Virtual Memory

- Page is not loaded until it is accessed
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code or data are needed to handle infrequently occurring cases.
- 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.
  - Need to allow pages to be swapped in and out.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O is needed
  - Less memory is needed
  - Faster response
  - More users
- Page is needed ⇒ Reference to it
  - Invalid reference ⇒ Abort
  - Not-in-memory ⇒ Page fault. Bring to memory

Valid-Invalid Bit

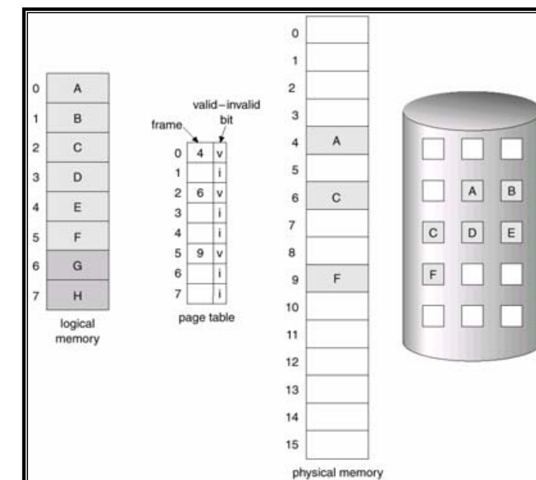
- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries.
- Example of a page table snapshot.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

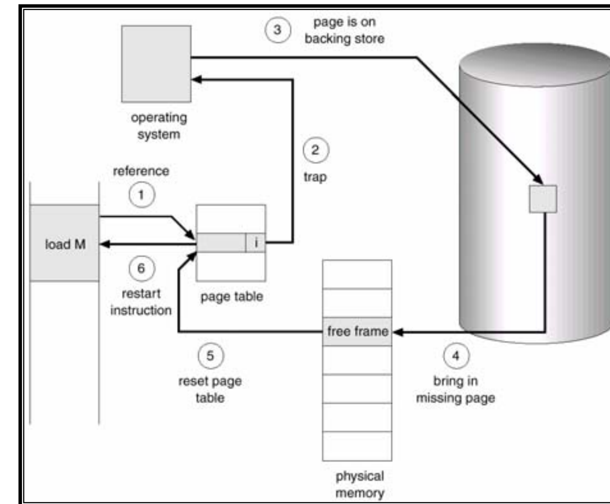
Page Table When Some Pages Are Not in Main Memory



Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at the page table to decide:
  - Invalid reference ⇒ Abort.
  - Just not in memory ⇒ Page fault:
    - ❆ Get empty frame.
    - ❆ Swap page into frame.
    - ❆ Reset tables, validation bit = 1.

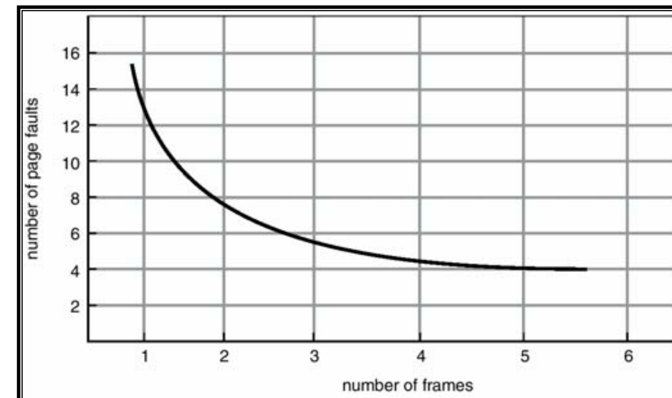
Steps in Handling a Page Fault



What happens if there is no free frame?

- Page replacement – Try to predict which page in the memory will not be used soon and swap it out.
- Performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.
- Use *dirty* (*modify*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.

Graph of Page Faults Versus The Number of Frames



Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  – if $p = 0$ no page faults
  – if $p = 1$, every reference is a fault
- Effective Access Time for One Page (EATOP)

EATOP = (1 – p) x memory access
+ p (page fault overhead
+ [swap page out if needed]
+ swap page in
+ restart overhead)

