Scheduling Bits & Pieces

Windows Scheduling

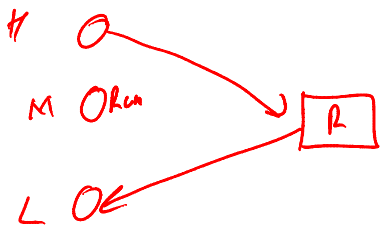
		Win32 process class priorities					
		Real-time	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Figure 11-27. Mapping of Win32 priorities to Windows priorities.

Windows Scheduling

- Priority Boost when unblocking
  - Actual boost dependent on resource
    - Disk (1), serial (2), keyboard (6), soundcard (8)…..
    - Interactive, window event, semaphore (1 or 2)
  - Boost decrements if quantum expires
- Anti-starvation hack
  - If a ready process does not run for long time, it gets 2 quanta at priority 15

Priority Inheritance



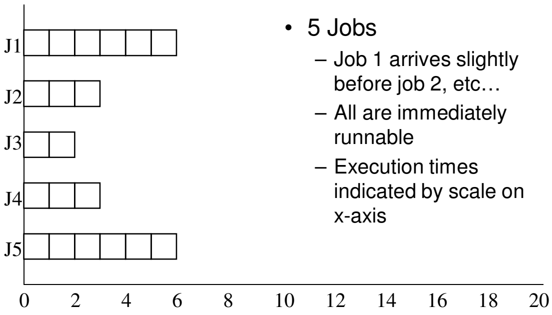
Batch Algorithms

- Maximise *throughput*
  - Throughput is measured in jobs per hour (or similar)
- Minimise *turn-around time*
  - Turn-around time (T_r)
    - difference between time of completion and time of submission
    - Or waiting time (T_w) + execution time (T_e)
- Maximise *CPU utilisation*
  - Keep the CPU busy
  - Not as good a metric as overall throughput

First-Come First-Served (FCFS)

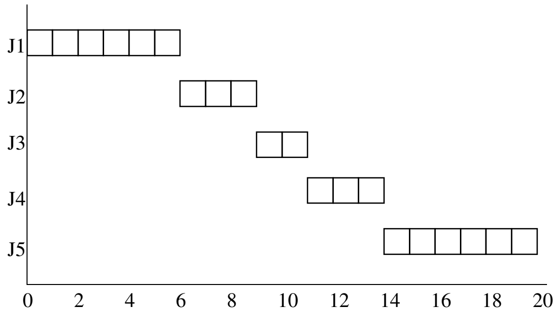
- Algorithm
  - Each job is placed in single queue, the first job in the queue is selected, and allowed to run as long as it wants.
  - If the job blocks, the next job in the queue is selected to run
  - When a blocked jobs becomes ready, it is placed at the end of the queue

Example



- 5 Jobs
  - Job 1 arrives slightly before job 2, etc…
  - All are immediately runnable
  - Execution times indicated by scale on x-axis

FCFS Schedule



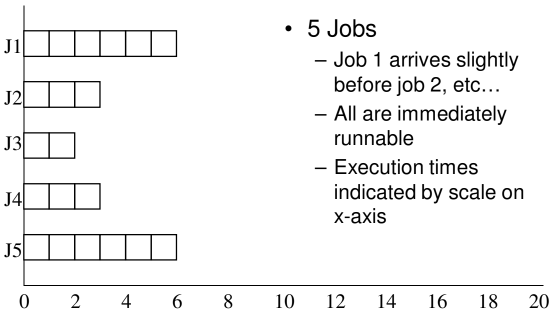
FCFS

- Pros
  - Simple and easy to implement
- Cons
  - I/O-bound jobs wait for CPU-bound jobs
  - ⇒Favours CPU-bound processes
    - Example:
      - Assume 1 CPU-bound process that computes for 1 second and blocks on a disk request. It arrives first.
      - Assume an I/O bound process that simply issues a 1000 blocking disk requests (very little CPU time)
      - FCFS, the I/O bound process can only issue a disk request per second
        - » the I/O bound process take 1000 seconds to finish
      - Another scheme, that preempts the CPU-bound process when I/O-bound process are ready, could allow I/O-bound process to finish in 1000* average disk access time.

Shortest Job First

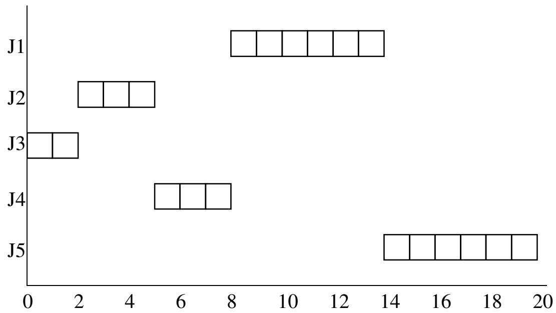
- If we know (or can estimate) the execution time *a priori,* we choose the shortest job first.
- Another non-preemptive policy

