●●CPU Scheduling: •FCFS: non-preemptive •Round robin: preemptive. When the quantum time allocated for a process finishes. •Shortest Remaining Time Next: preemptive. When a new process arrives and it has a shorter service time as compared to the running process. What happens if the time slice allocated in a Round Robin Scheduling is very large? And what happens if the time slice is very small? If time slice is very large, it results in FCFS scheduling. If time slice is too small, the processor through put is reduced, since more time is spent on context switching. Assume that P1 and P3 are ready for execution, but P2 becomes ready after 2 time units. Can SPN still perform well? How can we modify the scheduling algorithm to still achieve an optimal execution order that minimizes the average turnaround time. What is the resulting execution schedule? If SPN is used, the turnaround time will be similar to that of the FIFO. P1 runs from 0 to 7, P2 runs from 7 to 8, and finally P3 runs from 8 to 24. The scheduling algorithm can be modified by preempting the running process if a shorter process is added to the system, or shortest remaining time next. In this case: •P1 runs from 0 to 2 •P2 preempts P1 and runs from 2 to 3 – turnaround time =1 •P1 resumes from 3 to 8 - turnaround time = 8 •P3 runs from 8 to 24 - turnaround time = 24. Therefore, the average turnaround time is 11. Race condition: race conditions when data are shared among several processes Amdahl's law:60% parallel gives clear idea that remaining 40% is serial. Formula for speedup = 1/(S + (1-S)/N) S = 40%, N = number or processing cores which is 4 1/(0.4 + (1-0.4)/4) = 1/0.55 = 1.818
CPU Scheduling
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? 1. Switches from running to waiting state 2. Switches from running to ready state 3. Switches from waiting to ready 4. Terminates
Running-Waiting & Terminates: non-preemptive Running-Ready & Waiting-Ready: preemptive Running-Waiting & Terminates: non-preemptive Running-Ready & Waiting-Ready: preemptive CPU-I/O Burst Cycle: cycle of CPU execution and I/O wait
CPU Burst distribution: main concern in CPU scheduling dispatcher: gives control of the CPU to the process selected by the short-term scheduler dispatcher underlying processes: 1. switching context 2. switching to user mode 3. jumping to proper location in the user program dispatch latency: time it takes for the dispatcher to stop one process and start running another one Scheduling Criteria: 1. CPU utilization 2. Throughput 3. Turnaround time 4. Waiting time 5. Response time CPU utilization: keep CPU busy as possible
Throughput: number of processes that complete their execution Turnaround time: time to execute a process Waiting time: time taken by the process waiting in ready queue
Response time: time take from when a request was given until first response is produced
Scheduling Criteria Optimization: 1. Max CPU utilization 2. Max throughput 3. Min. turnaround time 4. Min. waiting time 5. Min. response time Scheduling algorithms: 1. First-Come, First-Serve (FCFS) 2. Shortest-Job-First (SJF) 3. Shortest-Remaining-Time-First 4. Round Robin (RR) First-Come, First Serve: where processes are done in who gets scheduled first (technically like a queue) convoy effect: FCFS problem where short processes are behind long ones exponential averaging: used to determine the length of the next CPU burst in SJF Shortest-job-first: example of priority scheduling
Shortest-remaining-time-first: preemptive version of SJF priority scheduling: priority number assigned to each process Problem in priority scheduling: starvation (low priority processes may never execute) Solution to priority scheduling problem: aging (increase process priority as time goes by) time quantum (q): a small unit of CPU time used in RR
Round-Robin performance issues: 1. large q - degenerate to FCFS (FIFO) 2. small q - always do context switch -> dispatch latency -> overhead multilevel queue: process permanently assigned in either ready queue partitions: foreground (interactive) or background (batch) Scheduling algorithms in multilevel queues: 1. foreground - RR 2. background – FCFS Scheduling between queues in multilevel queue: 1. fixed priority scheduling (possibility of starvation) 2. time slice (each queue with certain times) 3. 20% to background in FCFS multilevel feedback queue: process can move between the various queues (aging can be implemented) multilevel feedback queue parameters: 1. no. of queues 2. scheduling algorithm for each queue 3. method to know: - when to upgrade a process - when to demote a process - which queue a requesting process will enter
process contention scope (PCS): scheduling user-level threads within the process (to run on LWP's)
system contention scope (SCS): scheduling kernel threads in system
PTHREAD_SCOPE_PROCESS: PCS scheduling for pthreads
PTHREAD_SCOPE_SYSTEM: SCS scheduling for pthreads
Linux and Mac OS X: only allow PTHREAD_SCOPE_SYSTEM
multiple-processor scheduling: involves homogeneous processors within a multiprocessor
asymmetric multiprocessing: only one process accesses the system data structures ymmetric multiprocessing: most common; each process is self-scheduling, in common ready queue or private queue with ready processes processor affinity: - process has an affinity for a processor - uses cache Types of processor affinity: 1. soft affinity 2. hard affinity 3. processor sets nice: Linux command to change priority taskset: Linux command to change to processor affinity psr: Linux command to know specific processor assigned to processor
ps - eopid, cmd, nice, rtprio: Linux commands that shows process-related info
load balancing: attempts to keep workload evenly distributed