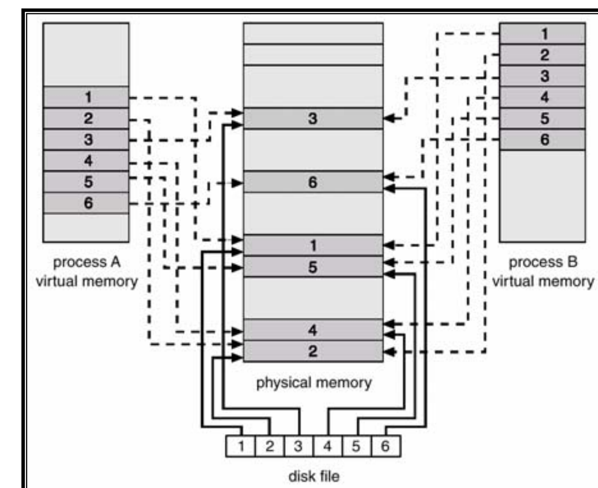
Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.
- If one of the processes modifies a shared page, the shared page will be copied.
- COW allows more efficient process creation as only modified pages are copied.
- COW makes the process creation faster.

Memory-Mapped Files

- 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 files' accesses by treating I/O of files through memory rather than **read()** and **write()** system calls.
- Also allows several processes to map the same file allowing the pages in the memory to be shared.
- UNIX implements this feature by the **mmap()** system call.

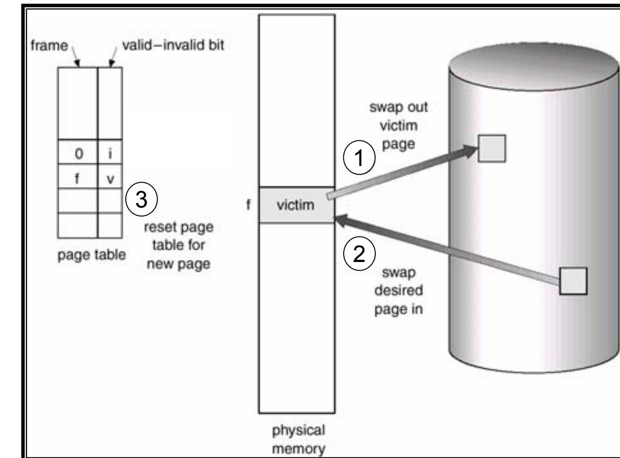
Memory Mapped Files



Page Replacement

- Find the location of the desired page on disk.
- 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.
- Read the desired page into the (newly) free frame. Update the page and frame tables.
- Restart the process.

Page Replacement



Page-Replacement Algorithms

- Wanted 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.

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)

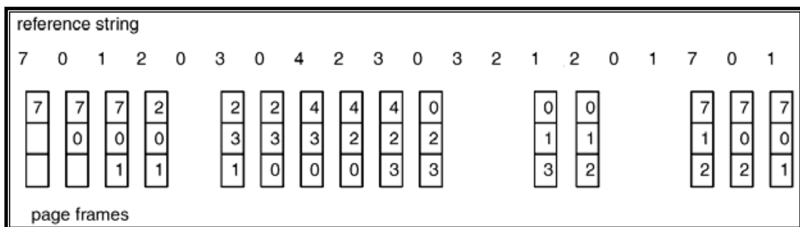
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

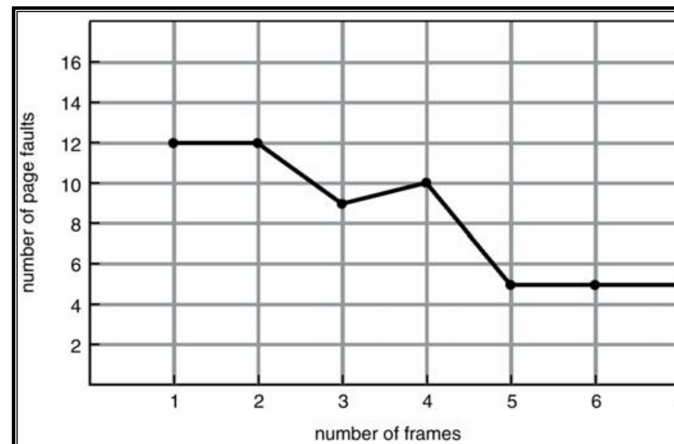
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement – Belady's Anomaly
  - more frames $\not\Rightarrow$ less page faults

FIFO Page Replacement



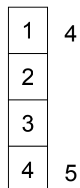
FIFO Illustrating Belady's Anamoly



Optimal Algorithm

- 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



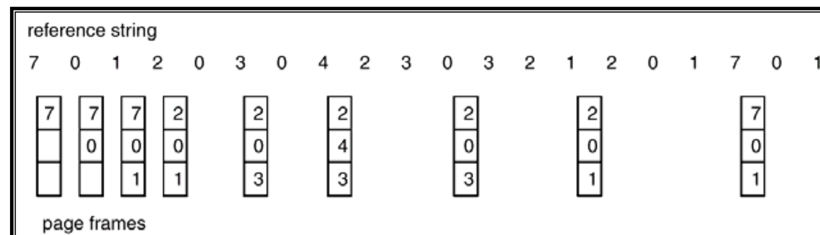
4

5

6 page faults

- How do we know this?
- Used for measuring how well an algorithm performs.

