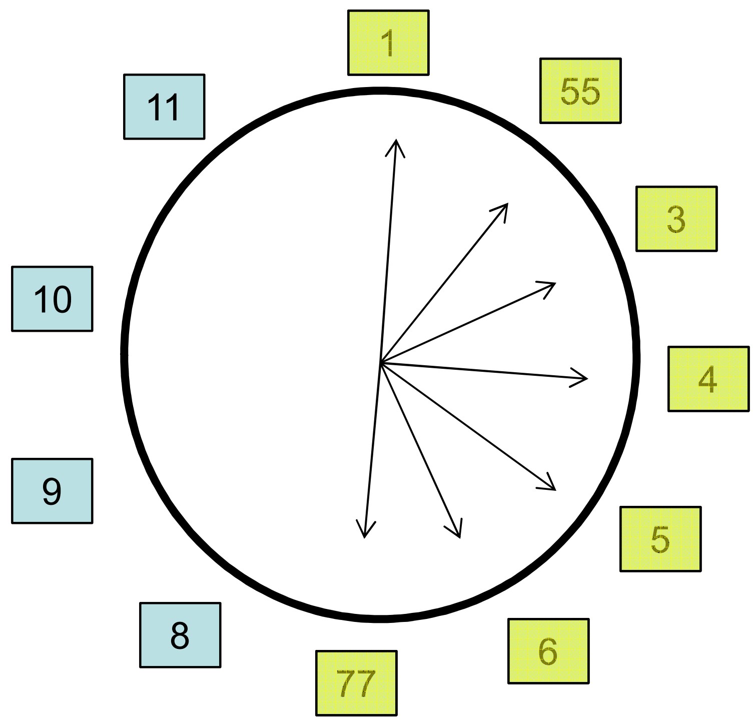
VM: The Final Frontier

- Topics
  - Clock
  - Thrashing
  - Working sets
- Learning Objectives:
  - Demonstrate how a clock algorithm works.
  - Define thrashing and explain how it can happen in a system (both with respect to paging and TLBs).
  - Explain working sets and how they help you manage memory.
  - Tackle Assignment 3.

Clock

Use bit = 0

Use bit = 1



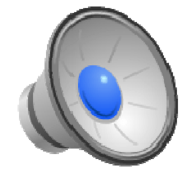
Page reference stream 1 3 1 1 6 55 4 5 77

Styles of Replacement

- Global replacement
  - All pages from all processes are grouped into a single replacement pool
  - Processes compete with each other for page frames.
- Per-process replacement
  - Each process has a separate pool of frames.
  - A page fault in one process can only evict one of its own frames.
  - No interference between processes.
- Per job replacement
  - Put all users' pages in a single pool.
  - Probably need a mechanism to move pages from one pool to another in this model (i.e., another user logs in).
- Global replacement provides most flexibility, but no pig protection.

Thrashing Returns

- Thrashing is when performance degrades, because two (or more) entities (in this case processes) are fighting over resources (e.g., memory).
- Suppose that there are many users, and that between them, their processes are making frequent references to **50** pages, but there are only **40** pages of memory.
  - Each time a page is brought in, another is evicted.
  - Assume that every memory reference takes 100 nanoseconds (.1 microseconds) and that a disk access takes 10 ms (10,000 microseconds).
  - What is the average memory access time (assuming 80% hit rate)?
  - What is the average memory access time (assuming all pages are misses, but you have 1000 accesses per page)?
- The system will spend all its time reading and writing pages and will get very little useful work done.
- We get the bad illusion: memory is as small as physical memory and as slow as disk!
- Thrashing was a severe problem in early demand paging systems.



