Lecture 12 — CS 347/333 — 30.8.2022

(*) **Scheduling**
- scheduler (the mechanisms)
  - events
  - context-switch
- scheduling policies.

(*) (i) **metrics**:
- utilization
- fairness (no starvation)
- finish time
- waiting time
- response time
- throughput
- # deadlines met.

execution time
- service time
  - actual time to do work.

yield()

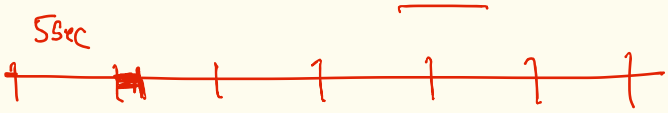
(ii) **categories**:

(i) non-preemptive vs. **preemptive**

- non-preemptive: schedule only when current process yields, exits or blocks.
- preemptive: Schedule a new process even if current process is not blocked/done (basically wants to continue running).

(ii) **non-work conserving** vs **work conserving**.

- non-work conserving: if resource is available & work to be done, may not schedule work on resource.
- work conserving: if resource is available & work is to be done, schedule work, no resource idling.



(iii) policy criteria/challenges.
- interactive/bursty vs. CPU-intensive
- real time vs. best effort

Examples :

① **FCFS** : first-come-first-serve

- add to end of ready queue (on block/exit/yield)
- dequeue from head of queue.

(*) work-conserving, no starvation, waiting time could be high!
(~non-preemptive)

② **RR** ~ Round Robin

~ preemptive at every time interval.

- dequeue from head of ready queue & execute for a specific max. time interval.
- on time-out or block (move to end of queue).
  ↳ dequeue next ready process.

(*) work-conserving, no starvation,
pre-emptive,
good for: fairness, response time.

execution & finish time can suffer (esp. at high load).

- time interval → 0 fair scheduler (context switch overheads increase) high.
- time interval >> 0 (large) ⇒ FCFS.

③ **Priority-based scheduler**

e.g. priority levels: gold, silver, bronze

~ a separate ready queue per priority level.
~ each process has a tagged priority.
~ FCFS/RR used for each ready queue.
(*) ~ always schedule process from higher priority first!

(+) pre-emptive, priority sensitive, (higher priority processes get priority for CPU usage).

(−) starvation! - lower priority process may never get CPU.

(4) proportionate scheduling.

WRR

weighted round robin

- one ready queue per priority.
- time slice per queue (weight) is proportional to priority.

e.g: 4:2:1 for 3 queues.

- 4 tasks of priority 1
- 2 tasks of — " — 2
- 1 task of — " — 1

and repeat (round robin).

(+) priority sensitive
no starvation.

(−) finish time depends on weights & load.

Lottery Scheduling

- # tickets issued to each process.
- random number generated to select winner/ticket ⇒ process.
- # tickets ⇒ probability of scheduling.

tickets > 0 ⇒ no starvation!

(*) probabilistic & proportionate scheduling.

more policies:

- SJF: shortest/smallest job first.
- STCF: Smallest Time to Completion first.
  (Shortest)

(#) real word

- arrival of processes (in ready queue) non-deterministic
- process execution time unknown. (and execution times are different for different processes)
- process behaviour (interactive vs cpu intensive) unknown.

(*) real scheduler example

(i) # multi-level feedback queue based scheduler. (MLFQ).

- part of Linux for a long time
- multiple queues (one per priority)
- priority of each process is dynamic/changing.

the MLFQ scheduling policy —

1. if priority (process A) > priority (process B)
   A gets CPU.

2. if priority (A) = priority (B), A & B run in RR fashion using a time slice for the given queue.

3. a process entering the system (READY for first time), Placed at highest priority (topmost queue).

4. Once process uses time slice / allotment at a given level it's priority is reduced (moved down one queue / level).

5. After some time period, move all processes to topmost queue
   or wrap around after last queue

(*) note: each queue has a different time slice for RR.

(+) no starvation, preemptive, priority sensitive & fair.
short jobs finish quickly, CPU-intensive long jobs make regular progress.

(ii) CFS — current Linux scheduler
Completely Fair Scheduler. (default for non real time tasks).

(iii) multi CPU (multiprocessor) scheduling.
- multiple queues (one per CPU)
- which queue a process should be placed in? (load balancing).
- should process move across queues? (migration).