# Some Other Topics in Demand Paging

**COMP 321**

**Dave Johnson**

1

## Global vs. Local Replacement

***When selecting a victim to evict in handling a page fault***

- ***Global replacement***: Select the globally "best" victim
  - Even if that physical page is in use for a virtual page belonging to some other process
  - Number of physical pages each process has will dynamically grow and shrink
- ***Local replacement***: Select the "best" among pages in own address space
  - Evict one of your own virtual pages to make room for another virtual page you need now
  - Number of physical pages each process has remains constant from before to after any page fault

2

## Global vs. Local Replacement

***Example: CLOCK page replacement***

- ***Global replacement***: Select the globally "best" victim
  - A single global set of the algorithm's data structures
  - One global FIFO-ordered list of all physical pages, one global clock hand
- ***Local replacement***: Select the "best" among pages in own address space
  - A separate set of the algorithm's data structures for each process
  - In (or attached to) each PCB: FIFO-ordered list of just that process's physical pages, and a clock hand just for that list





## Global vs. Local Replacement

***So which is better, global replacement or local replacement?***

- ***Global replacement***: Select the globally "best" victim
  - Somewhat simpler to implement
  - Generally better overall utilization of memory
  - More common than local replacement
- ***Local replacement***: Select the "best" among pages in own address space
  - Arguably more "fair" (for at least one definition of "fair")
  - The behavior of one process can't cause another process to get more page faults

VAX example: VMS used local FIFO, Unix used global CLOCK

# The Working Set Model

***How many physical pages does a process "need" when executing?***

- Just 1 text page, 1 data page, 1 stack page?
- What about a loop spanning across a text page boundary?
- This number is important, such as if using local page replacement

***The "working set" for a process is those pages it is "actively" using***

- The definition of "actively" is related to "***recently*** used" (temporal locality means "likely to use again in the ***near*** future")
- Example: Using reference-bit-history, any page for which most significant *k* bits are all 0 could be considered no longer part of the working set

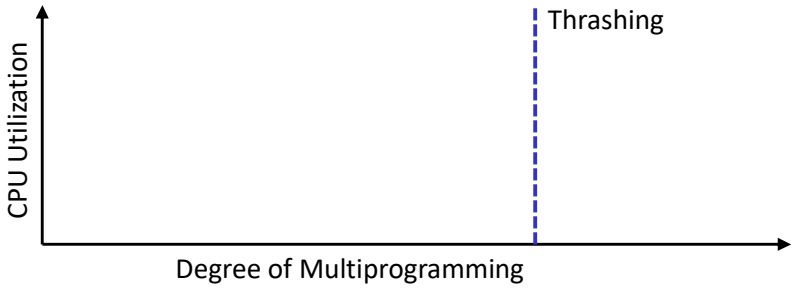
**0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 1 . . .**



# Effect of the Working Set

***Does a process have enough physical pages to hold its working set?***

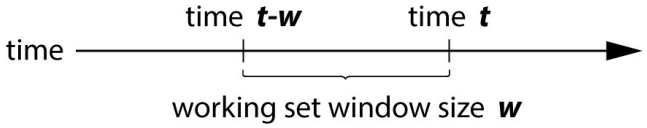
- If yes: Few page faults
- If no: Many page faults




***Thrashing***: spending more time page faulting than executing program

# Formal Definition of the Working Set

The ***working set*** of some process, written as ***WS(t, w)***




***WS(t, w)*** = the virtual pages referenced by the process over this time

How do we measure time?

- Elapsed real time, elapsed CPU time, number of instructions executed?

How big should ***w*** be?

- 1/100 second, 1/10 second, 1/2 second, … ?

