assembler directive that

- Indirectly assign values to symbols
- Reset the location counter to the specified value-ORG value
- Value can be: constant, other symbol, expression
- No forward reference

Assemblers scan the source program, generating machine instructions. Sometimes, the assembler reaches a reference to a variable, which has not yet been defined. This is referred to as a **forward reference** problem. It is resolved in a **two-pass assembler** as follows:

On the first pass, the assembler simply reads the source file, counting up the number of locations that each instruction will take, and builds a **symbol table** in memory that lists all the defined variables cross-referenced to their associated memory address. On the second pass, the assembler substitutes opcodes for the mnemonics,

and variable names are replaced by the memory locations obtained from the symbol table.

Consider the following program

```
          ORG   $800
N:        DS    1
M:        DS    1
COLUMN:   DS    50
ODD:      DS    50
*
          CLR   M              ; initialize M
          LDAB  N              ; Put N into B
          LDX   #COLUMN        ; Point X to COLUMN
          LDY   #ODD           ; Point Y to ODD
LOOP:     LDAA  1,X+           ; Next number of COLUMN into A
          BPL   JUMP           ; Go to next number if positive
          BITA  #1             ; Z = 1 if, and only if, A is even
          BEQ   JUMP           ; Go to next number if even
          STAA  1,Y+           ; Store odd, negative number
          INC   M              ; Increment length of ODD
JUMP:     DBNE  B,LOOP         ; Decrement counter; loop if not done
          BGND                 ; Halt
```

On the first pass, the ORG statement sets the location counter to $800. Thus the label N has the value $800, the label M has the value $801, the label COLUMN has the value $802, and the label ODD has the value $834. The instruction CLR M will take three bytes , the instruction LDAB N will take three bytes, and so forth. Similarly, we see that the first byte of instruction we see that the first byte of instruction

```
LOOP:   LDAA  1,X+
```

will be at location $872. Thus the symbolic address LOOP has the value $872 and so on. Continuing in this way, we come to

```
BPL   JUMP
```

This program searches the array COLUMN looking for odd, negative, one-byte numbers, which then are stored in array ODD. The length of COLUMN is N and the length of ODD is M, which the program calculates. We do not know the second byte of this instruction because we do not know the value of the address JUMP yet. (This is called a forward reference, using a label whose value is not yet known.) However, we can leave this second byte undetermined and proceed until we see that the machine code for DBNE is put into location $87f, thus giving JUMP the value $87f. As we continue our first pass downward, we allocate three bytes for DBNE B,LOQP.

We do not find this instruction's offset yet, even though we already know the value of LOOP.

Scanning through the program again, which is the second pass, we can fill in all the bytes, including those not determined the first time through, for the instructions BPL JUMP, BEQ JUMP, and DBNE B,LOOP. At this time, all object code can be generated.

Q.106. What criteria should be adopted for choosing type of file organization **(8)**

Ans:
Choosing a file organization is a design decision, hence it must be done having in mind the achievement of good performance with respect to the most likely usage of the file. The criteria usually considered important are:
1. fast access to single record or collection of related records
2. Easy record adding/update/removal, without disrupting (1)
3. Storage efficiency
4. Redundancy as a warranty against data corruption.

Needless to say, these requirements are in contrast with each other for all but the most trivial situations, and it's the designer job to find a good compromise among them, yielding an adequate solution to the problem at hand. For example, easiness of adding/ etc. is not an issue when defining the data organization of CD ROM product, whereas fast access is given the huge amount of data that this media can store. However as it will become apparent shortly, fast access techniques are based on the use additional information about the records, which in turn competes with the high volumes of data to be stored.
Logical data Organization is indeed the subject of whole shelves of books in the "Database" section of your library. Here we'll briefly address some of the simpler used techniques, mainly because of their relevance to data management from the lower-level (with respect to a database's) point of view of an OS. Five organization models will be considered:

(i) Pile.
(ii) Sequential.
(iii) indexed-sequential.
(iv)Indexed
(v) Hashed.

Q.107 Compare pre-emptive and non-preemptive scheduling policies. (4)

Ans:
In preemptive scheduling we preempt the currently executing process. In non-preemptive we allow the current process to finish its CPU burst time.

