



UNIVERSITY OF REGINA


CS330-001

INTRODUCTION TO OPERATING SYSTEMS

ANDRÉ E. **DOS SANTOS**

dossantos@cs.uregina.ca

andreeds.github.io

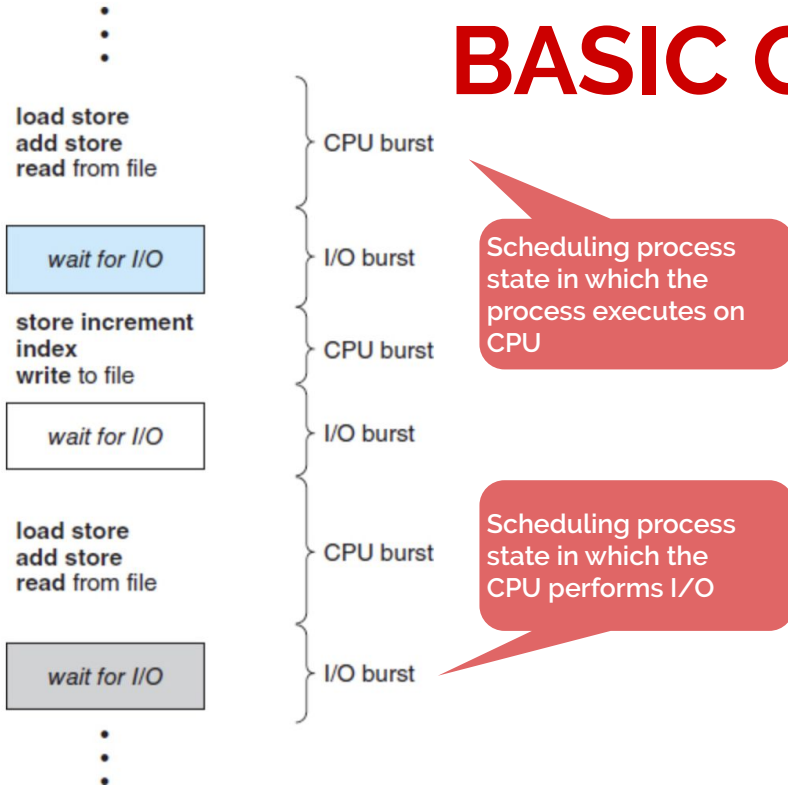
An aerial photograph of a city, likely Regina, Saskatchewan, showing a large park with a lake in the foreground and various city buildings in the background. The image is in black and white and serves as a background for the slide.

CS330-001
INTRODUCTION TO
OPERATING SYSTEMS

CPU SCHEDULING

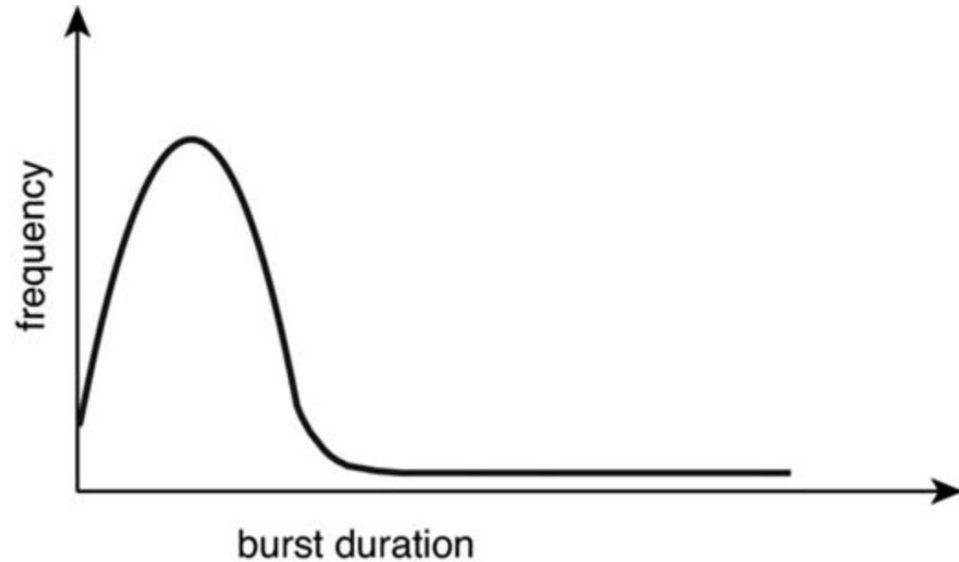
ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreedsgithub.io

