Eric Paik – Lecture 14 – 3-16-09 - cs162-an

Working Sets are a solution proposed by Peter Denning

- "the set of pages that a process is working with, and which must thus be resident if the process is to avoid thrashing"
- Idea is to use the recent needs of a process to predict its future needs.
- More formally "exactly that set of pages used in the preceding tau virtual time units" (usually given in units of memory references)
- Choose tau, the working set parameter. At any given time, all pages referenced by a process in its last tau seconds of execution are considered to comprise its working set.

Working Set page number keeps in memory exactly those pages used in preceding tau time units.

Process will never be executed unless its working set is in main memory.

Working set paging requires that the sum of the sizes of the working sets of the jobs eligible to run balance set be less then to equal the amount of space available.

- Algorithm needed for moving processes into and out of balance set.

Working set adjusts amount of space dynamically, LRV is static space.

How do we implement working sets?

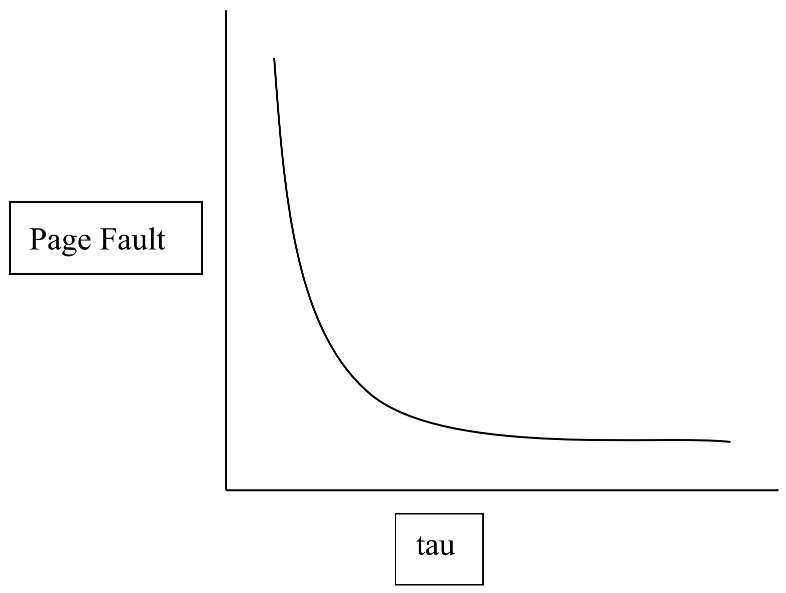
- First suggestion was to use capacitor on each memory page and have it get charged on each reference.

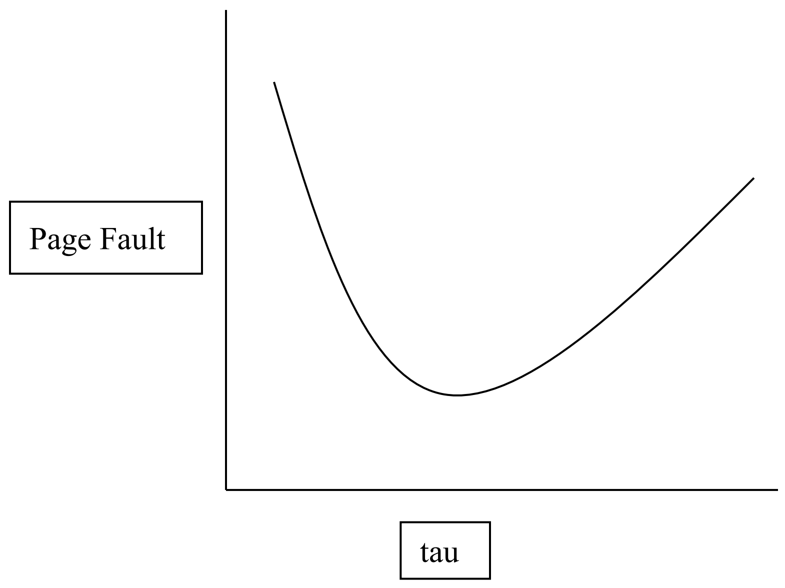
Actual Solution

- Use "used" bits
- Every once in awhile, san all pages of a process. For each "used" bit on, clear page's idle time.
- In practice, not implemented. Inefficient as hell. Used for simulation.

