Virtual Memory

Gordon College
Stephen Brinton

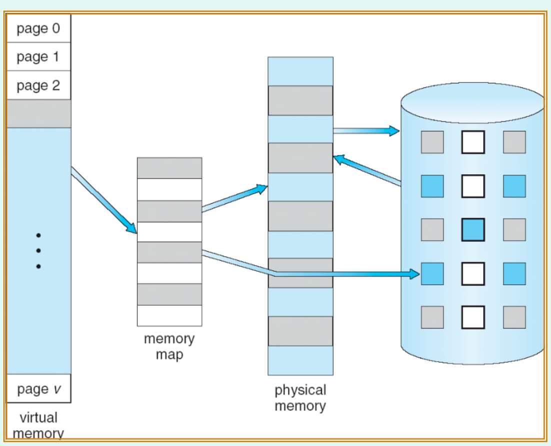
Virtual Memory

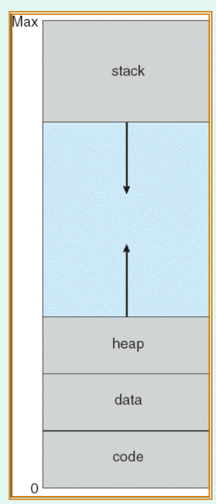
- Background
- Demand Paging
- Process Creation
- Page Replacement
- Allocation of Frames
- Thrashing
- Demand Segmentation
- Operating System Examples

Background

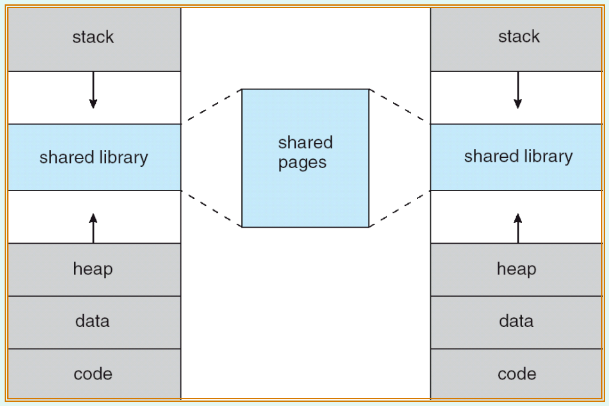
- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needed
  - Logical address space > physical address space.
    - (easier for programmer)
  - shared by several processes.
  - efficient process creation.
  - Less I/O to swap processes
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Larger Than Physical Memory





Shared Library Using Virtual Memory



Demand Paging

- Bring a page into memory <u>only when it is needed</u>
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users (processes) able to execute
- Page is needed ⇒ reference to it
  - Page available ⇒ immediate access
  - Invalid reference ⇒ abort
  - Not-in-memory ⇒ bring to memory

Valid-Invalid Bit

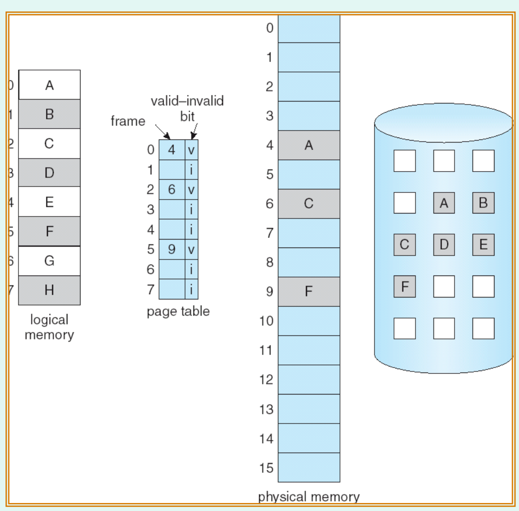
Example of a page table snapshot:

Frame #	valid-invalid bit
33	1
34	1
36	1
23	1
	0
⋮	
	0
	0

page table

- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries
- During address translation, if valid–invalid bit in page table entry is 0 ⇒ <u>page-fault trap</u>

