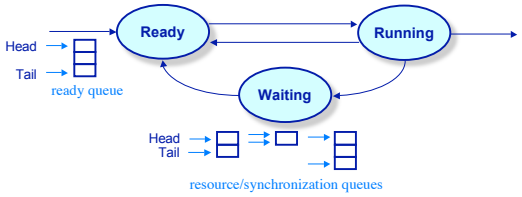
Process Scheduling

Processes and State Transitions



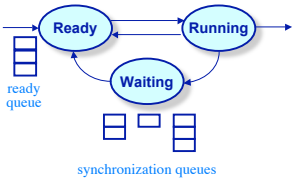
- Three states: Ready, Running, and Waiting

When a process makes a transition:
1. from *running* to *waiting*
2. from *running* to *ready*
3. from *waiting* to *ready*
   (3a. a process is *created*)
4. from *running* to *terminated*

Why a process makes a transition:
1. an action of the *process*
   non-preemptive scheduling
2. occurrence of an external *event*
   preemptive scheduling

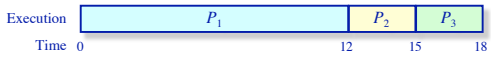
Process Scheduling

- Process scheduling
  - Select a process from ready queue for execution
- Evaluation metrics
  - CPU/device utilization
    - Time busy/time observed
  - System throughput
    - Jobs completed/time observed
  - Turnaround time
    - Start-to finish time for onn-interactive jobs
  - Waiting time
    - Time spent in ready queue
  - Response time
    - Start-to-finish time for interactive jobs



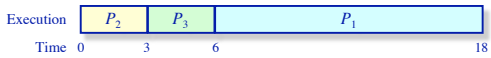
Scheduling Policies
First-Come-First-Served (FCFS)

- The discipline corresponding to FIFO queuing
- If a process blocks while executing, CPU is given to next in queue
- Example — 3 processes w/ compute times 12, 3, and 3
  - Job arrival order P_1, P_2, P_3



Average response time = (12 + 15 + 18)/3 = 15

Job arrival order P_2, P_3, P_1



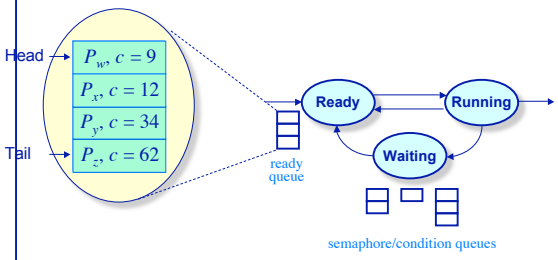
Average response time = (3 + 6 + 18) /3 = 9

FCFS Scheduling (Cont'd.)

- Advantage:
  - Simple!
- Disadvantages:
  - Average waiting time is highly variable
    - Short jobs may wait behind long ones !!
  - May lead to poor overlap between I/O and CPU processing
    - CPU bound processes will make I/O bounds processes to wait ⇒ I/O devices remain idle

Scheduling Policies
Shortest-Job-First (SJF)

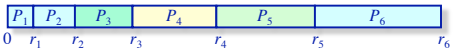
- Select the shortest job first
  - Enqueue jobs in order of estimated completion time



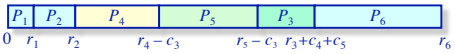
Shortest-Job-First Scheduling
Optimal Mean Turnaround--when jobs are available simultaneously

- Intuition: Consider an SJF execution of a set of processes

Mean turnaround time = $(r_1 + r_2 + r_3 + r_4 + r_5 + r_6)/6$



Can switching the execution order reduce response time?



Mean turnaround time = $(r_1 + r_2 + r_4 - c_3 + r_5 - c_3 + r_4 + c_4 + c_5 + r_6)/6$
= $(r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + (c_4 + c_5 - 2c_3))/6$

SJF Scheduling --- The Catch

- It's unfair !!
  - Continuous stream of short jobs will starve long jobs
- Needs clairvoyance
  - Need to know the execution time of a process
  - Simple solution: ask the user !
  - Yeah, right !!
- So, what if you don't subscribe to the Psychic Network ??

Shortest Process Next Scheduling
Estimating execution time

- Jobs are enqueued in order of estimated completion time
  - "Recent history is a good indicator of the near future"

```
process P
begin
  loop
    <read input from user>
    <process input>
  end loop
end P
```

t_n — duration of the n^{th} CPU burst

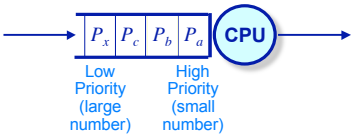
τ_{n+1} — predicted duration of the $n{+}1^{st}$ CPU burst

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, \text{ for } 0 \le \alpha \le 1$$

Scheduling Policies
Priority Scheduling (PS)

- Assign a priority (a number) to each job and schedule jobs in order of priority
  - Typically low priority values = "high priority"

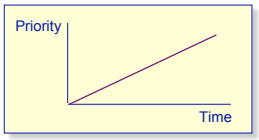
    E.g., if priority = τ_n, then a priority scheduler becomes a SJF scheduler.

