Scheduling

Sanzheng Qiao

Department of Computing and Software

January, 2013

Introduction

In this part, we'll talk about *resources*, the things operated upon by processes (CPU time, disk space, etc).

Resources fall into two classes:

- Preemptive: Can be taken away, use it for something else, then give it back later (eg. CPU).
- Non-preemptive: Once given, it can't (or it is difficult to) be reused until process gives it back.

Why scheduling?

Limited resources, multiprogramming

OS makes two related kinds of decisions about resources:

- Allocation: who get what. Given a set of requests for resources, which process should be given which resources in order to make most efficient use of resources? Implication is that resources are not easily preemptible.
- Scheduling: how long can they keep it? When more resources are requested than can be granted immediately, in which order should they be served? (CPU scheduling) Implication is that resource is preemptible.

CPU scheduling

Allocates cpu among the processes in the ready state.

Goals: Maximize the resource (CPU) utilization and throughput; minimize overhead (context switches).

An observation: a typical execution of a program consists of cpu bursts and I/O bursts. Usually, there are many short cpu bursts.

Process execution begins with a CPU burst (start up) followed by an I/O burst and then a CPU burst and so on. Finally, it ends with a CPU burst (finish).

Statistics

Response time Time spent in the ready queue waiting for the first chance to execute.

Waiting time Total time spent in the ready state.

Running time Total time spent in the running state.

Turnaround time The sum of the waiting time and running time.

Algorithms

Assumptions: There are n ready processes. (Steady state)
Average execution time (CPU burst) is t.

Non-preemptive algorithm FCFS

response time: nt
wait time: nt
fast, little overhead
This is short processes' disadvantage.

Algorithms

Either preemptive or non-preemptive

- Shortest-job first (SJF)
  This algorithm minimizes average wait time.

$$\text{average wait time} = \frac{(n-1)t_1 + \cdots + t_{n-1}}{n}$$

  This is long processes' disadvantage.
- Priority
  The SJF is a special case of priority scheduling where the next CPU burst is the priority (assuming low numbers represent high priority).

Synchronization and scheduling

Priority inversion problem

A high priority thread is waiting for a low priority thread, e.g., waiting in `Join()`, and middle priority threads are on the ready queue. Then the high priority thread cannot run before the middle priority threads, because the low priority thread cannot run until the middle priority threads finish.

A partial solution

Have the waiting thread “donate” its priority to the low priority thread while it is holding the resource (lock, semaphore).

Preemptive algorithms

time quantum: q

(a) Round-Robin
response time: nq
wait time: $\approx nt$
More overhead, long processes have to go through the ready queue several times.
The choice of q: 80% of cpu bursts $t \leq q$ (10000-100000 instr's).

(b) Modified Round-Robin
The time quantum is increased (e.g., doubled) each time a process reenters the ready queue.
Shorter waiting time for long processes, longer response time for short processes.

Preemptive algorithms

(c) Multilevel priority queues
The ready queue is divided into several queues with different priorities. A process is permanently assigned to one queue. Sacrifice overall system efficiency to give us better performance with respect to some other parameters, say, safety.

(d) Multilevel feed back queues (adaptive)
The ready queue is divided into several queues with different time quanta, scheduling policies. Processes move between queues. When a process is submitted, it is put on q_1. If it does not finish in time quantum, it is moved to q_2, ...

Example: SUN

A fair-share scheduling.

- Keep history of recent CPU usage for each process. Forget 90% of recent CPU time in $5n$ seconds, where n is the average number of ready processes in the last minute. The decaying rate r:

$$0.1 = r^{5n}.$$

- Adjust the base priority (-20–$+20$, high–low). In t seconds,

$$\text{adj} = \text{recent CPU usage} \times r^t.$$

Example: SUN

Implications:

- In a heavy load (n large), the recent CPU usage is forgotten slowly.
- A CPU intensive process (recent CPU usage large) has low priority. This policy favors I/O bound processes.
- The CPU usage is forgotten (adj $\rightarrow 0$) as time passes ($t \rightarrow \infty$).

Summary:

- The algorithms have strong effects on the system's overhead, efficiency, and response time.
- The best scheme is adaptive.