5: CPU Scheduling

Last Modified:
6/2/2004 3:01:20 PM

-1

Scheduling Policy

- We've talked about the context switch mechanism
  - How we change which process or thread is executing on the CPU
- Today, we will talk about scheduling policies
  - How do we choose which process or thread to execute next
  - Unit of scheduling = process or thread

-2

Scheduler

- Scheduler = the module that moves jobs from queue to queue
- Scheduler typically runs when:
  - A timer interrupt occurs
  - A process/thread blocks on a request (transitions from running to waiting)
  - A new process/thread is created or is terminated

-3

Scheduling Algorithm

- The scheduling algorithm examines the set of candidate processes/threads and chooses one to execute
- Scheduling algorithms can have different goals
  - Maximize CPU utilization
  - Maximize throughput (#jobs/time)
  - Minimize average turnaround time (Avg(EndTime – StartTime))
  - Minimize response time
- Recall: Batch systems have which goal? Interative systems have which goal?

-4

Starvation

- Starvation = process is prevented from making progress towards completion because another process has a resource that it needs
- Scheduling policies should try to prevent starvation
  - E.g. Even low priority processes should eventually get some time on the CPU

-5

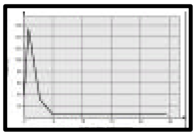
Brainstorm

- What are some different ways to schedule access to a resource?
  - First Come First Serve
    - Many services humans use are like this?
  - Prefer Short Jobs
    - Express lane at the grocery store
  - Important Jobs First
    - Order you do your TODO list? Maybe round robin?
- Now what about scheduling processes?

-6

Process Model

- Think of a process/ thread as an entity that alternates between two states: using the CPU and waiting for I/O (not a bad model)
- Most “CPU bursts” are short



First Come First Serve (FCFS)

- Also called First In First Out (FIFO)
- Jobs scheduled in the order they arrive
- When used, tends to be non-preemptive
  - If you get there first, you get all the resource until you are done
  - “Done” can mean end of CPU burst or completion of job
- Sounds fair
  - All jobs treated equally
  - No starvation (except for infinite loops that prevent completion of a job)

Problems with FCFS/FIFO

- Can lead to poor overlap of I/O and CPU
  - If let first in line run till they are done or block for I/O then can get convoy effect
  - While job with long CPU burst executes, other jobs complete their I/O and the I/O devices sit idle even though they are the “bottleneck” resource and should be kept as busy as possible
- Also, small jobs wait behind long running jobs (even grocery stores know that)
  - Results in high average turn-around time

Shortest Job First (SJF)

- So if we don’t want short running jobs waiting behind long running jobs, why don’t we let the job with the shortest CPU burst go next
  - Can prove that this results in the minimum (optimal) average waiting time
- Can be preemptive or non-preemptive
  - Preemptive one called shortest-remaining-time first

Problems with SJF

- First, how do you know which job will have the shortest CPU burst or shortest running time?
  - Can guess based on history but not guaranteed
- Bigger problem is that it can lead to starvation for long-running jobs
  - If you never got to the head of the grocery queue because someone with a few items was always cutting in front of you

Most Important Job First

- Priority scheduling
  - Assign priorities to jobs and run the job with the highest priority next
  - Can be preemptive such that as soon as high priority job arrives it get the CPU
- Can implement with multiple “priority queues” instead of single ready queue
  - Run all jobs on highest priority queue first

Problems with Priority Scheduling

- First, how do we decide on priorities?
  - SJF is basically priority scheduling where priority determined by running time – also a million other choices
- Like SJF, all priority scheduling can lead to starvation
- How do we schedule CPU between processes with the same priority?
- What if highest priority process needs resource held by lowest priority process?

-13

Priority Inversion

- Problem: Lowest priority process holds a lock that highest priority process needs. Medium priority processes run and low priority process never gets a chance to release lock.
- Solution: Low priority process "inherits" priority of the highest priority process until it releases the lock and then reverts to original priority.

-14

Dealing with Starvation

- FCFS has some serious drawbacks and we really do like to be able to express priorities
- What can we do to prevent starvation?
  - Increase priority the longer a job waits
  - Eventually any job will accumulate enough "waiting points" to be scheduled

-15

Interactive Systems?

- Do any of these sound like a good choice for an interactive system?
- How did we describe scheduling on interactive systems?
  - Time slices
  - Each job given a its share of the CPU in turn
  - Called Round Robin (RR) scheduling
- No starvation!