Q.108. Explain the differences between:

(i) Logical and physical address space.
(ii) Internal and external fragmentation.
(iii)Paging and segmentation. (6)

Ans:

Logical Vs physical address space

(1) An address generated by the CPU is commonly referred to as a logical a logical address. The set of all logical addresses generated by a program is known as logical address space. Whereas, an address seen by the memory unit- that is, the one loaded into the memory-address register of the memory- is commonly referred to as physical address. The set of all physical addresses corresponding to the logical addresses is known as physical address space.

(2) The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, in the execution-time address-binding scheme, the logical and physical-address spaces differ.

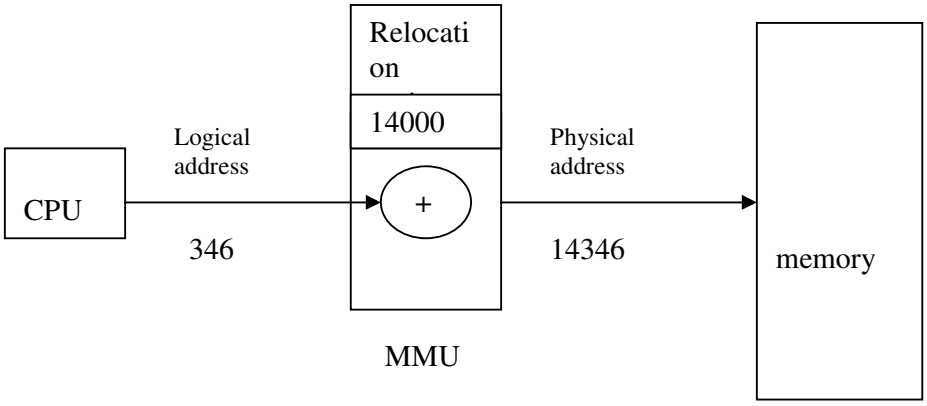
(3) The user program never sees the physical addresses. The program creates a pointer to a logical address, say 346, stores it in memory, manipulate it, compares it to other logical addresses- all as the number 346.

Only when a logical address is used as memory address, it is relocated relative to the base/relocation register. The memory-mapping hardware device called the memory-management unit(MMU) converts logical addresses into physical addresses.

(4) Logical addresses range from 0 to max. User program that generates logical address thinks that the process runs in locations 0 to max.

Logical addresses must be mapped to physical addresses before they are used. Physical addresses range from (R+0) to (R + max) for a base/relocation register value R.

(5) Example:



Mapping from logical to physical addresses using memory management unit (MMU) and relocation/base register

The value in relocation/base register is added to every logical address generated by a user process, at the time it is sent to memory, to generate corresponding physical address.

In the above figure, base/ relocation value is 14000, then an attempt by the user to access the location 346 is mapped to 14346.

(ii) Internal and external fragmentation

(1) When memory allocated to a process is slightly larger than the requested memory, space at the end of a partition is unused and wasted. This wasted space within a partition is called as internal fragmentation. When enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes. This wasted space not allocated to any partition is called external fragmentation.

(2) Internal fragmentation is found in multiple fixed partition schemes where all the partitions are of the same size. That is, physical memory is broken into fixed-sized blocks.
External fragmentation is found in multiple variable partition schemes. Instead of dividing memory into a fixed set of partitions, an operating system can choose to allocate to a process the exact amount of unused memory space it requires.
(3) In multiple fixed partition scheme, the partition table needs to store either the starting address for each process or the number of the partition allocated to each process.
In multiple variable partition scheme, the overhead of managing more data increases. The partition table must store exact starting and ending location of each process and data about which memory locations are free must be maintained.
(4) In multiple fixed partition schemes, size/limit register is set at boot time and contains the partition size. Each time a process is allocated control of CPU, the operating system only needs to reset the relocation register. In multiple variable partition schemes, each time a different process is given control of the CPU, the operating system must reset the size/limit register in addition to the relocation register. The operating system must also make decisions on which partition it should allocate to a process.
(5) Internal fragmentation can be reduced using multiple variable partition method. However, this solution suffers from external fragmentation. External fragmentation can be solved using compaction where the goal is to shuffle the memory contents to place all free memory together in one large block. Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non contiguous. This solution is achieved by paging and segmentation.