push migration: check load on each processor and push found task from overloaded CPU to other CPU's
pull migration: idle processors pulls waiting task from busy processor multicore processors: each core has hardware threads where they take advantage of memory stall to make progress with another thread soft real-time systems: no guarantee as to when critical real-time process will be scheduled hard real-time systems: task must be serviced by its deadline
types of latency in real-time CPU scheduling: interrupt latency & dispatch latency interrupt latency: time from interrupt arrival to start of interrupt service routine dispatch latency in real-time CPU scheduling: time for scheduler to take a process off the CPU and switch to another real-time scheduling: scheduler must support preemptive and priority-based scheduling hard real-time scheduling: must provide ability to meet deadlines periodic processes: require CPU at constant intervals virtualization: - schedules multiple guests onto CPU(s) with each guest having its own scheduling - can undo good scheduling efforts by guests rate monotonic scheduling: - assigning priority based on the inverse of its period - shorter periods = higher priority, vice versa earliest deadline first scheduling: the earlier the deadline, the higher the priority and vice versa proportional share scheduling: shares are allocated among all processes in the system deterministic evaluation: - analytic evaluation where it takes a particular workload and defines the performance of each algorithm for it - requires exact numbers for input queuing models: describes process arrivals and CPU & I/O bursts probabilistically Little's law: in a steady state, processes leaving the queue must equal processes arriving simulation: programmed model of a computer system that is more accurate than queuing models implementation: accurate yet high risk and cost method that implements a new scheduler and tests it in real systems Describe the cycles of CPU and I/O bursts? First CPU will start, alternating between I/O as it's needed, and back to CPU. Finally, the CPU will terminate the processes execution. Compare I/O bound programs with CPU bound programs in terms of CPU bursting habits? I/O bound will have many small CPU bursts. CPU bound will tend to have few, long CPU bursts. What are some advantages of non-preemptive (aka cooperative) CPU scheduling? No special hardware needed (for interrupts, etc). No race conditions. What are the Pros and Cons a FCFS CPU scheduling algorithm? Pros: simple to implement, fair. Cons: wait time is quite high, convoy effect when you have a CPU bound process hogging the CPU. FCFS is non-preemptive. SJF scheduling is optimal. Why can't it be successfully implemented by the short-term scheduler? AKA shortest-next-CPU-burst. No way to reliably predict which job is the shortest or what jobs will be queued up subsequently. What is preemptive SJF called? shortest-remaining-time-first How does SJF relate to Priority Scheduling? SJF is a version of priority scheduling where priority is the inverse of the next predicted CPU burst. What is aging? A method to avoid indefinite blocking/starvation, where low priority processes are eventually moved to higher priority so they will be executed. How does RR scheduling add on to FCFS? Round robin slices up CPU time into time quantums or time slices. Each process is placed in a circular queue and gets preemptive access to the CPU for their time slice. How does multilevel queue scheduling work? Different types of processes (say, background and foreground) can be placed into difference scheduling queues, using different algorithms and priorities. How do multilevel feedback queues modify multilevel queues? plain multilevel queues typically have static queue assignments for processes. With the feedback version there is dynamic process assignment between queues, depending on the status of the system. This can help optimize resource usage and avoid starvation. What is the difference between PCS and SCS? Process-contention scope pertains to the implementation of many-to-one and many-to-many models where user-level threads must be assigned a LWP. So only user-level threads are involved. System-contention scope pertains to all threads on the system, including kernel threads. By the way one-to-one model OSs only use SCS (so Windows, Linux, Solaris). Compare asymmetric and symmetric multiprocessor scheduling? Asymmetric involves a "master server" processor making all scheduling decisions, I/O processing and other system activities, while the other processors execute only user code. This can be good since only once process will need access to the system data structures. Symmetric: all processors handle their own scheduling. All processes may be in a common ready queue, or each processor may have its own private queue of ready processes. Regardless, scheduling proceeds by having the scheduler for each processor examine the ready queue and select a process to execute What is the point of processor affinity? To keep a process preferentially on the processor it has been already running on, preserving the cache availability of its data. Soft affinity: prefers but not bound. Hard affinity: assigned set of processors. How does NUMA pertain to processor affinity? Non-uniform memory access just reflects the fact that often a CPU has different access speeds to different parts of main memory. Compare push and pull migration for load balancing. push: A task examines the load of each processor and will move a process from a busy to a less busy processor. pull: where an idle processor will snag a process from a busy processor. load balancing can counteract the benefits of processor affinity.