BASIC CONCEPTS



- Maximum CPU utilization obtained with multiprogramming
- CPU-I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by I/O burst
- CPU burst distribution is of main concern

HISTOGRAM OF CPU-BURST DURATIONS



CPU Scheduler

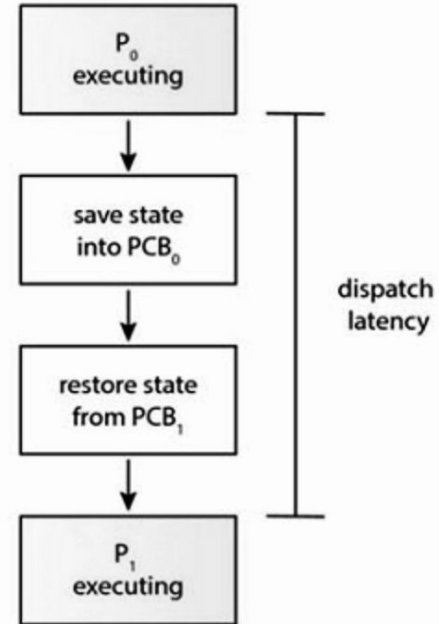
- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
 - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
 1. Switches from running to waiting state
 2. Switches from running to ready state
 3. Switches from waiting to ready
 4. Terminates
- Scheduling under **1** and **4** is **nonpreemptive**
- All other scheduling is **preemptive**
 - Consider access to shared data
 - Consider preemption while in kernel mode
 - Consider interrupts occurring during crucial OS activities

Scheduling in which, once a core has been allocated to a thread, the thread keeps the core until it releases the core either by terminating or by switching to the waiting state

A form of scheduling in which processes or threads are involuntarily moved from the running state

Dispatcher

- 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

Process	Burst Time
P1	24
P2	3
P3	3

- Suppose that the processes arrive in the order: **P1 , P2 , P3**
- The *Gantt Chart* for the schedule is:



- Waiting time for **P1 = 0; P2 = 24; P3 = 27**
- Average waiting time: **$(0 + 24 + 27)/3 = 17$**

First- Come, First-Served (**FCFS**) Scheduling

- Suppose that the processes arrive in the order: **P₂** , **P₃** , **P₁**
- The *Gantt Chart* for the schedule is



- Waiting time for **P₁** = **6**; **P₂** = **0**; **P₃** = **3**
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
 - Consider one CPU-bound and many I/O-bound processes

A scheduling phenomenon in which a number of threads wait for one thread to get off a core, causing overall device and CPU utilization to be suboptimal

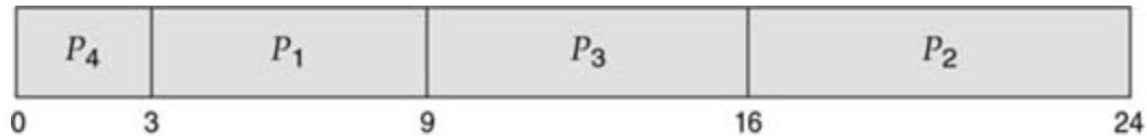
Shortest-Job-First (**SJF**) Scheduling

- Associate with each process the length of its next CPU burst
 - Use these lengths to schedule the process with the shortest time
- 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
 - Could ask the user

