TRUE

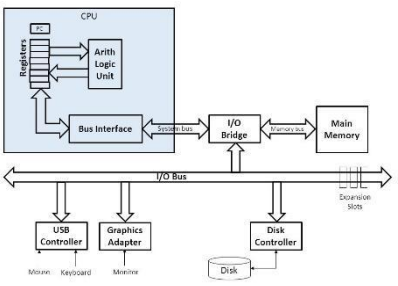
<> in sql is ≠

1. SCAN Disc Scheduling == Elevator [[Scan: end tak, look: jaha tak data hai]]
2. Necessary condition: इतना तो होना ही चाहिए [[All true satisfies it, satisfying it may always not be TRUE]]
3. Necessary and sufficient: Agar satisfy kare toh always TRUE
4. From(X) □ Where(σ/condition/for all tuples) □ GroupBy □ Having(for groupby groups) □ Select (π/projection) □ Distinct □ Orderby (Sorting)
5. A page table must fit in a memory frame
6. AVL tree is height balanced tree with height difference of 1
7. B/B+ tree is height balanced tree with ***height difference of 0***
8. ***A given CFL is regular is undecidable***
   {Ex: 'If L is CFL $\bar{L}$ is CFL'; means we know $\bar{L}$ is CSL, so asking is CSL a CFL: Undecidable}
9. ***A given CFG is finite ≠ A given CFG is Regular {The before one is finiteness problem of CFG}***
10. ***In TM no. of steps =33(decidable); no of states =33 (undecidable), may get stuck on some state=24***
11. ***L={ww} is CSL; $\bar{L}$ is CFL***
12. ***L={<M>|M accepts Σ*; <M>|M accepts Φ/ε} == {decidability problem of completeness; emptiness; emptiness(ε)}***
13. ***Rice theorem ::*** If potential to gets stuck on some state q (may not halt on q): Undecidable problem (possible in not-halting TM) :: State Entry problem:: We cannot say for sure a given string enters that state
14. Decidable□ TM halts, it is definitely a REC language
15. Undecidable□ TM may halt for some strings and it may not halt for some strings (partial decidable) (+ve condition) ex (string of length 33, may accept some, may halt for some) (Turing machines which don't halt on Σ*) □ RE
16. Undecidable□ TM may not halt for any string (completely undecidable) (-ve condition) ex (Turing machines which don't halt on the empty input) □ Not RE
17. If does not halt + positive condition == Recursive Enumerable (RE)
18. If does not halt - negative condition ==Not Recursive Enumerable (Not-RE)
19. ***CAUTION== Not RE ≠ Not recursive (Not recursive = RE, not RE)***
20. ***L={<M>|L(M) is not recursive; given M is a TM} is undecidable. □ for a non-halting TM, membership does not hold, so we cannot say the given lang is RE or not-RE using a given TM***
21. ***Every infinite regular language contains an undecidable language as a subset*** {Assume a RE lang as a part of inf regular language, now we have non-halting TM, so there may be some state q where our TM is stuck, hence lang may contain undecidable lang}
22. GroupBy/Aggregate functions/Orderby/duplicate/insert/update □ Not possible in TRC/RA
23. Expressive power:: Basic RA == Safe TRC== Safe DRC ≠ SQL
24. {t|~(t ∈ R1)} says that get me all integers that do not belong to R1, can be infinite (Unsafe TRC)
25. Locking at the innermost level provides highest degree of concurrency. (Sabse andar wale darwaze mein taala, more people will be able to reach there, if tala at main door, only one person with chabi will enter at a time)
26. The number of relations that are both reflexive and asymmetric is 0
27. The number of relations that are both symmetric and asymmetric is 1 (i.e. null relation)
28. If R and S are equivalence relations then R ∩ S is also an equivalence relation
29. If R and S are equivalence relations then R U S is ***not*** an equivalence relation
30. ***A but not B = (A-B) = A-(A∩B) = A∩$\bar{B}$***
31. ***Regular – CFL □ Reg ∩ $\overline{CFL}$ □ R ∩ CSL = CSL***
32. ***CFL – Regular □ CFL ∩ $R\bar{E}\bar{G}$ □ CFL ∩ Reg = CFL***
33. ***L= {a*} ∪ {$a^n b^n$} then it is same as {Regular} ∪ {CFL} == CFL <<**imp, do this way**>>***
34. ***If A is reducible to b then □ A <$_p$ B ; If A is decidable, B may/maynot be; If A undecidable, B is surely undecidable (B is polynomially bigger problem than A)***
35. No. of strings over Σ*: Countable
36. No. of languages over Σ*: Uncountable
37. No of Non-regular languages over (0/1)*: Uncountable
38. No of Recursive Enumerable languages over (0/1)*: Countable
39. No of Recursive Enumerable languages: Countable
40. Pascal, Fortran, and C are Context ***Sensitive*** Languages (CSL)
41. Regular expressions are used in lexical analysis.
42. Pushdown automata is related to context free grammar which is related to syntax analysis.
43. Dataflow analysis is done in code optimization {DAG}
44. If the given schedule is conflict serializable, then it is surely recoverable

Halting TM: REC
Turing REcognisable: RE
Turing DECidable: REC

Type 0, and Unrestricted Grammar RE (last wala)

Overflow flag
"$Carry_{in} \oplus Carry_{out}$"



45. ***2PL and strict 2PL are sterilisable, strict 2PL in guaranteed recoverable, both are not free of deadlock and starvation.***
46. ***2PL: All locks before all unlocks, strict 2PL: exclusive lock unlock after commit, rigorous 2PL: All unlock after commit***
47. Register allocation is done in code generation.
48. Every left-recursive grammar can be converted to a right-recursive grammar and vice-versa
49. The language generated by a context-free grammar all of whose productions are of the form X --> w or X --> wY (where, w is a string of terminals and Y is a non-terminal), is always regular
50. The derivation trees of strings generated by a context-free grammar in Chomsky Normal Form are always binary trees (Chomsky Normal Form= A □a; A□AB)
51. There are two approaches by which PDA accepts the input:
    a. Acceptance of an empty stack
    b. Accepting by entering final state
52. Balancing parenthesis: Syntax Phase: DPDA/DCFL : {S□aSb|SS|∈} == {S□(S)|SS|∈}=={ (),()(),(())…}
53. Sα □ B {Violates CFL} ; Sα □ b {Violates CSL} ; { α = Terminal/Non Terminal}
54. det(AB) = det(A) det(B)
55. rank(A + B) ≤ rank(A) + rank(B) [AIR, Addition, Rank]
56. Optimal binary search tree construction can be performed efficiently using dynamic programming
57. ***Without inorder we can't construct binary tree, can construct BST with pre and post order, not Binary tree***
58. As the number of entries in a hash table increases, the number of collisions increases
59. Every regular set has a LR(1) grammar
60. Every left-recursive grammar can be converted to a right-recursive grammar and vice-versa
61. The language generated by a context-free grammar all of whose productions are of the form X □ w or X □ wY is always regular (left-regular grammar)
62. LR(1) parsing is sufficient for deterministic context-free languages. (Parsing -> Syntax -> DCFL)
63. Generation of intermediate code based on an abstract machine model is useful in compilers because = “it enhances the portability of the front end of the compiler.”
64. A linker is given object modules for a set of programs that were compiled separately. What *information needs to be included* in an object module>> “Names and locations of all external symbols defined in the object module”
65. Data flow analysis is used in control flow graphs, for code optimization.
66. Heap can have repeated elements (but not preferred)
67. In max Heap smallest element is at the leaf node
68. In max Heap second largest element always the child of root (Can be found in 3 comparisions)
69. In max heap min element can be found in the leaves (needs n/2-1 comparisions on n/2 leaves)
70. *** IN GATE SYLLABUS, HEAPS DO NOT CONTAIN REPEATED ELEMENTS ***

$$\log_a b = \frac{1}{\log_b a}$$

71. Max Heap can be build from Binary search tree in θ(n)
    [as all elements are available, we use bottom up]
72. Kruskal’s Algo normally =θ(elog(v)) , If edges are sorted θ(e+n)
73. Inorder traversal of BST is sorted array
74. Infix to Postfix: Operator Stack (Operator=+,-,*,/)
75. Postfix to Infix: Operand Stack

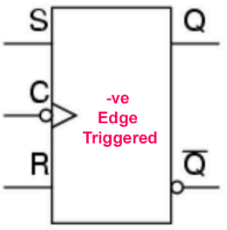
For a recurrence relation of the form,

$$T(n) = T(\alpha\, n) + T((1-\alpha)\, n) + cn$$

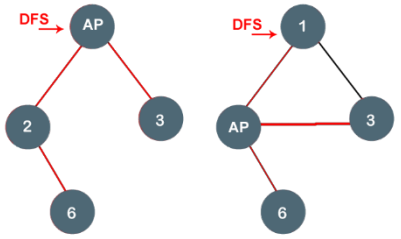
$$T(n) = \theta\left(n \log_{\alpha}\left(\frac{1}{n}\right)\right)$$

76. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations. (Dynamic Lists- Constant Delete and Insert)
77. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.
78. In a vectored interrupt the interrupting source supplies the branch information to the processor through an interrupt vector.
79. BFS traversal can find shortest path from a single source, if graph is unweighted and Directed (In place of DJ’s)
80. G is a weighted connected undirected graph with distinct positive edge weights. If every edge weight is increased by the same value, then "Minimum spanning tree of G does not change"
81. G is undirected simple graph in which each edge has a distinct weight, and 'e' is a particular edge of G, then "If 'e' is the heaviest edge of some cycle in G , then every MST of G excludes 'e' "
82. Static allocation of all data areas by a compiler makes it impossible to implement recursion ***(Recursion needs Dynamic Stack)***
83. Dynamic allocation of activation records is essential to implement recursion
84. Encoder and decoder do not have select lines, MUX single output, Encoder multiple output
85. (Encoder 2^n X n), not (m X n)

86. Static RAM (Cache): Flip flops, Dynamic RAM: Capacitors (Clear cache needs to be done, RAM gets cleaned automatically)
87. RAM is a sequential circuit and PLA is a combinational circuit (Seq: Flip Flops, Comb: MUX)
88. Static RAM is easier to use than dynamic RAM. Static RAM has shorter read and write cycles than that of dynamic RAM
89. Connected undirected weighted graph G has a unique minimum spanning tree (All statements "OR")
    a. if no two edges of G have the same weight.
    b. "OR" if, for every cut of G, there is a unique minimum-weight edge crossing the cut
    c. "OR" The edge with the second smallest weight is always part of any minimum spanning tree of G
    d. One or both of the edges with the third smallest and the fourth smallest weights are part of any minimum spanning tree of G
    e. Suppose S C V be such that S ≠∅ and S ≠ V . Consider the edge with the minimum weight such that one of its vertices is in S and the other in V \ S . Such an edge will always be part of any minimum spanning tree of G.
90. Bit Synchronization is always handled by Physical Layer of OSI model but not Data Link Layer
91. End – to – End Process Communication is handled by Transport Layer.
92. Medium Access Control (MAC) sub-layer functionality "Channel sharing"
93. The sequence of procedure calls corresponds to a preorder traversal of the activation tree.
94. The sequence of returns corresponds to a postorder traversal of the activation tree.
95. The path from the root to a node N shows the activations that are live at the time N is executing.
96. Procedure calls and returns are managed by a control stack, i.e., run-time stack.



