OPERATING SYSTEMS

VIRTUAL MEMORY

9 Virtual Memory

1

- All the memory management policies try to keep a number of processes in the memory at the same time to allow multiprogramming. But they require process to be loaded in the memory before execution.
- With the virtual memory technique, we can execute a process which is only partially loaded in memory. Thus, the logical address space may be larger than physical memory, and we can execute more processes in memory at a time.

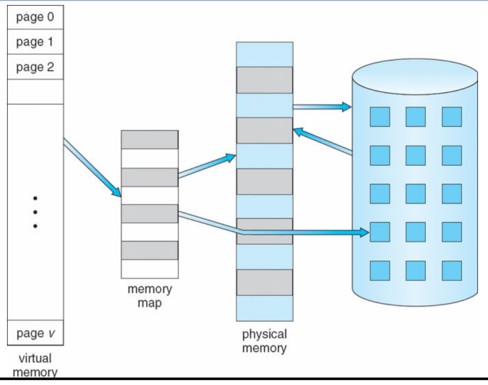
9 Virtual Memory

2

- **By means of virtual 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
  - A greater degree of multiprogramming can be possible
  - OS allows more efficient process creation
- **Virtual memory can be implemented by:**
  - Demand paging
  - Demand segmentation

9 Virtual Memory

3



9.1 Demand Paging

4

- Demand paging is the most common virtual memory management system. In demand paging, programs reside on a swapping device commonly known as the backing store. The backing store, for most of today's operating systems is a disk.
- When the operating system decides to start a new process, it swaps only a small part of this new process into memory.

9.1 Demand Paging

5

- The page table of this new process is prepared and loaded into memory. In the page table, there is a column to show valid/invalid bits which show whether that page in memory (valid) or not (invalid).
- If the executing process tries to access a page which is not in the memory (invalid), a page fault occurs.

9.2 Valid-Invalid Bit

6

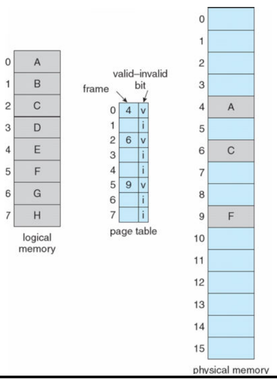
- With each page table entry a valid–invalid bit is associated (v ⇒ in-memory, i ⇒ not-in-memory)
- Initially valid–invalid bit is set to i for all entries
- During address translation, if valid–invalid bit in page table entry is i, it means that a page fault will occur for this entry.

Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
….	
	i
	i

9.2 Valid-Invalid Bit

7



9.3 Page Faults

8

- When a page fault happened, the OS finds the desired page on the disk and looks for a free frame on the main memory. If there is no free frame, the OS must choose a frame to swap it out to the disk.
- Then, the valid/invalid bit of the chosen page is changed as "i".
- Now the desired page is swapped into newly freed frame, its frame table is modified, and the valid/invalid bit of it is set to valid.

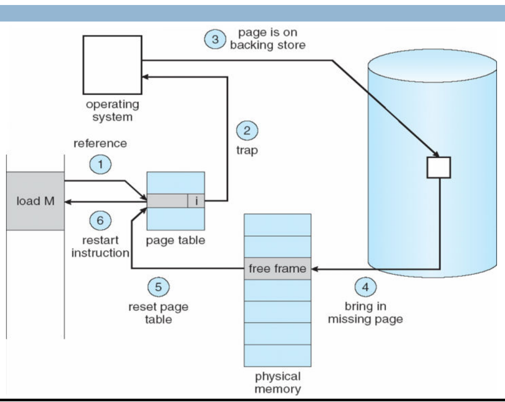
9.3 Page Faults

9

- Trap the OS. Save registers and process state for the current process.
- Check whether the page reference is valid, when a page fault occurred.
- If yes, determine the location of the required page on the backing store.
- Find a free frame and swap in the required page from the backing store into the free frame.
- When I/O is completed, restore registers and process state for the process which caused the page fault and save state of the currently executing process.
- Modify the corresponding page table entry to show that the recently copied page is now in memory.
- Restart instruction that caused the page fault.

