Chapter 4
Virtual Memory

All the memory management policies we have discussed so far, try to keep a number of processes in the memory simultaneously to allow multiprogramming. But they require an entire process to be loaded in the memory before it can execute.

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 have more processes executing in memory at a time, hence a greater degree of multiprogramming.

4.1 Demand Paging

Demand paging is the most common virtual memory management system. It is based on the locality model of program execution. As a program executes, it moves from locality to locality.

Locality is defined as a set of pages actively used together. A program is composed of several different localities which may overlap. For example, a small procedure when called, defines a new locality. A while-do loop when being executed defines a new locality.

As a program executes, it moves from locality to locality.

Procedural languages with while-do, repeat-until, for-do structures (i.e. Pascal, Algol, C, etc.) have less frequently changing localities than other high-level languages with go-to structure (i.e. Basic, Fortran).

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 (a few pages) into memory. The page table of this new process is prepared and loaded into memory, and the valid/invalid bits of all pages that are brought into memory are set to "valid". All the other valid/invalid bits are set to "invalid" showing that those pages are not brought into memory.

If the process currently executing tries to access a page which is not in the memory, a **page fault** occurs, and the OS brings that page into the memory. If a process causes page fault then the following procedure is applied:

1. Trap the OS.
2. Save registers and process state for the current process.
3. Check if the trap was caused because of a page fault and whether the page reference is legal.

4. If yes, determine the location of the required page on the backing store
5. Find a free frame.
6. Read (swap in) the required page from the backing store into the free frame. (During this I/O, the processor may be scheduled to some other process)
7. 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.
8. Modify the corresponding page table entry to show that the recently copied page is now in memory.
9. Resume execution with the instruction that caused the page fault.

While executing a process, in the case of a page fault, the OS finds the desired page on the backing store and if there is no free frames, the OS must choose a page in the memory (which is not the one being used) as a victim, and must swap it out (slow) to the backing store.

Then, the OS changes the valid/invalid bit for the victim page in the page table to indicate that it is no longer in memory. It swaps the desired page into newly freed frame, modifies the frame table, and sets the valid/invalid bit of the new page to valid. The executing process may now go on.

This operations can be summarized as:

1. Checking the address and finding a free frame or victim page (fast)
2. Swap out the victim page to secondary storage (slow)
3. Swap in the page from secondary storage (slow)
4. Context switch for the process and resume its execution (fast)

In servicing a page fault, the time is spent mainly for swap-out and swap-in. The time required for other operations are negligible.

Virtual memory can be implemented in:

- paging systems
- paged segmentation systems
- segmentation systems (However, segment replacement is much more sophisticated than page replacement since segments are of variable size.)

4.2. Performance Calculation of Demand Paging System

If there is no page fault, effective access time is effective memory acces time is

$$eat_{NO\text{-}PF} = emat$$

If there is a page fault, we have

$$eat_{PF} = pfst + emat \cong pfst$$

where pfst is page fault service time. Since emat is very small compared to pfst it can be ignored.

Now, let p be the probability for a page fault ($0 \leq p \leq 1$) to occur. Then, the effective access time must be calculated as follows:

$$eat = p * eat_{PF} + (1 - p) * eat_{NO\text{-}PF}$$
$$\cong p * pfst + (1 - p) * emat$$

Example 3.1:

Effective memory access time for a system is given as 1 microseconds, and the average page fault service time is given as 10 milliseconds. Let p=0.001. Then, effective access time is

$$eat = 0.001 * 10\ msec + (1 - 0.001) * 1\ \mu sec$$
$$= 0.001 * 10\,000\ \mu sec + 0.999 * 1\ \mu sec$$
$$\cong 10\ \mu sec + 1\ \mu sec = 11\ \mu sec$$

4.3 Dirty bit

In order to reduce the page fault service time, a special bit called the dirty bit can be associated with each page.

The dirty bit is set to "1" by the hardware whenever the page is modified. (written into).