Page Table: Some Pages Are Not in Main Memory

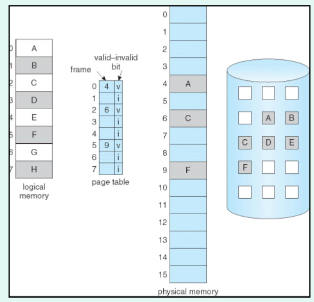


Page-Fault Trap

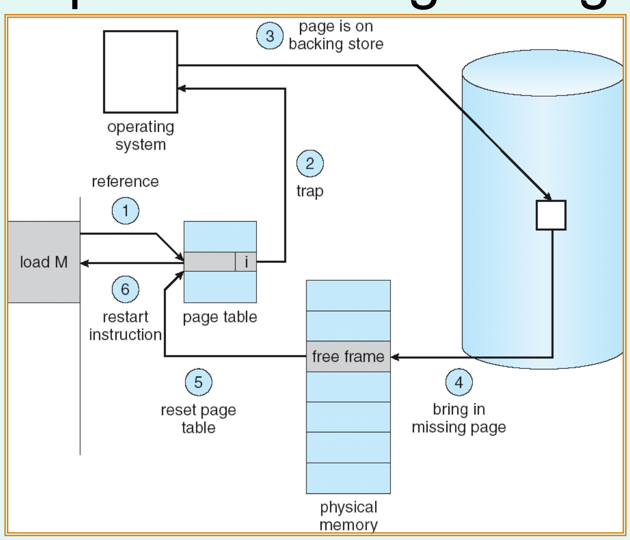
Reference to a page with invalid bit set - trap to OS ⇒ page fault

Must decide???:

- Invalid reference ⇒ abort.
- Just not in memory ⇒
  - Get empty frame.
  - Swap page into frame.
  - Reset tables, validation bit = 1.
  - Restart instruction:
    - what happens if it is in the middle of an instruction?



Steps in Handling a Page



What happens if there is no free frame?

- Page replacement – find some page in memory (not in use) & swap it out
  - Algorithm - must be speedy
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times
- LOCALITY OF REFERENCE principle

Performance of Demand Paging

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

$$\text{EAT} = (1 - p) \times \text{memory access} + p\ (\text{page fault overhead})$$

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out

Page-switch time: around 8 ms.

$$
\begin{aligned}
EAT &= (1 - p) \times (200) + p(8 \text{ milliseconds}) \\
&= (1 - p) \times (200) + p(8{,}000{,}000) \\
&= 200 + 7{,}999{,}800p
\end{aligned}
$$

$$220 > 200 + 7{,}999{,}800p$$

$$p < .0000025$$

Process Creation

- Virtual memory allows other benefits during process creation:

  - Copy-on-Write

  - Memory-Mapped Files

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

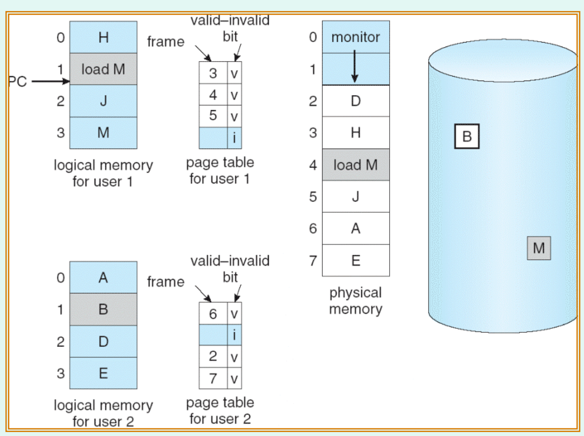
- COW allows more efficient process creation as only modified pages are copied

- Free pages are allocated from a **pool** of zeroed-out pages (the pool is kept in case of a need to copy)