(iii) Paging and segmentation

Paging	Segmentation
Computer memory is divided into small partitions that are all the same size and referred to as, page frames. Then when a process is loaded it gets divided into pages which are the same size as those previous frames. The process pages are then loaded into the frames.	Memory-management scheme that supports user view of memory. Computer memory is allocated in various sizes (segments) depending on the need for address space by the process.
Address generated by CPU is divided into: *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory. *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.	Logical address consists of a two tuple: <segment-number, offset>
Transparent to programmer (system allocates memory)	Involves programmer (allocates memory to specific function inside code)
No separate protection	Separate protection
No separate compiling	Separate compiling
No shared code	Share code

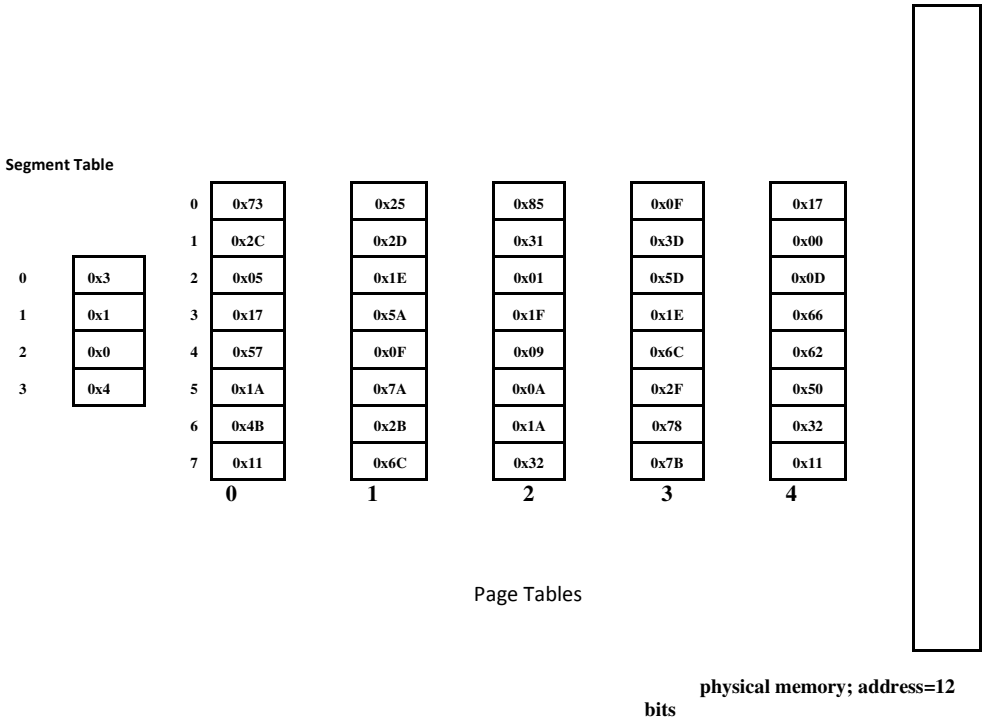
Q.109. Suppose that a process scheduling algorithm favors those processes that have used the least processor time in the recent past. Why will this algorithm favour /IO-bound processes, but not starve CPU-bound processes? (6)

Ans: It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

Q.110 List one advantage and one disadvantage of having large block size. (4)

Ans: The advantage of using a large block of memory is accommodation of maximum processes resulting is less number of page faults.
The disadvantage of using a large block of memory is occurrence of internal fragmentation i.e allocated memory may be slightly larger than requested memory.
e.g., suppose memory is allocated in blocks of 4K, a 1K process will waste 3K of space in its partition, as will a 5K process.

Q.111. Consider the following segmented paging memory system. There are 4 segments for the given process and a total of 5 page tables in the entire system. Each page table has a total of 8 entries. The physical memory requires 12 bits to address it; there are a total of 128 frames.