9.3 Page Faults

10



9.3 Page Faults

11

These operations can be summarized as:

- (FAST) Checking the address and choosing a frame
- (SLOW) Swap out the chosen frame to secondary storage (mostly disk is used)
- (SLOW) Swap in the page from secondary storage into the vacated frame.
- (FAST) Restart the process

In servicing a page fault, the time is spent mainly for swap-out and swap-in. The periods of other operations can be negligible.

9.4 Performance

- Page Fault Probability $0 \le p \le 1.0$
  - if $p = 0$, it means no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (eat) = $(1 - p) * \text{eat}_{\text{no-pf}} + p * \text{eat}_{\text{pf}}$
- $\text{eat}_{\text{no-pf}}$ = memory access time
- eat_{pf} = page fault overhead + swap page out
  + swap page in+ restart overhead

In briefly, eat_{pf} can be represented by "page fault service time".

9.4 Performance

Example:

- Memory access time = 200 ns
- Average page fault service time = 8 ms
  eat = (1 – p) x 200 ns + p x 8 ms
  = 200 (1 – p) + 8,000,000 p
  = 200 + 7,999,800 p ns
- If one access out of 1000 (p=0.001) causes a page fault, then
  eat = 8.2 ms

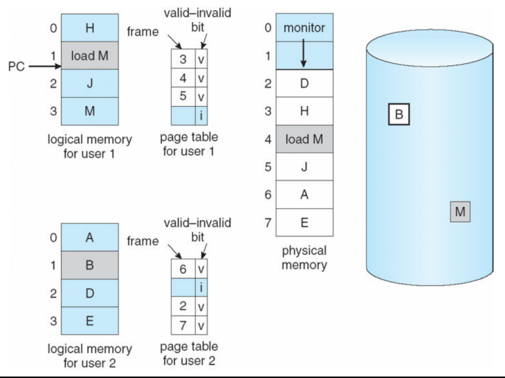
According to $\text{eat}_{\text{no-pf}}$, there is a slowdown by a factor of 40K.

9.5 Page Replacement

- A page replacement algorithm determines how the page to be replaced is selected when a page fault occurs. The aim is to minimize the page fault rate.

- The efficiency of a page replacement algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. Reference strings are either generated randomly, or by tracing the paging behavior of a system and recording the page number for each logical memory reference.

9.5 Page Replacement

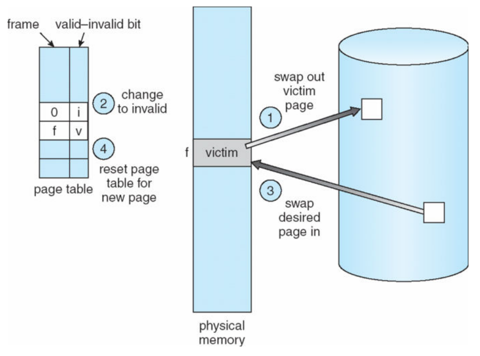


9.5 Page Replacement

Usual algorithm when a page fault occurred:

- 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, select a frame using a page replacement algorithm to move it into disk
- Bring the desired page into the free frame; update the page and frame tables
- Restart the process

9.5 Page Replacement



9.5.1 Dirty Bit

- The dirty bit allows performance optimization.
- This strategy requires that the backing store retain a copy of the page after it was swapped into memory.
- The dirty bit is set to “0” by the hardware when the page is swapped in.
- When we select a frame by using a page replacement algorithm, we examine its dirty bit.

9.5.1 Dirty Bit

- If it is “1”, that means the page has been modified since it was swapped in.
- In this case we have to copy that page into the backing store.
- However if the dirty bit is “0”, that means the page has not been modified since it was swapped in, so the copy in the backing store is valid.

