CS 350 Notes

Matthew Visser

Nov 8, 2011

1 Paging

1.1 Prefetching

- Prefetching means we move virtual pages before we need them
- Same or more amount of work, but decreases apparent latency

1.2 Page Size

- Large page sizes have less paging.
- Larger pages have fragmentation and an increased chance of paging memory you don't need.
- Super pages allow you to change page size at runtime.

1.3 How much memory does a process need?

Working set: W, Resident Set R.

$$W \subseteq R$$

The working set is the heavily used portion of the program's address space.

The program's resident set is the set of pages in memory.

We define $W(s, \Delta)$ to be the set of pages referenced by a process during a time interval $(t - \Delta, t)$. This is the working set of time t.

1.4 Thrashing and Load Control

We need a good level of the number of processes:

- Too low and we idle resources
- Too high and we have too few resources per process.
- A system spending too much time is said to be *thrashing.*
- Thrashing can be cured by shedding load:
  - killing processes
  - Suspending and swaping out processes.
- Performance drops *very* quickly when it starts thrashing.
- What processes do we suspend?
  - Low priority
  - blocked
  - Large (frees lots of resources)

2 Copy-On-Write

The reason to have this is that often, you want to duplicate an address space, *e.g.* `fork()`.

The way it works is it doesn't copy pages at first, but when one page tries to write, then it copies.

This is useful for filesystem snapshots as well.

3 Program Execution

A running thread can be modeled as CPU bursts and IO bursts.

Threads are scheduled when:

- They yield
- they are pre-empted after their quantum

Properties we want:

- Enforce priority

- prevent starvation
- be fair
- minimize wait time
- maxmize throughput, better (minimal) turn-around time.
- Enforce CPU quota

There are two basic strategies:

- FCFS — a FIFO redy queue, non-preemptive
- Round Robin
- SHortest Job First
  - Non-preemptive
  - ready threads are scheduled according to the length of their next CPU burst.
  - Requires knowledge of burst length:

    $$B_{i+1} = \alpha b_i + (1 - \alpha) B_i$$

    where B_i is the predicted length of the CPU burst and b_i is the actual length. $0 \leq \alpha \leq 1$.