Operating Systems

Dr. J.M. Khalifeh

Virtual Memory

Operating Systems

Dr. J.M. Khalifeh

Outline

- Background
- Demand Paging
- Page Replacement

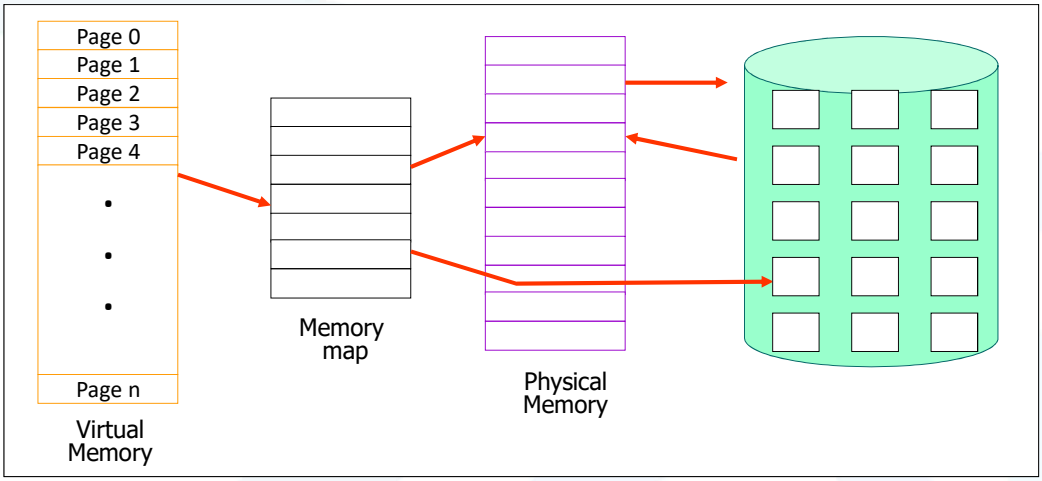
Operating Systems
Dr. J.M. Khalifeh

Remember!

- *Virtual memory* is a technique that allows the execution of processes that may not be completely in memory. The main visible advantage of this scheme is that programs can be larger than physical memory. The instructions being executed must be in physical memory (place the entire logical address space in physical)
- *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.

Operating Systems
Dr. J.M. Khalifeh

virtual memory larger than physical memory



Background

- *Overlays and dynamic loading* can help ease this restriction, but it limits the size of a program to the size of physical memory.
- The ability to execute a program that is only partially in memory would have many benefits:
  - Users would be able to write programs for an extremely large *virtual* address space, simplifying the programming task.
  - More programs could be run at the same time, with a corresponding increase in CPU utilization and throughput, but with no increase in response time or turnaround time.
  - Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

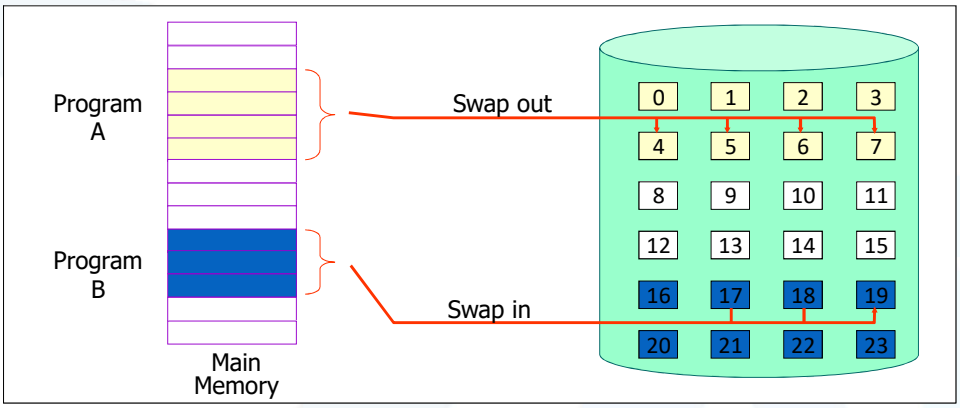
Virtual memory is commonly implemented:
- By demand paging
- In a segmentation system
- Demand segmentation

