| | SVKM's NMIMS<br>School of Technology Management &Engineering |
|---|---|
| | Lab Manual - Operating System (702CO1C002 & 702CO0C056) |

| Year:- | Academic Year- 2025-26 | Semester:- |
|---|---|---|

**Experiment # 5**

**PART B**

| | |
|---|---|
| Roll No: D084 | Name: Somish Jain |
| Class: Btech CE sec B | Batch: 2 |
| Date of Experiment: 18 Aug | Date of Submission: 18 Aug |

**Code:**

1. **Preemptive**

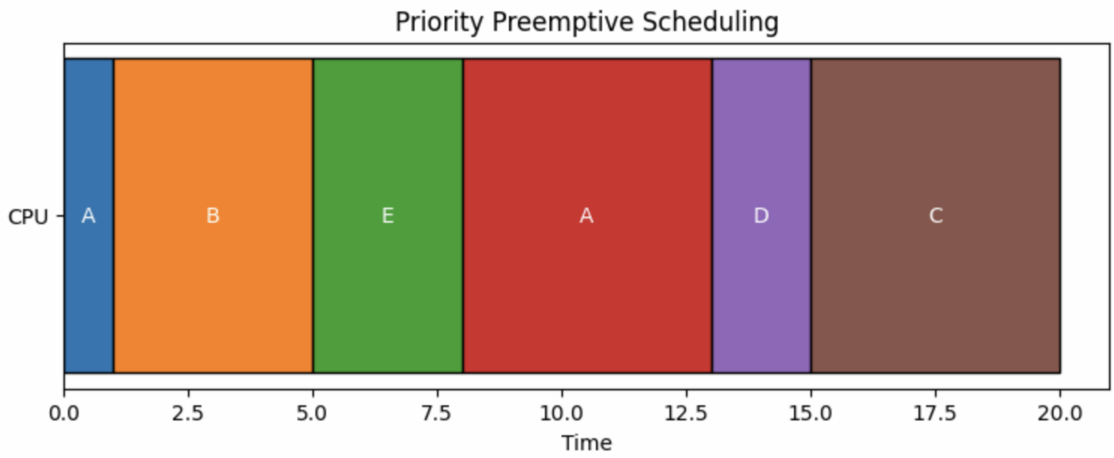
```
import requests
import pandas as pd
import matplotlib.pyplot as plt

API_URL = "https://jsonplaceholder.typicode.com/todos"

def load_processes(url):
try:
res = requests.get(url, timeout=5)
res.raise_for_status()
data = res.json()
if isinstance(data, list):
return [
{"id": "A", "arrival": 0, "burst": 6, "priority": 2},
{"id": "B", "arrival": 1, "burst": 4, "priority": 1},
{"id": "C", "arrival": 2, "burst": 5, "priority": 3},
{"id": "D", "arrival": 3, "burst": 2, "priority": 2},
{"id": "E", "arrival": 4, "burst": 3, "priority": 1},
]
return data
except Exception as e:
print(f"[INFO] Could not fetch from API: {e}. Using demo list.")
return [
{"id": "A", "arrival": 0, "burst": 6, "priority": 2},
{"id": "B", "arrival": 1, "burst": 4, "priority": 1},
{"id": "C", "arrival": 2, "burst": 5, "priority": 3},
{"id": "D", "arrival": 3, "burst": 2, "priority": 2},
```

```
{"id": "E", "arrival": 4, "burst": 3, "priority": 1},
]

def priority_preemptive_scheduler(process_list):
items = []
for p in process_list:
temp = dict(p)
temp["remain"] = p["burst"]
temp["first_seen"] = None
items.append(temp)

sequence = []
results = []
t = 0
while any(p["remain"] > 0 for p in items):
ready = [p for p in items if p["arrival"] <= t and p["remain"] > 0]
if not ready:
t += 1
continue
ready.sort(key=lambda x: (x["priority"], x["arrival"]))
current = ready[0]
if current["first_seen"] is None:
current["first_seen"] = t
start, end = t, t+1
current["remain"] -= 1
sequence.append((current["id"], start, end))
t = end
if current["remain"] == 0:
completion = end
tat = completion - current["arrival"]
wt = tat - current["burst"]
rt = current["first_seen"] - current["arrival"]
current.update({
"start": current["first_seen"],
"finish": completion,
"turnaround": tat,
"waiting": wt,
"response": rt
})
results.append(current)
return results, sequence

def show_table(data):
df = pd.DataFrame(data)
```

```
cols = ["id","arrival","burst","priority","start","finish","waiting","turnaround","response"]
print(df[cols])
print("\nAverages:")
print("WT:", df["waiting"].mean())
print("TAT:", df["turnaround"].mean())
print("RT:", df["response"].mean())

def gantt_chart(seq, title="Gantt Chart"):
fig, ax = plt.subplots(figsize=(9,3))
merged = []
for pid, s, e in seq:
if merged and merged[-1][0] == pid and merged[-1][2] == s:
merged[-1] = (pid, merged[-1][1], e)
else:
merged.append((pid, s, e))
for pid, s, e in merged:
ax.barh("CPU", e-s, left=s, edgecolor="black")
ax.text((s+e)/2, 0, pid, ha="center", va="center", color="white")
ax.set_xlabel("Time")
ax.set_title(title)
plt.show()

if __name__ == "__main__":
processes = load_processes(API_URL)
completed, seq = priority_preemptive_scheduler(processes)
print("=== Priority Preemptive Scheduling ===")
show_table(completed)
gantt_chart(seq, "Priority Preemptive Scheduling")
```