Optimal Page Replacement



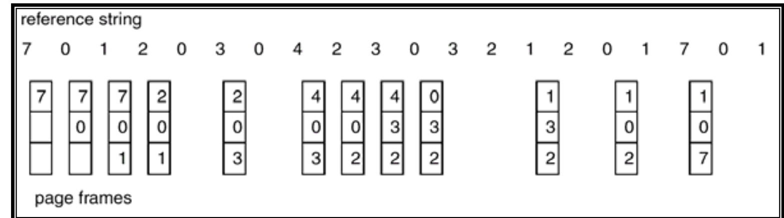
Least Recently Used (LRU) Algorithm

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

1		5
2		
3	5	4
4	3	

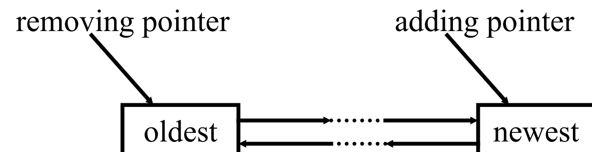
- Counter implementation
  - Each page entry has a time-stamp; when a page is referenced, copy the clock into the time-stamp field.
  - When a page needs to be replaced, look at the time-stamps to determine which one should be swapped out.

LRU Page Replacement

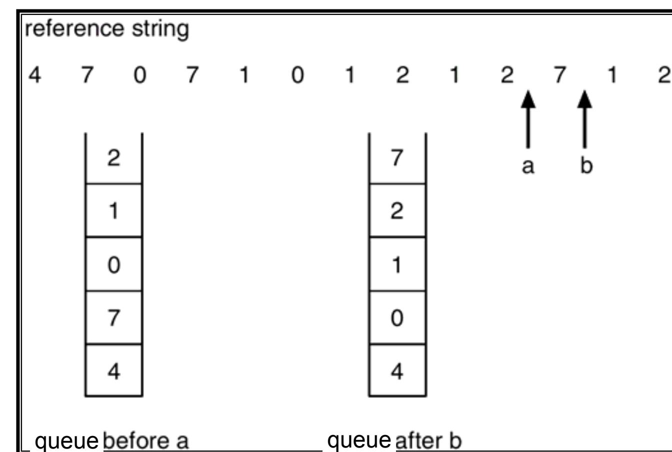


LRU Algorithm (Cont.)

- Queue implementation – keep a queue of page numbers in a double linked list in the form of:
  - If a page is referenced, move it to the top.
    - Requires 6 pointers to be changed. (2 that were pointing to this page, 2 that this page was pointing to, the "next" pointer of the old newest page and the adding pointer.)
  - If the page is not in the list, add it and remove the oldest page.
  - No search for the referred page, because the page table points it.



Use Of A Queue to Record The Most Recent Page References



Counting Algorithms

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

The Clock Algorithm

- The memory spaces holding the pages can be regarded as a circular buffer and the replacement algorithm cycles through the pages in the circular buffer, like the hand of a clock.
- Each page is associated with a bit, called reference bit, which is set by hardware whenever the page is accessed.
- When it is necessary to replace a page to service a page fault, the page pointed to by the hand is checked.
  - If its reference bit is unset, the page is replaced.
  - Otherwise, the algorithm resets its reference bit and keeps moving the hand to the next page.
- Linux and Windows employ Clock.

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.
  - $s_i =$ size of process p_i
  - $S = \sum s_i$
  - $m =$ total number of frames
  - $a_i =$ allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$
$$s_i = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Priority Allocation

- If process P_i generates a page fault,
  - Select for replacement a frame from a process with a lower priority value. If no such a frame:
  - Select for replacement one of its frames or a frame of another process with the same priority. If no such a frame:
  - Wait for a process with a higher priority value to finish. Then, select for replacement a frame from the finished process.