9.5.2 Optimal Page Replacement

- Optimal Page Replacement (OPT) chooses the page which will not be used for the longest period.
- For a fixed number of frames, OPT has the lowest page fault rate among the page replacement algorithms.
- But OPT is not possible to be implemented in practice. Because it requires future knowledge. However, it is used for performance comparison.

9.5.2 Optimal Page Replacement

Example

- Assume we have 3 frames and consider the reference string of 5, 7, 6, 0, 7, 1, 7, 2, 0, 1, 7, 1, 0
- Show the content of memory after each memory reference if OPT page replacement algorithm is used. Find also the number of page faults.

	5	7	6	0	7	1	7	2	0	1	7	1	0
f1	5	5	5	0	0	0	0	0	0	0	0	0	0
f2		7	7	7	7	7	7	2	2	2	7	7	7
f3			6	6	6	1	1	1	1	1	1	1	1
pf	1	2	3	4	same	5	same	6	same	same	7	same	same

OPT finds 7 page faults.

9.5.3 First-In-First-Out (FIFO)

The operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the oldest arrival in front. When a page needs to be replaced, the page at the front of the queue is selected.

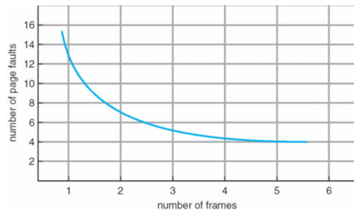
For the same example:

	5	7	6	0	7	1	7	2	0	1	7	1	0
f1	5	5	5	0	0	0	0	2	2	2	7	7	7
f2		7	7	7	7	1	1	1	0	0	0	0	0
f3			6	6	6	6	7	7	7	1	1	1	1
pf	1	2	3	4	same	5	6	7	8	9	10	same	same

FIFO finds 10 page faults.

9.5.3 First-In-First-Out (FIFO)

Normally, while the total number of frames increases, decreasing of the number of page faults is expected.



However, for FIFO, there are some cases where this generalization fails. This is called **Belady’s Anomaly**.

9.5.3 First-In-First-Out (FIFO)

As an exercise consider the reference string below, and find the number of page faults applying the FIFO method with 3 and 4 paged main memory. Then, examine whether the replacement suffer Belady's anomaly.

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

9.5.4 Least Recently Used (LRU)

In this algorithm, the chosen is the page that has not been used for the longest period.

The OS sets the reference bit of a page to "1" when it is referenced. This bit will not give the order of use but it will simply tell whether the corresponding frame is referenced recently or not. The OS resets all reference bits periodically.

9.5.4 Least Recently Used (LRU)

For the same string (5, 7, 6, 0, 7, 1, 7, 2, 0, 1, 7, 1, 0)

	5	7	6	0	7	1	7	2	0	1	7	1	0
f1	5	5	5	0	0	0	0	2	2	2	7	7	7
f2		7	7	7	7	7	7	7	7	1	1	1	1
f3			6	6	6	1	1	1	0	0	0	0	0
pf	1	2	3	4	same	5	same	6	7	8	9	same	same

LRU finds 9 page faults.

9.6 Frame Allocation

In order to be able to decide on the page replacement scheme of a particular reference string, we have to know the number of page frames available. In page replacement, some frame allocation policies may be followed.

- Global Replacement: A process can replace any page in the memory.
- Local Replacement: Each process can replace only from its own reserved set of allocated page frames.

9.6 Frame Allocation

In case of local replacement, the operating system should determine how many frames should the OS allocate to each process. The number of frames for each process may be adjusted by using two ways:

- Equal Allocation
- Proportional Allocation

9.6 Frame Allocation

- Equal allocation: If there are p processes and n frames, frame allocation for process p is as below:

  $f(p) = n / p$

- Proportional allocation: If v(p) is virtual memory size of process p, there are m processes and n frames, then the total virtual memory size will be:

  $V = \Sigma v(p)$

  Frame allocation for process p is as below:

  $f(p) = n * v(p) / V$

9.6 Frame Allocation

Example: Consider a system having 64 frames and there are 4 processes with the following virtual memory sizes: v(1) = 16, v(2) = 128, v(3) = 64 and v(4) = 48.

