Linux Scheduling (Commercial)

Two types of thread-based schedulers:

- **Linux Real-time Scheduler**: schedules *real-time* and *normal* threads.
    - *normal* threads subject to normal scheduler routine.
- **Linux Non-Real-time Scheduler**: schedules only *normal* threads

Linux Real-time Scheduler
Divides work into 3 queues:

- `SCHED_FIFO:` First-in, first-out real-time threads/tasks.
    - Scheduled as follows:
        - FIFO thread gets interrupted only if:
            - Higher priority FIFO thread is ready.
                - If multiple high priority threads of same priority read, choose one who waited *longer.*
            - FIFO thread gets blocked (e.g on I/O).
            - FIFO yields with `sched_yield`.
        - FIFO thread is interrupted, it is placed in the ready queue.
- `SCHED_RR:` Round-robin Real-time threads/tasks.
    - Same as FIFO scheduling, except implementation includes time-slices.
- `SCHED_OTHER:` Non-real-time threads.
    - A non-real time thread executes *only* if no *Round-robin* or *FIFO* threads are ready.

Linux Non-Real Time Scheduler:

- **Traditional:**
    - O(n) algortihm
    - Bad for *multi-processing* because:
        - Single queue
        - Single *mutex* access to run queue.
        - No *pre-emption*.
- **O(1) Algorithm:**
    - Guarantees *O(1)* runtime.
    - Implements a queue for 140 different priorities with *round-robin:*
        - *0-99:* Real-time*; 100-139:* Normal
        - A 'queue' is technically two queues:
            - **Active Queue:**
                - Queue of ready, unserved threads.
                - Pre-empted tasks go back to *active queue*
                - *All* scheduling happens here only.
                - When *active queue* empty, *active*, *expired queue* exchange spots.
            - **Expired Queue:** Queue of ready, unserved threads
    - Implements bitmap to indicate empty queues.
- **CFS (Complete Fair Schedule):**
    - Runs in *O(ln(n))* time
        - *Caching* used to improve performance.
    - Uses *red-black self-balancing* trees for *ready-queue.*
        - Threads sorted as per *virtual*-CPU running time used
            - **Time Recorded:**
                - *Higher* nice value means over-recorded CPU-runtime

    - *Lower* nice value means under-recorded CPU-runtime.
  - **History Decay:**
    - *Higher* priority means faster recorded-runtime decay
    - *Lower* priority means slower recorded-runtime decay
- *Round-robin-ing, Time-slice* vary in length to meet *Target Latency.*
  - **Target Latency:** total time to serve all ready processes.
- *Blocked* tasks removed from *ready-queue*
- *Pre-empted* tasks inserted again with updated CPU-runtime
- Trees periodically balanced.

- *I/O Bound* threads highly priortized to increase *responsiveness*
- ***Group Scheduling:***
  - Related threads spawning each other are part of a *group*
  - *Multiple* groups exist on system - groups are treated equally
    - *Groups* might have different number of threads in them
    - Within *groups,* threads have a fair competition

NOTE: you STILL NEED to complete the section about SMP