(i) How many bytes are contained within the physical memory?
(ii) How large is the virtual address?
(iii) What is the physical address that corresponds to virtual address 0x312?
(iv) What is the physical address that corresponds to virtual address 0x1E9? (8)

Ans:

(i) Number of bytes in physical memory is equal to 2 ^(7 + 7) = 16K bytes. This is because 12 bits are required to address physical memory location out of which 3 bits are to refer frame no. within page table + 2 bits to locate page from segment and remaining 7 bits for offset with frame.
(ii) The size of virtual memory is 2 ^ 20 (20 = 2 for segment index, + 3 for page table index + 3 for frame index in page table + 7 for frame number and 5 for offset within frame).
(iii) 312 (Hex) = 001100010010 = 00(segment) in table number 3 (refer to the data in question at entry 0 in segment table then find page 3, in page 3 find 110 (6th entry) which is 78; that is 120 th frame then the offset with the frame is given by the last 7 bits 0010010.
(iv) 312 (Hex) = 001100010010 = 00(segment) in table number 3 (refer to the data in question at entry 0 in segment table then find page 3, in page 3 find 110 (6th entry) which is 78; that is 120 th frame then the offset with the frame is given by the last 7 bits 0010010.

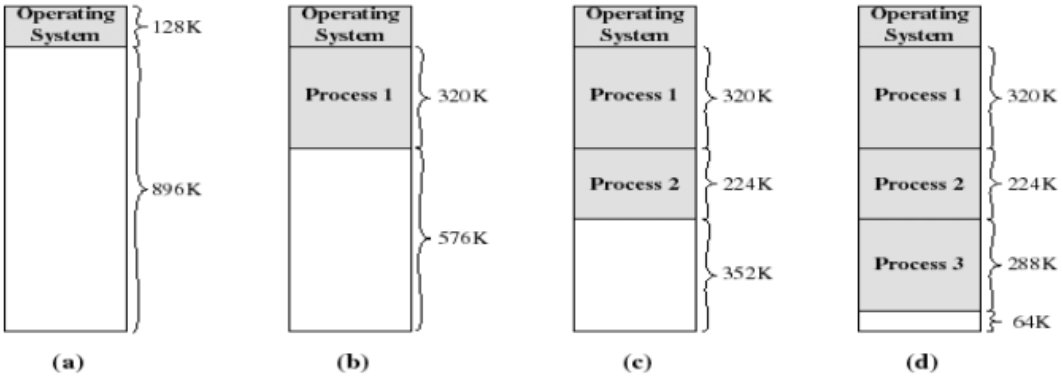
Q 112. Explain analysis and synthesis phase of a compiler. (7)

Ans:
The analysis and synthesis phases of a compiler are:
Analysis Phase: Breaks the source program into constituent pieces and creates intermediate representation. The analysis part can be divided along the following phases:
1. **Lexical Analysis**- The program is considered as a unique sequence of characters. The Lexical Analyzer reads the program from left-to-right and sequence of characters is grouped into **tokens**–lexical units with a collective meaning.
2. **Syntax Analysis**- The **Syntactic Analysis** is also called **Parsing**. Tokens are grouped into grammatical phrases represented by a **Parse Tree, which** gives a hierarchical structure to the source program.
3. **Semantic Analysis**- The **Semantic Analysis** phase checks the program for semantic errors (**Type Checking**) and gathers type information for the successive phases. **Type Checking** check types of operands; No real number as index for array; etc.
Synthesis Phase: Generates the target program from the intermediate representation. The synthesis part can be divided along the following phases:
1. **Intermediate Code Generator**- An intermediate code is generated as a program for an abstract machine. The intermediate code should be easy to translate into the target program.
2. **Code Optimizer**- This phase attempts to improve the intermediate code so that faster-running machine code can be obtained. Different compilers adopt different optimization techniques.
3. **Code Generator**- This phase generates the target code consisting of assembly code. Here
1. Memory locations are selected for each variable;
2. Instructions are translated into a sequence of assembly instructions;
3. Variables and intermediate results are assigned to memory registers.

Q.113. Explain the concept of variable-partition contiguous storage allocation. (6)