Shortest-Job-First (**SJF**) Scheduling

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

- SJF scheduling chart



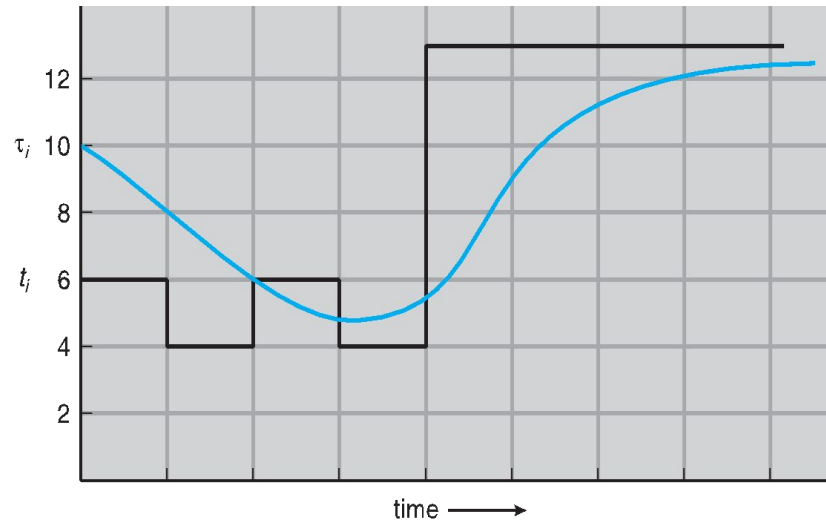
- Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
 - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
 1. t_n = actual length of n^{th} CPU burst
 2. τ_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 \leq \alpha \leq 1$
 4. Define:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$
- Commonly, α set to $\frac{1}{2}$
- Preemptive version called **shortest-remaining-time-first**

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	5	9	11	12	...

Examples of Exponential Averaging

- $\alpha = 0$

- $\tau_{n+1} = \tau_n$
- Recent history does not count

- $\alpha = 1$

- $\tau_{n+1} = \alpha \tau_n$
- Only the actual last CPU burst counts

- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} = & \alpha \tau_n + (1 - \alpha) \alpha \tau_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha \tau_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

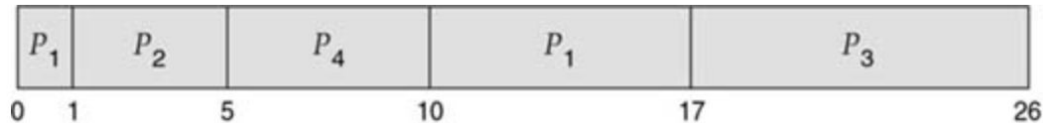
- Since both α and $(1 - \alpha)$ are less than or equal to **1**, each successive term has less weight than its predecessor

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- Preemptive SJF Gantt Chart**



- Average waiting time = $[(10-1)+(1-1)+(17-2)+5-3]/4 = 26/4 = 6.5$ msec

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority)
 - Preemptive
 - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem \equiv **Starvation** – low priority processes may never execute
 - Solution \equiv **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Priority scheduling *Gantt* Chart



- Average waiting time = $(1+6+16+18)/5 = 8.2$ msec

Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum q**), usually 10-100 milliseconds
 - After this time has elapsed, the process is preempted and added to the end of the ready queue
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once
 - No process waits more than $(n-1)q$ time units
- Timer interrupts every quantum to schedule next process
- Performance
 - q large \Rightarrow FIFO
 - q small $\Rightarrow q$ must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P1	24
P2	3
P3	3