How do multithreaded cores improve multicore CPU efficiency? Multicores are subject to memory stalls, due to cache misses, etc. How do coarse-grained and fine-grained multithreading differ? coarse-grain allows a thread to execute until a long latency event (memory stall) occurs. fine-grain is also known as interleaved threading, switches back and forth constantly, using specialized hardware. CPU Scheduling may happen when: 1. Switches from running to waiting state (wait for I/O; wait for child process to terminate): non-preemptive 2. Switches from running to ready state (when an interrupt occurs): preemptive 3. Switches from waiting to ready (completion of I/O): preemptive 4. Terminates: non-preemptive Dispatcher: -performs context switch -switch to user mode - start program at correct place Scheduling Criteria: CPU utilization, throughput, turnaround time, waiting time, response time Scheduling Criteria: throughput: processes over time Scheduling Criteria: turnaround time: time for one process Scheduling Criteria: Waiting time: time a process waits in the ready queue Scheduling Criteria: Time from when a request was submitted to its first response FCFS Scheduling: First come, first served. Intrinsically fair but is not the fastest SJF Scheduling: Shortest job first. Best because shortest wait time. Hard to know how future jobs will behave Formula to estimate the length of CPU bursts: 1/2 last estimated time + 1/2 last actual time SRTF Scheduling: Shortest Remaining Time First. Consider both arrival time and burst time. Preemptive SJF.

Round robin: Each process given fixed quantum of time
Multi-level queue: ready queue partitioned. Each has its own scheduling algorithm
Process-contention scope (PCS): The scheduling of user threads WITHIN a process to run on available LWPs, done by thread libraries.
system-contention scope (SCS): scheduling kernel threads in a SYSTEM
CPU Scheduling: The act of determining which process in memory is given access to the CPU so that it may execute
Preemptive scheduling decisions: Switching from running to ready -Switching from waiting to ready
nonpreemptive scheduling decisions: -Switches from running to waiting -Terminates
Dispatcher: The module that gives control of the CPU to the process selected by the short-term scheduler
Dispatch latency: time it takes for the dispatcher to stop one process and start another running
Scheduling Criteria: CPU utilization, throughput, turnaround time, waiting time, response time
First-Come First-Served: allocates resources to those who are first in line
Shortest-Job-First: A scheduling algorithm that deals with each user or task based on the getting the smaller ones out of the way
CPU burst: the amount of time the process uses the processor before it is no longer ready
Priority Scheduling Problem: Starvation- low priority processes may never execute
Priority Scheduling Solution: Aging- as time progresses increase the priority of the process
Round Robin Scheduling: Preemptive process scheduling algorithm. Each process is provided a fixed time to execute, it is preempted to the back of the queue
Multilevel Queue: Each queue has its own scheduling algorithm and scheduling is done between queues
Thread Scheduling: distinction between user-level and kernel-level threads
User-level thread scheduling: Process-contention scope
Kernel thread scheduling: System-contention scope
Asymmetric multiprocessing: only one processor accesses the system data structures, alleviating the need for data sharing
Symmetric multiprocessing: each processor is self-scheduling
Soft affinity: OS attempts to keep process on specific processor but not guaranteed.
Hard affinity: Process guaranteed to run on a specific type of processor.
Scheduling performance criteria are: -CPU utilization Keep the CPU as busy as possible. Higher is better. (close to 90% is ideal). -Throughput The number of processes that complete execution per time unit. Higher is better. -Turnaround time Amount of time needed to execute a particular process from time of submission to time of completion. Sum of times waiting in queues, waiting to get into memory, executing on the CPU and operating I/O. Lower is better. -Waiting time Amount of time a process waits in the ready queue before execution. Lower is better. -Response time Time from when a request was submitted until the first response is produced (not the total output). Lower is better -Fairness A measure of the probability that every job gets a chance to run. By default all processes are equiprobable to run. The probability distribution is subjective.