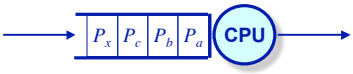


Low Priority (large number) — High Priority (small number)

Priority Scheduling
Avoiding starvation

- Aging
  - Gradually increase a process's priority (decrease its priority value) over time



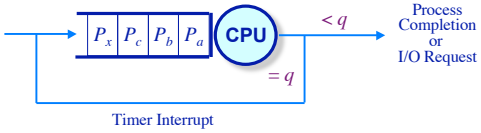


Non Pre-emptive vs. Pre-emptive Scheduling

- Non Pre-emptive Scheduling:
  - Once a process begins execution, it occupies CPU until it finishes or it blocks
  - Advantage: simplicity, but …
  - Creates problems … (like what?)
  - Examples: FCFS, SJF, PS, …
- Pre-emptive Scheduling:
  - A process is switched back and forth between running and ready states
  - Advantage: more efficient, better capabilities, but …
  - More complex and needs hardware support (e.g., timer interrupts)
  - Examples: Round Robin, Shortest Remaining Time First (SRTF), Multi-level Feedback Queue (MLF)

Scheduling Policies
Round-Robin Scheduling (RR)

- Allocate the processor in discrete unit called *quanta* (or *time-slices*)
- Switch to the next ready process at the end of each quantum
  - Processes execute every $(n-1)q$ time units

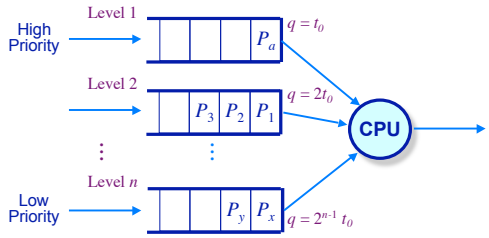


RR Scheduling: Selecting a Time Quantum

- Too large
  - Long waiting time
  - Degenerates to FCFS in the limit
- Too small
  - Responsive, but …
  - Throughput suffers due to large context switch overhead
- Goal:
  - Select a time quantum that balances this tradeoff
  - Rule of thumb: maintain context switch overhead to less than 1%

Scheduling Policies
Multi-level feedback queues (MLF)

- *n* priority levels — priority scheduling between levels, round-robin within a level
- Quantum size decreases with priority level
- Jobs are demoted to lower priority levels if they don't complete within the current quantum



Scheduling Policies
Real-time scheduling

- Real-time processes have *timing constraints*
  - Timing constraints are translated into *deadlines* or *rate requirements*
- A set of *m* periodic events is *schedulable* if:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1 \qquad \text{where}$$

C_i is the CPU time required to process event i
P_i is the period (1/rate) at which event i occurs

Two dominant real-time scheduling policies:

"Rate-Monotonic" scheduling
Priority = period

"Deadline" scheduling
Priority = *release time + deadline*

Rate monotonic is guaranteed to work if $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$ or, if $m \to \infty$ $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq ln2$

Scheduling Policies
Real-time scheduling

- Real-time processes have *timing constraints*
  - Timing constraints are translated into *deadlines* or *rate requirements*

Two dominant real-time scheduling policies

"Rate-Monotonic" scheduling
Priority = 1/*rate* (the "period")

"Deadline" scheduling
Priority = *release time* + *deadline*

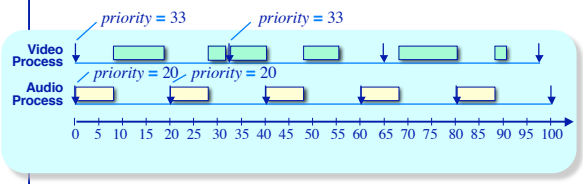
Example: Digital video playout

```
/* Main processing loop */
   loop
      data         = read( network)
      video_frame  = decompress(data)
      write( frame_buffer, video_frame)
   end loop
```

Timing constraint: Execute loop once every 33 ms.

Scheduling Policies
Real-time scheduling example

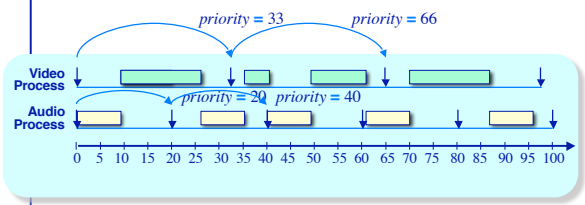
- Consider scheduling audio and video playout processes
  - The audio process processes 1 audio sample every 20 *ms*
  - The video process processes 1 video frame every 33 *ms*



Rate-monotonic scheduling: *priority* = 1 / *rate* = *period*

Scheduling Policies
Real-time scheduling example

- Consider scheduling audio and video playout processes
  - The audio process processes 1 audio sample every 20 *ms*
  - The video process processes 1 video frame every 33 *ms*



Deadline scheduling: *priority* = *release time* + *deadline*

deadline = *inter-arrival time (in this example)*

Scheduling Policies: Summary

- FCFS:
  - Not fair, and poor average waiting times
- Priority Scheduling:
  - Not Fair and starvation possible
- Round Robin:
  - Fair, but poor average waiting times
- SJF/SRTF:
  - Not fair, but average waiting time is minimized
  - Requires accurate prediction of computation times
  - Starvation is possible
- MLF:
  - An approximation to SJF
- Real-time scheduling:
  - Needed to meet timeliness constraints
- Lottery Scheduling
  - Assign tickets to processes in proportion that reflects the desired relative CPU usage