Scheduling Policies

Types of processes:

- interactive jobs
- low priority, cpu bound jobs that use excess processor capacity (e.g., calculating π to $10^{1000000}$ decimal places)
- somewhere in between

We will concentrate on scheduling at the level of selecting among a set of ready processes.

Scheduler is invoked whenever the operating system must select a user-level process to execute:

- after process creation/termination
- a process blocks on I/O
- I/O interrupt occurs
- clock interrupt occurs (if preemptive)

Distinguish between a *short* and *long* process. Based on the time a process runs when it gets the CPU. An I/O bound process is short and a CPU bound process is long.

Note; The idea of short vs. long is determined by how much of its time slice that a process uses, not the total amount of time it executes.

Criteria

Criteria for a good scheduling algorithm:

- fairness: all processes get fair share of the CPU
- efficiency: keep CPU busy 100% of time
- response time: minimize response time
- turnaround: minimize the time batch users must wait for output
- throughput: maximize number of jobs per hour

They are competing. Fairness/efficiency, interactive/batch

Look at Figure 2-38 for goals/criteria under different situations.

Measurements

In order to compare different short-term policies, we need a measure of performance. Assume that a process needs t time in execution before it leaves the ready list:

execution time (t) — execution time

response time (T) — finish time - arrival time. (wall clock time)

missed time (M) — T - t; time spend on the ready list or in blocked state.

penalty ratio (P) — T/t; penalty of 1 ideal (lower penalty is good)

response ratio (R) — t/T; response of 1 ideal (higher response is good)

Other useful measures:

- *kernel time* — amount of time the spent by the kernel in making policy decisions and carrying them out. Context switching. A well tuned O.S. uses between 10-30%.
- *system time* — kernel time devoted to a process.
- *Idle time* — amount of time spend when the ready list is empty. Thus running a NULL process or running NULL routine code.

First Come, First served (FCFS)

Also called FIFO. CPU services processes in the order they arrive. Processes run until they give up the CPU. If the policy is *non-preemptive*, a process is never blocked until it *voluntarily* gives up the CPU.

Non-preemptive policies resist change.

Advantages:

- easy to implement
- efficient—minimize context switching

Disadvantages:

- Penalty ratio for short jobs much greater than for long jobs
- favors computation intensive tasks over interactive tasks
- no way to break out of an infinite loop

Round Robin (RR)

CPU services a process for only a single *quantum* of time, q, before moving on to the next process. A quantum is usually 1/60 to 1 second.

The quantum q acts as a parameter that can be tuned. Long quantums result in FCFS. Short quantums result in an approximation of *processor sharing*, in which every process thinks it is getting constant service from a processor that is slower proportionally to the number of processes.

RR achieves a good penalty ratio by preempting processes that are monopolizing the CPU.

Is there a limit as to how small can we make the quantum value?

Note: we always start the job with a full time quantum, regardless of how long it took of the last quantum.

Shortest Process Next (SPN)

Also called Shortest Job First (SPF).

Preemption is *relatively* expensive. SPN attempts to avoid that cost.

The SPN policy always tries to select the shortest job for servicing.

Of course, we can't know absolutely how long a job will take! However, its recent behavior is likely to be a good approximation. For instance, one might compute an *exponential average*:

$$E_{smooth} = \alpha E_{smooth} + (1 - \alpha) E_{measured}$$

$E_{measured}$ is how long a process runs when it gets a chance.

α is called the smoothing (aging) factor and may be set higher or lower to make the estimate less or more responsive to change.

- when α is 0, we use only the recent value
- when α is .8, we treat the new value suspiciously — it may be only a temporary fluctuation.

Unfortunately, SPN may lead to *starvation*, a condition where some processes are never serviced.

In Unix, the command *uptime* uses an exponential decay to compute the average number of jobs in the run queue over the last 1, 5, and 15 minutes. For example:

```
% uptime
  1:02pm  up 18 days,  2:02,  4 users,
      load average: 0.48, 0.26, 0.03
%
```

Preemptive Shortest Process Next (PSPN)

Combine preemption of round robin (RR) with shortest process next (SPN).

PSPN preempts the current process when another process is available with a total service requirement less than the remaining service time required by the current process.

Multiple-level Feedback (FB)

Maintain multiple queues, with lower numbered queues having highest priority. When a process has used a certain quanta in its queue, the scheduler moves it to another queue.

Use Round Robin policy within each queue to select a process of the highest priority.

Look at Fig. 2-42.

Interactive processes have priority over longer ones, because long process migrate to lower priority queues. Give larger quanta to lower-priority.

Side note: CTSS system 1962, Policy that switched from CPU to interactive bound. Users found that it helped priority for long jobs to give keystrokes! Primitive way to detect I/O bound jobs.

Other Policies

- Lottery Scheduling—chance of a ready process being scheduled is in proportion to the number of lottery tickets held. Can allocate resources more accurately. Cooperating processes can pool tickets.
- Fair-Share Scheduling amongst users rather than processes. Similar issue for all threads of a process.
- Real-Time Scheduling—deadlines hard and soft.

Linux Process Scheduling

Two classes of processes:

- real-time (soft deadlines)
- normal

Always run real-time processes above normal. It uses priorities with either a FIFO or round-robin policy.

Normal process scheduling uses a prioritized, preemptive, credit-based policy:

- Scheduler always chooses process with the most credits to run.
- On each timer interrupt one credit is deducted until zero is reached at which time the process is preempted.
- If no runnable processes have any credits then for *every* process recredit as `credits = credits/2 + priority`.
- This approach favors I/O bound processes which do not use up their credits when they run.

Windows NT/2000/XP Scheduling

Windows NT/2000/XP uses *threads* as the basic scheduling unit. Threads have priorities and can be preempted (not always true for threads).

Evaluating Policies

Mathematical Analysis involves a mathematical formulation of the policy and a derivation of its behavior.

Queueing networks model the system as a set of queues to servers (e.g. CPU, I/O devices, etc.).

1. each server has an average *service time*
2. processes move from one queue to another according to probabilities that describe the breakdown of time spent at each server

The primary drawback of mathematical analysis is that the model is only an approximation to the actual system, and complex systems are often impossible to solve.

Simulation involves tracking a large number of processes through a model and collecting statistics.

Process execution can be driven by probabilities (as in mathematical analysis) or by actual trace data.

Experimentation is the "best" method, because it measures the real system. Unfortunately, because the system tested must be built, experimentation is usually expensive.

Guidelines

- Rule of thumb: preemption is worth extra switching costs. Clock devices are pretty much universal—not much more time to make a decision on an interrupt.
- Use large enough quantum to keep kernel time down. Example: process switch 500 us, quantum 10 ms, 5% scheduling cost. Want quantum small enough so the "interactive" does not "see" time slicing.
- ease of implementation—FCFS, RR easy
- must also consider space used by process—more space bigger quantum
- interactive—give priority. UNIX gives priority to processes with a high idle time/CPU ratio.
- want quantum as big as possible while still not causing noticable response problems.