When we select a victim by using a page replacement algorithm, we examine its dirty bit. If it is set, that means the page has been modified since it was swapped in. In this case we have to write that page into the backing store.

However if the dirty bit is reset, that means the page has not been modified since it was swapped in, so we don't have to write it into the backing store. The copy in the backing store is valid.

Let the probability of a page being dirty be d. In this case effective access time becomes:

$$eat = p * ((1-d) * swap\_in + d * (swap\_in + swap\_out)) + (1 - p) * emat$$

4.4 Page Replacement Algorithms

A page replacement algorithm *determines how the victim page (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.

The performance of a page replacement algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults.

Consecutive references to the same page may be reduced to a single reference, because they won't cause any page fault except possibly the first one:

(1,4,1,6,1,1,1,3) → (1,4,1,6,1,3).

We have to know the number of page frames available in order to be able to decide on the page replacement scheme of a particular reference string. Optionally, a frame allocation policy may be followed.

4.4.1 Optimal Page Replacement Algorithm (OPT)

In this algorithm, the victim is the page which will not be used for the longest period. For a fixed number of frames, OPT has the lowest page fault rate between all of the page replacement algorithms, but there is problem for this algorithm. OPT is not possible to be implemented in practice. Because it requires future knowledge. However, it is used for performance comparison.

Example 4.2

Assume we have 3 frames and consider the reference string below.

Reference string: 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

According to the given information, this algorithm generates a page replacement scheme with 7 page faults.

4.4.2 First-In-First-Out (FIFO)

This is a simple algorithm, and easy to implement. The idea is straight forward: choose the oldest page as the victim.

Example 4.3

Assume there are 3 frames, and consider the reference string given in example 4.2. Show the content of memeory after each memory reference if FIFO 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	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

10 page faults are caused by FIFO

Belady's Anomaly

Normally, one would expect that with the total number of frames increasing, the number of page faults decreases. However, for FIFO, there are cases where this generalization fails. This is called Belady's Anomaly.

As an exercise consider the reference string below. Apply the FIFO method and find the number of page faults considering different number of frames. Then, examine whether the replacement suffer Belady's anomaly.

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

4.4.3 Least Recently Used (LRU)

In this algorithm, *the* victim is the page that has not been used for the longest period. So, this algorithm makes us be rid of the considerations when no swapping occurs.

The OS using this method, has to associate with each page, the time it was last used which means some extra storage. In the simplest way, 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.

Example 4.4

Assume there are 3 frames, and consider the reference string given in example 4.2. Show the content of memory after each memory reference if LRU 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	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

This algorithm resulted in 9 page faults.

4.5 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.

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: If there are n frames and p processes, n/p frames are allocated to each process.

- Proportional Allocation: Let the virtual memory size for process p be v(p). Let there are m processes and n frames. Then the total virtual memory size will be: V = Σv(p). Allocate (v(p) /V) * n frames to process p.

Example 4.5

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.

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.

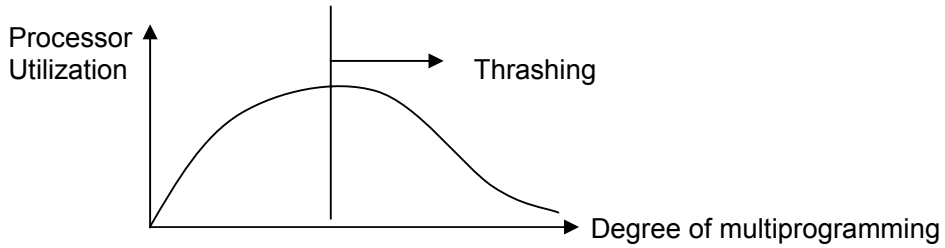
4.6.Thrashing

A process is thrashing if it is spending more time for paging in/out (due to frequent page faults) than executing.