Our Previous Example



- 5 Jobs
  - Job 1 arrives slightly before job 2, etc…
  - All are immediately runnable
  - Execution times indicated by scale on x-axis

Shortest Job First



Shortest Job First

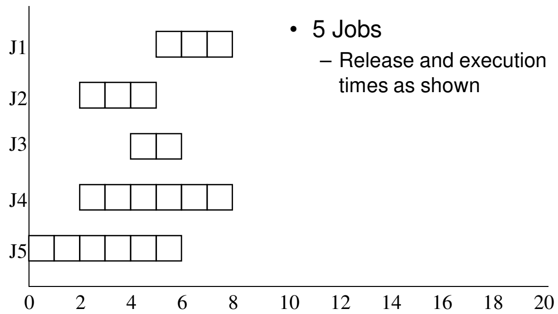
a | a
b | a + b
c | a + b + c
d | a + b + c + d

- Con
  - May starve long jobs
  - Needs to predict job length
- Pro
  - Minimises average turnaround time (if, and only if, all jobs are available at the beginning)
  - Example: Assume for processes with execution times of *a, b, c, d.*
    - *a* finishes at time *a*, *b* finishes at *a + b*, *c* at *a + b + c*, and so on
    - Average turn-around time is (4*a* + 3*b* + 2*c* + *d*)/4
    - Since *a* contributes most to average turn-around time, it should be the shortest job.

Shortest Remaining Time First

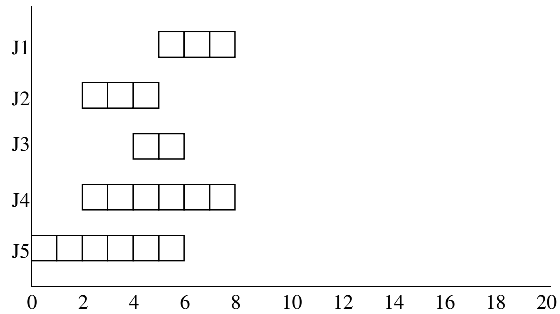
- A preemptive version of shortest job first
- When ever a new jobs arrive, choose the one with the shortest remaining time first
  - New short jobs get good service