97. IEEE 802.11 MAC protocol:
    a. At least three non-overlapping channels are available for transmissions
    b. Unicast frames are ACKed
98. IP packets from the same source to the same destination can take different routes in the network
99. Sending more than 1 byte at a time called message stream. TCP is byte stream in which 1 byte transferred at a time, In TCP, a unique sequence number is assigned to each, and it count each and every byte clearly.
100. Neither TCP nor UDP can give you any throughtput (communication rate) guaranatees.
101. TCP ensures in order delivery as it ensures the reliable delivery of a packet to other TCP connection.
102. TCP reacts to congestion by reducing sender window size (Slow start Exponencial Increase Algo for Congestion control)
103. TCP employs retransmission to compensate for packet loss
104. The DFS tree on undirected never contain cross edges of graph, may contain in directed graphs.
105. To solve DFS/BFS problems make graph like a tree
106. If the root of DFS tree is having multiple children, means while backtracking ***"koi aur node nahi mila jisse we can reach the other node 3",*** so the root is an articulation point.



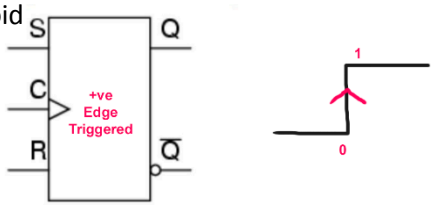
107. TCP handles both congestion and flow control.
108. DFS and BFS both can find if a graph is cyclic/acyclic of a graph (DFS preferred, BFS cannot find no. of cycles)
109. Both BFS DFS can find connected components (No of times BFS/DFS called)
110. ***Activity selection problem*** {Subject DA problem, place the most marks subject just before deadline (nlogn), Fractional Knapsack :: GREEDY, 0/1 Knapsack: Dynamic preferred, can be greedy also
111. UDP handles **none of** congestion and flow control
112. Fast retransmit deals with congestion but not flow control (It is congestion control technique and has no relation with Flow control)
113. Slow start mechanism deals with both congestion and **not** flow control.
114. Flow control { Stop&Wait, Go Back& N, Selective repeat}
115. Congestion control { Slow Start, Congestion Avoidence}
116. In TCP, if the estimated round trip time at any given point of time is t sec, the value of the retransmission timeout is always set to greater than or equal to t sec.
117. TCP connections are full duplex.
118. (TCP) & (Go Back & N) have "selective (i.e. individual) & cumulative acknowledgement" both.
119. RIP uses distance vector routing, OSPF uses link state routing.
120. RIP is sent using UDP. OSPF uses neither UDP nor TCP. (OSPF link state needs all routers access, not possible with TCP, needs to be reliable, not possible with UDP)
121. In TCP slow start, the congestion window doubles every round trip time (not propagation delay)
122. TCP has 2^{32} sequence numbers, one byte uses one sequence number.

123. IP fragmentation can cause excessive retransmissions when fragments encounter packet loss and *reliable protocols such as TCP must retransmit all of the fragments in order to recover from the loss of a single fragment.* (MTU IP-TCP transfer) In this process the identification number will also not be same as we are sending the entire packet again altogether.
124. PROMPT (Weather to prompt or not b/w a file transfer @FTP), HEAD (Meta data @HTTP), RCPT (Receipt @SMTP), HELO (Sender's Host Name @SMTP), PORT PASV (getting port privileges @FTP), PPP (Point to point @DLL), BGP (Border Gateway Protocol is used to Exchange routing information @NL)
125. Stateful: TCP(TL), FTP(AL), POP3(AL) {TL: Transport layer}
Stateless: IP(NL), HTTP/HTTPS(AL), DNS(AL), UDP(TL) {Do not need authentication}
126. HTTP & FTP uses multiple TCP connections.
127. Distance Vector Routing Protocol (DVRP) uses Bellmen Ford, it just doesn't have entire topology for the network. (Calculated at every node)
128. The Link State Routing Protocol (LSRP) uses Dijkstra's algorithm. (Calculated at every node)
129. DVRP requires lesser number of network messages to fully converge than LSRP (Since DVR is based upon local knowledge whereas LSR is based upon global knowledge i.e. a flooding is done)

No.	Routing Protocols	Persistent looping/Count to infinity problem
1	Distance vector routing protocol (DVRP)	Yes
2	Split horizon in DVR	Yes
3	Split horizon with poison reverse in DVR	No
4	Link State Routing Protocol	No

130. **1-Persistent:** It senses the shared channel first and delivers the data right away if the channel is idle. If not, it must wait and **continuously** track for the channel to become idle.
131. ***Non-Persistent***: It first assesses the channel before transmitting data; if the channel is idle, the node transmits data right away. If not, the station must wait for a ***random*** amount of time, and when it discovers the channel is empty, it sends the frames.
132. ***P-Persistent:*** It consists of the 1-Persistent and Non-Persistent modes combined. Each node observes the channel in the P-Persistent mode, and if the channel is idle, it sends a frame with a P probability. [waits for a ***fixed time slot***]
133. The data is not checked for any loss in the DLL/NL, it's only done at TL (Only header checksum(NL), CRC(DLL) is done)
134. It is not necessary for a router to implement any routing protocol (case of static routing where forward paths are pre-loaded/downloaded to routers)
135. Packet Switching results in more variation in delay (variation in delay: queueing, crowd in path, routing algo TC, etc) Circuit Switching don't have these delays, it reserves all of resources.
136. ***Record Route***: Options and padding (40B) (length, pointer and code consumes 3B) (Rem 37B Can store ***9 IPV4 IP's***) [37/4, 32bits=4B] ***{In IPV4 header}***
137. ***Traceroute***: It uses two ICMP messages, "time exceeded" and "destination unreachable" to find the route of a packet. This program is at ***application level*** and uses UDP. It uses ICMP messages and TLL field in the IP packet to find the route. The TTL value gives the number of routers in the path
138. Cannot use Record Route to trace route as we can store only 9 IP's.\
139. Traceroute reports a possible route that is taken by packets moving from some host 1 to some other host 2 By progressively querying routers about the next router on the path to host 2 using ICMP packets, starting with the first router.
140. No ICMP error message is generated for Loop-Back Addresses and DHCP client packets
141. As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection (in TCP)
142. For undirected graphs, each edge uv is stored twice, once in u's neighbor list and once in v's neighbor list; for directed graphs, each edge u --> v is stored only once, in the neighbor list of the tail u. For both types of graphs, the overall space required for an adjacency list is O(V + E)
143. Adj list
    a. 1 element TC = Degree of the node
    b. All elements TC = sum of degree of all vert : O(E+V) full list traversal
    c. For MST all elements traversal: ((V-1)+ V) = O(V) for MST (#E=#V-1)

144. Heap can be used as priority queue for dijkstra graph traversal algo and also for spanning tree algo {both are BST}
145. Level triggered in J-K Flip Flop with both J=K=1, and if clk=1 cause ***Race around condition***, THAT IS ***WHY EDGE TRIGGERED is PREFERRED***. In race around, the output Q will toggle as long as CLK remains high which makes the output unstable or uncertain. We can overcome this problem by ***making the clock =1 for very less duration***. The circuit used to overcome race around conditions is called the Master Slave JK flip flop.
146. Prop Delay of FF < Half Clock Pulse: Race Around will happen; reverse it to avoid
147. Toggling/Compliment operation, frequency of FF = frequency of clock/2
148. Counter1 Mod-N1, Counter2 Mod-N2, Cascaded(combined) = Mod – N1*N2
149. TCP is a connection oriented service
150. Persistent timer is for deadlock
151. Segmentation is done when transport layer is giving data to network layer



152. Fragment, process, byte :: NL,AL,TL :: Identification no, port no, sequence number (one byte uses one sequence number)
153. In lexical analysis compiler efficiency improved (Lexical Analyser scans through the symbol table in a sequential manner and replaces the identifiers present in the code with their location. This helps the rest of the phases to directly access the required identifier in a random manner and so saves time. Hence, efficiency is improved)
154. 'sfs' is a single token
155. **Synthesised attributes are analysed during BOTTOM-UP** (IT, SBI){IT: Inherited Top Down}
156. Intermediate Code :: Linear: Postfix and 3-Add code, Tree(Non-linear): AST, DAG
157. int a = "08" int b="0x33K" first one is octal, next is hexadecimal, both are Lexical Errors.

158. Single pass compiler: All stages of compiler in one go :: More Memory, More Speed.
159. Code optimization is an optional phase.
160. '7' distinct elements, no of heap/BST of height '6' = 2^6 {Standard Question, skewed tree}
161. Pre-processor –> Compiler/Interpreter –> Assembler –> Linker –> Loader –> Run-Time Environment
162. Complex loaders can perform Linking and Reallocation (Relocating: modifies the programme so that it can be loaded at an address different form the location originally specified)
163. If there is no compiler error we may have: {Logical Error, Runtime Error, Linking Error} {Logical Error: Sometimes, we do not get the output we expected after the compilation and execution of a program}
{Logical Err ≠ Semantic Err (Meaning)}
164. More the number of edges in a DAG, fewer the number of topological orders it has.
165. *Lemma:* If A is Adjacency matrix of a Graph G, Sum of all elements of A^3 is no. of path of length 3 in G {The A^3 Adjacency matrix will have 1's only for 3 length path of G, 0 for all others}
166. Quick sort works by ***comparing*** and swapping elements using a top down approach. Quick sort is a comparison based sorting algorithm.. ***Quick sort and Merge sort are TOP DOWN***
167. log*(log(N)) {log* means iterative log, In asymptotic notation we need to use many logs, not single log}
{$2^{\log^* n}$ =log(log..log(n))}
168. Distance Vector Routing (Good News Travels Fast, Bad news slow {Bad News: Increased cost slowly})
169. RARP(Net Layer) , BOOT (UDP Layer), DHCP (Net Layer) (changes w.r.t. time)
170. Network Address Translation (NAT) provides Security, advertise only one address for the network to the entire world
171. Flooding is static routing (not dynamic, not adaptive {dynamic==daptive}) (Flooding: Link State Routing) {Dynamic is optimum, but we send packets to all routers} {Selective routing solves this problem by flooding selectively only to some nodes}
172. Forward edge is not possible in BST
173. DJ's and Bellman will terminate if node unreachable from source. Bellman always do not detect a -ve cycle (if unreachable from source, does not detect)
174. If -ve weight cycles DJ's will not stop/terminate, Bellman will stop and report if reachable from source. (Both do not give result for -ve cycle)
175. If negative weight, DJ may/may not give correct result. Bellman ford always gives correct result for -ve weight.
176. Mod 5 counter cascaded to Mod 3 counter = Mod 15 counter
177. Flip-flop is a bi stable device with only two stable states.
178. Dynamic link (control link) field stores the address of activation record of the caller procedure. {DC}
179. Static Link (Access link} is used to access variables that are not local
180. Return value field used by the called procedure to return a value to the calling procedure.

