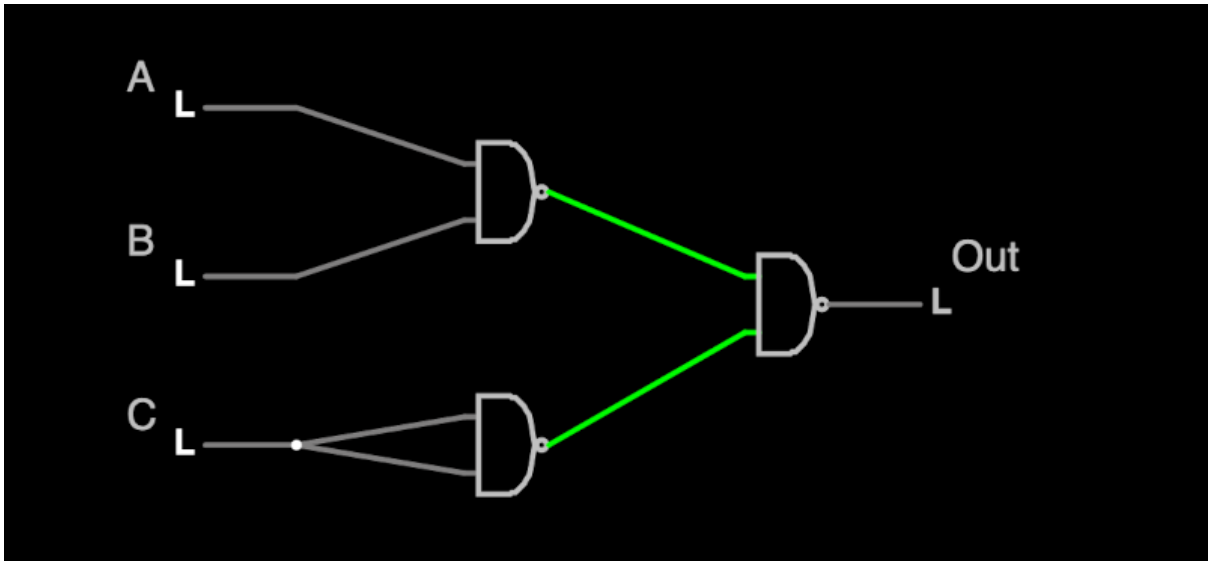
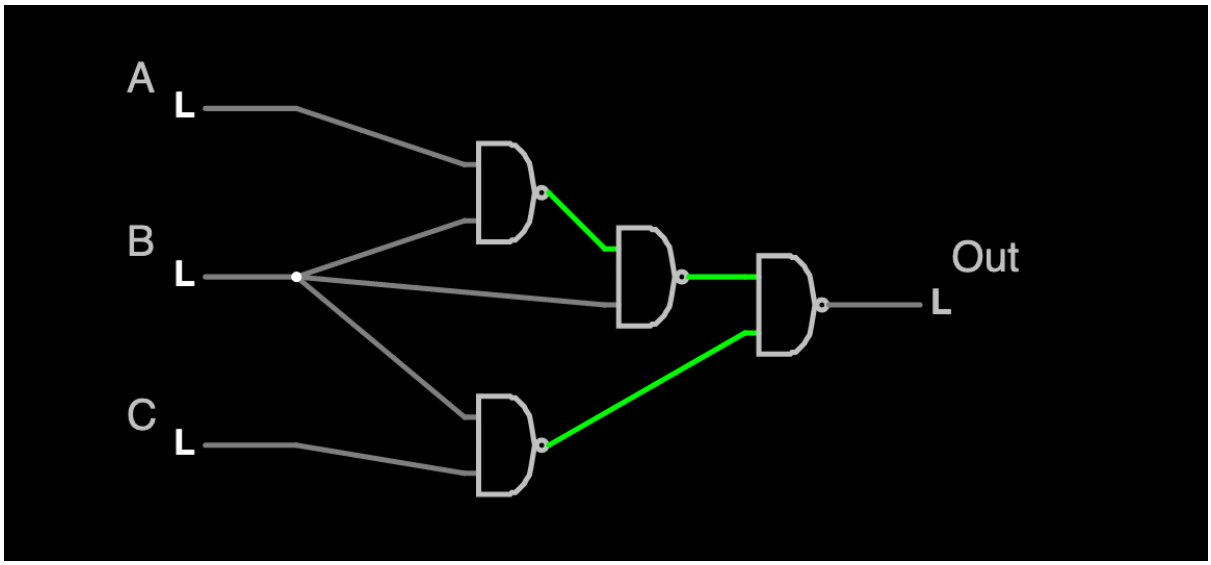
Q1) Write truth tables for the following circuits:
Circuit 1:



Answer:
Truth table

A	B	C	A NAND B	NOT C	(A NAND B) NAND (NOT C)
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	1

Circuit 2:



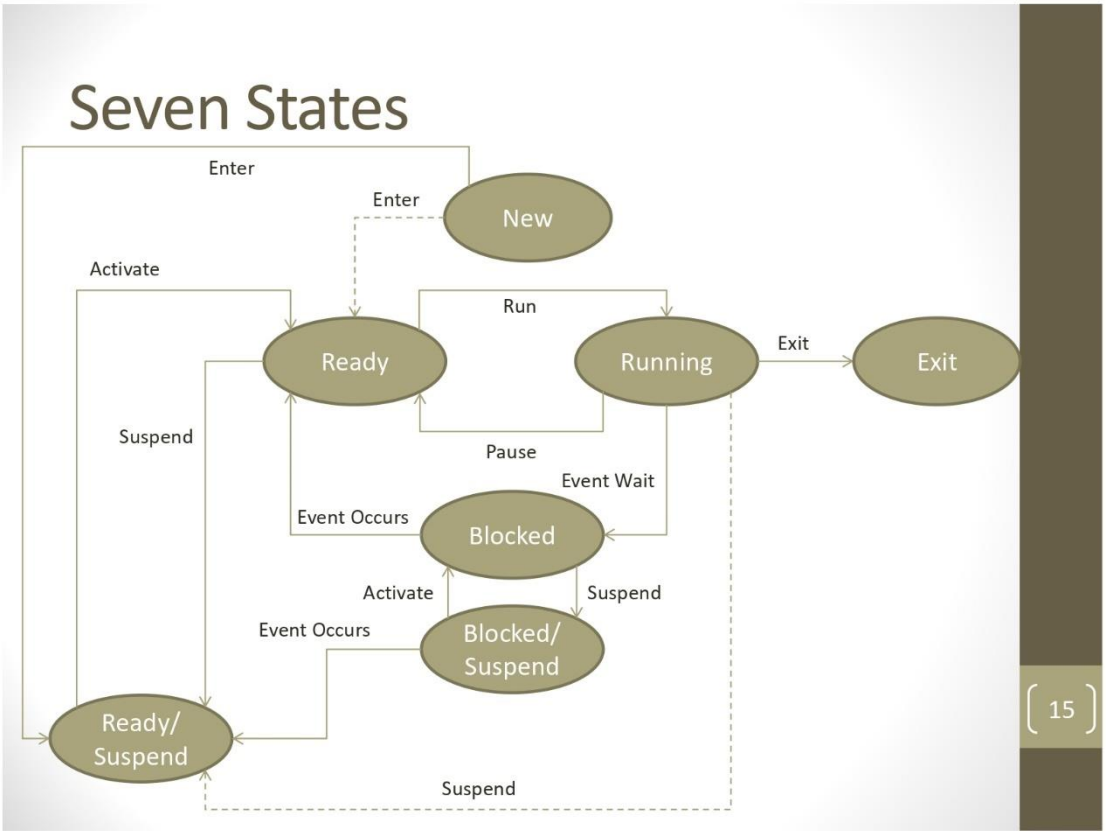
Answer:

Truth table

A	B	C	A NAND B	(A NAND B) NAND B	B NAND C	((A NAND B) NAND B) NAND (B NAND C)
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	0	1
1	0	0	1	1	1	0
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	1	0	1	0	1

Q2) Using the 7-state process model introduced in lecture, describe each state and the transitions between them. Explain how this model developed from the simple FCFS model implemented on batch systems.

Answer:



The 7-state process model is an advanced representation of process management in OS, designed to efficiently handle multiple processes by detailing various states and transitions. Here's a description of each state and the transitions between them:

1. **New**:
    - **Description**: A process has been created but not yet admitted to the ready queue.
    - **Transition**:
        - New -> Ready: When the operating system admits the process for execution.
        - New -> Ready/Suspend: When the operating system admits the process for execution, but the process is temporarily suspended due to other higher priority processes.
2. **Ready**:
    - **Description**: The process is loaded into memory and is waiting to be assigned to a CPU for execution.
    - **Transitions**:
        - Ready -> Running: When the scheduler selects the process to run.
        - Ready -> Ready/Suspend: If the process is temporarily suspended due to other higher priority processes.
3. **Running**:
    - **Description**: The process is currently being executed by the CPU.
    - **Transitions**:
        - Running -> Exit: If the process completes its execution.
        - Running -> Ready: If the process is paused by the scheduler to allow another process to run.
        - Running -> Blocked: If the process needs to wait for an event to occur.
        - Running -> Ready/Suspend: If the process is suspended.

4. **Exit**:
    - **Description**: The process has finished execution and is being removed from memory.
    - **Transitions**:
        - There are no transitions from Termination as the process lifecycle ends here.
5. **Blocked**:
    - **Description**: The process cannot continue until a certain event occurs, such as the completion of an I/O operation.
    - **Transitions**:
        - Blocked -> Ready: If the waiting event occurs.
        - Blocked -> Blocked/Suspend: If the process is suspended while waiting.
6. **Blocked/Suspend**:
    - **Description**: The process is suspended and waiting for an event to occur.
    - **Transitions**:
        - Blocked/Suspend -> Blocked: If the suspension is lifted but the process still needs to wait.
        - Blocked/Suspend -> Ready/Suspend: If the waiting event occurs.
7. **Ready/Suspend**:
    - **Description**: The process is in memory and ready to run but is currently suspended.
    - **Transitions**:
        - Ready/Suspend -> Ready: If the suspension is lifted and the process is ready to execute.