**Output:**

```
=== Priority Preemptive Scheduling ===
  id  arrival  burst  priority  start  finish  waiting  turnaround  response
0  B        1      4         1      1       5        0           4         0
1  E        4      3         1      5       8        1           4         1
2  A        0      6         2      0      13        7          13         0
3  D        3      2         2     13      15       10          12        10
4  C        2      5         3     15      20       13          18        13

Averages:
WT: 6.2
TAT: 10.2
RT: 4.8
```




2. Non preemptive

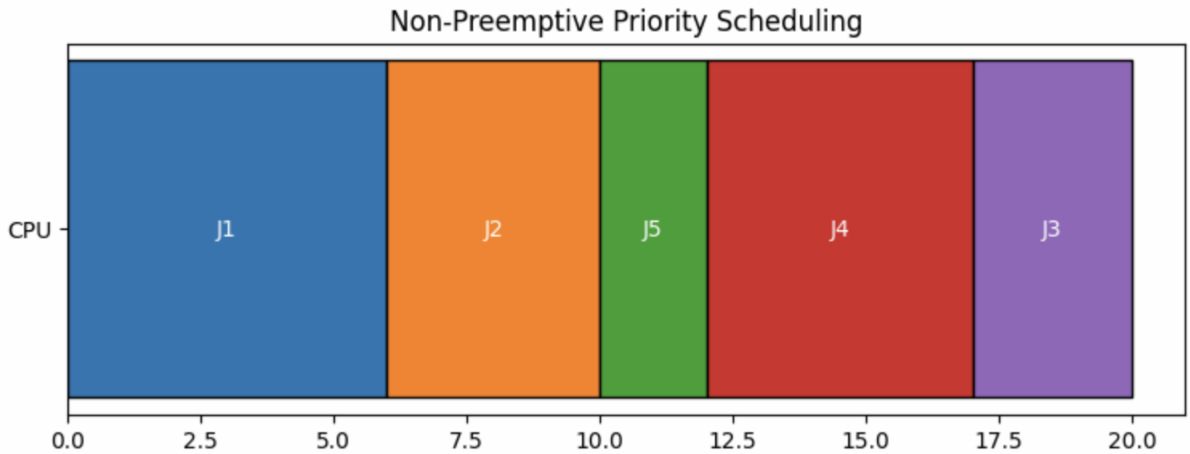
```
import requests
import pandas as pd
import matplotlib.pyplot as plt

API_URL = "https://jsonplaceholder.typicode.com/posts"

def load_jobs(url):
try:
r = requests.get(url, timeout=5)
r.raise_for_status()
data = r.json()
if isinstance(data, list):
return [
{"id": "J1", "arrival": 0, "burst": 6, "priority": 2},
{"id": "J2", "arrival": 1, "burst": 4, "priority": 1},
{"id": "J3", "arrival": 2, "burst": 3, "priority": 3},
{"id": "J4", "arrival": 3, "burst": 5, "priority": 2},
{"id": "J5", "arrival": 4, "burst": 2, "priority": 1},
]
return data
```

```
except Exception as e:
print(f"[WARN] API not usable: {e}. Using default job set.")
return [
{"id": "J1", "arrival": 0, "burst": 6, "priority": 2},
{"id": "J2", "arrival": 1, "burst": 4, "priority": 1},
{"id": "J3", "arrival": 2, "burst": 3, "priority": 3},
{"id": "J4", "arrival": 3, "burst": 5, "priority": 2},
{"id": "J5", "arrival": 4, "burst": 2, "priority": 1},
]

def priority_non_preemptive(jobs):
processes = []
for j in jobs:
task = dict(j)
task["done"] = False
processes.append(task)

chart, completed = [], []
t = 0

while not all(p["done"] for p in processes):
ready = [p for p in processes if p["arrival"] <= t and not p["done"]]
if not ready:
t += 1
continue
ready.sort(key=lambda x: (x["priority"], x["arrival"]))
current = ready[0]

start, end = t, t + current["burst"]
chart.append((current["id"], start, end))
current["done"] = True

finish = end
tat = finish - current["arrival"]
wt = tat - current["burst"]
rt = start - current["arrival"]

current.update({
"start": start,
"finish": finish,
"turnaround": tat,
"waiting": wt,
"response": rt
```

```
})
completed.append(current)
t = end
return completed, chart

def show_table(records):
df = pd.DataFrame(records)
cols = ["id","arrival","burst","priority","start","finish","waiting","turnaround","response"]
print(df[cols])
print("\nAverages:")
print("WT:", df["waiting"].mean())
print("TAT:", df["turnaround"].mean())
print("RT:", df["response"].mean())

def plot_chart(chart, title="Gantt Chart"):
fig, ax = plt.subplots(figsize=(9,3))
for pid, s, e in chart:
ax.barh("CPU", e-s, left=s, edgecolor="black")
ax.text((s+e)/2, 0, pid, ha="center", va="center", color="white")
ax.set_xlabel("Time")
ax.set_title(title)
plt.show()

if __name__ == "__main__":
jobs = load_jobs(API_URL)
results, gantt = priority_non_preemptive(jobs)
print("=== Non-Preemptive Priority Scheduling ===")
show_table(results)
plot_chart(gantt, "Non-Preemptive Priority Scheduling")
```