Questions about Working Sets/Memory

What should tau be?





Concept is widely used and understood even though not used.

Working Set Restoration – “swapping”

- Idea is when we remove a process from the in-memory queue, we know what its working set is.
- When we run the process again, we can restore the working set to memory all at once.
- Advantages
    - minimize CPU overhead
    - don't have to wait for each page faults, all transfers are at once

Approximate implementation is hard. Opderbach and Chu created Page Fault Frequency.

Page Fault Frequency: Let x be the virtual time since the last page fault for this process.

- At time of a page fault, if x>tau, remove all pages (of the process) with the bit off. Then get a page frame for the new new pages and turn off all reference bits for the process.
- Idea was to make this a quick and easy way to implement.
- Doesn't work as well as working set and is unstable.
- Problem is that process can fault frequently and still not need more pages.

What size should pages be?

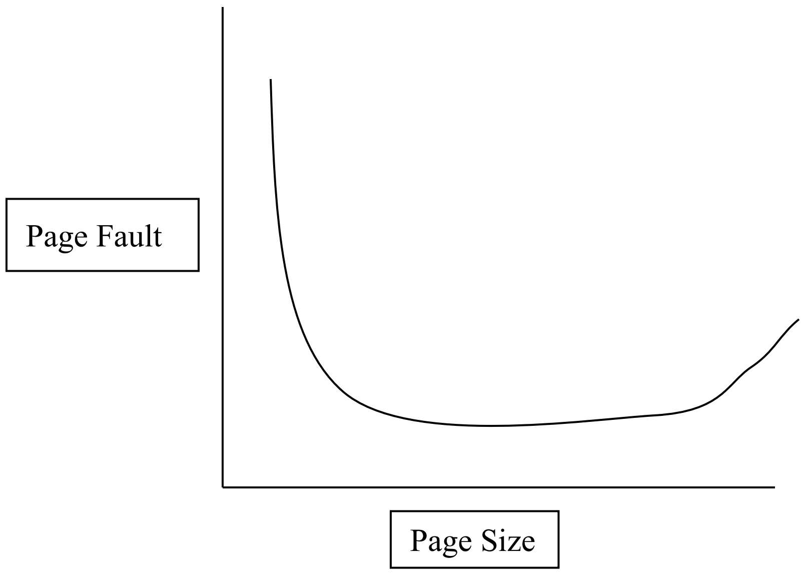
- Don't have choice, determined by hardware. Can simulate larger page size.

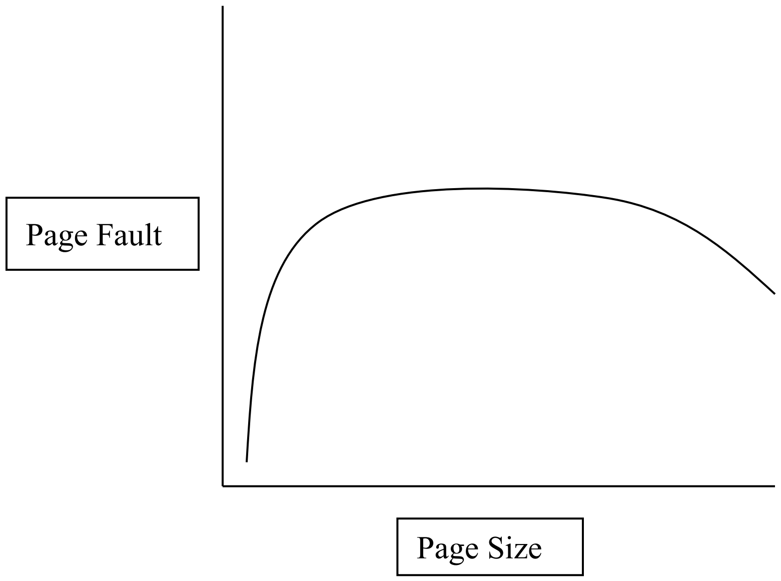
Larger Pages:

- Less overhead – same overhead per fault but fewer faults. More internal fragmentation.
- TLB covers more "address space" so fewer TLB misses.