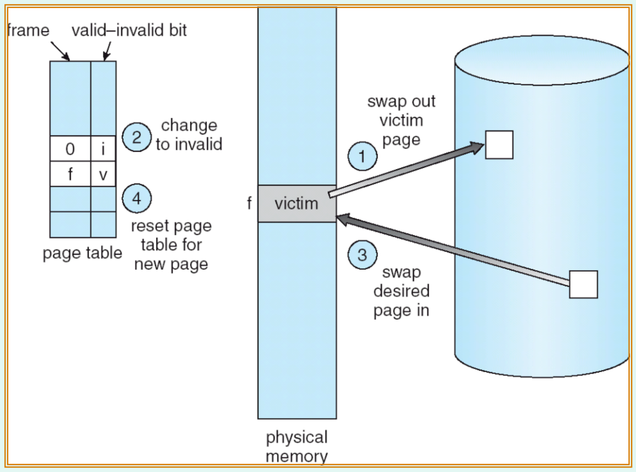
Need: Page Replacement

- 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



Basic Page Replacement

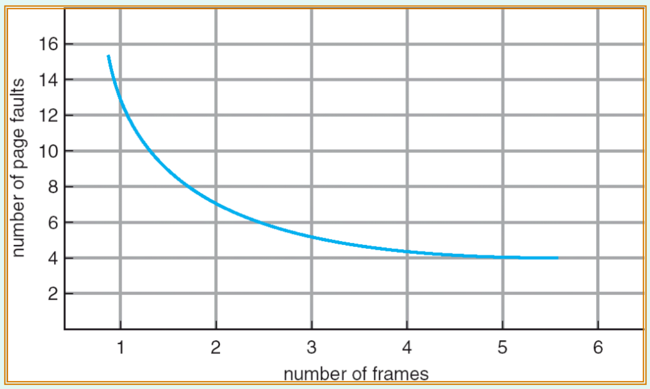


Page Replacement Algorithms

- GOAL: 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

Example string: 1,4,1,6,1,6,1,6,1,6,1

Graph of Page Faults Versus The Number of Frames



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)

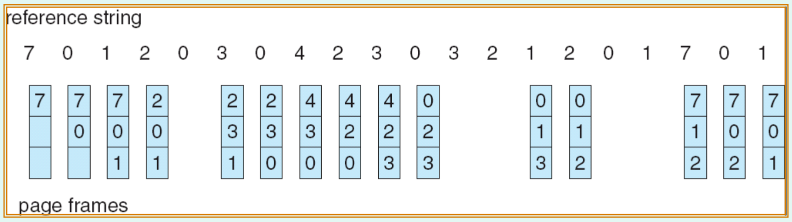
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

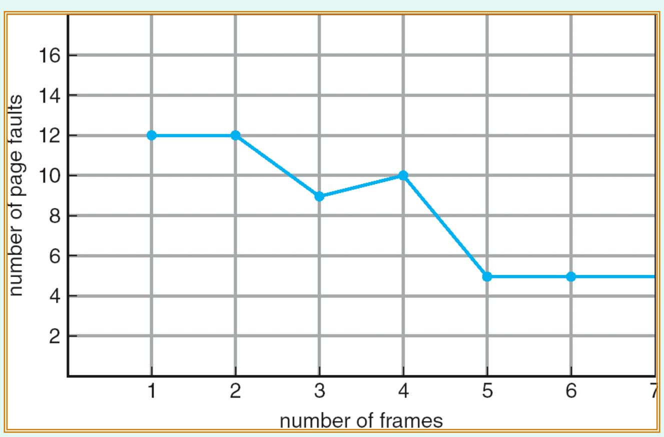
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement – Belady's Anomaly
  - more frames ⇒ more page faults

FIFO Page Replacement



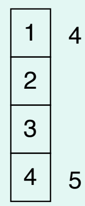
FIFO Illustrating Belady's Anomaly



Optimal Algorithm

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

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

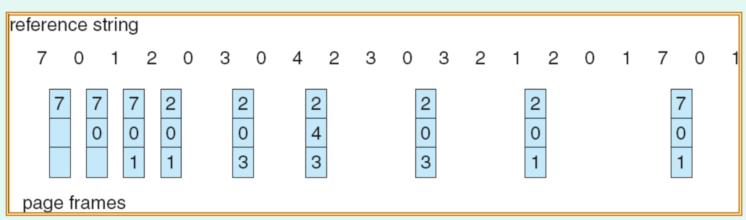


6 page faults

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

  “Well, is it at least 4% as good as Optimal Algorithm?”

