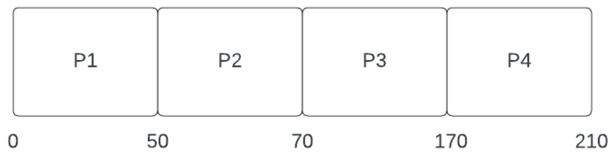
1. The concurrent execution of these two processes can result in one or both being blocked forever due to a deadlock situation. A deadlock occurs when each process holds a semaphore that the other process needs and vice versa. In this case, a deadlock can happen if the following sequence of events occurs:
    a. foo acquires semaphore S.
    b. bar acquires semaphore R.
    c. foo tries to acquire semaphore R but foo is blocked because bar holds R.
    d. bar tries to acquire semaphore S but bar is blocked because foo holds S.

    As a result, both processes are waiting for the semaphore held by the other which results in a deadlock, and neither can proceed.

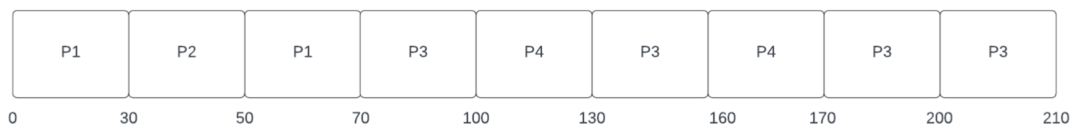
2.

3.
    a. **FCFS:**



Non-Preemptive Priority (Smaller Priority Number Implies Higher Priority):

Round Robin with Quantum 30 ms:



    b. **FCFS:**
    Avg. Waiting Time = (0 + 30 + 30 + 110) / 4 = 42.5 ms

    Non-Preemptive Priority (Smaller Priority Number Implies Higher Priority):
    Avg. Waiting Time = (0 + 30 + 70 + 10) / 4 = 27.5 ms

    Round Robin with Quantum 30 ms:

Avg. Waiting Time = (20 + 10 + 70 + 70) / 4 = 42.5 ms

c. **FCFS:**
Avg. Turnaround Time = (50 + 50 + 130 + 150) / 4 = 95 ms

Non-Preemptive Priority (Smaller Priority Number Implies Higher Priority):
Avg. Waiting Time = (50 + 50 + 170 + 50) / 4 = 80 ms

Round Robin with Quantum 30 ms:
Avg. Waiting Time = (70 + 30 + 170 + 110) / 4 = 95 ms

4.
a. The process can be scheduled and executed twice in a single round of the scheduler. This essentially allows the process to get a longer time slice or more CPU time in a single quantum compared to other processes.
b. The major advantage of this scheme is that it allows a specific process to have more consecutive time slices without being preempted. This can be beneficial in scenarios where a process requires a larger continuous chunk of CPU time, and frequent preemption might result in less efficient use of the CPU.
c. We can allocate longer quantum time for the high-priority processes

5.
a. Available = Total - Sum of Allocation = 12 12 8 10 - 9 9 6 9 = 3 3 2 1

b. CurrentNeed = Max - Allocation

Process	**Allocation**	**Max**	**CurrentNeed**
	A B C D	A B C D	A B C D
P0	2 0 0 1	4 2 1 2	2 2 1 1
P1	3 1 2 1	5 2 5 2	2 1 3 1
P2	2 1 0 3	2 3 1 6	0 2 1 3
P3	1 3 1 2	1 4 2 4	0 1 1 2
P4	1 4 3 2	3 6 6 5	2 2 3 3

c.

Process	Allocation	Available	CurrentNeed
	A B C D	A B C D	A B C D
P0	2 0 0 1	3 3 2 1	2 2 1 1
~~**P1**~~	~~3 1 2 1~~	~~5 3 2 2~~	~~2 1 3 1~~
~~**P2**~~	~~2 1 0 3~~	~~5 3 2 2~~	~~0 2 1 3~~
P3	1 3 1 2	5 3 2 2	0 1 1 2
P4	1 4 3 2	6 6 3 4	2 2 3 3
P1	3 1 2 1	7 10 6 6	2 1 3 1
P2	2 1 0 3	10 11 8 7	0 2 1 3

Safe State: P0, P3, P4, P1, P2

d.

Request (1, 1, 0, 0) <= Available (3 3 2 1) TRUE
Request (1, 1, 0, 0) <= Need (2, 1, 3, 1) TRUE
Need(i) = Need(i) - Request(i) = 2 1 3 1 - 1 1 0 0 = 1 0 3 1
Available = Available - Request(i) = 3 3 2 1 - 1 1 0 0 = 2 2 2 1
Allocation(i) = Allocation(i) + Request(i) = 3 1 2 1 + 1 1 0 0 = 4 2 2 1

Process	Allocation	Available	CurrentNeed
	A B C D	A B C D	A B C D
P0	2 0 0 1	2 2 2 1	2 2 1 1
~~**P1**~~	~~4 2 2 1~~	~~4 2 2 2~~	~~1 0 3 1~~
~~**P2**~~	~~2 1 0 3~~	~~4 2 2 2~~	~~0 2 1 3~~
P3	1 3 1 2	4 2 2 2	0 1 1 2
P4	1 4 3 2	5 5 3 4	2 2 3 3
P1	4 2 2 1	6 9 6 6	1 0 3 1
P2	2 1 0 3	10 11 8 7	0 2 1 3

Safe State: P0, P3, P4, P1, P2

e. Request (0, 0, 2, 0) <= Available (3 3 2 1) TRUE
Request (0, 0, 2, 0) <= Need (2, 1, 3, 1) TRUE
Need(i) = Need(i) - Request(i) = 2 2 3 3 - 0 0 2 0 = 2 2 1 3
Available = Available - Request(i) = 3 3 2 1 - 0 0 2 0 = 3 3 0 1
Allocation(i) = Allocation(i) + Request(i) = 1 4 3 2 + 0 0 2 0 = 1 4 5 2

Process	**Allocation**	**Available**	**CurrentNeed**
	A B C D	A B C D	A B C D
~~P0~~	~~2 0 0 1~~	~~3 3 0 1~~	~~2 2 1 1~~
~~P1~~	~~4 2 2 1~~	~~3 3 0 1~~	~~1 0 3 1~~
~~P2~~	~~2 1 0 3~~	~~3 3 0 1~~	~~0 2 1 3~~
~~P3~~	~~1 3 1 2~~	~~3 3 0 1~~	~~0 1 1 2~~
~~P4~~	~~1 4 5 2~~	~~3 3 0 1~~	~~2 2 1 3~~

Not Safe