181. Every change to next state is either shift/reduce {#states (I0,I1,..) in LR(0) = SLR(1) = LALR(1) :: maybe more in CLR(1) (because of look ahead, new states comes)}
182. #shifts in LR(0) = SLR(1) = LALR(1) :: may be more in CLR(1) because of look-aheads.
183. #reduce entries in CLR is less as it is put only in the look ahead symbol. Whereas in SLR, it is placed in all of the follow variables, in LR(0) placed in all variables.
184. Finding clique in a graph (completely connected part of a graph) cannot be found by DJ's
185. Maximum amount of ***flow from source to sink*** (***Cannot*** use Greedy)
186. Prims and Kruskal's are unaffected by negative weights
187. Sorting algo that is least dependent on initial ordering: Merge Sort
188. After first 3 iterations of Insertion sort (Taash patta one) first 4 elements are sorted.
189. Star needs a central controller or hub (no computer is connected to another computer directly)
190. Bus requires a multipoint connection (a single cable to which all the network nodes are connected)
191. Vulnerable time in pure aloha =2*tt, in slotted aloha = tt.
192. Pure aloha :: $S=Ge^{-2G}$ :: S_{max} @ G=1/2 [[G= Channel load; S Throughput]] :: Slotted Aloha :: $S=Ge^{-G}$ :: S_{max} @G=1
193. Decoder: Binary to Decimal (Give a BCD i/p respective Decimal o/p is enabled) {Encoder opposite of this}
194. DeMUX: Serial to Parallel data conversion {think of decoder, small i/p to large o/p}
195. Optimum binary search tree construction, Matrix chain Multiplication, Bellman Ford: Dynamic Programming Problems
196. Bellman ford can be used to find longest path in directed acyclic graph (just replace all weights from 'e' to '-e')
197. Round trip delay = 2*PD only, not TT+2*PD
198. In { E1 – R(M:1) – E2} , the relation (R) is merged with entity(E1) towards the many side, total participation is preferred over partial participation, as we may have null value in pk in case of partial participation.
199. In { E1 – R(1:1) – E2}, all tables could be merged together even if one side has total participation. Caution (All tables not merged together in case of (1:M) or (M:1)
200. Lossless, Decomposition preserving decomposition into 3NF is always possible.
201. Multitasking(new) = Multiprogramming(old) + Time Sharing (All Multi-tasking is multiprogramming)
202. Multi-Programmed
    a. More than one program may be loaded into main memory (Ready state) at the same time for execution.
    b. If a program waits for certain events such as I/O, another program is immediately scheduled for execution.
    c. If the execution of program terminates, another program is immediately scheduled for execution
    d. Just no time sharing
203. PCB is in Main Memory and uses doubly linked list, every program has it's own PCB
204. ***HRRN (Highest Response Ratio Next) is a modification of SJF to avoid starvation***, a large sized job may go into starvation in SJF, so we prefer jobs with "Shorter time" or one "who is waiting for long", one of most optimum job scheduling. {The highest-response ratio next scheduling policy favours short jobs, but it also limits the waiting time of long jobs} [There is short job computation]
205. As the time quantum goes on increasing the **Round robin acts like a FCFS, not SJF** *[No short job computation, naturally follows the queue methodology]*
206. SJF is most optimum thing needed in OS, everyone ultimately wants to be a SJF (FCFS is by default, queue)
207. No Starvation: HRRN, FIFO, LRTF (prem.), Round Robin (LRTF: There is a limit of heavy jobs in OS, upto memory capacity, so large jobs cannot be coming one after other infinitely, but short ones can, so SRTF/SJF may create starvation)
208. SJF/SRTF: Min Avg. WT, TAT and best Throughput, RR: Best Response Time
209. Interrupt is like a function, it stops current function, keeps the next instruction to be executed (PC) after interrupt in a Stack and executes itself.
210. **System calls** are usually invoked by using a **software interrupt** (We communicate to kernel via API)
211. Software interrupts are required by CPU to obtain System services which need execution of privileged instructions
212. Context switching from process A to process B OS performs:
    a. Saving current register values of A and restoring saved register values for process B.
    b. Changing address translation tables (Page Table, updated; each process has it's page table; context switch brings in new process, so old page table is flushed)
    c. Invalidating the translation look-aside buffer (***Resetting TLB, as we may not need them for next process***) (It is not resetted in case of an ongoing process, as we may need the same pages in upcoming instructions)
    d. ***[False]*** Swapping out the memory image of process A to the disk (A memory image is simply a copy of the process's virtual memory, saved in a file. It's used to debug a program fail, in case of a crash)

213. System call provides the services of the operating system to the user programs via Application Program Interface (API). Ex: Fork(), read(), wite(), chmod(), etc

214. User mode (Non-privileged mode), Kernel Mode (Privileged mode) – Dual Mode of OS {For security of system}

215. OS does resource management, we cannot access any hardware/software directly, we would exploit/misuse it, so we produce a system call like read(), and then via the OS we access the resources.

216. While going from user mode to kernel, the OS sets a timer, to set interrupt, for OS to take over the control. So, normally Kernel to User happens automatically, by a timer set by Kernel. If user explicitly wants to get out of Kernel mode from **Privileged to Non-privileged (Kernel to User)**: Non-privileged instruction is used, without interrupt.

217. **Non-privileged to Privileged (User to Kernel)**: TRAP instruction is used, software interrupt

218. Switching from user to kernel mode is a very fast operation (OS has to just change single bit **at hardware level**)

219. Process switches or Context switches can occur in only kernel mode. So, for process switches first we have to move from user to kernel mode. Then we have to save the PCB of the process from which we are taking off CPU and then we have to load PCB of the required process. (comparatively slower process)

220. Implementing preemptive scheduling (Context switches) require hardware support because of it we can swap the process between states. (Preemptive scheduling needs hardware support to manage context switch which includes saving the execution state of the current process and then loading the next process.) (periodic interrupts from the hardware)(periodic cycles that trigger an interrupt to the OS scheduler)

221. **Interrupt**
   a) **Software:** TRAP, Exception {Generated from execution of a program}
      - **Exceptions (unplanned, from within the system):** arithmetic overflow, undefined instruction, divide by zero, page fault, invalid memory access, invalid opcode
      - ***TRAPS (planned, from user to access OS) –*** special instructions requesting service from OS, ***TRAPS are called system calls***
   b) **Hardware** Maskable (Can be masked/ignored/ less priority), Non-maskable {Generated by a device outside the CPU} Ex: keyboard, timer, MM, HDD, I/O, [[power failure, real time application like power plant sensors, medical monitoring, missile (Non-Maskable)]]

222. An exception is an unexpected event from within the processor; e.g., arithmetic overflow

223. An interrupt is an unexpected event from outside the processor; e.g., I/O

***** Both these kinds of definition is there for interrupts/exception *****

224. An exception is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor.

225. Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. From user point of view we see our code is stored in continuous memory locations, but is actually not

226. Every FD is a MVD, reverse not True
   a. If A □ B and A □ C, then A □□ BC :: **TRUE** (Since A□ B and A□ C == A □BC which is a FD, so also a MVD)
   b. If A □□ B and A □□ C, then A □ BC :: **FALSE**

227. Any relation with two attributes is in BCNF

228. A relation in which every key has only one attribute is in 2NF (No proper subset of CK)

229. No partial dependency: 2NF, No transitive dependencies: 3NF (TD: BCNF + LHS not a NPA)

230. A prime attribute ***cannot*** be transitively dependent on a key in a BCNF relation ***(PA □ PA, not allowed in BCNF)***

231. CPU Utilization ↓ CPU Idle ↑ ; ** Throughput = #Jobs/Unit time **

232. Threads **do not share*****: Process Counter, Stacks, Registers*** {Process specific stuffs, as they act as an independent process)

233. Threads **share*****: Code, data, files heaps*** {Non-Process specific}

234. In Kernel level threads, they are given izzat as threads, so independent, so more variables, so more context, so more context switch time.

235. For user-level threads, a system call can block the entire process. Blocking a Kernel level thread, does not block all threads (treated as thread, independency) Every kernel level thread can be assigned to separate processors

236. Kernel supported threads can be scheduled independently.

237. User level threads are transparent to the kernel (Kernel is the boss, anyway 😉)

238. User level threads do not need any hardware support.

239. Gang Scheduling (Parallel scheduling in multiple processors) □ Thread/Process runs parallelly , Fair Share Scheduling (Multi-Tasking) □ Guaranteed Scheduling , Rate Monotonic □ Real Time Scheduling (Priority Scheduling)

240. Fork() Parent=1, Child=0

241. Synchronous Counter = All counters use same clock

242. Asynchronous Counter = Counters use O/P of prev. counter as clock
243. ***I/O:*** writes on Memory on Input, reads memory on Output, CPU is just a medium for transfer
244. ***Programmed I/O: Works with the help of a program [Transfer speed depends on CPU, I/O speed & Program T.C] [data brought as I/O ↔ CPU registers (because of program) ↔ Main Memory] **** □ □ IMP***

a. ***Synchronous I/O:*** I/O device always ready and always send data in same speed say X B/s [Practically Rare] ***Entire I/O transfer time CPU stops work, but puts the process in wait state entire CPU time is taken for I/O 100% CPU cycle waste, CPU and I/O maintains same clock signal, both knows each other's behaviour, both knows kab tak data aayega from I/O to CPU after that interrupt is made to put the process from blocked to ready, Full data transfer time CPU busy storing data in it's register and forwarding it to the MM. [[CPU in sync with I/O device]]***
b. ***Asynchronous I/O:*** After I/O ready, it sends a chunk of synchronous data. Has a start and end bit for CPU to know it wants to transfer data, CPU needs to be checking all the time for this bit [Status Bit] ***When no I/O is coming CPU can do it's work, but needs to check status bit baar baar. [[CPU out of sync with I/O device] [I/O happens separately, yet parallelly along with CPU work]]. Uses CPU registers to store and forward I/O data***
c. ***Interrupt driven:*** CPU initiates I/O and does it's work***, no check of status bit baar baar,*** when I/O is ready it produces an interrupt signal. But still CPU registers are used to store and forward data to MM. The part of the program that gets activated when interrupt request comes is called ***"Interrupt Handler" or "Interrupt service routine (ISR)" Process not sent to wait state, it continues it's work, same process ke other instructions chata hai, jo I/O se independent ho. When I/O ready, it is like an interrupt, ISR is called and the executing program is saved to handle the interrupt. Context needs to be saved.***

***** In interrupt driven & Asynchronous, process not sent to wait state, other instructions of same programme is executed***
***** Synchronous m jab I/O hota hai, toh full dedication to only I/O, no CPU work is done***

245. When interrupt comes, the PSW and PC is saved and ISR (Interrupt service routine) is called to handle and if it can, it will return back the control to previous running instruction. Cannot return in case of illegal memory access, divide by zero, etc. ***[[ISR is a code not a hardware]]***
246. [[In synchronous I/O process performing I/O operation will be placed in blocked state till the I/O operation is completed. An ISR will be invoked after the completion of I/O operation and it will place process from block state to ready state.]]
247. [[In asynchronous I/O, Handler function will be registered while performing the I/O operation. The process will not be placed in the block state and process continues to execute the remaining instructions. when the I/O operation completed signal mechanism is used to notify the process that data is available]]
248. In the case of synchronous I/O, the process waiting for the completion of I/O is woken up by the ISR that is invoked after the completion of I/O
249. An ISR (Interrupt Service Routine) is invoked on completion of I/O in synchronous I/O but not in asynchronous I/O (to inform that I/O operation is completed)
250. Direct Memory Access: An additional hardware DMA controller (is kind of a I/O module, but is kinda mini-processor for I/O operations ***[data brought as I/O ↔ DMA Controller (chota CPU) ↔ Main Memory]***
    - DMA Controller has: Read bit, Write bit (to decide nature of operation R/W), DMA REQ, DMA ACK, Interrupt, Data count register (kitna data ka len-den hoga), Address register (kaha se lana hai starting base address), Data Register (Substitute to CPU register, which will do the store and forward to the MM. [[All the info about the transfer will be given by actual CPU]]
    - DMA controller sends interrupt after it finishes
    - The data register, data count, address registers are connected to data bus of system, address register is connected to Address bus of the system
    - As data counter ==0 , DMA sends interrupt to CPU, saying data transfer is over.
    - In DMA, DMA controller is the master of System Bus, CPU is basically doing local operations, only in interleaving DMA, CPU is allowed to used system bus when it needs it, as a token of gratitude, but DMA controller is the master of bus.
    - No change of context, as I/O is not like an interrupt
    - A running program may be paused in b/w any machine cycle (IF,ID,EX,MA,WB) for DMA transfer, as no register/ flag of CPU is getting changed in the transfer, so no problem
251. DMA: No involvement of CPU, directly transfer to MM, DMA interface chip, transfer in bulk, uses system bus
    a. Burst mode: All data at once
    b. Cycle stealing mode: steals some cycle of CPU, wastes some CPU time, by using system bus, when CPU needs it.

c. Interleaving: When CPU uses local bus, DMA uses the sys bus, if CPU needs system bus, DMA waits.

