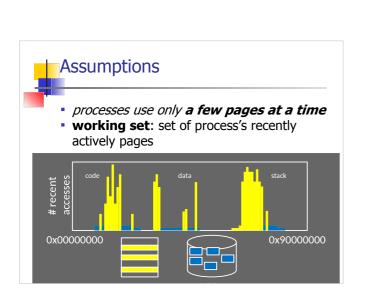


Virtual Memory That is Larger Than Physical Memory Page 0 page 1 page 2 Virtual memory map physical memory virtual memory



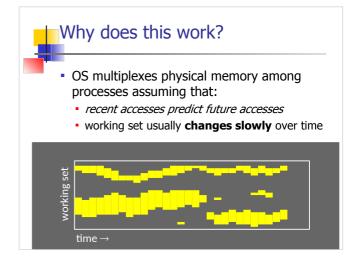
Bac

Background

- Virtual memory separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution.
 - Logical address space can therefore be much larger than physical address space.
 - Allows address spaces to be shared by several processes.
 - Allows for more efficient process creation.
- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation

Chapter 9: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples





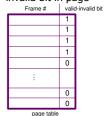
Demand Paging

- Bring a page into memory only when it is needed.
 - Less I/O needed
 - · Less memory needed
 - Faster response
 - More users
- Page is needed ⇒ reference to it
 - invalid reference ⇒ abort
 - not-in-memory ⇒ bring to memory



Valid-Invalid Bit

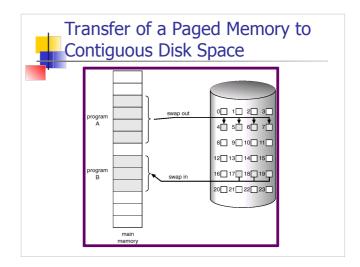
- With each page table entry a valid-invalid bit is associated
- $(1 \Rightarrow \text{in-memory}, 0 \Rightarrow \text{not-in-memory})$
- Initially valid—invalid but is set to 0 on all entries.
- Example of a page table snapshot.
 During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

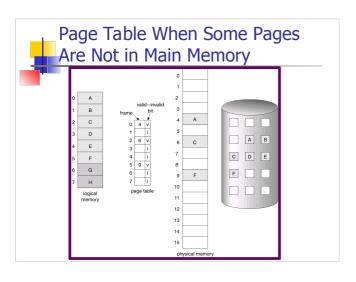


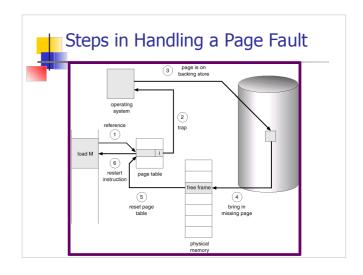


Page Fault

- If there is ever a reference to a page, first reference will trap to
 - OS ⇒ page fault
- OS looks at another table to decide:
 - Invalid reference ⇒ abort.
 - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- · Restart instruction: Least Recently Used
 - block move
 - auto increment/decrement location









What happens if there is no free frame?

- Page replacement find some page in memory, but not really in use, swap it out.
 - algorithm
 - performance want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.



Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = $(1 p) \times 200 + p$ (8 milliseconds) = $(1 - p \times 200 + p \times 8,000,000$ = $200 + p \times 7,999,800$
- If one access out of 1,000 causes a page fault, then
 - EAT = 8.2 microseconds.
 - This is a slowdown by a factor of 40!!



Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory.
 If either process modifies a shared page, only then is the page copied.
- COW allows more efficient process creation as only modified pages are copied.
- Free pages are allocated from a pool of zeroed-out pages.



Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
 - if p = 0 no page faults
 - if p = 1, every reference is a fault
- Effective Access Time (EAT)

 $EAT = (1 - p) \times memory access$

- + p (page fault overhead
- + [swap page out]
- + swap page in
- + restart overhead)



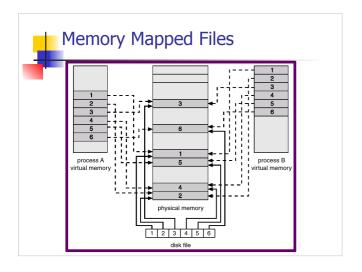
Process Creation

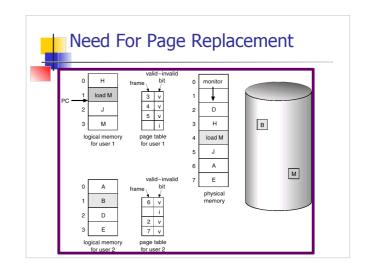
- Virtual memory allows other benefits during process creation:
 - · Copy-on-Write
 - Memory-Mapped Files

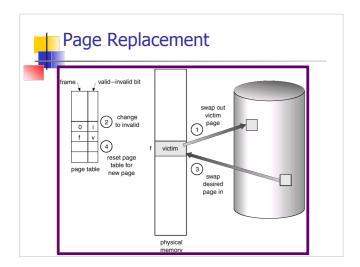


Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than read() write() system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared.









Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.



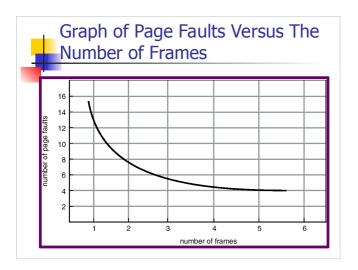
Basic Page Replacement

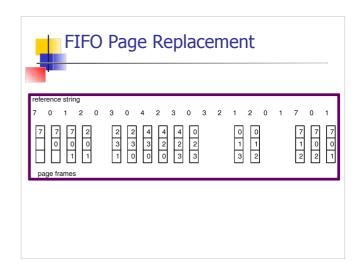
- 1. Find the location of the desired page on disk.
- 2. Find a free frame:
 - If there is a free frame, use it.
 - If there is no free frame, use a page replacement algorithm to select a *victim* frame.
- 3. Read the desired page into the (newly) free frame. Update the page and frame tables.
- 4. Restart the process.



Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is
 - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.







Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example

- How do you know this?
- Used for measuring how well your algorithm performs.

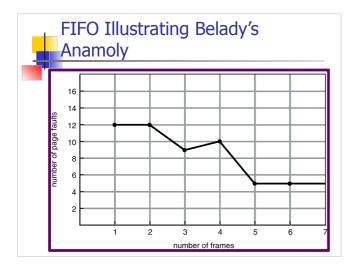
First-In-First-Out (FIFO) Algorithm

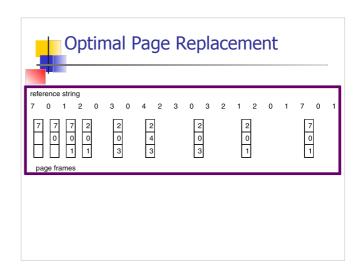
Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames (3 pages can be in memory at a time per process)
3 frames (3 pages can be in memory at a time per process)

• FIFO Replacement – Belady's 3 3 2 Anomaly 4 4 3

• more frames ⇒ less page faults







Least Recently Used (LRU) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Counter implementation
 - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
 - When a page needs to be changed, look at the counters to determine which are to change.



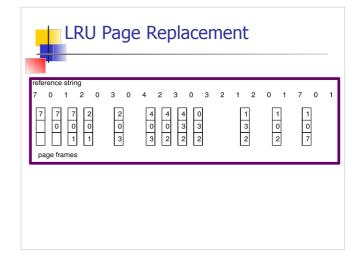
LRU Algorithm (Cont.)

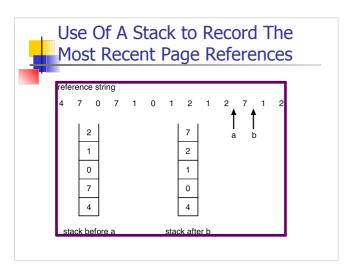
- Stack implementation keep a stack of page numbers in a double link form:
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - No search for replacement

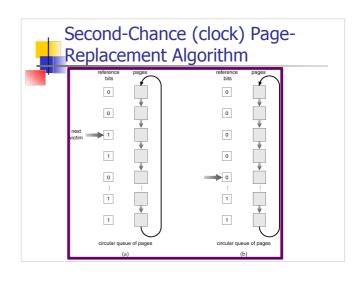


LRU Approximation Algorithms

- Reference bit
 - With each page associate a bit, initially = 0
 - When page is referenced bit set to 1.
 - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
 - · Need reference bit.
 - · Clock replacement.
 - If page to be replaced (in clock order) has reference bit = 1. then:
 - set reference bit 0.
 - leave page in memory.
 - replace next page (in clock order), subject to same rules.









Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.



Fixed Allocation

- Equal allocation e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation Allocate according to the size of process.



Global vs. Local Allocation

- Global replacement process selects a replacement frame from the set of all frames; one process can take a frame from another.
- Local replacement each process selects from only its own set of allocated frames.



Allocation of Frames

- Each process needs minimum number of pages.
- Example: IBM 370 6 pages to handle SS MOVE instruction:
 - instruction is 6 bytes, might span 2 pages.
 - 2 pages to handle from.
 - 2 pages to handle to.
- Two major allocation schemes.
 - fixed allocation
 - priority allocation



Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault,
 - select for replacement one of its frames.
 - select for replacement a frame from a process with lower priority number.



Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
 - low CPU utilization.
 - operating system thinks that it needs to increase the degree of multiprogramming.
 - another process added to the system.
- **Thrashing** ≡ a process is busy swapping pages in and out.



- Why does paging work? Locality model
 - Process migrates from one locality to another.
 - Localities may overlap.
- Why does thrashing occur?
 Σ size of locality > total memory size



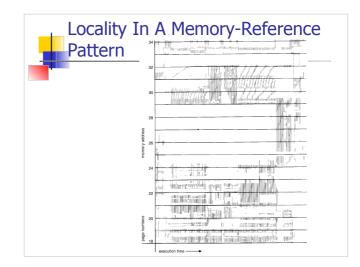
Working-Set Model

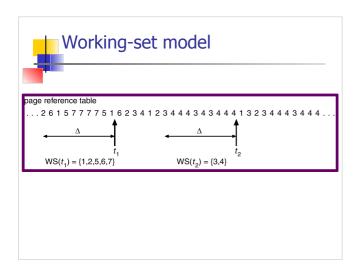
- Δ working-set window: a fixed number of page references
 - Example: 10,000 instruction
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
 - if Δ too small will not encompass entire locality.
 - if Δ too large will encompass several localities.
 - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma WSS_i \equiv \text{total demand frames}$
 - if $D > m \Rightarrow$ Thrashing
 - Policy if D > m, then suspend one of the processes.

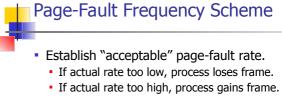


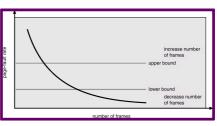
Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10,000$
 - Timer interrupts after every 5000 time units.
 - Keep in memory 2 bits for each page.
 - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
 - If one of the bits in memory = 1 ⇒ page in working set.
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units.











Other Considerations

- Prepaging
 - To reduce the large number of page faults that occurs at process startup
 - Prepage all or some of the pages a process will need, before they are referenced
 - But if prepaged pages are unused, I/O and memory was wasted
 - Assume s pages are prepaged and a of the pages is used
 - Is cost of s * a save pages faults > or < than the cost of prepaging
 - s * (1- a) unnecessary pages?
 - a near zero => prepaging loses



Other Considerations (Cont.)

- TLB Reach The amount of memory accessible from the TLB.
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB.
 - Otherwise there is a high degree of page faults



Other Considerations (Cont.)

- I/O Interlock Pages must sometimes be locked into memory.
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.



Other Considerations

- Page size selection
 - fragmentation
 - table size
 - I/O overhead
 - locality

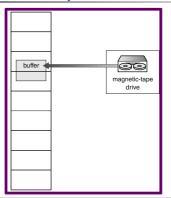


Increasing the Size of the TLB

- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size.
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.



Why Frames Used For I/O Must Be In Memory?





Operating System Examples

- Windows NT
- Solaris 2



Solaris 2

- Maintains a list of free pages to assign
- faulting processes. **Lotsfree** threshold parameter to begin
- Paging is peformed by *pageout* process.
- Pageout scans pages using modified clock algorithm.
- **Scanrate** is the rate at which pages are scanned. This ranged from slowscan to fastscan.
- Pageout is called more frequently depending upon the amount of free memory available.



Windows NT

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned working set minimum and working set maximum.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.