***Principle of locality: any "reasonable" answers should make little difference***





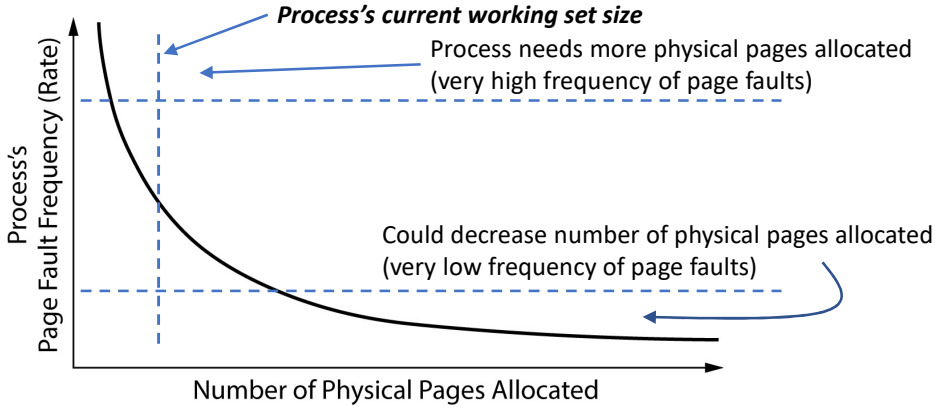
# Using Working Set to Control Multiprogramming

***OS can use working set sizes to control the degree of multiprogramming***

- Keep track of the ***size*** of every process's working set
- Compute the ***total size*** of ***all*** process working sets
- If the ***total*** **<** the size of all of physical memory
  - In a batch (non-interactive) system, can start a new job
- If the ***total*** **>** the size of all of physical memory
  - The degree of multiprogramming is too high (thrashing)
  - ***Entirely swap out*** some process for a short while
    - Reduces demand on physical memory so other processes run much more efficiently
    - Cycle among processes, swapping each out for a short while in turn

# The Page Fault Frequency Strategy

***An easier, more efficient way to accomplish the same thing***




***If all processes above top line (or none below bottom line), swap out a process***







# A Way to Find the PTE for Any Given PFN

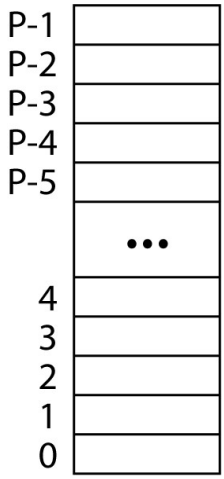
***Often need to be able to find the PTE for a given PFN***

- Example: CLOCK or reference-bit-history page replacement

***At boot time***

- The kernel creates an array of size P = # physical pages
- P is a constant while the computer is up and running

***P = # of physical pages***




Store address of PTE for each ***physical*** page *i* at index *i* in the array

Or store VPN and address of the page table or PCB for each ***physical*** page *i* at index *i* in the array

If multiple virtual pages mapped to some physical page, store address of beginning of a ***linked list*** of entries for each physical page *i* at index *i* in the array

## Background: Direct Memory Access (DMA)

***Making I/O faster (less CPU overhead)***

- Interrupts help reduce CPU overhead, but still one interrupt per character
- Example: read a sector from disk (4096 bytes = 4096 interrupts)

***Hardware support for Direct Memory Access (DMA) for I/O***

- OS gives to DMA controller hardware for the target I/O device
  - Memory address of a buffer (an area in memory)
  - Byte count size of that buffer
  - An operation to perform (e.g., READ or WRITE)
- DMA controller hardware does the entire transfer from/to the memory buffer
- ***Only one interrupt at completion of the entire command***



## Virtual Memory Mapping vs. DMA

***Normal DMA hardware requires a physically contiguous buffer***

- Buffer address, byte count, command (e.g., READ or WRITE)
- But buffer might be only ***virtually*** contiguous, not ***physically*** contiguous

***Two possible solutions***

- Only do DMA from ***kernel buffers*** known to be physically contiguous
- Or improvement to the DMA ***hardware: scatter/gather DMA***
  - Give hardware a ***list of multiple (buffer address, byte count) entries***, followed by the command
  - DMA "gathers" data on output, "scatters" data on input
  - Still a ***single I/O command*** and ***one interrupt for the whole buffer list***

# Demand Paging vs. DMA

***What if a page involved in DMA is selected as the victim on a page fault?***

- Output uses the wrong data from the wrong virtual page
- Input overwrites the wrong data in the wrong virtual page

***A simple solution***

- Keep a count, for each ***physical*** page, of the number of times that page is ***currently*** involved in DMA
- Skip pages with count > 0 in victim selection (page is "***pinned***" in physical memory)