Optimal Page Replacement



Optimal: 9 faults
FIFO: 15 faults
67% increase over the optimal

Optimal Page Replacement

- Requires FUTURE knowledge of the reference string
  - – therefore (just like SJF) – IMPOSSIBLE TO IMPLEMENT
- Therefore – used for comparison studies---

"…an algorithm is good because it is 12.3% of optimal at worst and within 4.7% on average"

Least Recently Used (LRU) Algorithm

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

1	5	
2		
3	5	4
4	3	

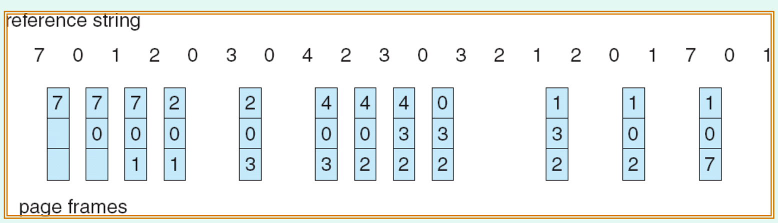
- <u>Counter implementation</u>
  - Every page entry has a <u>counter</u>; every time page is referenced through this entry: counter = clock
  - When a page needs to be changed, look at the counters to determine which are to change

LRU Page Replacement

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

LRU faults ?

LRU Page Replacement

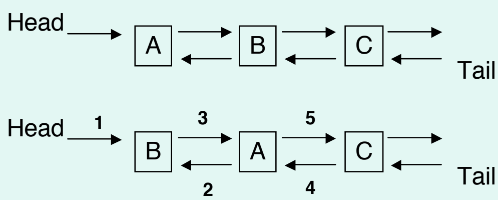


Optimal: 9 faults
FIFO: 15 faults

LRU: 12 faults

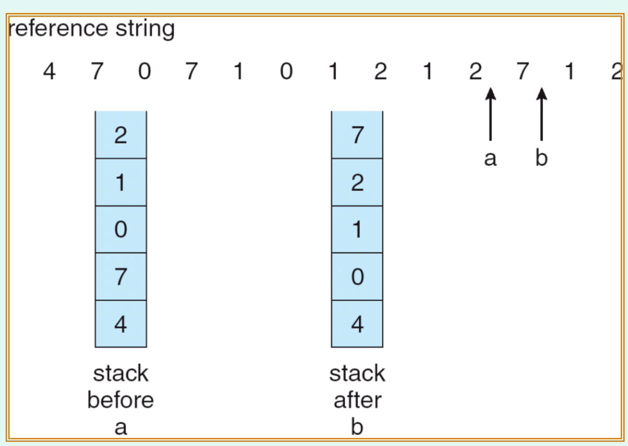
LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top (most recently used)
    - Worst case: 7 pointers to be changed
  - No search for replacement



Also B's previous link

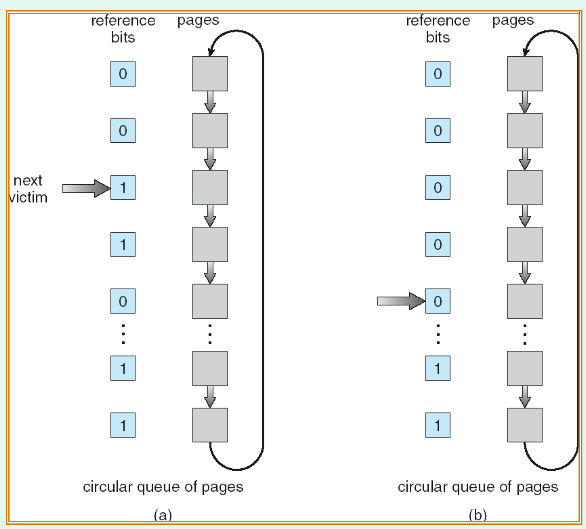
Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replacement: choose something with 0 (if one exists). We do not know the order, however.
- Second chance (Clock replacement)
  - Need reference bit
  - 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
  - Can use byte for more resolution