Thrashing Returns

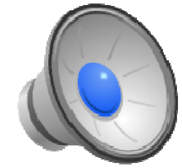
- Just like we could get TLB thrashing, we can get memory thrashing (only this is worse!)
- Suppose that there are many users, and that between them, their processes are making frequent references to **50** pages, but there are only **40** pages of memory.
  - Each time a page is brought in, another is evicted.
  - Assume that every memory reference takes 100 nanoseconds (.1 microseconds) and that a disk access takes 10 ms (10,000 microseconds).
  - What is the average memory access time (assuming 80% hit rate)?
    - 4 * .1 + 1 * 10000 = 10000.4 / 5 = 2000 microseconds.
    - That is 20,000 times slower than memory!
  - What is the average memory access time (assuming all pages are misses, but you have 1000 accesses per page)?
    - 999 * 0.1 + 1 * 10000 = 10099.9 microseconds for 1000 accesses = ~10.1 microseconds/access.
- The system will spend all its time reading and writing pages and will get very little useful work done.
- We get the bad illusion: memory is as small as physical memory and as slow as disk!
- Thrashing was a severe problem in early demand paging systems.

What Causes Thrashing?

- Two fundamentally different causes:
  1.
  2.

What Causes Thrashing?

- Two fundamentally different causes:
  1. A single process is too big.
  2. The sum of all processes it too big.
- What can you do?
  - One big process
    - Out of luck
    - That process is just going to thrash (buy more memory).
  - Combination of processes is too much
    - Figure out how much memory each process needs.
    - Change scheduling priorities to run processes in groups whose memory demands can be satisfied.
    - Diminish load on the OS (*load shedding*).



Working Sets

- Proposed by Peter Denning in 1960's when Multics exhibited thrashing.
- Informal definition: the collection of pages that a process is working with and that must therefore be resident if the process is to avoid thrashing.
- The idea is to use the recent memory needs of a process to predict its future needs.
- More formally:
  - Let τ (tau) be the working set parameter.
  - Let the working set of τ be all pages referenced by a process in its last τ seconds of execution.
  - A process will never be executed unless its working set is resident in main memory.
  - Pages outside the working set may be discarded at any time.
- Working sets are not quite enough to solve the problem…

Balance Sets

- If the sum of the working sets of all runnable processes is greater than the size of memory, refuse to run some processes (for awhile).
- Divide the runnable processes into two groups: active, inactive.
- When a process is made active, its working set is loaded.
- When it is made inactive, its working set is allowed to migrate to disk.
- The collection of active processes is called a balance set.
- Now, all you need is an algorithm for moving processes into and out of the balance set.
- What happens if the balance set changes too frequently?

- As working sets change, balance set changes too.
  - This has a problem that you need to constantly update the working set

Balance Sets

- If the sum of the working sets of all runnable processes is greater than the size of memory, refuse to run some processes (for awhile).
- Divide the runnable processes into two groups: active, inactive.
- When a process is made active, its working set is loaded.
- When it is made inactive, its working set is allowed to migrate to disk.
- The collection of active processes is called a *balance set*.
- Now, all you need is an algorithm for moving processes into and out of the balance set.
- What happens if the balance set changes too frequently?
  - You still get thrashing!
- As working sets change, balance set changes too.
  - This has a problem that you need to constantly update the working set

Working Set Theory

- The idea was that you stored some sort of capacitor with each memory page.
- When the page was referenced, the capacitor was charged.
- Then it would discharge slowly.
- τ would be determined by the size of the capacitor.
- In practice, you want separate working sets for each process, so capacitor should only discharge while process is running.
- Not clear what you do if a page is shared!

Working Sets in Reality

- Use use bits.
- OS maintains an idle time value for each page.
- This is the amount of CPU time received by the process since the last access to the page.
- Periodically, scan all the pages of a process.
- For each use bit that is set, set page's idle time to 0.
- If the use bit is clear, add the process' CPU time (since the last scan) to the idle time.
- Turn all bits off during scan.
- Scans happen on order of every few seconds (in UNIX, τ is on the order of a minute or more).

Parting Thoughts

- What should τ be?
- What happens if τ is too large?
- What happens if τ is too small?
- What algorithms should be used to determine which processes are in the balance set?
- How do we compute working sets if pages are shared?
- How much memory is needed in order to keep the CPU busy?
- In the working set model, the CPU may occasionally be idle even though there are runnable processes.
- Technology changes problems!
  - In a PC or even a VM, thrashing may be a less critical issue than in timesharing systems.
  - If one user has too many processes, she can just kill some!
  - With multiple users, the OS must somehow arbitrate fairly.