252. Efficiency for 1B transfer: Programmed>Interrupt I/O > DMA, generally (DMA>Interrupt>Programmed)

253. **In interrupt, PSW (program status word) and PC needs to be saved** (as an entirely new program starts to run), in normal function/subroutine call, only PC is saved, as the same program is running anyhow. ((The Program Status Word (PSW) contains status bits that reflect the current CPU status))

***** PSW Fags*****

254. ***6 conditional flags*** (carry, parity, auxiliary carry (carry from lower nibble to higher), zero, sign, overflow

255. ***3 control flags*** (TRAP (infinite halt in a state to allow debugging, Interrupt, Direction (for auto increment/decrement))

256. ***Priority interrupt Controller (PIC):*** A device b/w multiple I/O's and CPU, assigns priority to I/O's and if multiple interrupts comes at the same time, the one with higher priority is acknowledged first, If after processing first I/O, if the second one (the lower priority one) is still active, it will be processed. CPU sends ack to PIC, It knows which I/O it needs to forward the ack from CPU, i.e. the vector of the I/O receiver

257. ***Nested Interrupt:*** When device 1's interrupt is being services, device 2 has interrupted. Solutions:
   a. [[Service the interrupt of D1, [[when interrupt of D2 comes in between, service it]] Resume interrupt of D1]] (may be D1 was non maskable interrupt, and it will get delayed): Generally done only when higher priority interrupt has come in b/w a low priority one, not the other way
   b. When interrupt is being serviced, stop Interrupt Service temporarily (will not service the interrupt intruders), while returning from service enable the interrupt again

258. ***How to find out which device wants I/O?***
   a. ***Vectored:*** The PIC stores the vector/base address/Device ID of the device who interrupted {Fastly identified} and sends to CPU via data bus
   b. ***Polling:*** All the device has a status bit, if it has interrupted, the bit will go high, CPU needs to scan all devices to identify who the hell interrupted {slow process} [[Polling is not a part of interrupt]]. In polling, the CPU periodically checks the status bits to know if any device needs its attention.

259. DMA by default transfers in word (word by word)

260. DMA I/O: HDD transfers, Interrupt I/O: Printer, Keyboard

261. Vectored Interrupt: Generally, interrupt with complete info about I/O (location, service, I/O device info, etc)

262. Interrupt Priority: Temp. Sensor >> Hard Drive>> Keyboard/Mouse

263. The ***daisy-chaining*** method involves connecting all the devices that can request an interrupt in a serial manner. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on. The interrupt pin is common to all. (It gives non-uniform priority to various devices) {Uniform means 'same', not uniformly increases}

264. Daisy chain INTR (interrupt request) line is common to all device, INTA is not. INTR is an 'OR' Gate anyone can request, multiple also can. INTA is first connected to D1 it's o/p is connected to D2 and so on, like a train engine fashion, D1 having higher priority. If D1 had interrupted it will not let the INTA to go to D2, if it had not, it will let the INTA propagate to D2, D2 will do the same thing, and so on…

265. When it comes to an interrupt, the device informs the CPU that it needs its attention. When it comes to polling, the CPU keeps on checking if the device needs attention.

266. Minimal Cover is the process of eliminating redundant Functional Dependencies and Extraneous attributes in Functional Dependency Set. So each dependency of F = {AB->C, D->E, AB->E, E->C} should be implied in minimal cover (Minimum cover, reduce if possible, but include all directly or in an implied manner)

267. Speed: hard wired control > horizontal microprogramming > vertical microprogramming

268. Redundant array of independent disks (RAID) is a storage technology used to improve the processing capability of storage systems

269. Raid configurations of the disks are used to provide Fault-tolerance, High speed(One disk in US, one in India), High efficiency.

270. Characteristics used in the design of a RISC processor
   a. Register-to-register arithmetic operations only (All ALU in register only)
   b. Fixed-length instruction format
   c. Hardwired control unit

271. Events that occur after a device controller issues an interrupt while process L is under execution
   a. The processor finishes the execution of the current instruction
   b. The processor pushes the process status of L onto the control stack
   c. The processor loads the new PC value based on the interrupt
   d. The processor executes the interrupt service routine (ISR)

e. The processor pops the process status of L from the control stack

272. A ***foreign key*** needs to refer a ***candidate key*** (Unique, may be NULL), ***may not be primary key***, for ***referral integrity***

273. Prefix, Suffix, Half, Even len., Odd len. of a regular language L is also regular (Half of L means even length strings of R)

274. Σ^*-L = L^c

275. $\emptyset^* = \in$ (Klen Closure) **[CAUTION: $\emptyset^* \neq \emptyset^c$. Don't think $\emptyset^*$ is compliment if $\emptyset$ i.e. {a,b}*]**

276. $\emptyset.L1 = \emptyset$ (Concardination)

277. $\emptyset \cup \in = \in$

278. Will always invoke a system call when executed: exit(), sleep(). If heap needed to expand, malloc() may need sys. call.

279. The goal of CPU scheduling to maximize CPU utilization & throughput and to minimize turnaround time, waiting time, and response time

280. Turnaround time includes waiting time

281. Round-robin policy can be used even when the CPU time required by each of the processes is not known apriori

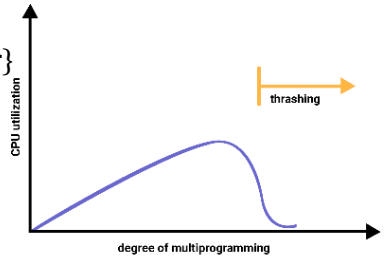
282. (0+1)* = (0+1)*0*1* {Expansion of (0/1)}

283. L={w|w has 3k+1 b's for some K∈N} is {No of b is, 3 mod No. of b=1, which is regular}

284. Absolute Addressing Mode: Memory Direct Addressing

285. In case of memory mapped I/O, the OS offers protection to I/O devices from user.

286. Thrashing is a condition in which ***excessive paging operations are taking place***. (more time solving page faults, throughput decreases drastically.)



287. LRU may cause Trashing

288. Trashing control:

(1) working-set strategy:

a. It initiates another process if there are enough extra frames.

b. It selects a process to suspend if the sum of the sizes of the working-sets exceeds the total number of available frames

(2) Page Fault Frequency

a. The problem associated with Thrashing is the high page fault rate and thus, the concept here is to control the page fault rate.

b. If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames

289. Bleady's Anamoly "More frames in FIFO = More page faults" {Even if we increase frame size in MM, the no of page faults may increase, logically weird as we are increasing MM, it should accommodate more pages, so less faults}

290. Bleady's Anamoly happens in non-stack based algo's; stack based algo's assign a priority, irrespective of no of frames, so never suffer from Bleady's Anamoly. (The smaller frame size elements say frame size =3 & frame content= {7,5,9} is always a subset of the larger frame say frame size=4 & frame content= {8,9,5,7}}

291. Locality of reference implies that the page reference being made by a process ***is likely*** to be to one of the pages used in the last few page references (not ***must find***)

292. The dirty bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified (helps avoid unnecessary write to the memory, we write only whose dirty bit is 1, ignore others) [dirty bit ==1 □ page is modified/written on]

293. Virtual Memory Space (LAS) of Paging ≠ Virtual Memory (Virtual RAM @HDD)

294. PAS = Main Memory size; LAS = Program/Process size

295. Every program has it's own Page table; likewise, every file has it's own index table (record □ index (using B/B^+tree))

296. ***MMU:*** CPU generates LA and then MMU converts it to PA

297. Virtual memory can give protected address spaces to processes (Without virtual memory, it is difficult to give protected address space to processes as they will be accessing physical memory directly) ***VM :: Security***

298. Because of virtual memory, Linker can assign addresses independent of where the program will be loaded in physical memory (can be done by Base relative and PC relative mode in MM itself) ***[imp: not only VM can do this]***

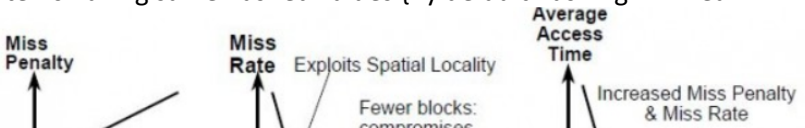
299. Virtual Memory slows access to memory, as if there was direct physical memory access, it would have been faster [virtual memory is RAM @HDD]

300. Programs larger than the physical memory size can be run ***[only virtual memory can make this possible]***

301. Virtual memory implements the translation of a program's address space into physical memory address space. Virtual memory require both hardwire (main memory and TLB) and OS support (to control those hardwire) , that actual translation from program's address space to physical address space is done by main memory.

302. Virtual memory increases the degree of multiprogramming {In VM we only load the process partially i.e only the selected number of pages as demanded by the process during execution and never load the entire process in MM and thus we can save some space in MM and load extra process' in that space and increase degree of MP}
303. ***Virtual memory increases the context switching overhead***. {VM increases the context switching overhead. VM is based on demand paging so more number of swaps == increase context switch, during page fault}
304. The total size of address space in a virtual memory system is limited by:
    - the length of MAR
    - the available secondary storage
    - ***not*** the size of main memory
305. Segmentation does not necessarily need virtual memory to implement it
306. ***Temporal Locality***: Temporal Locality means that an instruction which is recently executed have high chances of execution again. So, the instruction is kept in cache memory (only the instruction)
307. ***Spatial Locality:*** Spatial Locality means that all those instructions which are stored nearby to the recently executed instruction have high chances of execution. So, all the instructions are kept in cache memory (a word of nearby instructions)
308. Size: Primary memory < virtual memory < secondary memory
309. An SQL query can contain a HAVING clause only if it has a GROUP BY clause (SQL/92 Standard :: GATE Standard)
310. All attributes used in the GROUP BY clause must appear in the SELECT clause (SQL/92 Standard :: GATE Standard)
311. $\in.0(00)^* = 0(00)^*$; $\in+0(00)^*+0(00)^*0(00)^* = 0^*$; $0^*(\in+0(00)^*) = 0^*$
312. MPL (Minimum Pumping Length) is the least possible value of P such that Pumping Lemma is satisfies (P: length of string) ; Pumping Lemma uses Pigeon Hole Principle
313. Auto Indexed addressing, generally do not require an additional (extra) ALU (other than the user ALU) for effective address calculation, it uses the main ALU, but if we have a dedicated ALU for effective address calculation, computer will get faster (better pipelining). Address = address + step size (step size, depends on opcode/what we are performing)
314. In ***serial data transmission***, every byte of ***data is padded with a '0' in the beginning and one or two '1' s at the end*** of byte because receiver is ***to be synchronized for byte reception***. '0' is added in the beginning of data as start bit and '1' is added at the end as a stop bit. The start signal tells the receiver about the arrival of data and the stop signal resets the receiver's state for the arrival of a new data.
315. In 2' s complement addition, overflow cannot occur when a positive value is added to a negative value
316. ***Pipeline Flushing (Stall):*** It's a procedure enacted by a CPU when it cannot ensure that it will correctly process its instruction pipeline in the next clock cycle.
317. A ***data hazard(RAW,WAR,WAW):*** arises if one instruction needs data that isn't ready yet (Read before Write)
    a. ***Solved by data forwarding***, using buffers, if already value calculated by the last instruction, need not wait upto Write Back, use data from ALU stage
    b. ***Solved by "Stall/Bubble":*** We could delay the AND instruction by introducing a one-cycle delay into the pipeline. Waiting for one clock cycle.
318. ***Structural hazards*** result from not having enough hardware available to execute multiple instructions simultaneously.
319. ***Data hazards*** can occur when instructions need to access registers that haven't been updated yet.
320. ***Control hazards*** arise when the CPU cannot determine which instruction to fetch next (Conditional Jump)
321. ***WAR: Anti-dependency; RAW: True dependency; WAW: output dependency***
322. In ***dynamic linking*** (linking at runtime) since the path is not known at compile time, there is a risk that a *process might try to access unwanted memory locations during linking (the intruder may place a malicious packet while linking)*. That's why dynamic linking is less secure and needs to be handled carefully by the OS. {Static Linking: If path of libraries are known already, i.e., at compiler time then OS knows the specific memory locations it will have to go and it won't be dealing with other unknown locations.}
323. If we have only one free memory frame @Physical Address Space, All SSTF, FCFS, etc will give same number of page fault (as every different block access is a page fault, irrespective of any condition)
324. The ***minimum*** number of page frames is decided by ***Instruction Set Architecture*** whereas the ***maximum*** number of page frames is decided by ***Physical Memory Size***.
325. $\pi_{A1}(\sigma_{C1}(R1)) \rightarrow \sigma_{C1}(\pi_{A1}(R1))$ {LHS cannot be replaced by RHS} (C: condition, A: attribute) (Condition C1 may have condition with respect to other attribute, say A2, projecting A1 first will remove A2 and condition can't be checked.
326. σ is commutative :: π is not commutative (π has power of eliminating attributes)
327. In conditional Jump of pipeline, the next instruction's Instruction Decode is loaded into pipe by default, and if a branch occurs (i.e. branch returns TRUE), this instruction is flushed (but prefer not to flush/replace by some non-problem-creating instruction) and the branched instruction is brought into pipe.