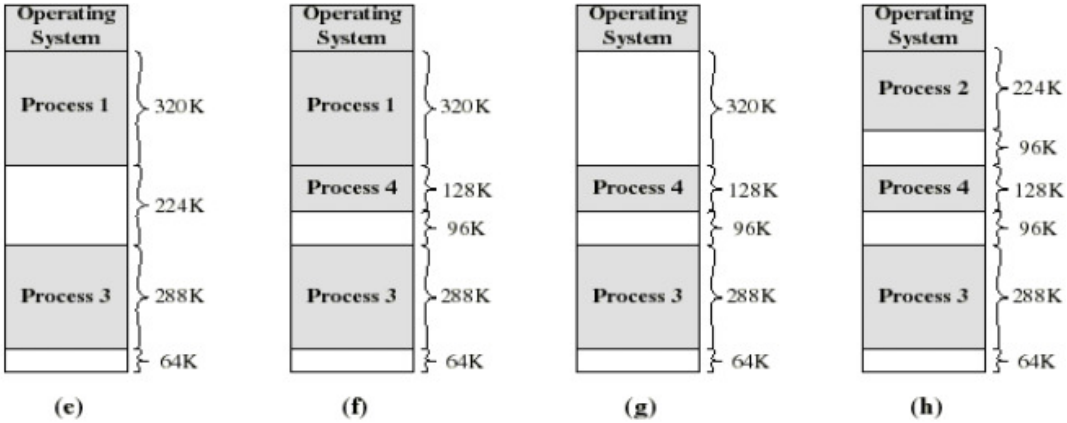
Ans:

Assume that we have 1024K main memory available in which 128K is occupied by operating system program. There are 4 jobs waiting for memory allocation in a job queue Applying FCFS scheduling policy, Process 1, Process 2 and Process 3 can be immediately allocated in memory. Process 4 cannot be accommodated because there is there is not enough space.



•A hole of 64K is left after loading 3 processes: not enough room for another process.

•Eventually each process is blocked. The OS swaps out process 2 to bring in process 4.



•Another hole of 96K is created.

•Eventually each process is blocked. The OS swaps out process 1 to bring in again process 2 and another hole of 96K is created.

Q.114. Suppose two processes enter the ready queue with the following properties:
Process 1 has a total of 8 units of work to perform, but after every 2 units of work, it must perform 1 unit of I/O (so the minimum completion time of this process is 12 units). Assume that there is no work to be done following the last I/O operation.
Process 2 has a total of 20 units of work to perform. This process arrives just behind P1. Show the resulting schedule for the shortest-job-first (preemptive) and the round-robin algorithms. Assume a time slice of 4 units of RR. What is the completion time of each process under each algorithm? (8)

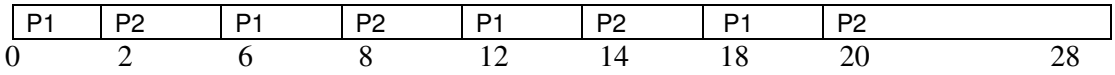
Ans:
S-J-F(pre emtive):

P1	P2	P1	P2	P1	P2	P1	P2	P2

0 2 3 5 6 8 9 11 12 28

Between 2-3unit of time, p2 is in ready queue and P1 has gone for I /O. So P2 should be executed as it is pre-emptive SJF algorithm.
Completion time of process p1 = 12 unit
Completion time of process p2 = 28 unit

Round- robin (R R) algorithm



Completion time of process p1 = 20 unit
Completion time of process p2 = 28 unit

Q.115. What are the relocation requirements in segmented addressing? (8)

Ans:
The relocation requirements of a program are influenced by the addressing structure of the computer system on which it is to execute. Use of the segmented addressing structure reduces the relocation requirements of a program
Consider the following assembly program of 8088. The ASSUME statement declares the Segment registers CS and DS to be available for memory addressing.

S.No.		Statement		offset
0001	DATA-HERE	SEGMENT		
0002	ABC	DW	25 0000	
0003	B	DW?	0002	
.				
.				
.				
0012	SAMPLE	SEGMENT		
0013		ASSUME	CS: SAMPLE, DS: DATA-HERE	
0014		MOV	AX, DATA-HERE	0000
0015		MOV	DS, AX	0003
0016		JMP	A	0005
0017		MOV	AL, B	0008
.		.		
.		.		
.		.		
0027	A	MOV	AX, BX	0196
.				
.				
.				
0043	SAMPLE	ENDS		
0044		END		

Translation time address of A is 0196. At S.NO 16, a reference to A is assembled as a displacement of 196 from the contents of the CS register. This avoids the use of an absolute address. Now no relocation is needed if segment SAMPLE is to be loaded in the memory starting at the address 2000 because CS register would be loaded with the address by a calling program (or by the OS). The effective operand address would be calculated as <CS> + 0196 = 2196. A similar situation for B in S.NO. 17. The reference to B is assembled as a displacement of 0002 from the contents of the DS register. Since the DS register would be loaded with the execution time address of DATA –HERE, the reference to B would be automatically relocated to the correct address.