CPU Scheduling

Tom Kelliher, CS 311

Mar. 2, 2012

Announcements:

From last time:

1. Processes and threads; context switching.

Outline:

1. Traditional process scheduling.
2. Comparison criteria.
3. Priority functions.
4. Thread scheduling.
5. Multiprocessor scheduling issues.
6. Gantt chart examples.

Assignment:

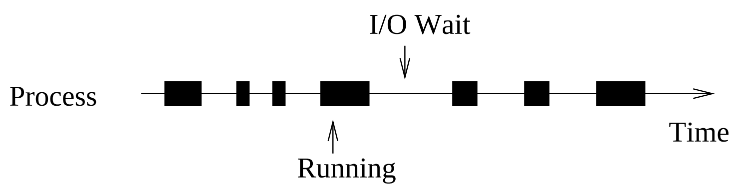
1 Traditional Process Scheduling

Motivation:

- Traditionally: keep the CPU busy.
- Now:
  - Promote modular design.
  - Increase throughput (a multi-threaded server).

What are the three schedulers, and how do they function?

Model of a process: CPU-I/O burst cycles:



Distribution of bursts.

CPU bursts terminate due to:

1. Process waits on event (blocked, suspended).
2. Process' quantum expires (back to ready Q).

1.1 Preemptive Scheduling

1. Non-Preemptive scheduling.

   Context switches occur:

   (a) Running process terminates.

   (b) Running process blocks.

I.e., running process controls the show.

New process takes over if running process blocks.

2. Preemptive scheduling.

   Principle: Highest priority ready process runs.

   Quantum timers come into play.

   Additional context switches:

   (a) Higher priority process changes state from blocked to ready, preempting running process.

   (b) Quantum expires (kernel preempts).

   Higher overhead.

3. Selective preemptive scheduling.

1.2 Comparison Criteria

User oriented, performance related criteria.

- Response time — time from submission of request to receipt of first output.
- Turnaround time — time from submission of request to its completion.
- Waiting time — turnaround time minus CPU time; time spent waiting for resources.
- Deadlines — When deadlines are specified, percentage of deadlines which are met.

User oriented, other criteria

- Predictability — the same job should run in about the same amount of time and cost regardless of other system activity.

System oriented, performance related criteria

- Throughput — number of processes completed per unit of time.
- CPU utilization — percentage of time CPU is performing *actual* work.

System oriented, other criteria

- Fairness — processes should generally be treated the same, with no process starving.
- Priority enforcement — when priorities are assigned, they should be adhered to.
- Balancing Resources — keep all system resources busy, adjust priorities accordingly. This can come in at the long-term scheduler level.

1.3 The Priority Function

- CPU time — usually the most important factor
- memory requirements — a major criterion in batch systems; in timesharing systems give a good measure of swapping overhead
- wall time — important for process "aging" — increase priority of older jobs.
- total required CPU time — specify max runtime for job (batch) or take some average of previous runtimes.
- external priorities — batch, interactive, realtime, VIP, etc. Let user pick priority and charge for higher priorities.
- system load — increasing quantum may offset swapping overhead, increasing utilization. Continue giving good response to high priority jobs, making others suffer. Graceful degradation.

1.4 Examples of Priority Functions

Implemented using queues or priority queues.

1. FCFS, FIFO — non-preemptive. Run oldest process. Standard batch priority function

    - Implemented with a simple queue for the ready Q

    - New jobs, jobs previously in wait or running state put at end of ready Q

    - Next job to run taken from head of ready Q

    - Priority function: time in ready Q

2. LIFO — non-preemptive. Run newest process. Not real useful.

3. SJF — shortest job first. Non-preemptive. Run process with shortest required CPU time.

    - Time is time of next CPU burst.

    - Implement with priority Q

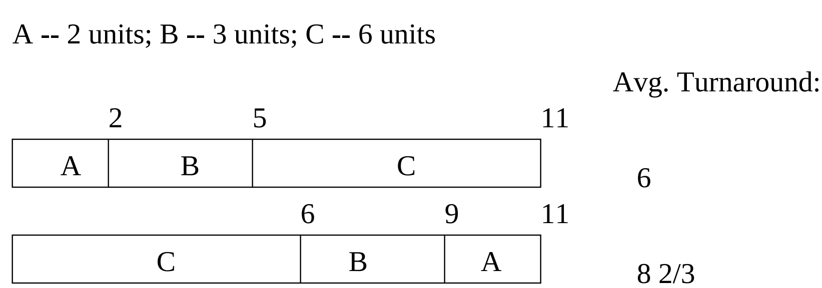
    - Estimate of next CPU burst:

      $$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n,$$

      where τ is the estimated time and t is the actual time. τ_n is an exponential average of history.

    - Priority function: $\frac{1}{\text{Next CPU Burst}}$

   Provably optimal from turnaround/waiting point of view:



4. SRT — (shortest remaining time) preemptive version of SJF.

- Another possibility — Time is remaining CPU time:

```
Current->RemainingTime -= LastCPUBurst;
```

- Total estimated CPU time is submitted with job. If exceeded, job is terminated.

5. RR — (round robin) preemptive FCFS with a time quantum limitation. Used in time sharing systems.

   - Uses FCFS's priority function

   - Additional factor in decision epoch: expiration of quantum timer

6. Multi-level queues — prioritized set of queues, Q_1 to Q_n.

   (a) Processes in queue i *always* have priority over queues $> i$.

   (b) A process remains within the same queue.

   (c) Each queue may have its own scheduling algorithm.

   (d) Alternative: each queue gets some fixed slice of the total CPU cycles.

   (e) Example: Queue for interactive jobs, RR scheduling; queue for batch jobs, FCFS.

7. Multi-level feedback queues — similar to multi-level queues, except that a process can move between different queues, based upon CPU usage.

   (a) Must specify rules for moving the processes between queues.

   (b) Ordinarily, lower priority queues have greater quantums, etc.

   (c) Linux uses this method, with a 100ms quantum for *all* queues. 141 priorities and run queues. A limited amount of dynamicism for non-realtime tasks. Higher priority tasks have longer quanta, but get "expired," preventing starvation.

1.5 Scheduling Examples

Suppose the following jobs arrive for processing at the times indicated and run with the specified CPU bursts (at the end of a burst a process waits for one time unit on a resource). Assume that a just-created job enters the ready queue after any job entering the ready queue from the wait queue.

Job	**Arrival Time**	**CPU Bursts**
1	0	1 2
2	1	1 3
3	2	1 1

Calculate the average turnaround time for each of the scheduling disciplines listed:

1. First Come First Served.
2. Shortest Remaining Time (assume that the running time is the sum of the CPU bursts).
3. Round robin with a quantum of 1.

Don't forget the "bubble" cycles (where no process is runnable), if required.

2 Thread Scheduling

Kernel-level (system scope) vs. user-level (process scope) threads.

pthread possibilities (implementation dependent):

1. Quantum allocation.
2. Process scope thread priorities; starvation.
3. Process scope threads with same priority: FIFO (no preemption) or RR (preemption) algorithms available.

3 Multiprocessor Scheduling Issues

1. Symmetric Multiprocessing vs. asymmetric multiprocessing: 1 or n run queues.

2. Processor affinity: maximize cache hit rates vs. load balancing vs. specialized devices attached to a single CPU.

3. Hyperthreading to reduce memory stall-forced CPU idling.

4. Virtualization: When a process quantum on a guest OS isn't all it's cracked up to be.