Chapter 6 summary

- Maximum CPU utilization obtained with multiprogramming.
- CPU burst followed by I/O burst.
- Short-term scheduler selects from among the processes in ready queue and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
    - Switches from running to waiting state (non-preemptive).
    - Switches from running to ready state (preemptive).
    - Switches from waiting to ready (preemptive).
    - Terminates (non-preemptive).
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
    - Switching context.
    - Switching to user mode.
    - Jumping to the proper location in the user program to restart that program.
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running.
- **Scheduling Criteria**:
    - **CPU utilization** – keep the CPU as busy as possible.
    - **Throughput** – # of processes that complete their execution per time unit.
    - **Turnaround time** – amount of time to execute a particular process.
    - **Waiting time** – amount of time a process has been waiting in the ready queue.
    - **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).
- **Scheduling Algorithm Optimization Criteria:**
    - Max CPU utilization.
    - Max throughput.
    - Min turnaround time.
    - Min waiting time.
    - Min response time.
- **First-Come, First-Served (FCFS) Scheduling:** **Important example in slide 10**
    - **Convoy effect** - short process behind long process.
- **Shortest-Job-First (SJF) Scheduling:** **Important example in slide 13**
    - Associate with each process the length of its next CPU burst.
    - SJF is optimal – gives minimum average waiting time for a given set of processes.
        - The difficulty is knowing the length of the next CPU request.

- Two schemes:
    - **Non-preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst. **Important example in slide 18**
    - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF). **Important example in slide 19 and 20**

- **Determining Length of Next CPU Burst:**
  - t_n is the actual value.
  - T_{n+1} is the predicted value.
  - Then for α, 0 <= α <=1, define :
    - $T_{n+1} = \alpha t_n + (1 - \alpha) T_n$
- **Priority Scheduling:** **Important example in slide 22**
  - A priority number (integer) is associated with each process.
  - CPU is allocated to the process with the highest priority (smallest integer highest priority)
    - Preemptive
    - Non-preemptive
  - SJF is priority scheduling where priority is the inverse of predicted next CPU burst time.
  - Problem → **Starvation** – low priority processes may never execute.
  - Solution → **Aging** – as time progresses increase the priority of the process.
- **Round Robin:** **Important example in slide 24 and 25**
  - Each process gets a small unit of CPU time (time quantum q).
    - After this time has elapsed, the process is preempted and added to the end of the ready queue.
  - Performance
    - q large → FIFO
    - q small → q must be large with respect to context switch, otherwise overhead is too high.
  - Time Quantum↑ Context switch ↓
    - 80% of CPU bursts should be shorter than q.
- **Multilevel queue:** **Important example in slide 29**
  - Ready queue is partitioned into separate queues:
    - foreground (interactive)
    - background (batch)
  - Each queue has its own scheduling algorithm:
    - foreground – RR
    - background – FCFS

- Scheduling must be done between the queues:
    - **Fixed priority scheduling** – (i.e., serve all from foreground then from background).
        - Possibility of starvation.
    - **Time slice** – each queue gets a certain amount of CPU time which it can schedule amongst its processes.
        - i.e., 80% to foreground in RR, 20% to background in FCFS
- **Multilevel Feedback Queue:** **Important example in slide 31 and 32**
    - A process can move between the various queues; aging can be implemented this way.
    - Multilevel-feedback-queue scheduler defined by the following parameters:
        - Number of queues.
        - Scheduling algorithms for each queue.
        - Method used to determine when to upgrade a process.
        - Method used to determine when to demote a process.
        - Method used to determine which queue a process will enter when that process needs service.
- **Multiple-Processor Scheduling:**
    - With multiple CPUs, **load sharing** becomes possible but CPU scheduling more complex:
        - **Homogeneous processors**
            - Can use any processor to run any process in the ready queue.
        - **Asymmetric multiprocessing**
            - Only one master processor accesses system data structures and other processors execute only user code.
        - **Symmetric multiprocessing (SMP)**
            - Each processor is self-scheduling, all processes in common ready queue, or each has its own private ready queue.
        - **Processor affinity** – process has affinity for processor on which it is currently running.
    - **Load balancing** attempts to keep workload evenly distributed across all processors.
    - **Push migration** – a task periodically checks load on each processor, and if found imbalance pushes task from <u>overloaded</u> CPU to other CPUs.
    - **Pull migration** – <u>idle</u> processors pull waiting task from busy processor.
- **Algorithm Evaluation:**
    - <u>Deterministic modeling:</u>
        - Takes a particular predetermined workload and defines the performance of each algorithm for that workload.

    - Simple and fast, but requires exact numbers for input, and its answers apply only to those cases.
- <u>Queueing Models:</u>
    - The distribution of CPU and I/O bursts can be measured and then simply estimated.
    - Little’s Formula:
        - n = average queue length
        - W = average waiting time in queue
        - λ = average arrival rate into queue

$$n = \lambda \times W$$

- <u>Simulations:</u>
    - Queueing analysis is useful but has limitations.
    - Simulations more accurate.
- <u>Implementation:</u>
    - Simulation can be expensive and limited accuracy.
    - Only completely accurate way to implement new scheduler and test in real systems.
    - Most flexible scheduling algorithms that can be altered by system managers and tuned for a specific set of applications.