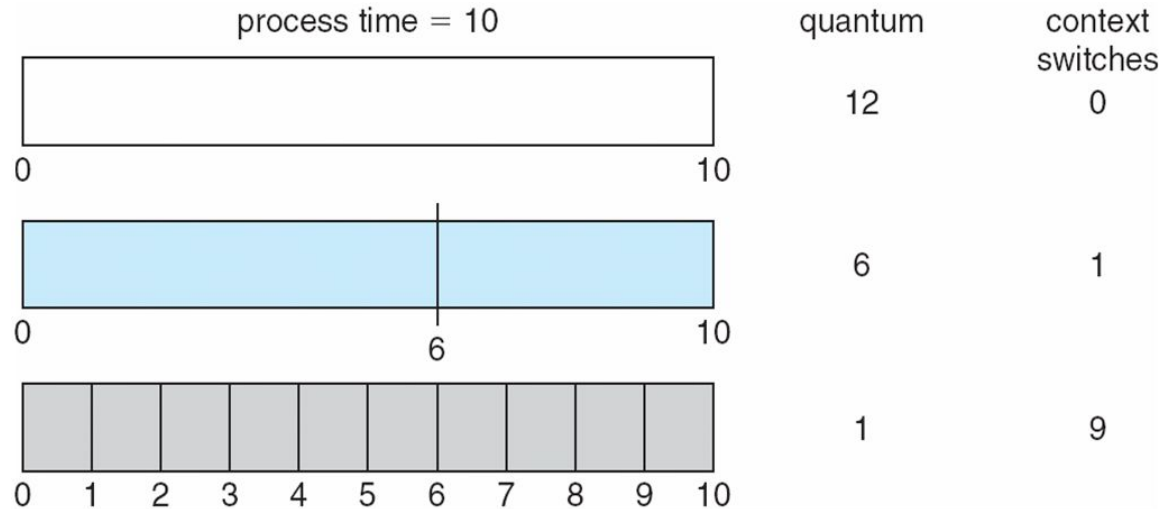
- The *Gantt* Chart is



- Typically, higher average turnaround than SJF, but better response
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time

The performance of the RR algorithm depends heavily on the size of the time quantum

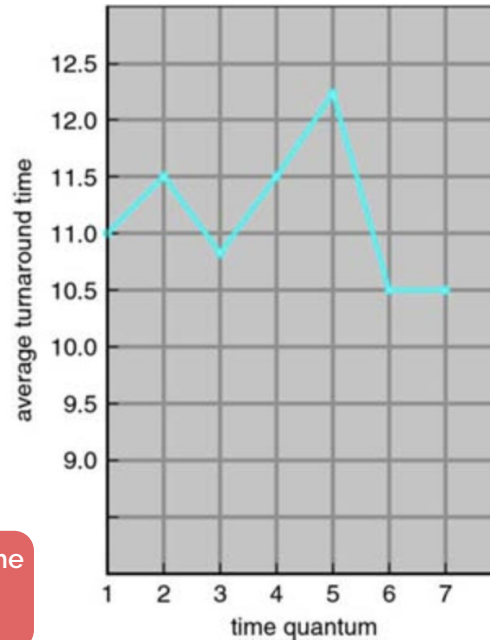


Time Quantum and Context Switch Time

The average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases

In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum

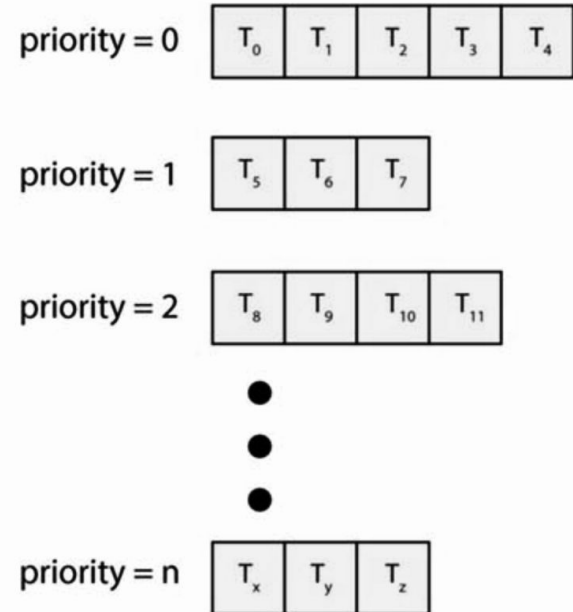
A rule of thumb is that 80% of the CPU bursts should be shorter than the time quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

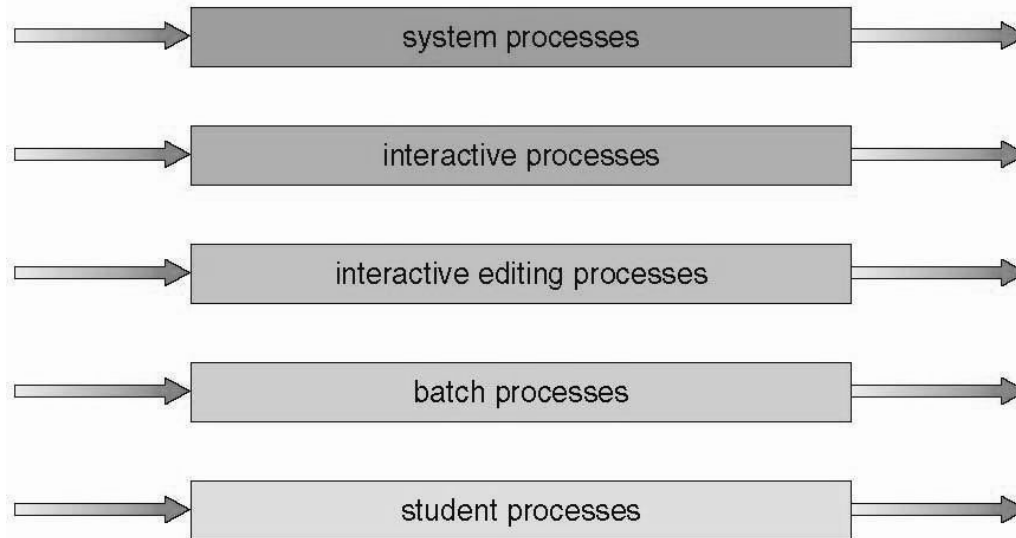
Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
 - **foreground** (interactive)
 - **background** (batch)
- Process permanently in a given queue
- 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 Queue

