**UNIT – III**

**Memory Management Part –II**

**Topics to be covered**

* Virtual Memory
* Demand Paging
* Page Replacement
* Allocation of Frames
* Thrashing

**VIRTUAL MEMORY**

Virtual memory is a technique that allows the execution of process that may not be completely in memory. The main visible advantage of this scheme is that programs can be larger than physical memory.

* Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available ( Fig ).
* Following are the situations, when entire program is not required to load fully.
  1. User written error handling routines are used only when an error occurs in the data or computation.
  2. Certain options and features of a program may be used rarely.
  3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
* The ability to execute a program that is only partially in memory would counter many benefits.

1. Less number of I/O would be needed to load or swap each user program into memory.
2. A program would no longer be constrained by the amount of physical memory that is available.
3. Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

### Demand Paging

A demand paging is similar to a paging system with swapping . When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally

* Similar to paging system with swapping (diagram below)
* Page is needed Þ reference to it
* invalid reference Þ abort
* not-in-memory Þ bring to memory
* **Lazy swapper** – never swaps a page into memory unless page will be needed
* Swapper that deals with pages is a

**pager**

### Basic Concepts

### With swapping, pager guesses which pages will be used before swapping out again

### Instead, pager brings in only those pages into memory

### How to determine that set of pages?

### Need new MMU functionality to implement demand paging

### If pages needed are already memory resident

### No difference from non demand-paging

### If page needed and not memory resident

### Need to detect and load the page into memory from storage

### 4 Without changing program behavior

### 4 Without programmer needing to change code

### With each page table entry a valid–invalid bit is associated (v Þ in-memory – memory resident, i Þ not-in-memory)

### Initially valid–invalid bit is set to i on all entries

### Example of a page table snapshot:

### During MMU address translation, if valid–invalid bit in page table entry is i Þ page fault

### Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

### Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

### Page Fault

### If there is a reference to a page, first reference to that page will trap to operating system:

### page fault

### Operating system looks at another table to decide:

### Invalid reference Þ abort

### Just not in memory

### Find free frame

### Swap page into frame via scheduled disk operation

### Reset tables to indicate page now in memory Set validation bit = v

### Restart the instruction that caused the page fault

### Steps in Handling a Page Fault

1. We check an internal table for this process to determine whether the reference was a valid or invalid memory access.
2. If the reference was invalid, we terminate the process. If .it was valid, but we have not yet brought in that page, we now page in the latter.
3. We find a free frame.
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been memory.

Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

The page replacement is used to make the frame free if they are not in used. If no frame is free then other process is called in.

### Page Replacement Algorithm

There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults. The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data.

1. For a given page size we need to consider only the page number, not the entire address.
2. if we have a reference to a page p, then any immediately following references to page p will never cause a page fault. Page p will be in memory after the

### first reference; the immediately following references will not fault.

Eg:- consider the address sequence

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102,

0103, 0104, 0104, 0101, 0609, 0102, 0105

and reduce to 1, 4, 1, 6,1, 6, 1, 6, 1, 6, 1

To determine the number of page faults for a particular reference string and page replacement algorithm, we also need to know the number of page frames available. As the number of frames available increase, the number of page faults will decrease.

### FIFO Algorithm

The simplest page-replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory.

The first three references (7, 0, 1) cause page faults, and are brought into these empty eg. 7, 0, 1, 2, 0, 3,

0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1 and consider 3 frames. This replacement means that the next reference to 0 will fault. Page 1 is then replaced by page 0.

### Optimal Algorithm

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN. It is simply

Replace the page that will not be used for the longest period of time.

Now consider the same string with 3 empty frames.

The reference to page 2 replaces page 7, because 7 will not be used until reference 15, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. Optimal replacement is much better than a FIFO.

The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

### LRU Algorithm

The FIFO algorithm uses the time when a page was brought into memory; the OPT algorithm uses the time when a page is to be used. In LRU replace the page that has not been used for the longest period of time.

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

Let SR be the reverse of a reference string S, then the page-fault rate for the OPT algorithm on S is the same as the page-fault rate for the OPT algorithm on SR.

### LRU Approximation Algorithms

Some systems provide no hardware support, and other page-replacement algorithm. Many systems provide some help, however, in the form of a reference bit. The reference bit for a page is set, by the hardware, whenever that page is referenced. Reference bits are associated with each entry in the page table Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware.

### Additional-Reference-Bits Algorithm

The operating system shifts the reference bit for each page into the high-order or of its 5-bit byte, shifting the other bits right 1 bit, discarding the low-order bit.

These 5-bit shift registers contain the history of page use for the last eight time periods. If the shift register contains 00000000, then the page has not been

used for eight time periods; a page that is used at least once each period would have a shift register value of 11111111.

### Second-Chance Algorithm

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page gets a second chance, its reference bit is cleared and its arrival e is reset to the current time.

### Enhanced Second-Chance Algorithm

The second-chance algorithm described above can be enhanced by considering troth the reference bit and the modify bit as an ordered pair.

* 1. (0,0) neither recently used nor modified best page to replace.
  2. (0,1) not recently used but modified not quite as good, because the page will need to be written out before replacement.
  3. (1,0) recently used but clean probably will be used again soon.
  4. (1,1) recently used and modified probably will be used again, and write out will be needed before replacing it

### Counting Algorithms

There are many other algorithms that can be used for page replacement.

* **LFU Algorithm:** The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.
* **MFU Algorithm:** The most frequently used (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

### Page Buffering Algorithm

When a page fault occurs, a victim frame is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out.

This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written out, its frame is added to the free-frame pool.

When the FIFO replacement algorithm mistakenly replaces a page mistakenly replaces a page that is still in active use, that page is quickly retrieved from the free-frame buffer, and no I/O is necessary. The free-frame buffer provides protection against the relatively poor, , but simple, FIFO replacement algorithm

**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

* + But then process execution time can vary greatly
  + But greater throughput so more common

Local replacement – each process selects from only its own set of allocated frames

* + More consistent per-process performance
  + But possibly underutilized memory

**Thrashing**

If a process does not have “enough” pages, the page-fault rate is very high

* + Page fault to get page
  + Replace existing frame
  + But quickly need replaced frame back
  + This leads to:

1. Low CPU utilization
2. Operating system thinking that it needs to increase the degree of multiprogramming
3. Another process added to the system

* **Thrashing** º a process is busy swapping pages in and out