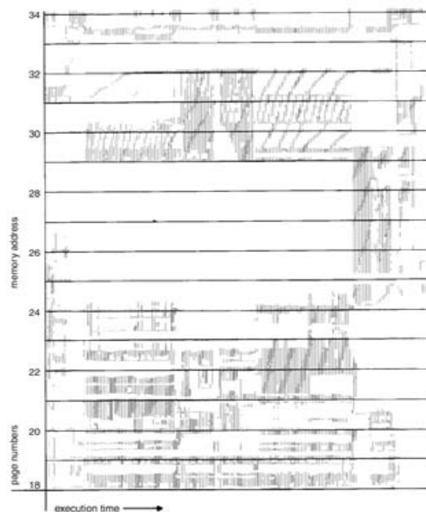
Global vs. Local Allocation

- 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.

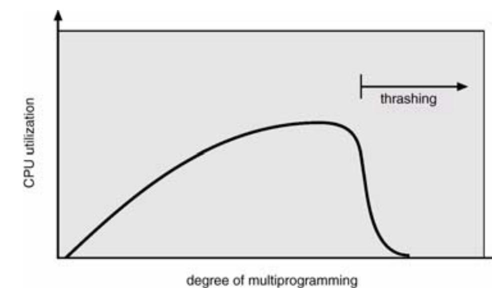
Locality Model

- If a value is referenced, there is a high probability that this value and the values in its neighborhood, will be referred soon.
- This neighborhood is called "locality".
- Why does paging work?
  - Process migrates from one locality to another.
  - Localities may overlap.

Locality In A Memory-Reference Pattern



Thrashing

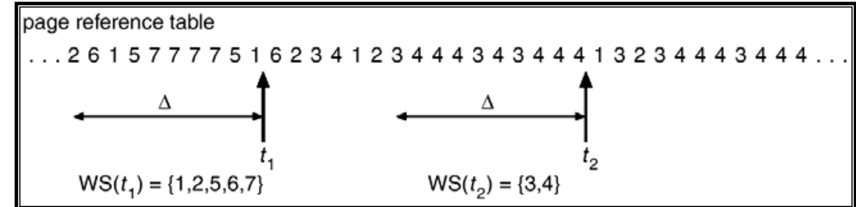


- If some processes do not have "enough" pages, the page-fault rate is very high. This leads to low CPU utilization.
- Thrashing ≡ Some processes are busy swapping pages in and out.
- Why does thrashing occur?
  Σ size of locality > total memory size

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ 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).
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D >$ *memory* $\Rightarrow$ Thrashing
- Possible policy: if D > memory, then suspend some of the processes.

Working-set model



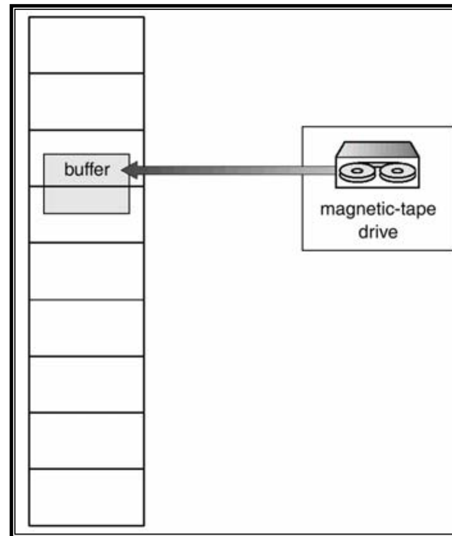
Page Size Selection

- Fragmentation
  - If the Page Size is increased, it may lead to an increase in fragmentation as not all applications require a large page size.
- Table size
  - TLB Coverage - The amount of memory accessible from the TLB.
  - TLB Coverage = (TLB Size) X (Page Size)
  - Ideally, the working set of each process is stored in the TLB. Otherwise the cost of page faults is high.
- I/O overhead

Page size selection (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.
- Locality
  - Larger pages contain more relevant data, hence less page faults.
  - Prepaging - Smaller pages, but brings more pages. Can save some fragmentation.

Why Frames Used For I/O Must Be In Memory



Other Consideration

- Program structure
  - Matrix A[1024, 1024] of integer
  - Each line of matrix is stored in one page
  - One frame
  - Program 1

```
for j := 1 to 1024 do
for i := 1 to 1024 do
    A[i,j] := 0;
```

    1024 x 1024 page faults

  - Program 2

```
for i := 1 to 1024 do
for j := 1 to 1024 do
    A[i,j] := 0;
```

    1024 page faults

Demand Segmentation

- OS/2 allocates memory in segments.
- Segment descriptor contains a valid bit to indicate whether the segment is currently in memory.
  - If segment is in main memory, access continues,
  - If not in memory, segment fault.
- Unix uses a combination of paging and segmentation.