Second-Chance (clock) Page-Replacement Algorithm



use a circular queue

Counting Algorithms

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

Allocation of Frames

- Each process needs *minimum* number of pages: <u>depends on computer architecture</u>
- **Example**: IBM 370 – 6 pages to handle special MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- Two major allocation schemes
  - <u>fixed allocation</u>
  - <u>priority allocation</u>

Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.
- Proportional allocation – Allocate according to the size of process

s_i = size of process p_i

$S = \quad s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \,!\, m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \quad 64 \,!\, 5$

$a_2 = \frac{127}{137} \quad 64 \,!\, 59$

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*”
  - Con: Process is unable to control its own page-fault rate.
  - Pro: makes available pages that are less used pages of memory
- **Local replacement** – each process selects from only its own set of allocated frames

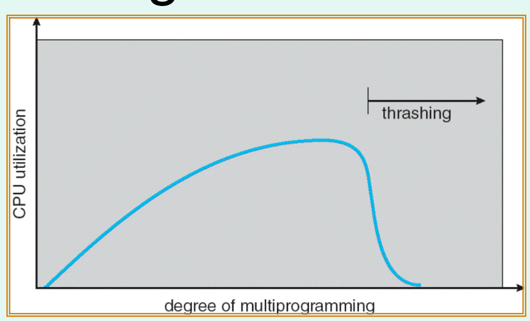
Thrashing

- Number of frames <u>less than</u> minimum required for architecture – must suspend process
  - Swap-in, swap-out level of intermediate CPU scheduling
- **Thrashing** ≡ a process is busy swapping pages in and out

Thrashing

<u>Consider this:</u>

- CPU utilization low – increase processes
- A process needs more pages – gets them from other processes
- Other process must swap in – therefore wait
- Ready queue shrinks – therefore system thinks it needs more processes



Demand Paging and Thrashing

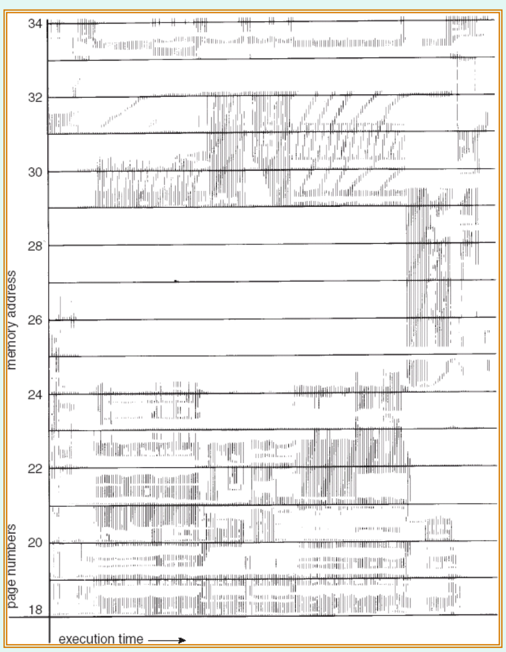
- Why does demand paging work?