328. In this instruction space, which gets flushed, to avoid flushing, or stall in pipelining, an instruction is brought such that it will not affect the branch condition, i.e. upcoming instructions of branch, if it somehow gets executed upto some extent, like a memory store is preferred rather than an ALU on the register value with respect to who we are branching.
329. Bypassing (Operand forwarding) can handle all RAW hazards: ***False*** { Suppose 'I1' is doing 'EX' and that value needs to be written to memory in some 'I3', in some loop of instructions or something, ***in the same cycle,*** we cannot pass this value to the 'I3' in the same cycle, as we will buffer it and send it in the next cycle.
330. Register renaming can eliminate all register carried WAR hazards.
331. Control hazards (CPU cannot determine which instruction to fetch next) can be reduced by 'dynamic branch prediction' not completely eliminated.
332. ***Paging*** is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous. {To give user the feel that the program code is contiguous}
333. ***Pages are logical division of the program,*** not actual, having index number, corresponding instruction in main memory (physical memory) is mapped inside a page table.
334. We have page table (Address Translation Table), whose ***index is page number (programme index)*** and the page table ***entries contain the frame number***, present bit ,modify bit, valid bit, protection bit, control bits.
335. Valid Bit: Says page available or not in main memory
336. Reference Bit: How many times page referred (useful in page replacement policies)
337. Paging has internal fragmentation in last page, no external fragmentation.
338. ***Segmentation*** has external fragmentation, ***no internal fragmentation.***
339. Fixed Partitioning (Static) has internal fragmentation, no external fragmentation
340. Variable Partitioning (Dynamic) has external fragmentation, ***no internal fragmentation.***
341. External Fragmentation: We have enough memory to store process Px but not contiguous, as some have left earlier from middle
342. Internal Fragmentation: The partitions will be wasted and remain unused if the process size is lesser than the total size of the partition.
343. In windows page size is 4KB
344. 256 eight-bit word means: Byte addressable memory and 256 such cells are present
345. If cache size is increased, hit ratio increases, miss penalty increases (cache bada ho gaya zyada search time in CM)
346. In main memory, speed is not a factor, but size is, In HDD size is not a factor, access speed is (Access Speed to CPU: MM:ns; HDD:ms; All operations happens in CPU anyway).(B,B+ tree have block amount of data in each child, if we stored just one or two keys in HDD Data Structure(i.e B/B+ tree), firstly, there would be internal fragmentation, and secondly we would bring very less data in every disk access, which is such a waste of time, since data is transferred block by block, block sized DS is preferred in HDD) perfect use of special locality)
347. BST,AVL are having 2 children, so are preferred in Main Memory; programs can access data faster anyway, as they are in MM, and they contain less chunk of data in them, CPU works word by word anyway, so DS with less data and faster operations(search/del max, etc) are preferred in MM
348. ***Range queries are faster in B+ Trees*** because of the Block pointer connects the next blocks in leaf node, whereas, in B tree if a block is exhausted, tree needs to be traversed again, atleast upto previous level (best case), worst case upto root node (if block is last leaf of left sub tree), even to access the next continuous block.
349. Associativity k means we have space for "k" blocks in a location (or) k-way set
350. Inverted Paging: Every processes do not have a page table, but entry in page table is made for all main memory frames. Size ↓ (as only one frame will have memory hole/internal fragmentation here); Search time ↑
351. The TLB performs an associative search in parallel (TLB is like a Fully Associative Cache. We know that in the fully associative cache, the search is made on the basis of CONTENT rather than the ADDRESS; in p-way, all p blocks are checked in parallel for row1, then row2, so on; in direct all lines are accessed in parallel)
352. If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory.
353. A TLB may reside between the CPU and the CPU cache, between CPU cache and the main memory or between the different levels of the multi-level cache ***{TLB only stores 'page no □ frame number', not cache address, raste mein cache mein bhi dekh leta hai, agar mil jaye toh badiya, nahi toh goes to MM}***
354. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, it means they will be present as a linked list of items having same hashed values {By default hashing == linear probing}
355. Miss Penalty: High, if we need to bring large blocks



from MM, if not found in CM, due to miss in CM

356. Miss Rate: Low, if block size inc, as more data brought to CM, in one access, as block by block access is done and this will increase spatial locality.
If block size increased so much that now we have only two blocks, and our process demands 4 blocks for running. We cannot bring that much data (So, temporal locality is compromised)

357. Avg. Access time: If block size is decreases, we have more blocks, hence more searching time (#block = CM size/ block size)

358. Page Table Servicing time, in page miss (includes bringing from MM to CM, replacement of page in page table, may also include HDD access)

359. Miss penalty from L2 cache to Main Memory = (extra time from CM L2 □ MM)
So, Avg. cost =L1 hit% (L1 time) +L1miss% (L2 hit% (L1 time+L2 time) + L2 miss% (L1 time+ L2 time + miss penalty))

360. ***Write Through*** (तुरंत करूँगा): In case of a write/update (dirty bit =1) update L1/L2 cache also the MM at the same time, as soon as program updates a block update it to MM (***Consistency*** ↑)

361. ***Write Back*** (बाद में करूंगा / back lag jayega): In case of a write/update (dirty bit =1) update L1/L2 cache also the MM after all operations completed by the program, if a block updated now, we'll wait, more updates may be done on same block, so older updates were unnecessary to update to MM, final block stuff after program execution is updated to MM (***Efficiency*** ↑)

362. ***Write Allocate:*** Writing back to MM after program completion (kind'a useless for Write Through; must for Write Back)

***** Assuming all exclusive cache (if miss in L1 and found in L2, brought from L2 to L1, no copy kept @L2)*****

363. ***Write Through cache (L1):***
   a. Read Miss: Bring from L2 or MM, L2 may be updated via MM, L1 will be updated via L2, ((L2 cannot be updated by dirty bits of L1 now, as we're not writing in this execution, previous dirty writes are already updated @MM))
   b. Write Miss: Bring from L2 or MM, write in L1 and MM right now ((nothing to write in L2 as we brought the block from there it is not there anymore))
   c. Read Hit: Enjoy!! Nothing to be done
   d. Write Hit: Write in L1 and in MM rt now (Nothing in L2)

364. ***Write Back cache (L1):***
   e. Read Miss: Bring from L2 or MM, L2 may be updated via MM, L1 will be updated via L2, ((L2 cannot be updated by dirty bits of L1 now, as we're not writing in this execution, will be written @MM after programme execution))
   f. Write Miss: Bring from L2 or MM, write in L1 as of now, not @MM rt now ((nothing to write in L2 as we brought the block from there, it is not there anymore))
   g. Read Hit: Enjoy!! Nothing to be done
   h. Write Hit: Write in L1 now MM not right now (Nothing in L2)

365. ***Write Through cache (L2):***
   i. Read Miss: Bring from MM to L2, L2 will be updated via MM, take from L2 to L1, L1 will be updated via L2 (nothing in L2, our block moved to L1)
   j. Write Miss: Bring from MM to L2, L2 will be updated via MM, take from L2 to L1, L1 will be updated via L2 (nothing in L2, our block moved to L1) L1 will update it's value and write to MM at the same time

***** Things will change in inclusive cache (if miss in L1 and found in L2, brought from L2 to L1, a copy kept @L2)*****

366. Eviction of a block @WB/WT cache (due to replacement) will not lead to any write operation

367. Eviction of a block @WB cache (due to replacement) is dangerous [back lag gaya!], the replaced block may have been dirty and replacing it will lead to lost update; no problem for WT cache (as it does write operation on time, immediately) but WT is slower ((less efficient)(data more reliable)(like circuit switching))

368. ***Memory refresh*** is the process of periodically reading information from an area of computer memory and immediately rewriting the read information to the same area without modification, In a ***DRAM*** chip, each bit of memory data is stored as the presence or absence of an electric charge on a small capacitor on the chip. As time passes, the charges in the memory cells leak away, so without being refreshed the stored data would eventually be lost. [like repeaters in CN]

369. ***Refresh is not done via data R/W lines, we have a separate architecture for that (Refresh: R1,R2,R3..) (R/W: C1,C2..)***

370. Interleaved Main Memory: Main memory content spread across banks (i.e. Caches)

371. ***DBMS Indexing:*** There are record files at database, and an index is made wrt some attribute, generally the key, for faster access, we have a search key and a pointer(the address of record), if for every block we have a key(sparse), key for every record(dense), wrt search key's ordering and uniqueness (unique and ordered: primary, not unique and ordered: Clustered, unordered: Secondary)(Generally sparse is preferred (from second level guaranteed sparse), ***if unorder and unique: level 1 needs dense indexing must.*** For indexing B/B+ tree is used, we have multiple levels like in paging, and all the indexes are stored in the database like page table is stored @MM

372. In a fixed block (every block can accommodate one file/ like fixed partitioning) if block size is increased, huge internal fragmentation, so poor disk utilisation, but as block size inc. #blocks decrease, as memory size is same, so seek time reduces, hence better disk throughput ***[[@CM to CPU if block increase, as Data BUS is of word size, transfer rate reduces, and temporal locality suffers MM to CM block by block; @HDD to MM (block by block) if block inc. seek time will reduce, as now #blocks dec. and will help in faster transfer, but a fixed architecture will increase memory hole]]***

373. At HDD to MM transfer "seek time >> transfer time", as they are magnetic disks

374. I/O redirection is like memory indirect of database, {can be employed to use an existing file as input file for a program}

375. Disk formatting is the process of preparing a data storage device such as a hard disk drive, solid-state drive, floppy disk, memory card or USB flash drive for initial use. In some cases, the formatting operation may also create one or more new file systems. The formatted disk capacity is always less than the "raw" unformatted capacity specified by the disk's manufacturer, because some portion of each track is used for sector identification and for gaps (empty spaces) between sectors and at the end of the track [Ex 32GB chip is always less than 32GB] formatting is not deleting

376. ***Swap space*** is a space on a hard disk that is a substitute for physical memory. It is used as virtual memory which contains process memory images

377. ***Disk Space Allocation***
   a. Contiguous Allocation [Data]: We put data in every bock in sequence, if in between data extracted External Fragmentation present. {**External frag.** ✓ Internal Frag. ✓ Seq. Access ✓ Random Access. ✓}
   b. Linked Allocation [Data]: Each file is a linked list of disk blocks which need not be contiguous. If a pointer of a block lost, the file gets truncated, more overhead {External frag. ✖ Internal Frag. ✓ Seq. Access ✓ **Random Access.** ✖} ***Connected like a train engine to boogie to next boogie and so on… c***
   c. Indexed Allocation: In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. {External frag. ✖ Internal Frag. ✓ Seq. Access ✓ Random Access. ✓} Like in questions, data set and a index block, no chance of external frag. [[See GFG photos for clarity]] ***Connected like a page table/index table/tree one connected to all***

378. Repeating file names are not allowed in a directory; हमारा laptop वाले folders {Rename/new file ke pehele all files scan}

379. Translation Look Aside Buffer (TLB): Address translation :: Normal TLB, program specific stuffs cached, temporal locality.

