



CS 3530 Operating Systems

OS Time Management

CS 3530 Operating Systems. Dr. Ken Hoganson, Copyright © 2008

- Time management is one of the functions of OS
- Efficiency is important
 - Time spent managing is “unproductive” time
 - User perceptions of system performance are critical to:
 - User satisfaction with system

- Operating systems are classified by time management/user management strategies

I. **Batch** operating system

- Not interactive - no user interaction
- processes (jobs) submitted into a job queue (waiting line)
- CPU completes one job before going on to the next
- poor efficiency - IF process does I/O, CPU waits
- poor performance - last process in the queue may wait a long time

- Enhances BATCH OS for better performance
 - Multiple programs (jobs) in the systems memory at the same time
 - When one job stalls for I/O, the CPU is switched to another process
 - Returns to first process when it is ready (an interrupt)
 - Efficiency: better - less wasted time waiting for I/O
 - More memory = more processes in memory = better chance that at least one process will be available for CPU work at all times
 - Goal is 100% CPU utilization
 - Still not interactive
 - Performance - much better, less wasted CPU time shows up as a faster system

- Improves Multi-Programming
- Of those programs in memory and available for CPU execution:
 - Each process gets a slice of time (time-slice or time-quantum)
 - Ordered “turns” - run through a list of processes, when at bottom of list, return to the top and repeat
- Works for interactive processing
 - each user (interface process) gets a little time-slice
 - computer time scale much faster than human time scale - appears that all users are working and getting computer service simultaneously
- Efficiency - good, little CPU wait time
- CPU utilization good

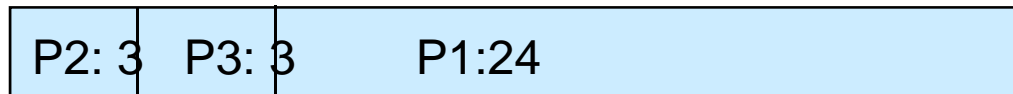
- Improves on time-sharing
- Allows prioritization of processes
 - some get more CPU time (bigger time slice)
 - OR, some get more CPU time slices
- Interactive user processes get high priority - fast response
- Other processing can occur at low priority in the background
- Operating system tasks at the highest priority
- Efficient - little wasted CPU time
- Performance - excellent - users perceive a very fast system
- Priorities can also be used to manage time allocations most efficiently

- How to allocate time among competing processes (jobs or user interfaces)
- Consider three processes:
 - Process 1 (P1) requiring 24 time quantum
 - Process 2 (P2) requiring 3 time quantum
 - Process 3 (P3) requiring 3 time quantum
- If executed in order: 30 total time

24	3	3
----	---	---

- CPU wait time over the 30 time quantum is zero, utilization is 100%
- The number of processes completed = 3 (throughput)
- Average wait time before getting the CPU is:
 - P1: 0
 - P2: 24
 - P3: 27 Average = $(0+24+27)/3 = 51/3 = 17$
- Average completion time (includes wait time) is:
 - P1: 24
 - P2: 27
 - P3: 30 Average = $(24+27+30)/3 = 81/3 = 27$

- How to allocate time among competing processes (jobs or user interface processes)
- Same three processes:
 - Process 1 (P1) requiring 24 time quantum
 - Process 2 (P2) requiring 3 time quantum
 - Process 3 (P3) requiring 3 time quantum
- If executed in different order: ^{30 total time}



- CPU wait time over the 30 time quantum is zero, utilization is 100%
- The number of processes completed = 3 (throughput)
- Average wait time before getting the CPU is:
 - P2: 0
 - P3: 3
 - P1: 6 Average = $(0+3+6)/3 = 9/3 = 3$
- Average completion time (includes wait time) is:
 - P2: 3
 - P3: 6
 - P1: 30 Average = $(3+6+30)/3 = 39/3 = 13$

- Even though the amount of work accomplished is the same, the second time allocation will be perceived by users as better - faster
- **Conclusion:** System Performance depends upon the workload mix and order that processes are executed in.
- The best performance is **Shortest Job First** (also Shortest cpu-Burst First, or Shortest Process First) **SJF**
- So, the OS should allocate time using SJF
- Problems?