-16

Problems With RR

- First, how do you choose the time quantum?
  - If too small, then spend all your time context switching and very little time making progress
  - If too large, then it will be a while between the times a given job is scheduled leading to poor response time
  - RR with large time slice => FIFO
- No way to express priorities of jobs
  - Aren't there some jobs that should get a longer time slice?

-17

Best of All Worlds?

- Most real life scheduling algorithms combine elements of several of these basic schemes
- Examples:
  - Have multiple queues
  - Use different algorithms within different queues
  - Use different algorithm between queues
  - Have algorithms for moving jobs from one queue to another
  - Have different time slices for each queue
  - Where do new jobs enter the system

-18

Multi-level Feedback Queues (MLFQ)

- Multiple queues representing different types of jobs
  - Example: I/O bound, CPU bound
  - Queues have different priorities
- Jobs can move between queues based on execution history
- If any job can be guaranteed to eventually reach the top priority queue given enough waiting time, them MLFQ is starvation free

-19

Typical UNIX Scheduler

- Processes with highest priority always run first; Processes of same priority scheduled with Round Robin
- Reward interactive behavior by increasing priority if process blocks before end of time slice granted
- Punish CPU hogs by decreasing priority of processes that use the entire quantum

-20

priocntl

```
> priocntl -l
CONFIGURED CLASSES
==================

SYS (System Class)

TS (Time Sharing)
        Configured TS User Priority Range: -60 through 60

IA (Interactive)
        Configured IA User Priority Range: -60 through 60

RT (Real Time)
        Maximum Configured RT Priority: 59
```

-21

priocntl

```
:~> ps
   PID TTY      TIME CMD
 29373 pts/60   0:00 tcsh
 29437 pts/60   0:11 pine
:~> priocntl -d 29373
TIME SHARING PROCESSES:
    PID    TSUPRILIM    TSUPRI
  29373      -30         -30
:~> priocntl -d 29437
TIME SHARING PROCESSES:
    PID    TSUPRILIM    TSUPRI
  29437      -57         -57
:~> priocntl -d 1
TIME SHARING PROCESSES:
    PID    TSUPRILIM    TSUPRI
      1        0           0
```

-22

nice

- Users can lower the priority of their process with nice
- Root user can raise or lower the priority of processes

-23

Some Special Cases

-24

Real Time Scheduling

- Real time processes have timing constraints
  - Expressed as deadlines or rate requirements
- Common Real Time Scheduling Algorithms
  - Rate Monotonic
    - Priority = 1/ RequiredRate
    - Things that need to be scheduled more often have highest priority
  - Earliest Deadline First
    - Schedule the job with the earliest deadline
    - Scheduling homework? ☺
- To provide service guarantees, neither algorithm is sufficient
  - Need admission control so that system can refuse to accept a job if it cannot honor its constraints

-25

Multiprocessor Scheduling

- Can either schedule each processor separately or together
  - One line all feeding multiple tellers or one line for each teller
- Some issues
  - Want to schedule the same process again on the same processor (processor affinity)
    - Why? Caches
  - Want to schedule cooperating processes/threads together (gang scheduling)
    - Why? Don't block when need to communicate with each other

-26

Algorithm Evaluation: Deterministic Modeling

- Deterministic Modeling
  - Specifies algorithm *and* workload
- Example :
  - Process 1 arrives at time 1 and has a running time of 10 and a priority of 2
  - Process 2 arrives at time 5, has a running time of 2 and a priority of 1
  - …
  - What is the average waiting time if we use preemptive priority scheduling with FIFO among processes of the same priority?

-27

Algorithm Evaluation: Queueing Models

- Distribution of CPU and I/O bursts, arrival times, service times are all modeled as a probability distribution
- Mathematical analysis of these systems
- To make analysis tractable, model as well behaved but unrealistic distributions

-28

Algorithm Evaluation: Simulation

- Implement a scheduler as a user process
- Drive scheduler with a workload that is either
  - randomly chosen according to some distribution
  - measured on a real system and replayed
- Simulations can be just as complex as actual implementations
  - At some level of effort, should just implement in real system and test with "real" workloads
  - What is your benchmark/ common case?

-29

One last point: Kernel vs User Level Threads

- Recall: With kernel level threads, kernel chooses among all possible threads to schedule; with user level threads, kernel schedules the process and the user level thread package schedule the threads
- User-level threads have benefit of fast context switch at user level
- Kernel-level threads have benefit of global knowledge of scheduling choices and has more flexibility in assigning priorities to individual threads

-30

Outakes

- Windows 2000 priority classes
- Linux source code: kernel/sched.c
  - How to find
  - How to read online

-31