380. Translation Look-Ahead Buffer Pages in the disk cache ahead of its access (probably based on ***spatial locality***) (Since block is brought, spatial elements also brought in; ahead:: Samay se pehele / Bina kisi zarurat ke jo pura block aa jata hai, we may not need full block, but comes before hand; ahead

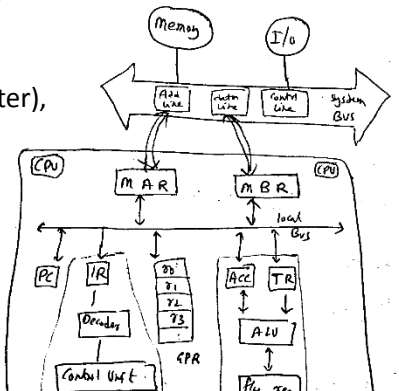
381. ***Buddy System:*** If we have a chunk a memory S KB and Physical memory 2^x KB, allocate the chunk iff $2^{x-1} < S \leq 2^x$; or recursively go on making it half and check the condition. Ex. Chunk- 18KB Phy Mem: 128MB; { 64 < 18 ≤ 128 □ No}, recursively break memory into half: 128(F)□64(F)□32(T) { 16 < 18 ≤ 32 } So put the chunk into 32KB [[Note: Int Frag]]

382. Premption of Round Robin is actually an interrupt caused by the time slice timer; No Premption scheduling is possible without interrupt. This timer used for Premption in a hardware, so Premption is not possible without hardware

383. Demand paging is not possible without interrupt, as if page is not present, we need to send error message, which is an interrupt, else the programme will keep on waiting infinitely. Even in Round Robin it will keep on coming and may lead to a dead lock, by circular wait (This program is holding some resource R1, and some other programme demands it, but as this is stuck infinitely in Memory access, so other process can never get this resource: Goes into Starvation)

384. Instruction Fetch
   I1: PC □ MAR
   I2: M[MAR]□MBR {System Bus}
   PC=PC+1 {Local Bus}
   I3: MBR □ IR

CPU has 9 registers: PC, IR (Instruction register), ACC, TR, MAR, MBR, Stack Pointer, Flag Register, GPR



385. MIPS is of RISC, having 32bit architecture, i.e word length =32b =4B
every instruction is of 32 bits

386. Instruction = 1 word; instruction is communication medium with CPU,

instruction set architecture (@control unit) is the vocabulary {All possible instructions}

387. ***Semaphore***: [[Up(),V(),signal(),release()]] [[Down(), P(), wait()]]
388. Binary Semaphore: At a time only one process allowed in critical state. 0 block, 1 allow
389. Counting Semaphore: Multiple processes allowed, at a time. 0 and -ve :: no resource available, block, +ve: allow
390. S= +1, 3D() 2U(); 3D() □ one process goes in, two are in wait state, 2U() □ 2 process pulled out of wait state. [[S=0]]
391. S=+2, 3D() 2U(); □ 2 D() both process goes in, S=0, next 1D() makes S=-1 one process goes to wait; next 1U() makes S=0; and pulls process out of wait and to ready. 1U() makes S=+1;
392. When process in suspended/wait state and some other process wakes it up, it comes to ready state, in this iteration it failed to gain access the critical, now in ready state, will attempt when scheduler schedules it again.
393. If two or more processes in wait state and no process to wake 'em up, they are in a DEADLOCK
394. Semaphore operations are atomic, even tough they contain more than one micro-instructions, because they are implemented inside OS Kernel
395. Dining philosophers are thinking now and sitting in a circle, all of 'em have spoons on both their sides, if all of 'em feels hungry together and picks their left/right spoon, all will be waiting for the right/left spoon and will result in a deadlock, solution: Give one philosopher right spoon first then left spoon, all others first left spoon then right spoon, ones holding both the spoons will finish in a finite time and will free resources for others.
396. Every semaphore will have it's own suspend list :: D(a),V(b) □ a will be put into suspend, b will take CS, a will not be released from suspend list; D(a),V(a) □ a is put in wait, a is released from wait and goes to ready, no one in CS
397. If in deadlock state □ In Banker's unsafe state (Of banker's algo; more recourse requested than available) :: Converse not true
398. ***Deadlock Characteristics (like cancer symptoms, better caught early)***
    a. Mutual Exclusion: At a time, non-sharable resource is held by only one process
    b. Hold & Wait: Process is holding a resource and waiting for some other
    c. No Premption: A resource can be released only voluntarily by the process holding it, no ज़बरदस्ती छींन लेना
    d. Circular wait: P1 holding Rx, waiting for Ry, P2 holding Ry, waiting for Rx
    e. ALL OF THESE FOUR INDIVIDUALLY ARE NECESSARY CONDITION FOR DEADLOCK, CUMILATIVELY THEY ARE NECESSARY AND SUFFICIENT FOR A DEADLOCK
399. If all deadlock characteristics holds, surely deadlock
400. Try to prevent ***Deadlock characteristics*** from happenin' □ ***Deadlock Prevention***
401. ***Deadlock Avoidance:*** Do not allow unsafe state in Banker's algo. Bankers algo in unsafe state is just a necessary condition, not sufficient, i.e. ***unsafe state not always lead to deadlock,*** but better to avoid it. [Plane stall not always crash] :: *Stall hone ke baad, trying avoid to crash, better prevent stall in first place.*
402. ***Unsafe state not always lead to deadlock:*** Pre deadlock things, (there is a chance of deadlock, has not happened yet) so with time need of a resource may decrease/not needed anymore, alternate resource used, etc. [like if you go this road, you may get into an accident, but not happened yet, may be the rider gets into someone's house.]
403. Safe state means deadlock can be avoided if resources are managed properly { i.e. deadlock free as of now, may occur in future }
404. ***Deadlock Prevention is more strict/restrictions than deadlock avoidance [[both are pre deadlock measurements]]***
405. ***Priority: Deadlock prevention >> Deadlock Avoidance (less restrictive) [[prevention is better than cure (avoidance)]]***
406. ***Deadlock detection*****:** If a cycle in Resource allocation graph (or) Banker's algo in unsafe state
407. ***Deadlock recovery***: Kill process, prempt process who is holding the resource and waiting for other resource or holding but not using it (Premption in ***Deadlock recovery may lead to starvation***)[[Here deadlock ho gaya, uske baad ke steps]]
408. ***Who to kill/prempt*****?** * With low priority *Having more resource request *Holding more resources
409. ***Necessary condition policies for preventing deadlock in a system with mutually exclusive resources:***
    a. Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released (hold and wait will never happen)
    b. The resources are numbered uniquely, and processes are allowed to request for resources only in increasing/decreasing resource numbers
410. Instances in the resource allocation graph may mean holding a request/requesting for a resource

***** Dissatisfying the deadlock preventions:***

411. Number the resources uniquely and never request a lower numbered resource than the last one requested (Circular wait) [the increasing order/decreasing order thing]
412. Release all resources before requesting a new resource (Hold and wait) [leads starvation]
413. Request and all required resources be allocated before execution (Hold and wait)

414. ***Stack pointer*** is a 16-bit register, which contains the address of stack top. i.e. the memory address of last byte entered in stack. It is important for a subroutine/nested subroutine/interrupt, as we need to store the address from where we'll begin after servicing the interrupt/executing the subroutine/nested subroutine. It is possible to implement all of 'em by creating an auxiliary stack @MM, but if using auxiliary space is not allowed, none of 'em is possible without SP.

415. Virtual memory is a combination of RAM and disk space that running processes can use***. Swap space*** is the portion of virtual memory that is on the hard disk, used when RAM is full

416. Mutual Exclusion: Jab koi process 'CS' mein ho, koi aur na aa pawe.

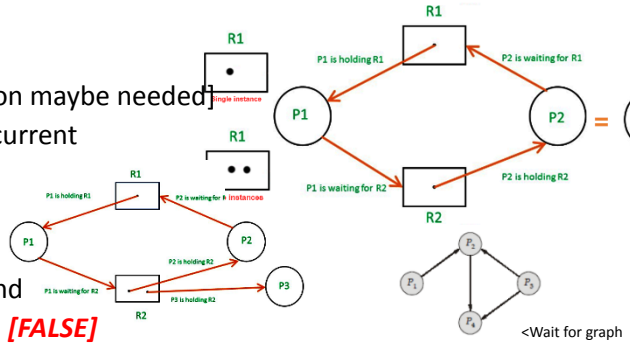
417. Progress: Every process apne marzi se 'CS' mein ja sakta hai

- Fail Case: Agar P1 'CS' gaya, toh hi P2 jaa payega [[***leads to Deadlock***; two process dependent on each other to go in]]

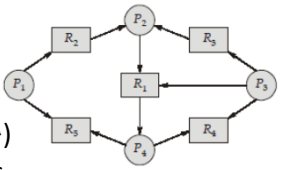
418. Bounded wait: No one waits infinitely to get into 'CS' :: ***Leads to Starvation***

419. If deadlock found, no need to check for bounded wait/progress, or if all exists, prefer deadlock

1. Segmentation requires memory compaction. [Since external fragmentation possible compaction maybe needed]
2. Each resource in system has *single instance* and current resource-allocation graph contains cycles ***[TRUE]*** cycle is cyclic of the process graph after removing resources from graph.
3. Each resource in system has *multiple instances* and
4. current resource-allocation graph contains cycles ***[FALSE]***