List 6 Scheduling Algorithms: First Come First Serve (FCFS), Shortest-Job-First (SJF), Priority, Round-Robin, Multi-Level Queue, Multi-Level Feedback Queue
Convoy effect: short process behind long process
There are two variations of SJF scheduling method: -Non-preemptive: Once the process is in the running state, it continues to execute until it terminates or it blocks itself to wait for I/O etc. -Preemptive (Shortest Remaining-Time-First): If a new process arrives with a CPU burst time less than the remaining time of the currently executing process, then we preempt and move it back to the ready state, and run the newly arrived process instead.
main problem with SJF? Solution? we don't know the remaining length of the CPU burst, or even the entire CPU burst!
We can expect that the next burst will be similar to previous bursts, and by looking at the average of the most CPU bursts, and the trend (increasing or decreasing) we can make a prediction how long the process will run for.
Priority Scheduling: The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
Priority Scheduling. Problem & Solution: Starvation (low priority processes may never execute) - Aging (as time progresses increase the priority of the process)
CPU-I/O Burst Cycle: Process execution consists of a cycle of CPU execution and I/O wait. Success depends on ability to observe cycles of I/O on CPU activity
Preemptive scheduling: CPU scheduling that occurs when the operating system decides to favor another process, preempting the currently executing process
CPU Scheduling Criteria: 1) MAX CPU Utilization 2) MAX Throughput (process/time) 3) MIN Turnaround time (how long to execute a process) 4) MIN Waiting Time (time in ready queue, not executing) 5) Response Time (first time user can respond)
FCFS Scheduling: Jobs are processed in the order in which they arrive. Very simple to implement, but long avg. waiting time
SJF (Shortest Job First Scheduling): The process with the next shortest CPU burst will go first. FCFS used to break ties. This is optimal, but it's nearly impossible to determine the CPU burst time. It can be approximated by TAUn+1 = alpha Tn + (1 - alpha) TAUn
Priority Queue Scheduler: Each process gets a priority and the CPU gets allocated to process with highest priority.
Improving Fairness: -give each queue a fraction of the CPU time -> only works if jobs are equally distributed -adjust priority of jobs as they do not get services -> reduces starvation but avg waiting time suffers when system overloaded because everything is given high priority
Multilevel Feedback Queues: - multiple queues w/ individual priorities - OS does Round Robin at each level - then moves down to next level - time slices increases as you go down
SJF: - least amount of CPU work goes firt - I/O bound jobs go before CPU bound jobs - works for both premptive and non-premptive - adaptive -> uses past to predict future
Round Robin: - timer -> premptive policy - time slices selection:
too large: p much FIFO
too small: too much overhead b/c of
context switches
balance: context switch 1% of time slice
today -> typically 10-100 ms w/ a context switch of .1 to 1 ms
FCFS: - FIFO - arrival order - run until complete or I/O - initially didn't even stop for I/O
Real CPU schedules: - minimize response time - minimize variance of average response time - predictability -> low avg and high variance - maximize throughput - minimize overhead - efficient use of resources - minimize wait time - fairness -> equal wait times
Ideal CPU Schedule: - maximizes CPU utilization & throughput - minimizes turnaround time, wait time, and response time - requires knowledge of processes that we usually don't have
Two Kinds of Scheduling Policies: *non-preemptive: - scheduler only runs when process blocks or terminates (not on interrupts) *preemptive: - scheduler runs on timer interrupt mostly, but also system calls, and other hardware interrupts - most OSes
When does the scheduler execute? - a process switches from running to blocked- a process is created or terminated- an interrupt occurs- timer- I/O
Booting OS Kernel? - boot from ROM- boot program steps- examines machine configuration (# of CPUs, amt. of memory, # and type of hardware services - builds a configuration structure describing the hardware - boots the OS kernel & gives it the configuration structure
Booting OS Kernel (Continued): - OS Initialization- initialize kernel data struct- initialize state of all hardware devices- creates a number of processes to start operation (Getty in Unix) - then OS runs user programs, if not, enters idle loop (OS does system management and profiling and halts processor for low power mode) - wakes upon hardware interrupt
What is Allocation & how does it defer from Scheduling? Allocation is, given a set of jobs and resources it has to use, allocation gets to decide which jobs get which resources and how much of it scheduling on the other hand determines the sequence of executions (jobs)
Goal of CPU Scheduling: maximize cou utilization, minimize job response times, maximize throughput