highest priority



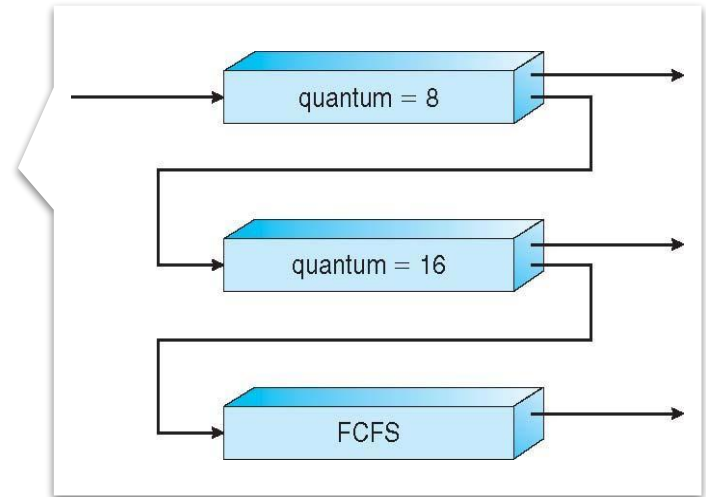
lowest priority

Multilevel Feedback Queue Scheduling

- 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

Example of Multilevel Feedback Queue

- Three queues:
 - Q_0 – **RR** with time quantum 8 milliseconds
 - Q_1 – **RR** time quantum 16 milliseconds
 - Q_2 – **FCFS**
- Scheduling
 - A new job enters queue Q_0 which is served FCFS
 - When it gains CPU, job receives 8 milliseconds
 - If it does not finish in 8 milliseconds, job is moved to queue Q_1
 - At Q_1 job is again served FCFS and receives 16 additional milliseconds
 - If it still does not complete, it is preempted and moved to queue Q_2



THREAD SCHEDULING

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on **LWP**
 - Known as **process-contention scope (PCS)** since scheduling competition is within the process
 - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

