Scheduling

COMP450 Operating Systems

Goals of Scheduling

- User Oriented
  - minimize response time (interactive)
  - minimize Turnaround time (batch jobs)
  - Meet deadlines
- System Oriented
  - Predictable
  - maximize Throughput
  - Processor Utilization
  - Fair
  - Enforce priorities
  - Balance resources
  - Balance user groups

Levels of Scheduling

- Long Term scheduling
- Medium Term scheduling
- Short Term scheduling
- I/O scheduling

Long-term scheduling

- Only relevant for systems that have a backlog of batch jobs to run
- Deals with creating a new process
- Controls the degree of multiprogramming
- Interactive systems tend to accept users unless the system is swamped
- The more processes, the smaller percentage of time each process is executed

This is all we are going to say about long-term scheduling

Medium-term scheduling

- The decision to add to the number of processes that are partially or fully in main memory
- Deals with swapping processes in and out
- A swapped out process cannot be executed until it is swapped back into RAM

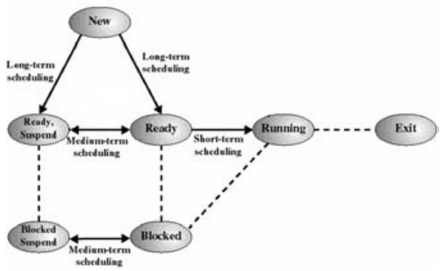
Medium-term scheduling will be discussed when we talk about memory management.

Short-Term Scheduling

- Determines which process is going to execute next
- The short term scheduler is known as the dispatcher
- Is invoked on an event that may lead to choosing another process for execution:
  - clock interrupts
  - I/O interrupts
  - operating system calls and traps
  - signals

Short Term Scheduling is the subject of this chapter

Classification of Scheduling Activity



- Long-term: which process to admit
- Medium-term: which process to swap in or out
- Short-term: which ready process to execute next

Types of Dispatch Algorithms

- Non-preemptive
  - Once the thread is started, it continues to run until it voluntarily gives up control.
  - May cause one thread to monopolize the system.
  - Useful for real-time systems
- Preemptive
  - The dispatcher may select another thread to run after a while.
  - Most systems are preemptive.

Dispatcher Algorithms

- FCFS
- Round robin
- Feedback queues
- Priority
- Shortest Job First
- Shortest Process Next (shortest estimated CPU burst)
- Real time scheduling

Common Elements

- Each algorithm selects a thread from the ready list and runs it.
- None of the scheduling algorithms will allow the processor to be idle if there is a thread to run.
- The CPU can execute in parallel with I/O.
- When a program is waiting for I/O to complete, another thread can run.
- If there is only one thread on the ready list, it doesn't matter what algorithm you use.

Scheduling Algorithm Evaluation

- Average time to completion
  - minimize
- Average total wait time
  - minimize
- CPU utilization
  - maximize

Priority Scheduling

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority
- May be preemptive or non-preemptive
- Lower-priority may suffer starvation
  - allow a process to change its priority based on its age or execution history

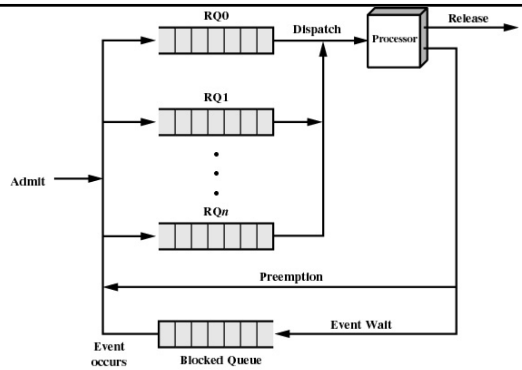


Figure 9.4 Priority Queuing

First-Come-First-Served (FCFS)

- The FCFS algorithm runs the oldest thread until it blocks and then runs the next oldest thread until it blocks etc.
- This is usually a non-preemptive algorithm.

Round-Robin

- Clock interrupts are generated at periodic intervals
- When a clock interrupt occurs, the currently running process is placed in the read queue. The next ready job is selected
- Known as time slicing

Shortest Job First

- This algorithm selects the thread that has the least amount of processing required to run to completion.
- This is the optimal algorithm. It provides the minimum average wait time.
- Since we can't look into the future, we don't easily know which thread has the least amount of time left to finish.
- Impossible to implement.

Shortest Process Next

- An attempt to implement the shortest job first algorithm.
- The thread that is likely to use the least amount of CPU time is selected.
- The amount of CPU time a thread is likely to use until it is blocked is computed by a time weighted average of the length of its last several CPU bursts.

Feedback Queues

- The dispatcher maintains several queues, often a short, medium and long queue.
- Threads in the short queue are executed for a short time quantum. Threads in the medium and long queue are executed for successively longer time quantums.
- New threads enter in the short queue.
- When a thread moves from the blocked state to the ready state, it enters the short queue.

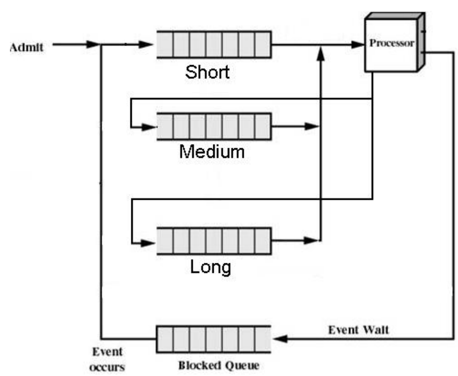
Feedback Queues

- The dispatcher runs threads in the short queue in the FCFS order. As long as there are threads in the short queue, they are executed.
- If a thread from the short queue uses all of its time quantum without relinquishing the CPU, it is moved to the medium queue.