What kind of information do we need for CPU Scheduling? given the fact you are scheduling jobs(processes) some useful information about them must be needed - you need resources -cpu itself-i/o resources-memory jobs -number of processes-process creation rate and termination rate-how long a process runs -how long a process uses CPU vs I/O
Scheduling approaches: im sure theres no single way to schedule so what are some common ways to do so-preemptive-nonpreemptive
main difference can be described as, one runs one by one where scheduler is not in ultimate command and the other runs with the scheduler running this bish
non-preemptive:-scheduler does not interrupt running process-current process runs until blocks, waiting for an event or a resource, or it terminates
preemptive:-scheduler forces current process to release the cpu -eviction of process occurs at clock interrupt, i/o interrupts
first in first out (fifo): simplest scheduling policy
arriving jobs are inserted into the tail(rear) of the ready queue
-job to be executed next is removed from the head**performs better for long jobs**relative importance of jobs measured only by arrival time
Problems with it:
-a long cpu bound job may hog the cpu
-short (i/o bound) jobs have to wait for long periods
-can lead to a lengthy queue of ready jobs known as the convoy effect
shortest job first: selects job with the shortest expected processing time first problem with this: -need to know or estimate the processing time of each job -long running jobs may starve for the cpu when there is a steady supply of short jobs

Priority based each process is assigned a priority: -process chosen for execution based on propriety (highest first) -propriety + preemption allows preemption when a higher priority process comes in -priority +aging boosts the priority as a process stays in the ready queue -.to prevent starvation

Comparison of scheduling policies: performance of scheduling depends on job characteristics
FCFS: -performs better for long processes -not suitable for interactive jobs -negligible processing overhead
RR: -not suitable for long batch jobs -good for interactive jobs -medium processing overhead
SJF: -long processes can starve -can have high processing overhead
how would u classify schedulers? given that we have seen many kinds of schedulers ant what not we can classify them from the perspective of
long term job scheduling: -done when a new process is created, it controls the degree of multi-programming

medium term: -involved suspend/resume processes by swapping them out of or in to memory
short term: -occurs most frequent and decides which process to execute next
Multi level feedback queue
scheduler design is based on a lot of different things one being the system used...
we will look at a multi queue priority based scheduler
it has many queue and each corresponds to a priority: -if a job shows up at a higher priority queue it will preempt a job that is running from a lower priority -when a job is selected to run does it run ti completion?
**two rules: one is preemptive and the other isn't the preemptive one is like round robin with a quantum
how to set priority?: -depends on system -isolate tasks that are interactive from the one that are non interactive
multi level feedback queue design: pick a job from queue k only when there is no jobs in queue 1 to k-1
run a job from queue k to maximum time where T is the time for which queue 1 jobs are run: if a rob runs for more than allowed amount of time , it is put into the next queue so the job goes from k to k+1
a job arriving in queue 1 would preamp the execution of job from queue k where k1 this is an adoption of the ml discipline of running the highest priority condition for certain duration of time, we have elf designs that allow running job to continue until the next time quantum and then selects the highest priority jobs
mlf parameters
number of queen
scheduling algothigm for each queue
method used to determine when to promote a process to a higher queue
lower)
knowing which process will work
What does process execution consist of? a cycle of CPU execution and I/O wait
CPU scheduling decisions take place when a process does what? 1) switches from running to waiting, 2) switches from running to ready, 3) switches from waiting to ready, 4) terminates. The dispatcher module does what? it gives control of the CPU to the process selected by the short-term scheduler. Define dispatch latency: the time it takes for the dispatcher to stop one process and start another. Scheduling algorithms are chosen based on what? optimization criteria. List some examples of scheduling algorithms: FCFS, SJF, Round Robin, Priority, Multi-level Queue, Multi-level Feedback-Queue. What is exponential averaging? an algorithm that determines the length of the next CPU burst
Define Exponential Averaging $t(n-1) = \alpha*t(n) + (1-\alpha)t(n)$; $t(n)$ = actual length of nth CPU burst; $t(n-1)$ = predicted value for the next CPU burst, α, $0 \leq \alpha \leq 1$ (commonly α set to 1/2)
Priority scheduling can result in _____ which can be solved by _____ a process? starvation, aging.
What does aging a process do? as time progresses, the process increases in priority
What happens in round robin scheduling? small time quantums can result in large amounts of context switches; time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum.