  <u>Locality model</u>
  - as a process executes it moves from locality to locality
  - Localities may overlap

- Why does thrashing occur?
  Collective size of localities > total memory size

all localities added together

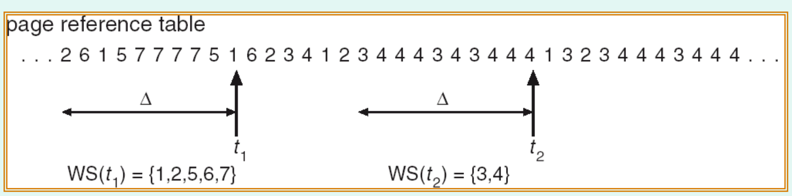
Locality In A Memory-Reference Pattern



Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (change 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 = Total Sum of $WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes

Working-set model

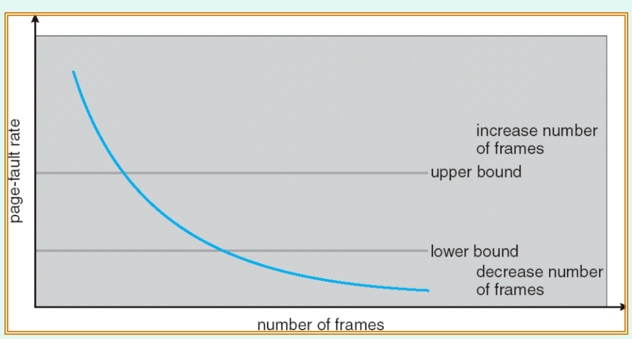


Keeping Track of the Working Set

- Approximate WS with interval timer + a reference bit
- Example: Δ = 10,000 references
  - 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 $\Rightarrow$ page in working set
- Why is this not completely accurate?
  - because a page could be in and out of set within the 5000 references.
- Improvement = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme

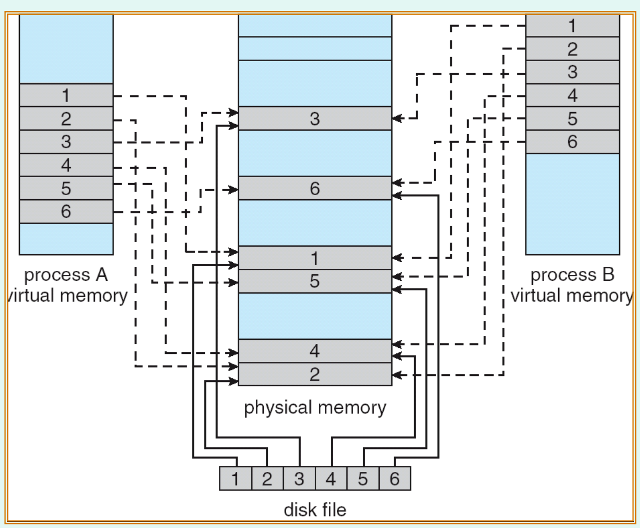
- Establish "acceptable" page-fault rate
  - If actual rate too low, process loses a frame
  - If actual rate too high, process gains a frame
- Used to tweek performance



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
- How?
  - 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 (less overhead)
- Sharing: Also allows several processes to map the same file allowing the pages in memory to be shared

Memory Mapped Files



WIN32 API

- Steps:
  Create a file mapping for the file
  Establish a view of the mapped file in the process's virtual address space

A <u>second process</u> can the open and create a view of the mapped file in its virtual address space

Other Issues -- Prepaging

- <u>Prepaging</u>
  - 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

Other Issues – Page Size

- Page size selection must take into consideration:
  - fragmentation
  - table size
  - I/O overhead
  - locality

Other Issues – Program Structure

- Program structure
  - Int[128,128] data;
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

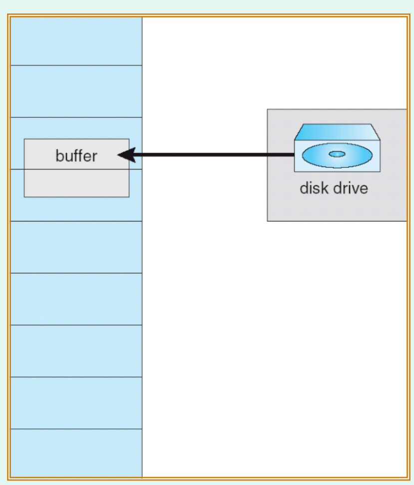
128 page faults

Other Issues – I/O interlock

- **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.

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



Other Issues – TLB Reach

- 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.
- 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.