Thrashing causes considerable degradation in system performance. If a process does not have enough number of frames allocated to it, it will issue a page fault. A victim page must be chosen, but if all pages are in active use. So, the victim page selection and a new page replacement will be needed to be done in a very short time. This means another page fault will be issued shortly, and so on and so forth.

In case a process thrashes, the best thing to do is to suspend its execution and page out all its pages in the memory to the backing store.

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.



To prevent thrashing, we must provide a process as many frames as it needs. For this, a model called *the working set model* is developed which depends on the locality model of program execution. But here we only mention its name and skip its details to limit the scope.

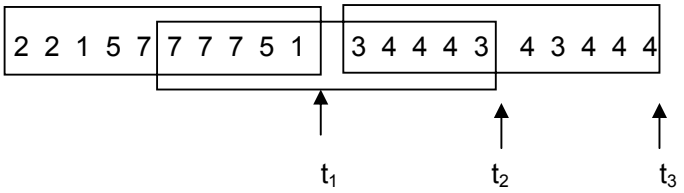
4.7 Working Set Model

To prevent thrashing, we must provide a process as many frames as it needs. For this, we shall use the working set model, which depends on the locality model of program execution, discussed earlier.

We shall use a parameter, Δ, called the working set window size. We shall examine the last Δ page references.

The set of pages in the last page references shall be called the working set of a process.

Example 4.6 : Assume $\Delta = 10$, and consider the reference string given below, on which the window is shown at difeferent time instants



Working sets of this process at these time instants will be:

WS(t1) = {2,1,5,7}
WS(t2) = {7,5,1,3,4}
WS(t3) = {3,4}

Note that in calculating the working sets, we do not reduce consequent references to the same page to a single reference. Choice of Δ is crucial. If Δ is to small, it will not cover the entire working set. If it is too large, several localities of a process may overlap. Madnick and Donovan suggested Δ to be about 10.000 references.

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

$$D(t_{now}) = \sum_{i=1}^{p} WSS_i(t_{now})$$

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.

In order to be able to use the working set model for virtual memory management, the OS keeps track of the WS of each process. It allocates each process enough frames to provide it with its WS.

If at one point D > n, OS selects a process to suspend. The frames that were used by the selected process are reallocated to other processes.

We can also use the page fault frequency to decide on decreasing or increasing the no. of frames allocated to a process.

QUESTIONS

1. Consider a demand paging system with associative registers.

a. If the hit ratio for the system is 80%, register access time is 50 nanoseconds and the main memory access time is 950 nanoseconds, calculate the effective memory access time (emat). Assume that associative registers can be loaded in parallel with an access to the main memory.

b. Let the probability of a page fault be 0.005 and calculate the effective access time if the page fault service time is given as 5 msec. Use the emat calculated in part 'a'.

2. Consider a demand paging system. assume working set window is 7, and the following reference string is given for process P:

1, 2, 1, 4, 3, 4, 1, 2, 1, 4, 5, 2, 5, 3, 5, 2, 3
t_{now}

a. What is the current working set of this process?

b. What is the current working set size of this process?

c. What happens if the summation of working set sizes for all the currently executing processes is larger than the total number of frames in the memory?

d. What would you suggest to solve the problem in 'c'?

3. The following reference string is given for a process executing in a demand paging system which uses FIFO page replacement algorithm:

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

a. If there are three memory frames allocated to these process, give a picture of all pages in memory for the given reference string, after each page fault.

b. Repeat part 'a' for four memory frames.

c. Comment on the total number of page faults in part 'a' and part 'b'.

4. Consider a demand paging system with the following observed average device utilization values:

processor utilization = 30%
disk utilization = 95%
I/O devices utilization = 10%

Discuss whether the following measures can improve the processor utilization: (considering them one by one)

a. Replace the processor with a slower one.

b. Replace the processor with a faster one.

c. Replace the disk with a faster one.

d. Increase the degree of multiprogramming.

e. Decrease the degree of multiprogramming.

f. Replace the I/O devices with faster ones.