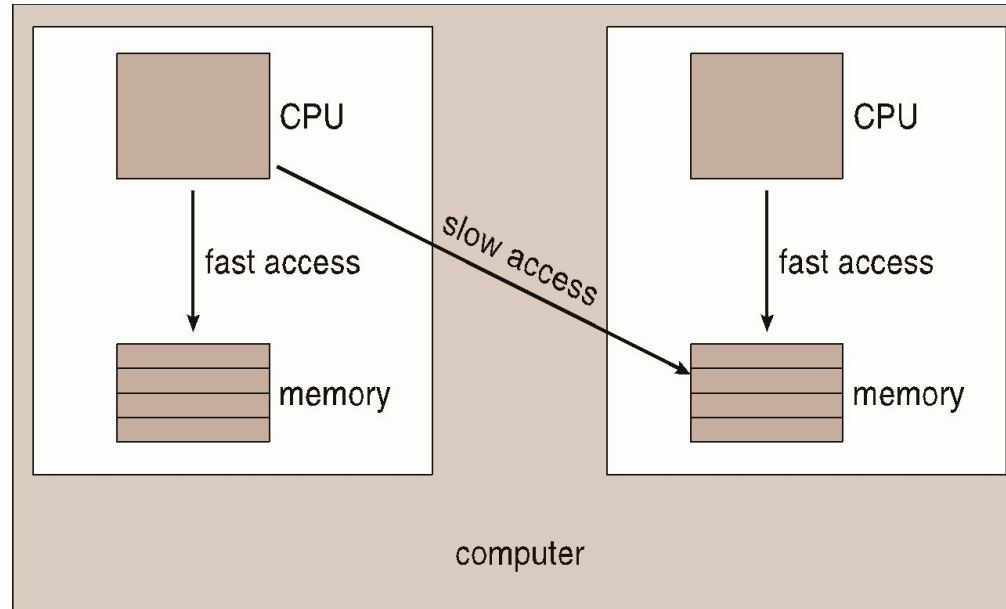
EXTRA

MULTIPLE-PROCESSOR SCHEDULING

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
 - Currently, most common
- Processor affinity – process has affinity for processor on which it is currently running
 - **soft affinity**
 - **hard affinity**
 - Variations including processor sets

EXTRA

NUMA and CPU Scheduling



Note that memory-placement algorithms can also consider affinity

EXTRA

Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

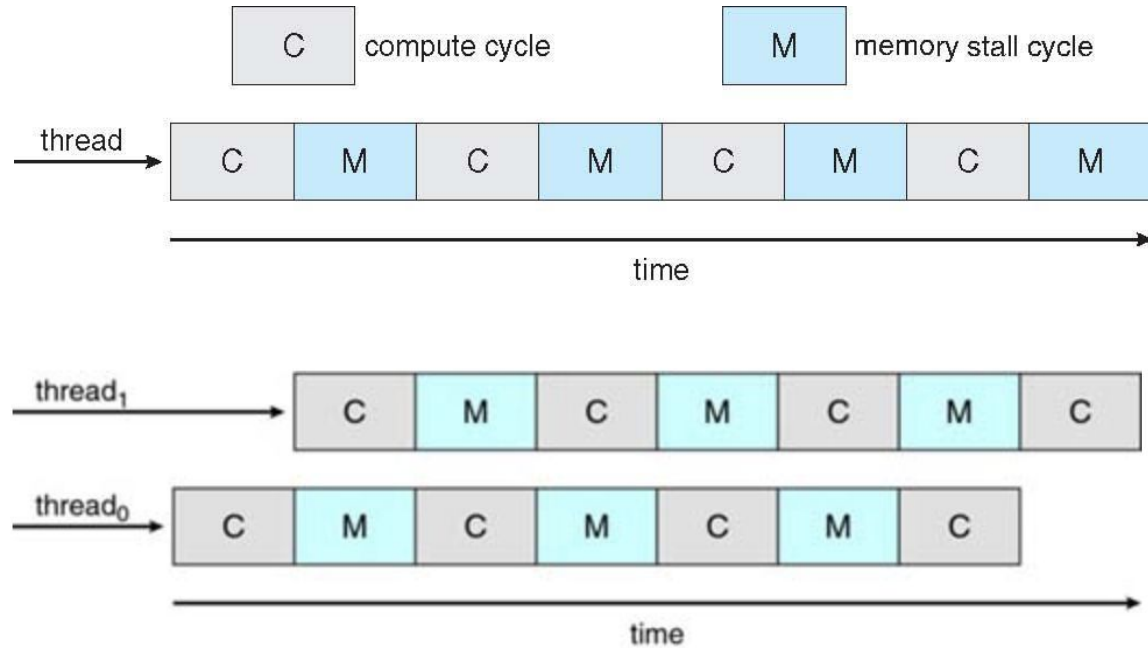
EXTRA

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
 - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

EXTRA

Multithreaded Multicore System




EXTRA




REVIEW QUESTIONS

CPU SCHEDULING




Which of the following is true of cooperative scheduling?

- A) It requires a timer.
- B) A process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- C) It incurs a cost associated with access to shared data.
- D) A process switches from the running state to the ready state when an interrupt occurs.




In preemptive scheduling, the sections of code affected by interrupts must be guarded from simultaneous use. True or False?



_____ is the number of processes that are completed per time unit.

- A) CPU utilization
- B) Response time
- C) Turnaround time
- D) Throughput




_____ scheduling is approximated by predicting the next CPU burst with an exponential average of the measured lengths of previous CPU bursts.