Feedback Queues

- Threads in the medium queue are executed only when the short queue is empty.
- Threads in the long queue are executed only when the short and medium queues are empty.

Feedback Queue



Feedback Analysis

- Threads that do lots of I/O and very little processing will be in the short queue. They will get the CPU ahead of CPU bound threads.
- Threads using the CPU for longer periods will run for longer time quantums to reduce the number of context switches.
- CPU bound threads run only when there are no other threads to run.

Fair Share Scheduling

- Imagine your program and my program are competing for the CPU.
- If each program has one thread, then a fair scheduling algorithm will give each thread 50% of the available time.
- Assume my program creates two more threads for a total of three. If each thread gets an equal share of the CPU, my program gets 75% of the CPU.

Prioritized, Shortest Process Next, Multilevel Feedback Queue Algorithm

- The best of the described algorithms can be combined to create a scheduling algorithm sure to please everyone.

A Dr. Williams original.

PSPNMFQ Algorithm

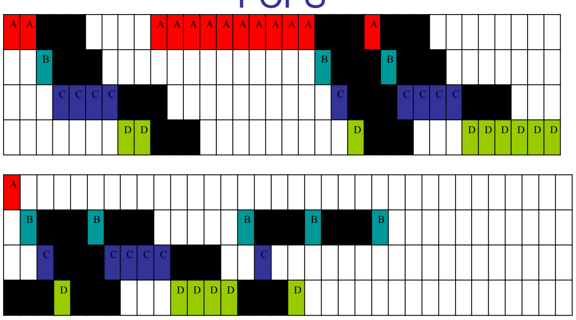
- Each thread is assigned a priority number.
- When a thread uses its time quantum, the priority number is lowered.
- When a thread is blocked, its priority number is raised.
- If a thread hasn't been executed for a while, its priority number is increased.
- The highest priority numbered thread is selected.
- Time quantum length is inversely proportional to priority.

Example Programs

- *Assume no overlap of I/O and processing.*
- *All I/O requests take 3 units of time.*

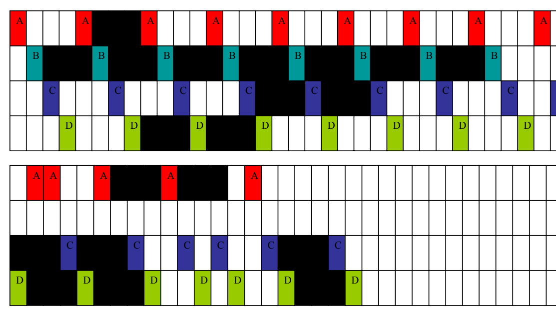
A starts at 0 priority low
2 I/O 10 I/O 1 I/O 1
B starts at 1 priority medium
1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1
C starts at 2 priority high
4 I/O 1 I/O 4 I/O 1 I/O 4 I/O 1
D starts at 3 priority medium
2 I/O 1 I/O 6 I/O 1 I/O 4 I/O 1

FCFS



A 2 I/O 10 I/O 1 I/O 1 **B** 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1
C 4 I/O 1 I/O 4 I/O 1 I/O 4 I/O 1 **D** 2 I/O 1 I/O 6 I/O 1 I/O 4 I/O 1

Round Robin



A 2 I/O 10 I/O 1 I/O 1 **B** 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1 I/O 1
C 4 I/O 1 I/O 4 I/O 1 I/O 4 I/O 1 **D** 2 I/O 1 I/O 6 I/O 1 I/O 4 I/O 1

Evaluation

- Average time to completion
  - FCFS= 35 + 57 + 50 + 52 = 48.5
  - RR= 49 + 30 + 54 + 55 = 47
- Processor Utilization
  - FCFS = 52/57 = 91.2%
  - RR = 52/55 = 94.5
- Context Switches
  - FCFS = 27
  - RR = 53

Multiprocessors

- There are many different types of computers with multiple processors. The most common system in use today is the **S**ymmetric **M**ulti**P**rocessor (SMP).
- All of the CPUs in an SMP system are identical and can address all of the RAM

Multiprocessor Scheduling

- **Very little has to be done to schedule a multiprocessor system.**
- Whenever a CPU needs a process to run, it takes the next task from the ready list.
- The scheduling queue must be accessed in a critical section. Busy waiting is usually used.

Considerations for SMP scheduling

- With multiple CPUs, it is not as likely that a short task will get stuck waiting for a long task to complete. Therefore the selection of the next task is not as important.
- Any task can run on any CPU thereby allowing load balancing.
- Tasks should stay with a single CPU when feasible to take advantage of cache loading.

Real Time Programming

- Real-time computing requires that the result not only be correct, but produced within a specific time limit.
- Real-time programming is used in process control to ensure that the system reacts to an input in time.
- Examples are chemical plant control or robotic control.

Real Time Scheduling

- Many real time systems run a known collection of tasks. The execution time of the tasks is frequently known ahead of time.
- Tasks have deadlines by which they must complete.
- If a task that runs for 3 time units must be done at time 10, it must start by time 7.
- If two tasks that runs for 3 time units each must be done at time 10, one must start by time 4.

Static R/T Scheduling

- **Static predetermined schedules** - A schedule is devised before hand from a list of known tasks, execution times and deadline times.
- **Static predetermined priorities** - Using a list of known tasks, execution times and deadline times, priorities are assigned to each task. A regular priority based scheduler is used.

Dynamic R/T Scheduling

- **Dynamic planning based** - When a task is scheduled, the algorithm determines if it can be feasibly executed in time.
- **Dynamic best effort** - The scheduler tries to meet deadlines by raising the priority of tasks as they approach their deadline. Tasks are aborted if they don't or can't meet their deadline.