Equal Allocation

Assume that there are n frames, and p processes, then n/p frames are allocated to each process allocates 64 / 4 = 16 frames to each process.

9.6 Frame Allocation

Example: Consider a system having 64 frames and there are 4 processes with the following virtual memory sizes: v(1) = 16, v(2) = 128, v(3) = 64 and v(4) = 48.

Proportional Allocation

V = 16 + 128 + 64 + 48 = 256. It allocates:

(16 / 256) * 64 = 4 frames to process 1,

(128/ 256) * 64 = 32 frames to process 2,

(64 / 256) * 64 = 16 frames to process 3,

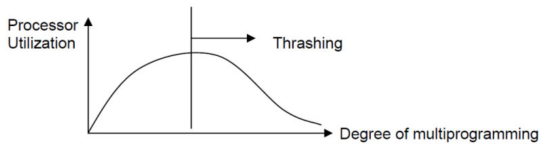
(48 / 256) * 64 = 12 frames to process 4.

9.7 Thrashing

Because of frequent page faults, if a process is spending more time for paging in/out than executing, then it is called as thrashing. Thrashing causes considerable decreasing in system performance. If a process does not have enough number of frames, it will send a page fault. Although a page replacement is done, if all pages are in active use, another page selection to replace will be needed in a very short time. This means another page fault will be issued shortly, and so on.

9.7 Thrashing

Local replacement algorithms can limit the effects of thrashing. If the degree of multiprogramming is increased over a limit, processor utilization falls down considerably because of thrashing.



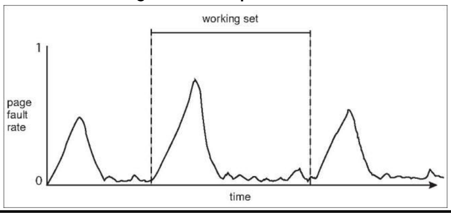
9.8 Working Set

To prevent thrashing, the process must be supported by providing as many frames as it needs. For this, a model called the working set model is developed. The working set of a process is the set of pages of the process that are currently resident in physical memory.

The working set contains only pageable memory allocations; nonpageable memory allocations or large page allocations are not included in the working set.

9.8 Working Set

- We shall use a parameter (T) called the working set window size.
- We shall examine the last T page references.
- The set of pages in the last page references shall be called the working set of a process.



9.8 Working Set

T parameter is chosen so that all pages referenced in the last T seconds comprise the working set.

- If T is chosen too small, it won't cover entire working set.
- If T is too large, several localities of a process may overlap.

Then, compute the WS size (WSS) for each process, and find the total demand (D) of the system at that time instance, as the summation of all the WS sizes.

$D(t_{now}) = \Sigma\ WSS_i(t_{now})$ for $i=1,\ldots,p$

9.8 Working Set

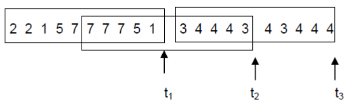
If the number of frames is n, then

a. If $D > n$, the system is thrashing.
b. If $D < n$, the system is all right, the degree of multiprogramming can possibly be increased.

If $D > n$ at any time instant, OS selects a process to suspend for a while. The frames that were used by the selected process are reallocated to other processes.

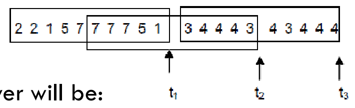
9.8 Working Set

Example: Assume T = 10 , and consider the reference string given below, on which the window is shown at different time instants. Find the working sets at these time instants.



9.8 Working Set

Example:



Answer will be:

$WS(t_1) = \{2,1,5,7\}$
$WS(t_2) = \{7,5,1,3,4\}$
$WS(t_3) = \{3,4\}$

So, WS size of this process have been 5 at most.