SCHDULING POLICIES

1. What is SCHED_FIFO?

- **SCHED_FIFO** is a **real-time scheduling policy** in Linux.
- It stands for **First-In-First-Out**, and it is **non-preemptive among same-priority threads**.
- **Key characteristics:**
  1. **Priority-based:** Higher priority threads preempt lower priority threads.
  2. **FIFO within same priority:** Threads with the same priority run in the order they become runnable.
  3. **Non-preemptive among same priority:** Once a thread starts executing, it runs until it blocks, yields, or finishes.
  4. **Real-time policy:** It is meant for time-critical tasks, not regular user tasks.

2. How it works

1. Each thread is assigned a **priority** (1–99 in Linux, higher number = higher priority).
2. Scheduler always runs the **highest-priority runnable thread**.
3. If two threads have the same priority, they execute **in the order they became ready**.
4. The thread keeps running until:
   - It **blocks** (e.g., waiting for I/O), or
   - It **yields** voluntarily, or
   - It **finishes execution**.

Key Observations for SCHED_FIFO:

- Higher priority threads **preempt** lower-priority threads.
- Threads of same priority execute **FIFO**.
- Waiting time is affected by **preemption**.

1. Process Details:

Process	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	6	40
P2	2	8	60
P3	4	7	50
P4	5	3	60

Policy: SCHED_FIFO (higher priority preempts lower, same priority → FIFO)

2. Determine Execution Order:

Step by step:

1. **Time 0:** Only P1 has arrived → P1 starts.
2. **Time 2:** P2 arrives (priority 60 > 40) → **preempts P1** → P2 runs.
3. **Time 4:** P3 arrives (priority 50 < 60) → P2 continues.
4. **Time 5:** P4 arrives (priority 60 = P2) → **P2 continues** (FIFO: P2 arrived earlier).
5. **Time 10:** P2 finishes (ran 2–10) → choose next highest priority: P4 (priority 60) → P4 runs.
6. **Time 13:** P4 finishes → next highest priority: P3 (priority 50) → P3 runs.
7. **Time 20:** P3 finishes → remaining: P1 (priority 40) → P1 resumes.
8. **Time 24:** P1 finishes.

3. Completion Time (CT):

Process	CT
P1	24
P2	10
P3	20
P4	13

4. Turnaround Time (TAT)

TAT =CT−AT

Process	CT	AT	TAT = CT-AT
P1	24	0	24
P2	10	2	8
P3	20	4	16
P4	13	5	8

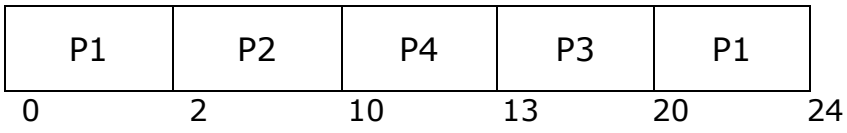
5. Waiting Time (WT)

Process	TAT	BT	WT = TAT-BT
P1	24	6	18
P2	8	8	0
P3	16	7	9
P4	8	3	5

6. Average Waiting Time (AWT)

AWT=(18+0+9+5)/4=32/4=8 ms

7. Gantt Chart:



Completely Fair Scheduler(CFS)

What is CFS?

- **CFS** stands for **Completely Fair Scheduler**.
- It is the **default scheduler for Linux** (for normal, non-real-time tasks).
- Goal: **Divide CPU time fairly among all processes**.

Key features:

1. **Fairness:** Each process gets a fair share of CPU time proportional to its weight (priority).
2. **No fixed time slices:** Uses **virtual runtime** instead of fixed quanta.
3. **Preemption:** Higher priority or "less virtual runtime" processes can preempt the current one.
4. **Red-black tree:** Linux maintains **all runnable tasks in a balanced tree**, sorted by **virtual runtime**.

2. How CFS Works:

1. Each process has a **weight** based on its **nice value** (priority).
2. **Virtual runtime (vruntime):** Time a process has spent on CPU **weighted by its priority**.
    - Lower vruntime → process has received less CPU → higher chance to run next.
3. The scheduler always picks the **process with the minimum vruntime**.
4. Over time, all processes get a **fair share of CPU**.

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	2	8
P3	4	7
P4	5	3

Step 1: Assume Target Latency

Target Latency (TL) = 12 ms

Step 2: Calculate Time Slice:

Time Slice=Target Latency/Number of runnable processes

Time Slice at Different Times:

Time	Runnable Processes	Time Slice
0	P1	12 / 1 = **12 ms**
2	P1, P2	12 / 2 = **6 ms**
4	P1, P2, P3	12 / 3 = **4 ms**
5	P1, P2, P3, P4	12 / 4 = **3 ms**

Step 3: CFS Scheduling (Using vruntime)

✓ **0 – 2**

- Only **P1** is available
- P1 executes

  0 ——— 2 → P1

✓ **2 – 8**

- Runnable: **P1, P2**
- Time slice = 6 ms
- **P2 has lower vruntime**
- P2 runs for 6 ms

  2 ——— 8 → P2

Remaining BT of P2 = 2 ms

✓ **8 – 11**

- Runnable: **P1, P2, P3, P4**
- Time slice = 3 ms
- **P4 has least vruntime**
- P4 runs completely (3 ms)

  8 ——— 11 → P4

✓ **11 – 14**

- Runnable: **P1, P2, P3**
- Time slice = 4 ms
- **P3 has lowest vruntime**
- P3 runs for 3 ms

  11 ——— 14 → P3

Remaining BT of P3 = 4 ms

✓ **14 – 16**

- Runnable: **P1, P2, P3**
- **P2 has lowest vruntime**
- P2 runs remaining 2 ms (finishes)

14 ——— 16 → P2

✓ **16 – 20**

- Runnable: **P1, P3**
- Time slice = 6 ms
- **P3 has lower vruntime**
- P3 runs remaining 4 ms (finishes)

16 ——— 20 → P3

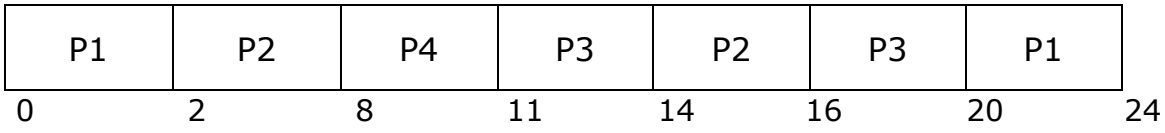
✓ **20 – 24**

- Only **P1** remains
- P1 finishes

20 ——— 24 → P1

Final Gantt Chart (Derived Using Time Slice)

Gantt Chart:



Process	Completion Time (CT)	Turn Around Time TAT = CT−AT	Waiting Time WT = TAT−BT
P1	24	24 − 0 = **24**	**18**
P2	16	16 − 2 = **14**	**6**
P3	20	20 − 4 = **16**	**9**
P4	11	11 − 5 = **6**	**3**