Smaller page tables

- More total memory needed to cover same active localities.
- Greater delay for page fault.
- Less overhead for running replacement algorithm





Size of pages from 4-16k. Once larger then 16, transfer time starts slipping.

Can you page table the operating system?

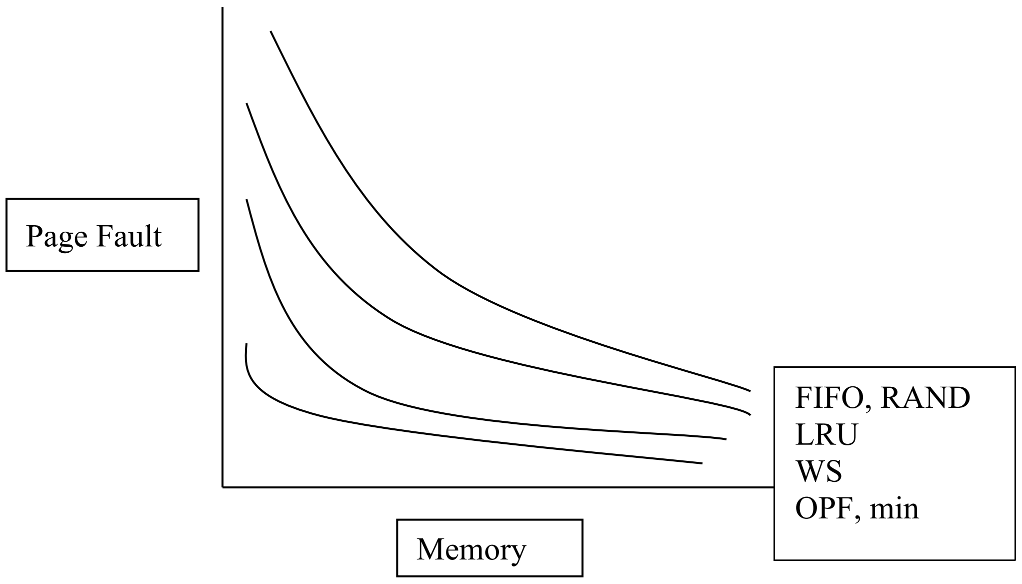
- Yes, traps, page handlers etc.
- Problem is you do this, breaks or hurts performance.

What happens if we're doing I/O while using virtual memory.

- I/O is always writing to random addresses.
- OS must translate virtual buffer address to real buffer address.
- We'd better make sure it doesn't get paged out.
- Usually done with lock bit. “Locks” page into memory.
- Must be careful with page boundaries since I/O is done with real addresses.
    - Break transfers into several pieces, each into a page.
    - Transfer is non-contiguous
    - Make sure T/O buffers are page aligned.
    - Put in contiguous area of real memory, managed by OS.

How do we study paging algorithm

- Could use math model, but no such acceptable model
- Could yse random number driven simulation, same problem as mathematical models.
- Could do experiments on real system, but difficult since time consuming. Need access to machine, may not be reproducible.
- Use trace driven simulation. Get a program address. Program address trace on the sequence of virtual addresses generated by the program.
- We get a program address storage, Write a machine interpreter.
- Hardware monitors
- Trace trap facility – trap on every instruction
- Use microcode modifications (popular).
    - Real instruction set and programmer instruction set.
- Instrument the object code or assembly code to write trace records either for every instruction, or for every load, store, and break. Get page fault on every reference and generate trace record. (popular but high overhead)



Minimize Page Faults?

- Write algorithms in efficient manner
    - Matrix transpose
- Program restructuring, organize pieces of a program with the pages, so that number of faults is minimized.
- Combine memory allocation with schedules to produce good results.

I/O

- Terminals – one character is sent at a time, one interrupt per character.
- 10-1900 characters per second.
- Keyboard and display independent in most systems
- Used to be handled with one interrupt per character but often DMA nowadays.