Output:

```
=== Non-Preemptive Priority Scheduling ===
   id  arrival  burst  priority  start  finish  waiting  turnaround  response
0  J1        0      6         2      0       6        0           6         0
1  J2        1      4         1      6      10        5           9         5
2  J5        4      2         1     10      12        6           8         6
3  J4        3      5         2     12      17        9          14         9
4  J3        2      3         3     17      20       15          18        15

Averages:
WT: 7.0
TAT: 11.0
RT: 7.0
```




**QA:**

## 1. Difference between Preemptive and Non-preemptive Priority Scheduling

- **Preemptive Priority Scheduling**
  - The scheduler may interrupt a running process the moment a more important (higher-priority) job arrives..
  - Advantage → urgent tasks get CPU quickly.
  - Drawback → extra overhead due to frequent context switches.
- **Non-preemptive Priority Scheduling**
  - Once a process starts, it will hold the CPU until it finishes or goes into waiting state.
  - Advantage → simple design, low overhead.
  - Drawback → if a low-priority process is running, even urgent tasks must wait.

## 2. How Starvation Occurs in Priority Scheduling

- Starvation happens when lower-priority processes keep getting pushed back because higher-priority ones constantly enter the system.
  Example → imagine "OS services" with priority 1 always arriving, then "user background tasks" with priority 5 may never get CPU time.

## 3. Strategies to Avoid Starvation

**I. Aging:**

- Gradually increase the priority of waiting processes over time
- Formula: new_priority = original_priority + (waiting_time / aging_factor)
- Ensures that even low-priority processes eventually get CPU time
- Most commonly used anti-starvation technique

**II. Priority Ceiling:**

- Set a maximum priority level that processes can reach through aging
- Prevents aged processes from becoming too dominant

**III. Time-based Priority Adjustment:**

- Reset priorities periodically
- Implement priority decay for processes that have executed recently

**IV. Multi-level Feedback Queues:**

- Use multiple priority levels with different scheduling algorithms
- Move processes between queues based on behavior and waiting time

## 4. Response Time in Priority Scheduling vs FCFS or SJF

**I. Priority Scheduling vs FCFS:**

- Priority scheduling generally provides better average response time
- High-priority processes get much faster response times
- Low-priority processes may have worse response times than FCFS
- Variance in response times is higher

**II. Priority Scheduling vs SJF:**

- SJF optimizes average waiting time but doesn't consider urgency
- Priority scheduling can provide faster response for critical tasks
- SJF may perform better for throughput, priority scheduling for responsiveness
- Preemptive priority scheduling often outperforms non-preemptive SJF for interactive systems

## 5. Impact of Incorrect Priority Assignment

If everything is marked "high priority," the system behaves like FCFS with no real differentiation. If an unimportant task is given very high priority, it can delay critical system processes. Overall, poor configuration can cause starvation, low throughput, and inconsistent user experience.

## 6. How Dynamic Workload Simulation Helps in Performance Tuning

Through scheduler testing with varied workloads and priorities, you can:

**Monitor and Identify:**

- Track which processes face delays or starvation
- Spot performance bottlenecks and unfair resource allocation

**Optimize Dynamically:**

- Adjust priority rules in real-time based on observed behavior
- Balance system responsiveness against process fairness

**Apply Practically:**

- **Databases:** Optimize query scheduling and background task management
- **Cloud platforms:** Improve VM allocation and tenant resource sharing
- **Real-time systems:** Ensure critical processes meet deadlines while maintaining efficiency