5. In a demand paging system, the page replacement policy is: examine each page in memory regularly, and swap those pages, which have not been used since the last examination, to disk. Discuss whether this policy would result in better performance than LRU page replacement policy or not.

6. Consider a demand paging system. The effective memory access time is 8000 nanoseconds. It takes 9 milliseconds to service a page fault if a free frame is found in memory, or the page to be replaced is not dirty, and takes 22 milliseconds to service a page fault if there are no free frames and the victim page is dirty. In 7 out of every 10 page faults, a victim page has to be chosen and in 5 of these 7 cases, the victim page is dirty. What is the acceptable page fault rate for this system in order to achieve an effective access time of no more than 2 milliseconds ?

7. In a virtual memory system, what determines the size of a logical address space? What determines the size of a physical address space? What (if any) relationship is there between them?

8. A working set window is the most recent k page references made by a process. Assume the most recent page references of a process are :

..., 6, 5, 3, 3, 4, 3, 2, 1, 4, 2 and Δ is 7.

a. What is the current working set window of this process?

b. What is the current working set of this process?

9. a. Explain how the LRU page replacement algorithm works.

b. Why do we need extra hardware such as a dirty bit and a reference bit in order to implement the LRU page replacement algorithm in a demand paging system?

c. A demand paging system with 4 page frames uses the LRU algorithm for page replacement. Give a diagram of all pages in memory after each page fault, for the following reference string:

2,4,5,4,5,9,2,7,4,8,9,1,6,2,2,5,3,8

d. In a demand paging system, why is it generally more desirable to replace an unmodified page than a modified one?

e. Under what circumstances might it be more desirable to replace a modified page?

10. An n_bit counter is associated with each frame in a virtual memory system with f frames to implement demand paging. Every time any page is referenced by a process , a '1' is shifted into the most significant bit of the counter associated with the referenced page and

zeros are shifted into the counters of all other frames allocated to the referencing process. This operation simply discards the least significant bit. The lowest numbered frame with the lowest counter value is replaced in case of a page fault.

a. With this implementation, show the frame contents after each page reference for the following reference string for n=2 (i.e. 2 bit counters) and f=4. Assume that initially all frames are empty with counters=00. There is only one process.

Reference string: 4,3,4,7,2,3,5,8.

b. If this implementation is to be used in a multiprogramming environment with local page replacement and equal allocation, give an expression for the maximum sensible allocation, a, per process, in terms of n and f. Also give an expression for the minimum multiprogramming degree p so that this implementation to function exactly as LRU. Explain your reasoning.

c. How would your answer to part **b.** change if the underlined word lowest is changed to highest in the explanation above ?

11. In each of the following cases, show the memory contents for the given page reference string, initial memory contents and page replacement policy:

a. Assume that there is single process in the system.

Page reference string: 1,6,1,4,1,2,5,7,3
Initial memory contents

Frame no:	1	2	3	4
Content::	4	8	3	5

Apply OPT policy.

b. Assume that there are two process in the system say A and B.

Page reference string: A4, A8, B2, B3, A3, B1, A5, A7, B3, B8, B7, B1, A4, A3
Initial memory contents

Frame no:	1	2	3	4
Content:	A5	A2	B3	B4

Apply local OPT policy with equal frame allocation

c. Same as **b.** with global OPT policy

d. Same as **b.** with local LRU policy

e. For the cases **b.**, **c.** and **d.**, calculate the average access time if total of page swap in and swap out time is 40 msec, memory access time is 10 μsec and an average of 100 accesses are made to a page once it is called.

12. A working set demand paging algorithm with window size = 4, (that is, the last 4 page references of each process constitute the windows of those processes) is to be applied on the reference string

time:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ref:	A3	B1	A1	A2	B3	B4	A3	A4	B2	B4	A1	A3	A1	B2

Assuming that this working set size correctly reflects program locality characteristics,

a. write the working sets
- **i.** for t=9
- **ii.** for t=14

b. Under the conditions and at the time instants in part **a.** should the operating system increase the multiprogramming degree, or decrease it to improve CPU utilization? Why?

c. Answer part **b.** if real memory is extended to 7 frames.