Example

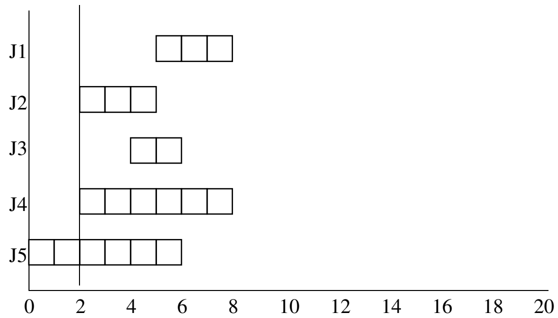


- 5 Jobs
  - Release and execution times as shown

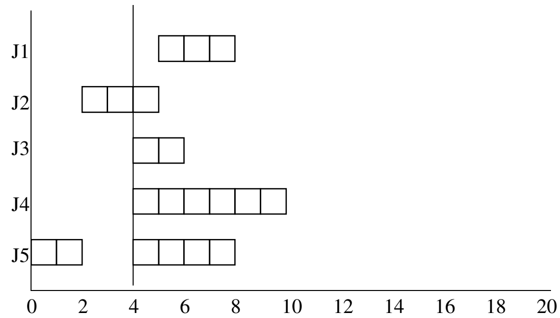
Shortest Remaining Time First



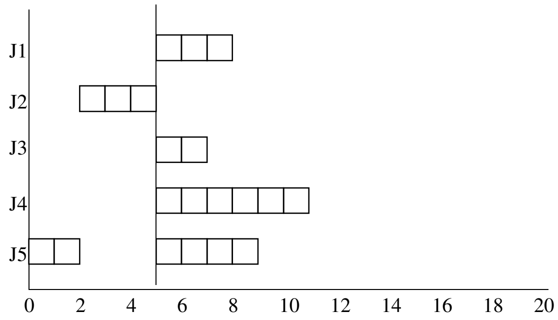
Shortest Remaining Time First



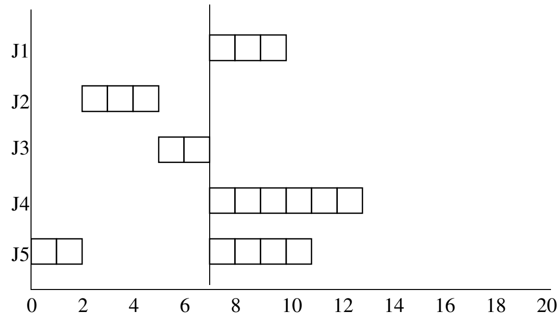
Shortest Remaining Time First



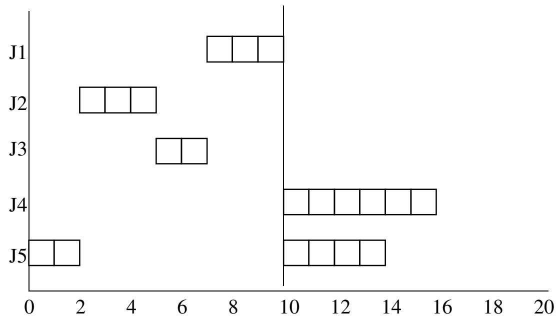
Shortest Remaining Time First



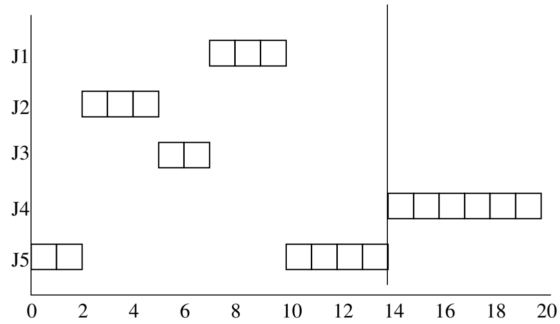
Shortest Remaining Time First



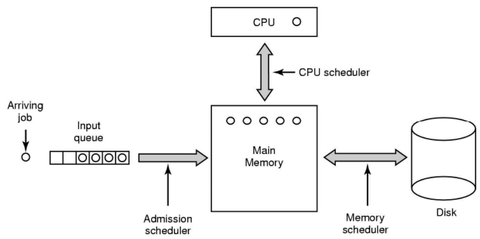
Shortest Remaining Time First



Shortest Remaining Time First



Scheduling in Batch Systems



Three level scheduling

Three Level Scheduling

- Admission Scheduler
  - Also called *long-term* scheduler
  - Determines when jobs are *admitted* into the system for processing
  - Controls degree of multiprogramming
  - More processes $\Rightarrow$ less CPU available per process

Three Level Scheduling

- CPU scheduler
  - Also called *short-term* scheduler
  - Invoked when ever a process blocks or is released, clock interrupts (if preemptive scheduling), I/O interrupts.
  - Usually, this scheduler is what we are referring to if we talk about a *scheduler*.

Three Level Scheduling

- Memory Scheduler
  - Also called *medium-term* scheduler
  - Adjusts the degree of multiprogramming via suspending processes and swapping them out

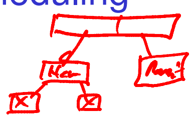
Some Issues with Priorities

- Require adaption over time to avoid starvation (not considering hard real-time which relies on strict priorities).
- Adaption is:
  - usually ad-hoc,
    - hence behaviour not thoroughly understood, and unpredictable
  - Gradual, hence unresponsive
- Difficult to guarantee a desired share of the CPU
- No way for applications to trade CPU time

Lottery Scheduling

- Each process is issued with “lottery tickets” which represent the right to use/consume a resource
  - Example: CPU time
- Access to a resource is via “drawing” a lottery winner.
  - The more tickets a process possesses, the higher chance the process has of winning.

Lottery Scheduling



- Advantages
  - Simple to implement
  - Highly responsive
    - can reallocate tickets held for immediate effect
  - Tickets can be traded to implement individual scheduling policy between co-operating threads
  - Starvation free
    - A process holding a ticket will eventually be scheduled.

Example Lottery Scheduling

- Four process running concurrently
  - Process A: 15% CPU
  - Process B: 25% CPU
  - Process C: 5% CPU
  - Process D: 55% CPU
- How many tickets should be issued to each? 100

Lottery Scheduling Performance

Observed performance of two processes with varying ratios of tickets

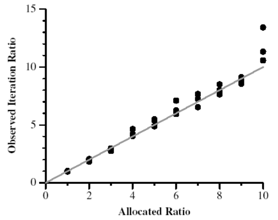


Figure 4: **Relative Rate Accuracy.** For each allocated ratio, the observed ratio is plotted for each of three 60 second runs. The gray line indicates the ideal where the two ratios are identical.

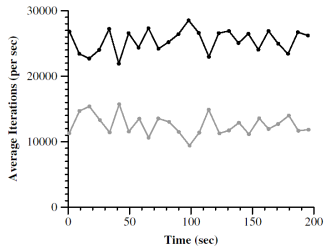


Figure 5: **Fairness Over Time.** Two tasks executing the Dhrystone benchmark with a 2 : 1 ticket allocation. Averaged over the entire run, the two tasks executed 25378 and 12619 iterations/sec., for an actual ratio of 2.01 : 1.

Fair-Share Scheduling

- So far we have treated processes as individuals
- Assume two users
  - One user has 1 process
  - Second user has 9 processes
- The second user gets 90% of the CPU
- Some schedulers consider the owner of the process in determining which process to schedule
  - E.g., for the above example we could schedule the first user's process 9 times more often than the second user's processes
- Many possibilities exist to determine a *fair* schedule
  - E.g. Appropriate allocation of tickets in lottery scheduler