5. CPU Scheduling

Turnaround time

: T(completion) - T(arrival)

Response Time

: T(firstrun) - T(arrival)

- firstrun : the job produces some kind of "response"

Introduction of time-shared machines
=> users sit at a terminal & demand interactive performance from the system => waiting for a long time would unpleasant

Convoy effect

: a number of relatively-short potential consumers of a resource get queued behind a heavyweight resource consumer

Preemptive Scheduler

A scheduling method where the currently running process can be forcibly interrupted and the CPU can be reassigned to another process.

Non-preemptive Scheduler

A scheduling method where the currently running process runs to completion before switching to another process. It cannot be taken away unless it voluntarily yields the CPU.

Shortest Job First

the perceived turnaround time per customer matters
=> less average turnaround time

- non-preemptive scheduler
  - Optimal if all jobs arrive at the same time
  - If not, turnaround time might be longer. =>Can't ensure that we're running Shortest Job

All modern schedulers are Preemptive

willing to stop one process from running in order to run another => context switch

Shortest Time-to-Completion First (STCF) / Preemptive Shortest Job First (PSJF)

SJF + add preemption

Any time a new job enters the system, the STCF scheduler determines which of the remaining jobs (including the new job) has the least time left, and schedules that one.

Great for turnaround time

Bad for response time & interactivity

- If three jobs arrive at the same time
- Third job has to wait for the previous two jobs to run in their entirely before being scheduled just once.

Round Robin (RR) a.k.a Time-Slicing

instead of running jobs to completion,
runs a job for a time slice => switches to the next job in the run queue.
=> repeatedly does so until the jobs are finished.

sensitive to response time

The length of a time slice matters

- must be a multiple of the timer-interrupt period => k*n
- the shorter -> the better the performance under the response-time metric
- BUT, making it too short is problematic : cost of context switching dominates overall performance

=> Deciding on the length of the time slice presents a trade-off
: making it long enough to amortize the cost of switching without making it so long that the system is no longer responsive

When it comes to turnaround time : RR is one of the worst policies

- what RR is doing is stretching out each job as long as it can
- by only running each job for a short bit before moving to the next
- even worse than simple FIFO in many cases

Trade-off

unfair (SJF, STCF) : optimizes turnaround time / bad response time
fair (RR) : optimizes response time / bad turnaround time

Incorporating I/O

scheduler make decision when

- a job initiates an I/O
    - the currently-running job won't be using the CPU during the I/O
    - It is blocked waiting for I/O completion
    - the scheduler should propably schedule another job on the CPU at that time
- I/O completes
    - interrupt is raised
    - the OS runs and moves the process that issued the I/O from blocked back to the ready state
    - Decide to run the job at that point

=> How should the OS treat each job?
: Overlap, with the CPU being used by one process while waiting for the I/O of another process to complete

=> Incorporate I/O
: By treating each CPU burst as a job, the scheduler makes sure processes that are "interactive" get run frequently.
While those interactive jobs are performing I/O, other CPU-intensive jobs run, thus better utilizing the processor