<Wait for graph

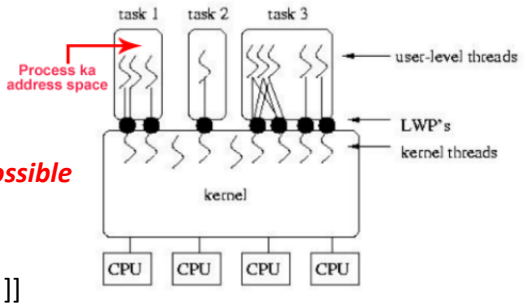


5. Wait-for graph of the Resource allocation graph does not have nodes for resources (See the up diagram ^)
6. Frame table maintains allocated frames and also free frames, If a frame is allocated, then frame table has an entry for process id to which it allocated.
7. Frame table can be accessed and modified only by operating system.
8. Relative path name and absolute path name for a file or directory ***may be*** same. (an absolute path refers to the same location in a file system relative to the root directory, whereas a relative path points to a specific location in a file system relative to the current directory you are working on) (abs: C/user/desktop/img.png; rel: abs: …*/desktop/img.png*)
9. Sharing of file or directories is allowed in acyclic-graph directories, but not allowed in tree structured directories. (In graph two nodes can lead to same node, not in tree)
10. An algorithm for searching a file or directly in general-graph directories structure may lead to infinite loop of searching in cycle and never terminate. [Graph may have cycles]
11. In acyclic-graph directories structure, a file may have multiple absolute path and different file names can refer to same file.
12. **In local page replacement policy: Trashing of one process will not affect others [FALSE]** [points: each process has separate page table, round robin is used, TLB is flushed after premption (TLB is a shared hardware)] BUT [a program has many lines of codes, will need memory frames, size of a frame >> size of page table, if many programs comes, will face scarcity of memory to place frames, not page table. Depending on amount of MM a prog. gets, only those many pages can be mapped, so if more programs, less MM space, we'll have less pages mapped, hence will lead to page fault, this is why increasing degree of multi programming, increases trashing. If a page fault occurs, even in case of a premption, the TLB is flushed, but we need to update the page tables, this request goes in a queue to MMU, now if next process also gets page fault, it's request will stand in a queue, so TLB overhead, will slow down the entire system.
13. If a process is in thrashing then in ***global replacement*** it can get frames from other processes. [local m khud pe dependent, no help from others, global m help leta hai]
14. In page-fault frequency strategy, page-fault rate can go beyond decided lower and upper bound. But, when exceed upper bound then allocate new frames to process and if go beyond lower bound then remove frames from process.
15. Fork creates exact replica of parent, which does not share the address space of parent, even has their own global variables.
16. By default join is a conditional join, and natural join is a conditional join with condition R1.c1=R2.c1 [Join does AND operation]
17. For Multivalued Dependencies we need to create a extra table, we can avoid the join in 1NF, as redundancy is not an issue here.
18. Used to evaluate indexing: Access time, access type, insertion time, space overhead (B/B+ tree indexing)
19. A primary key is the field in a database that is the primary key used to uniquely identify a record in a database. A secondary key is an additional key, or alternate key, which can be use in addition to the primary key to locate specific data
20. (Secondary Key ≠ Secondary Index)

21. PC is a binary counter, it may increment when a pulse is applied to its COUNT UP input. {PC increment do not need local bus support}
22. Dead configuration: a situation where a state do not have a transition for an input symbol. DFA have transitions for all inputs on all states, but may not in NFA. (Not equal to dead state)
23. Single tape TM, multi tape TM are equally powerful; TM has atleast two states. {read FC what cannot reduce TM power}
24. DCFL is regular: ***Decidable*** :: CFL is regular: ***Undecidable***
25. Intersection with regular, Inverse homomorphism:: Closed for all Regular, DCFL, CFL,CSL, REC, RE
26. A is reducible to B, B is reducible to C then, if C is decidable, A ∩ B is decidable [[Since A∩B is polynomially smaller problem than C]] , { Not this: conditions are getting intersected, increasing conditions} {This: the intersection of two problem is a smaller problem}
27. TM: Can the tape alphabet be the same as the input alphabet? ***No***, I/p: 00110, Tape: B00110B (Blank to mark the end can only be in tape alphabet, not the i/p string). Can the 'B' of i/p tape be printed as o/p? ***YES***, anything can be printed using TM
28. Join is commutative and associative (DBMS)
29. Only dirty read is rolled back in cascading rollback [[W1(A), R3(A) :: T1 rolls-back, T3 will also roll back, cause of dirty read]], lost update is not rolled back, as anyway we're updating the value and it doesn't matter if we loose an intermediate update [[W1(A), W3(A) :: T1 rolls-back, T3 will not]]
30. Complement of CFL can be CFL
31. WW^r is CFL (Even Length Palindromes) :: It's compliments are:
    a. Odd length string {Regular}
    b. Even length string which are not palindrome {Can be made by PDA, accepting upto when stack is not empty, rejecting the stack empty state}
32. ***EXCEPT/MINUS command in SQL removes duplicates << ** IMP ****
33. Sign extension is not used in floating point multiplication, booths multiplicand method is used
34. Booths multiplication is used both for +ve and -ve numbers. Provides excellent result for strings with strings of 0's and 1's since in booth multiplicand 0 0 = 0 and also 1 1= 0.
35. ***In the case of direct mapping, there is no requirement for a replacement algorithm***. It is because the block of the main memory would be able to map to a ***certain specific line of the cache only*** (MM loc % no.of lines). In full associative, we can fill to any line that's free, and once full, we need to decide, who to replace, in set- associative, we have done questions of replacing. Direct:(MM loc % no.of lines) :: Fully associative: Any line :: Set-Associative: (MM loc % no.of sets)
36. ***Non preemptive kernel is free from race around condition***
37. List of free frames is maintained by OS, not PCB. PCB stores:
    a. Page table base address and length of page table register, process state, Gen purpose reg values
    b. I/O devices allocated to process, memory management info
    c. Amount of CPU used by process. [CPU scheduling info]
38. In fork(), parent can run parallelly with child, or may even wait for child to complete and then complete itself, Child process can be loaded with program other than parent program. [[suppose if we use (if fork()==0 {execute Process P2();}) only child will execute this, so it can load program other than what parent program loads, as, parent will not load this program. Child process may/maynot be exact duplicate of parent, may do extra/less stuffs.
39. Mutex lock is a software tool for ***solution*** to critical-section problem. Mutex ensures that only one thread has access to a critical section or data by using operations like a lock and unlock. Deadlock is not yet solved.
40. Suppose MS Word as a **process**, Insert, design format, all operations of word are **threads** **[Thread uses address space of the process]**
41. ***User level threads are application specific and independent of kernel.*** User-level threads only managed by ***user space (created in the address space of user)*** and Kernel is not aware about this. So, a single process can get only a single processor. And one processor can execute one thread only. ***More than one user-level threads of a process cannot run concurrently.***
42. ***User threads*** are the threads created by the user with help from a user library and are visible just to the creating process and it's run time environment (***the kernel has no idea about the creation of these threads***). User threads just stay in the address space of the creating process and are run and managed by the creating process without kernel intervention i.e., any problems with the execution of these threads are not kernel's headache.
43. ***User level threads are transparent to the kernel: [TRUE] Not forever invisible, as we make thread table after a process gets in kernel (CPU), once thread comes into execution it is visible. {transparent == visible}***
44. ***Kernel threads*** on the other hand are created by the kernel and are visible to it. A user process with the help of a provided library asks kernel to create an executable thread for that process and the kernel in turn creates the thread on behalf of the process, and puts it onto it's list of the available executable threads present. Here the creation, execution and management of the thread is taken care of by the kernel.

45. In a nutshell user threads need to be mapped to kernel threads because it's the kernel that schedules the thread for execution onto the CPU and for that it must know about the thread that it is scheduling
46. The CPU scheduler in the kernel schedules threads onto the CPU for execution, but the catch here is that the scheduler being part of the kernel knows only about the kernel level threads because, the kernel has no idea about the existence of user threads since they are created in the address space of the creating process, hence the kernel has no control over them. The kernel level threads are scheduled to the CPU.
47. Multi-level threading: (Kernel threads are created by the CPU) (User threads are created by process/application)
    a. ***Many to One:*** Many user threads are mapped to a single kernel thread {No parallelism/concurrency, cannot used in multiprocessor system architecture} We have one processor/kernel.
    b. ***One to One:*** Each user thread is mapped to a separate kernel thread. ***Solves the problem of parallelism/concurrency***, can be used in a multiprocessor architecture. ***If one thread is blocked, remaining threads can be still used***. Problem: If we create more number of threads, CPU performance is degraded, as we will need so many kernel threads.
    c. ***Many to Many:*** Many user threads are mapped to ***equal/less*** kernel threads. [BEST APPROACH]

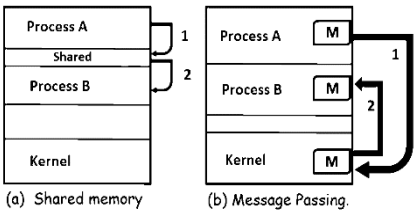
48. A system is using many-to-one pure user-level thread model in which Kernel is not aware of any user-level thread. The following situations/states is/are possible:



- P is in running state and no thread of P is in running state.
- P is in block state and a thread of P is in running state.
- P is in ready state and a thread of P is in running state.
- P is in running state & more than one threads of P are in running state << ***Never possible***

49. In fixed partitioning: One process in one block [[Internal Frag]]
50. In variable partitioning: Multiple processes are allowed in one block [[External Frag]]
51. Two or more processes can concurrently execute the same program. [Swiggy, Zomato, Uber runs code to access location], also think of fork call(), the codes are same but one is in child memory space other in parent's
52. Class ***inheritance*** is a way for one class to extend another class. So we can create new functionality on top of the existing. When ***fork( )*** system call creates a ***child process***, then child process ***inherits*** all the open file descriptors of ***parent*** process. [Makes an exact copy of parent] ***[[Inherits/copies are correct ; shares is wrong]] [share bole toh, code ek location mein pada hua hai and both are using them, running same code matlab same hai but not in same location, Swiggy and Zomato wanting locating access, both are executing the same code, but not sharing the same code, Zomato ke memory space m bhi location access ka code hai, swiggy m bhi hai, the code is technically the same, but are in different program memory space]***
53. User-level thread switching do not require user-mode to Kernel-mode conversion of their process.
54. PCB is a doubly linked list, stored in Main Memory.
55. Threads of a process share the protected access to other processes which are connected to process of these threads as inter-process communication.
56. Shell is an application who helps to communicate with the kernel. When we write something in the shell, it is considered as a file name and the current directory is searched for the file name, if present, is opened as a program. ***Exec()*** is a system call, it is called as exec(VLC), calls VLC and terminated the Shell, to avoid this, fork is made. In shell, int k=fork(), if(k==0){ exec(VLC)} :: here the child calls the VLC, and suicides itself, leaving the parent shell alive, which goes on spawning childs and killing them with time.
57. Calling of exec() will change: (i) PC of Kernel stack (ii)Page table entries, will not change (i) Parent's PID [^ Shell is the parent]
58. exec( ) system call used to ***replace the process's memory space*** with new program. New memory space will lead to new program counter, now page tables, etc. Process inside a process will flush page tables.
59. Initially for communicating within processes, Pipe() was used, which was poor in data transfer concurrency, as after a process executes, other would be able to use it. So, later sharing of some chunks of memory, say a register within the address space of a process, was used. [Shared resource==critical section]. This helped user level thread to share data among themselves, reducing the need of pipe(), making data sharing concurrent. But we are trading security for this.
60. User level thread, if calls exit, will lead to termination of entire process, as exit() is a system call, so if this reaches the kernel, the kernel will assume, the entire process wants to exit. This is why a thread level exit, exists thread_exit(). Now we also may have a scheduler at the application level. Context switching @user level is of no use, as we aren't executing anything in the user level, so it's fast. ***There is no concurrency in user level threads. If these threads passes to the kernel level, they are mapped to kernel threads, and if we have a 1-1 or 1-M architecture, single process ka multiple threads (kernel thread) can execute together.***
61. User-Level threads are managed entirely by the run-time system (user-level library) :: Run of a user level thread is limited to context switch, and thread scheduling.