- A) Multilevel queue
- B) RR
- C) FCFS
- D) SJF




REVIEW QUESTIONS

CPU SCHEDULING




The ____ scheduling algorithm is designed especially for time-sharing systems.

- A) SJF
- B) FCFS
- C) RR
- D) Multilevel queue



Which of the following is true of multilevel queue scheduling?

- A) Processes can move between queues.
- B) Each queue has its own scheduling algorithm.
- C) A queue cannot have absolute priority over lower-priority queues.
- D) It is the most general CPU-scheduling algorithm.



CPU burst duration prediction.

Let $\alpha = 0.5$ (i.e., the actual duration of the most recent CPU burst and the expected duration of the most recent CPU burst are given equal weight). Also, let $n = 0$ to start. Assume that the initial expected CPU burst duration was $\tau_0 = 10$ time units, and that the seven observed CPU bursts for t_0 to t_6 were 6, 4, 6, 4, 13, 13, and 13, respectively.

Calculate τ_1 to τ_7 .

REVIEW QUESTIONS

FCFS



Throughput =
Average waiting time =
Average work ratio =

Process	Arrival Time	Actual Time	Waiting Time	Turnaround Time	Work Ratio
A	0	7			
B	4	4			
C	5	1			
D	9	1			
E	12	3			

Process	Time															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																

running
waiting

REVIEW QUESTIONS

SJF

- Here we assume that the actual duration and expected duration of the CPU bursts are equal
- We also assume that processes cannot be preempted



Throughput =
Average waiting time =
Average work ratio =

Process	Arrival Time	Actual Time	Waiting Time	Turnaround Time	Work Ratio
A	0	7			
B	4	4			
C	5	1			
D	9	1			
E	12	3			

Process	Time															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																

running
waiting

REVIEW QUESTIONS

SRTF

- This is preemptive SJF



Throughput =
Average waiting time =
Average work ratio =

Process	Arrival Time	Actual Time	Waiting Time	Turnaround Time	Work Ratio
A	0	7			
B	4	4			
C	5	1			
D	9	1			
E	12	3			

Process	Time															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																

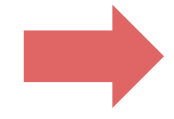
running
waiting



REVIEW QUESTIONS

RR

- 1 time quantum = 1 time column



Throughput =
Average waiting time =
Average work ratio =

<i>Process</i>	<i>Arrival Time</i>	<i>Actual Time</i>	<i>Waiting Time</i>	<i>Turnaround Time</i>	<i>Work Ratio</i>
A	0	7			
B	4	4			
C	5	1			
D	9	1			
E	12	3			

[illegible]

running
waiting



REVIEW QUESTIONS

PRIORITY SCHEDULING

- Assume the priority is the actual time



Throughput =
Average waiting time =
Average work ratio =

Process	Arrival Time	Actual Time	Waiting Time	Turnaround Time	Work Ratio
A	0	7			
B	4	4			
C	5	1			
D	9	1			
E	12	3			

[illegible]

```
running
waiting
```

REVIEW QUESTIONS

MLFB

Assume the following:

- Number of queues = 3 (i.e., 1, 2, 3)
- Priority queue 1 > queue 2 > queue 3
- Time quantum for queue 1 = 1 unit (FCFS algorithm)
- New processes are always appended to the tail of queue 1
- Time quantum for queue 2 = 2 units (Preemptive Priority algorithm, where the priority is the actual time)
- Processes are always inserted into queue 2 in priority order
- Time quantum for queue 3 = 1 unit (RR algorithm)
- Processes are always appended to the tail of queue 3
- Demote a process to a lower priority queue after 1 time quantum except when there are no other processes in the queue and in any of the lower level queues
- If a process is running and a new process arrives in a higher priority queue, the running process is always interrupted before the end of its time quantum to let the new process run



Throughput =
Average waiting time =
Average work ratio =

Process	Arrival Time	Actual Time	Waiting Time	Turnaround Time	Work Ratio
A	0	7			
B	4	5			
C	5	4			
D	9	1			
E	12	4			

Process	Time															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																

r_1 unning queue 1
 r_2 unning queue 2
 r_3 unning queue 3
 w_1 aiting in queue 1
 w_2 aiting in queue 2
 w_3 aiting in queue 3