1.

- Preemptive Scheduling: The operating system can interrupt and suspend an active process, even if that process has not finished its CPU burst, to assign the CPU to another method. This is usually done based on priority, time slicing, or when an I/O operation is needed.
- Non-preemptive Scheduling: Once a process is assigned the CPU, it retains control until it completes or voluntarily yields the CPU. The operating system does not forcibly remove the process from the CPU.

2.

- I/O-bound Programs: These programs spend more time performing I/O operations than utilizing the CPU. They tend to voluntarily yield the CPU often, as they frequently request I/O operations, allowing the OS to perform a context switch while waiting for I/O completion.
- CPU-bound Programs: These programs spend most of their time performing computations on the CPU and use I/O infrequently. The OS may need to forcibly preempt the process if time-sharing or priority scheduling is used, leading to nonvoluntary context switches.

3.

- Option 1: Each core has its ready queue
    - Advantages:
        - Since each core has its queue, there's less contention for the ready queue, leading to enqueue faster and dequeue operations.
        - A process assigned to a specific core may perform better since it can stay on the same core without being moved, reducing cache misses.
        - The system can better optimize task scheduling based on the characteristics of individual cores (e.g., cores with different speeds or power profiles).

    - Disadvantages:
        - Some cores may end up with more tasks than others, leading to load imbalances where some cores are underutilized, and others are overburdened.
        - If a core's ready queue becomes full while others are relatively idle, the system cannot easily shift tasks between cores, potentially leading to inefficiency.
- Option 2: A single ready queue shared by all cores
    - Advantages:
        - A shared ready queue allows processes to be distributed evenly across all cores, reducing the risk of underutilization or bottlenecking on any particular core.
        - Tasks can be dynamically assigned to whichever core is available, ensuring efficient use of all cores.
    - Disadvantages:
        - With multiple cores accessing the same queue, there could be contention, leading to delays in accessing the ready queue.
        - Processes may get moved between cores more frequently, leading to higher cache misses and reduced performance due to loss of cache data on core switches.

4.

a.

1. FCFS (First-Come, First-Served)
   | P1 | P2 | P3 | P4 | P5 |
   0 5 8 9 16 20

2. SJF (Shortest Job First)
   | P3 | P2 | P5 | P1 | P4 |
   0 1 4 8 13 20

3. Non-Preemptive Priority
   | P2 | P3 | P4 | P5 | P1 |
   0 3 4 11 15 20

4. RR (Round Robin with Quantum = 2)
   | P1 | P2 | P3 | P4 | P5 | P1 | P4 |
   0 2 4 5 7 9 11 16 20

b.

Process	FCFS TAT	SJFTAT	Priority TAT	RR TAT
P1	20	13	20	16
P2	8	4	3	4
P3	9	1	4	1
P4	16	20	11	20
P5	20	13	15	16

c.

Process	FCFS WT	SJFWT	Priority WT	RR WT
P1	15	8	15	11
P2	5	1	0	2
P3	8	0	3	0
P4	9	13	4	13
P5	16	9	11	12

d.

Average Waiting Time Calculation:

- FCFS: (15+5+8+9+16)/5=10.6ms
- SJF: (8+1+0+13+9)/5= 6.2 ms
- Non-Preemptive Priority: (15+0+3+4+11)/5=6.6 ms
- RR: (11+2+0+13+12)/5=7.6ms
- Minimum Average Waiting Time: SJF with 6.2 ms.

5.

a.

Utilization Factor Calculation: The utilization factor U for a set of periodic tasks is given by the formula:

$U=\sum(ti/pi)$

UP1=t1/p1=2550=0.5UP2=t2/p2=3075=0.4

Now, calculate the total utilization:

U=UP1+UP2=0.5+0.4=0.9

For RMS, the maximum utilization U for n tasks should not exceed:

$U\leq n \cdot (2^{(1/n)} -1)$

For n=2:

$U\leq 2 \cdot (2^{(½)}-1))\approx 0.828$

Since U=0.9>0.828, these processes cannot be scheduled without missing deadlines.

b.