What schedule types have multiple process queues that have different priority levels? multi-level queues and multi-level feedback queues
In a feedback queue, priority is fixed. True or False? false; processes can be promoted and demoted to different queues
In what ways can multiprocessor scheduling be done? asymmetric multi-processing, symmetric multi-processing, and processor affinity
Define asymmetric multi-processing: only one processor accesses system data structures; no need to data share
Define symmetric multi-processing: each processor is self-scheduling (currently the most common method)
Define processor affinity: A process running on one processor is more likely to continue to run on the same processor (so that the processor's memory still contains data specific to that specific process)
What does process execution consist of? a cycle of CPU execution and I/O wait
CPU scheduling decisions take place when a process does what? 1) switches from running to waiting, 2) switches from running to ready, 3) switches from waiting to ready, 4) terminates
The dispatcher module does what? it gives control of the CPU to the process selected by the short-term scheduler
Define dispatch latency: the time it takes for the dispatcher to stop one process and start another
Scheduling algorithms are chosen based on what? optimization criteria
List some examples of scheduling algorithms: FCFS, SJF, Round Robin, Priority, Multi-level Queue, Multi-level Feedback-Queue
What is exponential averaging? an algorithm that determines the length of the next CPU burst
Priority scheduling can result in ____ which can be solved by _____ a process? starvation, aging
What does aging a process do? as time progresses, the process increases in priority
What happens in round robin scheduling? small time quantums can result in large amounts of context switches; time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum
What schedule types have multiple process queues that have different priority levels? multi-level queues and multi-level feedback queues
In a feedback queue, priority is fixed. True or False? false; processes can be promoted and demoted to different queues
In what ways can multiprocessor scheduling be done? asymmetric multi-processing, symmetric multi-processing, and processor affinity
Define asymmetric multi-processing: only one processor accesses system data structures; no need to data share
Define symmetric multi-processing: each processor is self-scheduling (currently the most common method)
Define processor affinity: A process running on one processor is more likely to continue to run on the same processor (so that the processor's memory still contains data specific to that specific process)
What are simulations? programmed models of a computer system with variable clocks
What are simulations used for? used to gather statistics indicating algorithm performance; more accurate than queueing models; disadvantage - high cost and high risk
Given n processes to be scheduled on one processor, how many different schedules are possible? Explain the difference between preemptive and nonpreemptive scheduling: preemptive scheduling- allows process to be interrupted in the middle of execution, taking the CPU away and allocating it to another process; nonpreemptive scheduling- ensures process relinquishes control of CPU only when it finishes its current CPU burst
What advantage is there in having different time-quantum sizes at different levels of multi-threaded queueing system? Processes that need more frequent servicing can be in a queue with a small time quantum; processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing thus more efficient. What is the relationship between priority and SJF algorithms? Shortest job has highest priority.
What is the relationship between multi-level feedbacks and FCFS algorithms? The lowest level of MLFQ is FCFS.
What is the relationship between priority and FCFS algorithms? FCFS gives the highest priority to the job having been in existence the longest.
What is the relationship between RR and SJF algorithms? None. Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past.
Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs? it will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.
Distinguish between PCS and SCS scheduling? PCS is done local to the process - it is how the thread library schedules threads onto available LWPs; SCS is the situation where the operating system schedules kernel threads; on systems using many-to-one or many-to-many, the scheduling models are different; on one-to-one systems, PCS and SCS are the same
Define CPU scheduling: the act of determining which process in the ready state should be moved to the running state
Only one process can be in the running state, true or false? True; in single processor systems many processes can be in the ready state but only one can be in the running state
Define nonpreemptive scheduling: the currently executing process gives up the CPU voluntarily
Define preemptive scheduling: the operating system decides to favor another process, by preempting the currently executing process.
Define turnaround time: the amount of time between when a process arrives in the ready state for the first time and when it exists the running state for the last time
Define waiting time: the amount of time a process has been waiting in the ready state.
Define FCFS: first-come, first-served; processes are moved to the CPU in the order in which they arrive in the ready state
Define service time (aka burst time): total amount of time units a process needs on the CPU
What is the difference between waiting time and turnaround time? waiting time is the average time each process is in the waiting state (waiting time for each process / # processes); while turnaround time is the average time it takes for each process to complete: (waiting time for process + running service time)/# process
Define SJN or SJF: shortest job next or shortest job first; process with shortest estimated running time in the ready state is moved into the running state first
What a running process has to wait, what happens? the OS takes the CPU away from that process and gives the CPU to another process in memory