- The time quantum required per process is not known until the process (job) completes
- Cannot see the future
- Need to predict or estimate the future
- Based on what? - Past performance
 - assume same as last time
 - average of last n bursts
 - complex trend analysis
 - random?
- Important to Minimize Overhead - minimize calculation time and memory required to save past data

- Uses a weighted average -
 - most recent CPU burst for a process t_n
 - cumulative old CPU past data for same process (also the previous prediction) p_n
 - Prediction for next burst is a weighted average of the two:

$$P_{n+1} = \text{alpha} * t_n + (1 - \text{alpha}) * p_n$$

Where *alpha* is the weighting factor (a fraction between 0 and 1).

- Low overhead: easy calculation, save only one old datum (p_n).

- Example: First burst was 4 quantum, second burst was 6
 - Prediction for third burst:
 $0.6(4) + 0.4(6) = 4.8$
 - 4.8 used to schedule by SJF.
- Third burst was actually 7
 - Prediction for fourth burst:
 $0.6(\text{old}-4.8) + 0.4(\text{recent}-7) =$
 $2.88 + 2.8 = 5.68$
 - Schedule fourth burst based on prediction
- Actual fourth burst and prediction used to predict fifth burst
 - etc. etc.

P1: 24 (5,5,4,6,4)

P2: 3 (2,1)

P3: 3 (1,2)

