CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Effective Access Time

You should see many similarities between demand paging and caching (such as valid bits in page table)

Everything process might need is stored on relatively slow disk, but we keep most pertinent information in faster memory, going to disk only as a last resort

In this case, memory is a cache for the disk

Demand paging works for same reason caching works -- memory accesses display **locality**

Virtual memory has many benefits, but results in performance penalty due to page faults

Each page fault results in access to slow disk

Without virtual memory, page faults could not occur

If page faults are too common, system will slow down unacceptably

To determine whether virtual memory is viable, need to consider

- cost of page fault
- frequency of page faults

Page fault cost

OS must service interrupt

Involves passing control to OS, saving process state, running interrupt handler

OS makes I/O request and missing page is read from disk

Page table updated

Process state restored and control returned to process

Page fault cost

Main cost is I/O

According to textbook, 8 milliseconds is reasonable approximation

This allows us to determine performance penalty corresponding to given frequency of page faults

Determine performance as **effective access time** (EAT)

Effective access time is how long memory access will take on average (sound familiar from discussions of caches and TLB?)

Page fault rate -- proportion of memory accesses that result in page faults. Denoted p

(Analogous to "cache miss rate", *not* "cache hit rate")

Memory access time denoted `ma`

$$EAT = (1 - p)ma + p(\text{page fault time})$$

Already stated that page fault time is approximately 8 milliseconds

According to textbook, reasonable range for memory access time is 10 - 200 nanoseconds

Assume memory access time is 200 nanoseconds

EAT = (1 - p)*200* + *p*(8,000,000) nanoseconds

= 200 + 7,999,800 * p

What if p = 1/1000?

EAT = 200 + 7,999,800 * (1/1000)

= 8200 nanoseconds (approximately)

This is 40x slower than the 200 ns required to access memory directly!

What if we want < 10% slowdown? What is the maximum page fault rate we could allow?

$$EAT = 200 + 7{,}999{,}800 * p$$

We want $EAT < (1.1 * 200)$

Solving for p gives

$$p < 0.0000025$$

To keep performance penalty below 10%, page fault rate must be below 2.5 * 10^(-6)

Can afford to miss just 1/399,990 accesses

Virtual memory *is* used, so must be possible to keep miss rate extremely low

See similar behavior many places where caching-type behavior applies

Strategies for keeping page fault rate low will be discussed later