Development from First Come First Serve (FCFS)

The FCFS model is one of the simplest scheduling algorithms where processes are executed in the order they arrive. This model worked well for batch systems where there was a single queue, and processes were executed one after another until completion.

Limitations of FCFS:

- **Long Waiting Time**: Once a process started, it could not be stopped until completion, which could lead to inefficiency and long wait times for other processes.
- **Lower Device Utilization**: Long-running processes could monopolize the CPU, causing delays for other processes.
- **Favours CPU over I/O**: There was no mechanism to handle processes that were waiting for I/O operations or other events, leading to potential idle CPU time.
- **Not Ideal for Time-Sharing Systems:** The FCFS algorithm, being non-preemptive, could leave dependent systems idle and waiting, which is not ideal for running processes smoothly and efficiently because time-sharing systems need to get a share of the CPU at regular intervals.

(Referenced from: https://blog.flexis.com/advantages-and-disadvantages-of-fcfs-order-scheduling)

Evolution to 7 State Model:

- **Preemption**: Introduced the ability to suspend processes and switch between them, improving CPU utilization and responsiveness.
- **Process States**: Introduced various states to better manage processes, such as handling blocked and suspended processes.
- **Efficiency**: Allowed for better handling of I/O operations and multitasking, leading to more efficient system performance.
- **Flexibility**: Provided a more flexible and responsive system capable of handling a variety of process needs and improving overall system performance.

The transition from FCFS to the 7-state process model represents a significant evolution in process management, aimed at optimizing system resources, improving responsiveness, and providing a more efficient and flexible operating environment.

Q3) Five batch jobs, A through E, arrive in alphabetical order at a computer at almost the same time (i.e. they all arrive before the next scheduling event occurs). They have estimated running times of 100, 300, 400, 200, and 500 ms, respectively. Using diagrams, show how the process scheduling algorithms below would schedule these jobs, and calculate the mean process turnaround time (assuming no process switching overhead):
1) FCFS (First Come First Serve)

Answer:

0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400ms	Wait	Turnaround
A															0	100
	B	B	B												100	400
				C	C	C	C								400	800
								D	D						800	1000
										E	E	E	E	E	1000	1500

Here,
Mean Process Turnaround Time = (100+400+800+1000+1500)/5=760ms

2) SPN (Shortest Process Next)

Answer:

0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400ms	Wait	Turnaround
A															0	100
			B	B	B										300	600
						C	C	C	C						600	1000
	D	D													100	300
										E	E	E	E	E	1000	1500

Here,
Mean Process Turnaround Time = (100+600+1000+300+1500)/5=700ms

3) RR with a quantum of 100ms

Answer:

0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400ms	Wait	Turnaround
A															0	100
	B				B				B						100	1000
		C				C				C		C			200	1300
			D				D								300	800
				E				E			E		E	E	400	1500

Here,
Mean Process Turnaround Time = (100+1000+1300+800+1500)/5=940ms

4) RR with a quantum of 200ms

Answer:

0	100	300	500	700	900	1000	1200	1400ms	Wait	Turnaround
A									0	100
	B				B				100	1000
		C				C			300	1200
			D						500	700
				E			E	E	700	1500

Here,
Mean Process Turnaround Time = (100+1000+1200+700+1500)/5=900ms

Q4) Does the following pseudocode solve the mutual exclusion problem for two processes (0 and 1)? Why or why not?

```
flag = {false, false}
process p(i):
   while (flag[1-i]) {
      // do nothing
   }
   flag[i] = true
   while (flag[1-i]) {
      // do nothing
   }
   <critical section>
   flag[i] = false
```

Answer:

No, the pseudocode provided above does not solve the mutual exclusion problem. It's because, if p(0) and p(1) start at same time, then in first while loop i.e. while (flag[1-i]) result in false for both of them and so, both of them will set themselves as true. And then both p(0) and p(1) will enter the second while loop. And now in second while loop as both of them are true and they will continue to wait for one another which will result in both the process to be in waiting condition forever. This results in a deadlock where both processes are stuck waiting for each other. Hence, the pseudocode provided above does not solve the mutual exclusion problem.

Q5) Suppose a 20MB file is stored on a disk on the same track (track 20) in consecutive sectors and the disk arm is currently situated over track 28. Assume that moving the arm from one cylinder to the next takes 2ms, it takes 10ms for the sector where the file begins to rotate under the disk head, and the drive has a reading rate of 100MB/s. If there are no other disk requests, how long will it take to retrieve this file from the disk?

Answer:
Here,
No. of tracks to move = 28-20 = 8
Seek time = 8*2 = 16ms
Rotational Delay = 10ms
File Size = 20MB
Reading rate = 100MB/s
Then,
Date Transfer Time = 20MB/(100MB/s) = 0.2s = 200ms
So, the total time will be = 16ms + 10ms + 200ms = 226ms.
Hence, if there are no other disk requests, it will take 226 milliseconds to retrieve the given file from the disk.