13. In a demand paging system with equal local frame allocation policy, a process is allocated f frames. All these f frames were initially empty. The process has a page reference string length=s. There are exactly n distinct page numbers in this string. consider OPT, LRU, and FIFO page replacement algorithms.

a. If n<= f,

> **i.** What is the minimum number of page faults for this process with this reference string? With which page replacement policy(ies) can this minimum number of page faults be achieved?
>
> **ii.** What is the maximum number of page faults for this process with this reference string? Which page replacement policy(ies) may cause this many page faults?

b. If n>f, repeat **i.** and **ii.** of part **a.**.

14. Consider a demand paging system with 3 frames in memory. Page replacement policy is LRU, and a global replacement is done. However, it is given that each process needs at least one page in memory for execution. There are two processes, P1 and P2 executing in the system. Assume the following reference string is given.

> (1,4,r) (1,4,w) (1,2,r) (1,3,w) (2,1,r) (2,1,w) (2,2,r) (2,3,w) (1,4,r) (1,2,w) (1,3,r) (2,1,w)

where it means :

> (i,j,r): i'th process references page j for a read.
> (i,j,w): i'th process references page j for a write.

Find and indicate the page faults caused by this string, and after each page fault, show the page tables for P1 and P2, indicating the values of valid/invalid and the dirty bits. Assume both processes have 5 pages.

Initially assume that all the 3 frames are empty. Also assume that the processes have not completed their execution at the end of the given reference string.

Is this system trashing? If yes, what is the minimum number of frames this system should have to execute this reference string of P1 and P2 efficiently?

15. Consider a computer system which uses demand paging system. Assume the system also has associative registers to implement the page table. Given :

Associative register access time=100 nanoseconds
Main memory access time = 300 nanoseconds
Hit ratio in associative registers =80%.
Page fault ratio = 0.1%
Page-fault service time= 200 000 nanoseconds

a. What is the total memory access time if there is a hit in associative registers and no page fault?

b. What is the total memory access time if there is a miss in associative registers but no page fault?

c. What is the effective memory access time, no page fault?

d. What is the effective access time when page faults are considered?

16. In a system having 4 frames, consider the following page references of process A and B.

time:	1	2	3	4	5	6	7	8	9	10	11	12
ref:	A1	A2	B1	B2	A1	A3	B3	B1	A1	A3	B1	B4

Show the content of memory, and indicate whenever there is a page fault if

a. FIFO algorithm is used with global replacement
b. FIFO algorithm is used with local replacement with equal allocation
c. repeat **a.** with LRU
d. repeat **b.** with LRU

17. A working set demand paging algorithm with window size=4 (that is, last 4 page references of each process constitute the windows of those processes) is to be applied on the reference string

time:	1	2	3	4	5	6	7	8	9	10	11	12
ref:	A1	A2	B1	B2	A1	A3	B3	B1	A1	A3	B1	B4

a. write the working sets

i. for t=8, **ii.** for t=12.

b. if the system has 4 frames, decide on if the system is trashing or not for

i. t=8, and **ii.** t=12

18. Consider a computer system having virtual memory with paging. Assume that the system also has the associative registers. Let:

associative register access time=50 nanosec
memory access time=350 nanosec
hit ratio=90%
swap_in= 100000 msec
swap_out= 200000 msec

a. calculate effective memory access time if no page fault occurs

b. calculate effective access time if page-fault rate=0.001

c. calculate effective access time if dirty bit is used, and the probability for a page being dirty is %60

19. In a system using working set model for frame allocation, explain what happens if the window size is chosen

a. smaller than the proper size

b. larger than the proper size