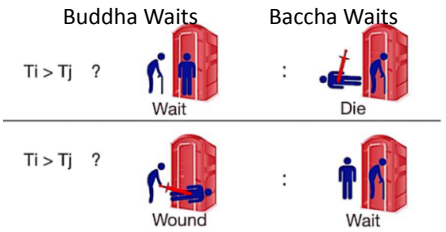
62. ***User level thread switch do not need help of the kernel.***
63. ***If two processes writing same location in shared memory, then it can lead to race condition. So, simultaneous write should be avoided.***
64. User level: User Level Thread Lightweight Process Lightweight Processes Schedules User Level threads to the Kernel
65. Kernel level: Kernel Level Thread CPU Scheduler CPU Scheduler Schedules Kernel Level threads to the Processor
66. If a process is I/O bound, a LWP may be blocked in an I/O operation. So we may need multiple LWP's
67. **LWP has a single 'kernel level thread' attached to it, it acts as an interface b/w User level threads and kernel level threads, you can think it as a micro controller of the User level. Application developer encodes it according to needs. All context switch of user level and scheduling at user level is handled by LWP code, The microcontroller code is kinda called here as 'user level thread library'.**
68. If a process P is in running state, and user level context switching is taking place of two threads using the user level thread library, and round robin time slice expires, the Process will not wait for the context switch to complete, immediately next process will come in and P will move to wait state. The program P will start from exact same point in next time slice. ***So, a user level thread executes only if the process is in running state.***
69. ***More than one user-level threads of a process can run concurrently False. Concurrency a problem in user threads [pure User Level Threads: Process cannot take advantage of multi-processing, one process always mapped to one Kernel thread***
70. User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed. User-level threads can be run on any operating system. There are no kernel mode privileges required for thread switching in user-level threads.
71. Kernel level threads are OS specific.
72. A process in user mode cannot execute privileged instructions. (if an attempt to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treat it as illegal and traps it to the operating system) (The Instructions that can run only in Kernel Mode are called Privileged Instructions) Ex: Halt instructions, Turn off all Interrupts, Set the Timer, Context Switching.
73. Exit(), Block() by system call always leads to context switch. Timer not always leads to context switch (may be timer timeout, just places a process in ready queue. If a process finishes I/O it comes to ready state, not always causing immediate premption/context switch. [[Round Robin timer, prempts]]
74. Priority inversion is a solution to starvation, as if we give priority to big timestamp(newer) and new programs keeps on coming, old programs fall into starvation, so we invert the priorities, to deal with starvation.
75. Suppose only a process P1 is executing, using Round Robin scheduling. After the time slice expires, the same process comes back into execution. This could be considered as: (Time Slice of RR is smaller than burst of P1) [This is both context switch, and a premption as: ***(In computing, preemption is the act of temporarily interrupting an executing task, with the intention of resuming it at a later time)***
76. ***Multi-level Queue Scheduling***: Scheduling for group of processes. Foreground needs to be quick (lower response delay) as it interacts with user, and background may be slower. Multi-level scheduling partitions the ready queue into several separate queues, maintained for different group of processes [they cannot change queue, once placed] . Each queue will have it's own scheduling algorithm. [Starvation possible]
    Foreground: RR is best (as we need better response time), Background: FCFS will also work. [[Topmost queue has highest priority among all queues]] – [[there is also scheduling among the separate queues themselves]]
77. ***Multi-level Feedback Scheduling:*** In ***Multi-level Queue Scheduling***, once a process gets into a queue, it is not taken out of it, till the end, If P1 was in Q1 and P2 was in Q2, they remain in the same queue till the end. But here, we can jump b/w queue if needed. [No starvation]
78. ***Inter-process Communication: [pipe() b/w processes & shared chunk of memory]***
    - Shared Memory: Shared Chunk of memory, shared among processes, P_A will write in the shared memory and P_B will read it. Only one system call, less load on OS.
    - Message passing: Messages transferred b/w processes via kernel, P_A will send message to kernel and kernel will forward it to the P_B, Multiple system calls, more burden to OS, better for smaller messages.



79. Splitting the memory into separate data memory and instruction memory reduces impact of Structural hazards, not data.
80. A collection of tracks on various surfaces is called cylinder, need not necessarily say relative to one particular track
81. The Word count register of DMA is controller is the amount of data transferred in one DMA cycle. After every DMA cycle, the CPU is disturbed to ask about further operation, and no of DMA cycles decides how many time DMA takes control over the system bus. ***If Word count register is of 16 bits and memory is Byte addressable, 2^{16}B of data can be sent in one DMA cycle***
82. How to overcome Structural Hazard: IF and MA trying to access memory at same time (Memory conflict/Structural Hazard), what we do is, make a separate memory for data and opcode, so MA goes to data memory and IF goes to opcode memory, so duplication of resource, helps to deal with resource conflict.

83. FIFO and second chance replacement algo falls under Belady's Anamoly
84. $a^n b^n c^n$ is a CSL, {We put two a's for a 'a' in stack and pop out one if 'b' comes and one if 'c' comes **<<WRONG**} Standard CSL reason: what if 'abcabc' comes? Even this is not CFL: {(a+b+c) |a|=|b|=|c|} as ordering is not guaranteed, they are CSL
85. Concardination is placing a string behind other. If L1 is Regular and L2 is CFL, L1.L2 is always Regular: False {Ex. $\phi.S1=\phi$}
86. Relational Algebra: Always Finite and Procedural, TRC: Unsafe and non-procedural and may be infinite
87. Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected (Ex. Not Nul, Primary key, Foreign Key,etc)
88. Tag and Set is used to define a block, if both are same and offset differs, both are actually same block, but different byte (at a different offset)
89. View Serialisable: Initial Read (IR), Update Read (UR) & Final Write (FW)
    S1: R1(A) [IR], R2(A) [IR], R3(A) [IR], W1(A), W2(A), W3(A) [FW]
    S2: R1(A) [IR], W1(A), R2(A) [UR], W2(A), R3(A) [UR], W3(A) [FW]
    Read is for all transactions, write is end specific.

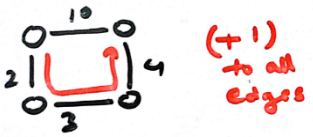
Bahar Waiting one is requesting ↓



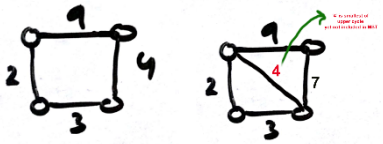
90. Rice's Theorem says that a trivial problem is always Decidable. Ex. "M is Turing Machine and is the only Turing Machine which accepts L(M) is decidable" is a ***ALWAYS FALSE*** statement, so it is **decidable**. As we can go on adding redundant states behind the actual TM, so a given TM can never be the only one to solve the problem. Always TRUE or Always FALSE are trivial statements.
91. Halting problem, and State-entry problem of TM is a is semi-decidable problem.
92. {ε} ∈ L(M), where M is a TM, is decidable: False :: (ε) may or maynot be part of a language, so TM may halt while finding it.
93. TM has odd number of states: Decidable [[Always True, go on adding redundant states until decidable]]

FALSE

1. All ε productions can be removed from any context-free grammar by suitable transformations "Cannot be removed if there is ε in the language itself"
2. rank(AB) = rank(A) rank(B)
3. det(A + B) ≤ det(A) + det(B)
4. Recursive programs are efficient (not always)
5. Binary search using a linear linked list is efficient (Not efficient for link list, as to find mid-element, we need O(n) time.
6. Every regular grammar is LL(1) { A regular grammar can also be ambiguous also, And LL(1) parses only unambiguous grammar }
7. The LALR(1) parser for a grammar G cannot have a reduce-reduce conflict if the LR(1) parser for G does not have reduce-reduce conflict.
8. The symbol table is accessed only during the lexical analysis phase.
9. Data flow analysis is necessary for run-time memory management.
10. Binary search tree can be constructed from max heap in θ(n)
    (Sort all elements of Heap: nlog(n) : Inorder BST)



11. G is a weighted connected undirected graph with distinct positive edge weights.
    a) If every edge weight is increased by the same value, then "Shortest path between any pair of vertices does not change".
    b) Shortest path between any two vertices of G is always unique



12. G is undirected simple graph in which each edge has a distinct weight, and 'e' is a particular edge of G, then "If 'e' is the lightest edge of some cycle in G, then every MST of G includes 'e' ."
13. Automatic garbage collection is essential to implement recursion (Garbage Collection: Objects no loonger being used are freed)
14. Let G be a connected undirected weighted graph "There exists a minimum weight edge in G which is present in every minimum spanning tree of G."
15. A station stops to sense the channel once it starts transmitting a frame in an Ethernet local area network.
16. IEEE 802.11 MAC protocol, the RTS-CTS mechanism is used for collision detection (no collusion detection in Wireless)
17. The packet source cannot set the route of an outgoing packets; the route is determined only by the routing tables in the routers on the way (Circuit Switching)
18. If the sequence number of a segment is m, then the sequence number of the subsequent segment is always m+1 (In TCP, may be sent out of order, but gets ordered at receiver's port before going to app. layer)

19. The size of the advertised window never changes during the course of the TCP connection (Window size in TCP is dynamic)
20. TCP contains message streams (It is a byte stream protocol) [True for TCP: Byte oriented, Full-Duplex, no multi-casting, suffers from silly window syndrome (problem with very small data)]
21. A connected UDP socket can be used to communicate with multiple peers simultaneously ***(The distinction is between connected and unconnected sockets. An unconnected socket can be used to communicate with any host; but a connected socket, because it has a dedicated destination, can send data to, and receive data from, only one host)***
22. A process cannot successfully call connect() function again for an already connected UDP socket (A process with a connected UDP socket can call connect() again for that socket for one of two reasons:
    a. To specify a new IP address and port
    b. To unconnect the socket
23. TCP destination port can be determined by analysing only the second fragment. (is incorrect because, when reassembly is done at Receiver, both fragments become a single TCP segment. based on that, destination port is found) (The protocol field in the IP header is an 8-bit number that defines what protocol is used inside the IP packet. Like TCP,UDP,ICMP…) (The port number is "tacked on" to the end of the IP address, for example, "192.168. 1.67:80" shows both the IP address and port number (i.e. port number is on the payload, an only be decoded after merging all segments at destination.) (Sniffing in an intermediate router, we can psiphon away all the segments, and can learn about the destination/source port)

24. A router does not modify the IP packets during forwarding (TTL, header checksum, etc)
25. A router should reassemble IP fragments if the MTU of the outgoing link is larger than the size of the incoming IP packet.
26. Pre-processor allows the user to use header files and it links all the object modules as single module. (Pre-processor allows the user to use header files but it dont link the object modules as single module. It will be done by link editor.)
27. If inorder traversal of a binary search tree T is given then it is possible to create T.
28. ICMP error reporting messages are used for error handling and debugging network problem (we can detect error using ICMP but not handle it (i.e correct it))
29. If graph G has negative weight cycle then Bellman-Ford algorithm will always report for negative weight cycle (If not reachable from source)
30. LALR(1) parser is more powerful than LL(1)
31. Lossless, Decomposition preserving decomposition into BCNF is always possible.
32. A relation scheme can have at most one foreign key.
33. A foreign key in a relation scheme R cannot be used to refer to tuples of R.
34. Relational algebra is more powerful than relational calculus
35. Relational algebra has the same power as relational calculus
36. ***(TRUE)*** Relational algebra has the same power as safe relational calculus ***(TRUE)***
37. If all attributes of a relation are prime attributes, then the relation is in BCNF (FD: AB □C, PA(A,B,D), key= ABD)
38. BCNF decompositions preserve functional dependencies
39. An SQL query automatically eliminates the duplicates
40. An SQL query will not work if there are no indexes on the relations {Index: clustered, primary, secondary}
41. SQL permits attribute names to be repeated in the same relation
42. Any implementation of a critical section requires the use of an indivisible machine- instruction ,such as test-and-set (We can use monitor also)
43. virtual memory makes faster access to memory on an average {Paging does, not virtual memory}
44. Every nondeterministic PDA can be converted to an equivalent deterministic PDA ***{Don't confuse this with DFA}***
45. Within an instruction pipeline an anti-dependence always creates one or more stalls. (WAR) {Not always, operand forwarding is used to avoid 'em}
46. All ∈ productions can be removed from any context-free grammar by suitable transformations {If the language itself has ∈ productions, nothing can be done, ex $(0+1)^* = \{\in,0,1,01,11,\ldots\}$ }
47. The flags are affected when conditional CALL or JUMP instructions are executed (Flags are tested during conditional call and jump not affected or changed)(effected during The ALU Operation only) [JMP NZ; NZ flag will be tested only]