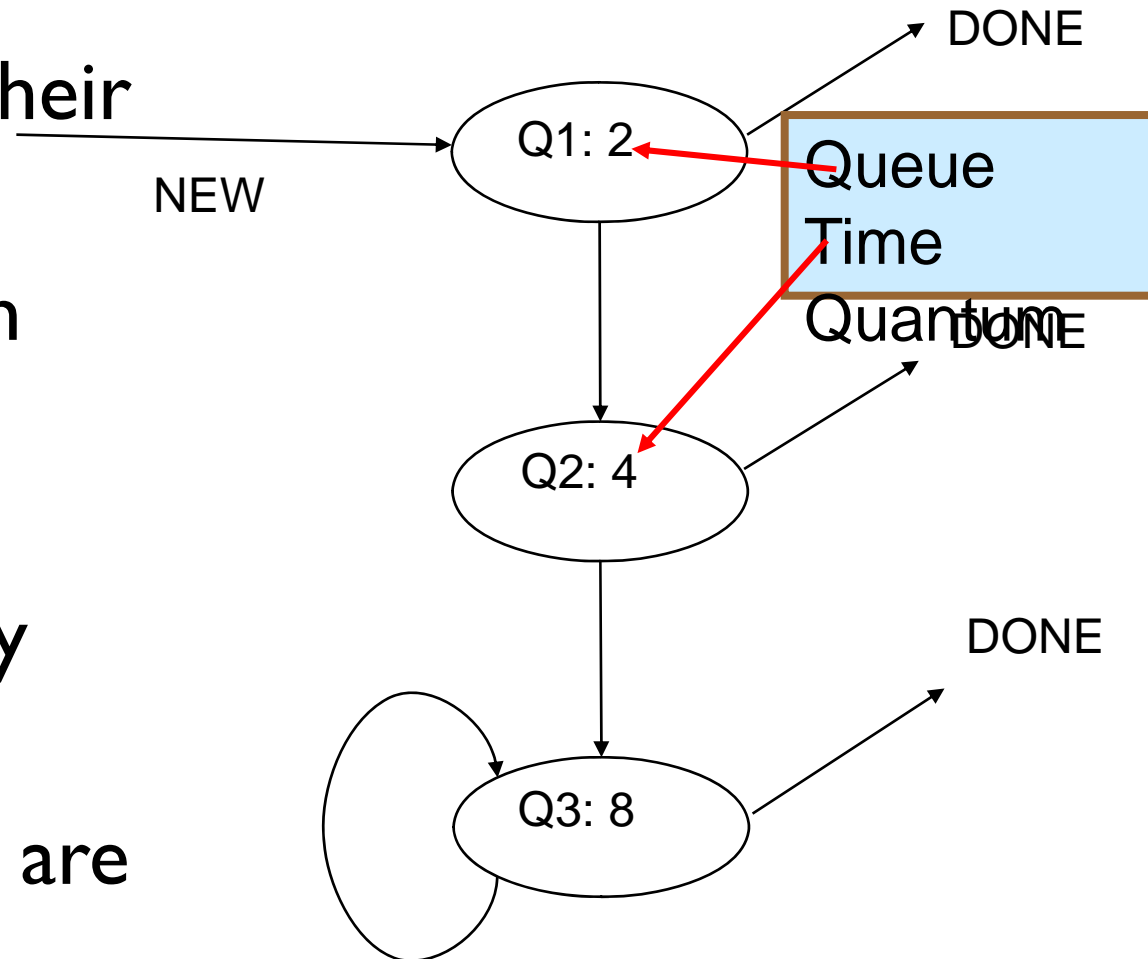
Default/Initial
Prediction=4

Alpha = 0.6

$$P_{n+1} = \text{alpha} * t_n + (1 - \text{alpha}) * p_n$$

1. All processes have the same prediction (default), so execute in order (P1 executes for 5). New prediction for P1 = $0.6(5) + 0.4(4) = 4.6$
2. Lowest prediction is 4 (P2&P3). Execute P2 for 2. New prediction for P2 = $0.6(2) + 0.4(4) = 2.8$
3. Lowest prediction is 2.8 (P2). Execute P2 for 1, P2 is complete.
4. Lowest prediction is 4 (P3). Execute P3 for 1. New prediction for P3 = $0.6(1) + 0.4(4) = 2.2$.
5. Lowest prediction is 2.2 (P3). Execute P3 for 2, P3 is now complete. Complete remaining process (P1).

- A non-computation method where processes “find their own level”
- Close to SJF, with low overhead
- Q2 gets no time until Q1 is empty
- Q3 gets no time until Q2 and Q1 are empty



- P1: 24, P2: 3, P3: 3
- Q1 gets P1, P2, P3
 - Q1 does P1 for 2, passes it to Q2
 - Q1 does P2 for 2, passes it to Q2
 - Q1 does P3 for 2, passes it to Q2
- Q2 does
 - P1 for 4, passes it to Q3 (Q1 is empty)
 - P2 for 1, it completes
 - P3 for 1, it completes
- Q3 does
 - P1 for 18, it completes (Q2 is empty)
- Average completion time = $11 + 12 + 30 = 53/3 = 17.67$
- compared to 13 for true SJF, and 19.6 for exp-ave.

- UNIX uses a priority queue somewhat similar to a MLFQ
- Each process has a priority number
- As process gets CPU time, its priority number becomes higher (lower priority)
- high priority processes (low numbers) get more CPU time
- long processes get low priorities (high number), thus favoring the shorter processes (SJF)

- For the following three jobs:
- P1: 10 (3,2,5)
- P2: 4 (2,2)
- P3: 16 (2,3,5,6)
- Execute the three jobs using Exp Ave and MLFQ. **ExpAve:** Alpha =0.6, default=2, ignore I/O wait times.

MLFQ: $q_1=1$, $q_2=2$, $q_3=4$. A process that exits the queue system due to a completed burst (for I/O wait), returns to Q_1 after 2 time quantum, but does NOT preempt the current queue that is executing, until that time slice is

**End
Of
Today's
Lecture.**



This slide intentionally left blank