Demand Paging

- ***A demand-paging system is similar to a paging system with swapping***

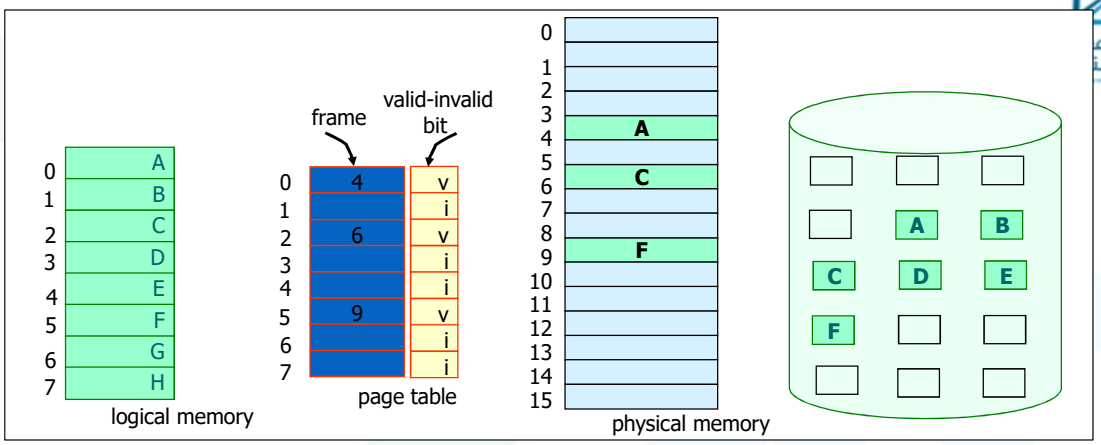
1. Processes reside on secondary memory (which is usually a disk).
2. When we want to execute a process, we swap it into memory.
3. Rather than swapping the entire process into memory, however, we use a lazy swapper. *(A lazy swapper never swaps a page into memory unless that page will be needed.)*
4. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
5. 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 anyway, decreasing the swap time and the amount of physical memory needed.
6. Here we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk.

Transfer of a paged memory to contiguous disk space.



A *swapper* manipulates entire processes, whereas a *pager* is concerned with the individual pages of a process. We shall thus use the term *pager,* rather than *swapper,* in connection with demand paging.

valid-invalid bit

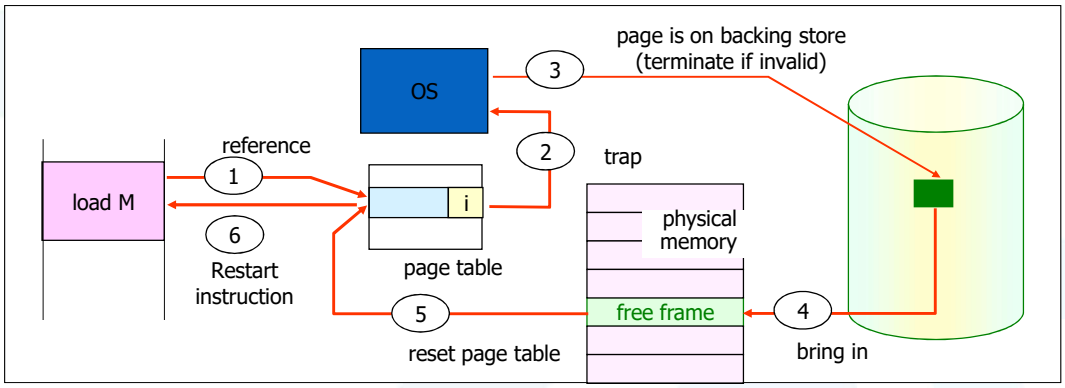


"valid" indicates that the associated page is both legal and in memory.
"invalid" indicates that the page either is not valid (that is, not in the logical address space of the process), or is valid but is currently on the disk.
The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simply marked invalid, or contains the ad dress of the Page on disk.

Page Fault Handling

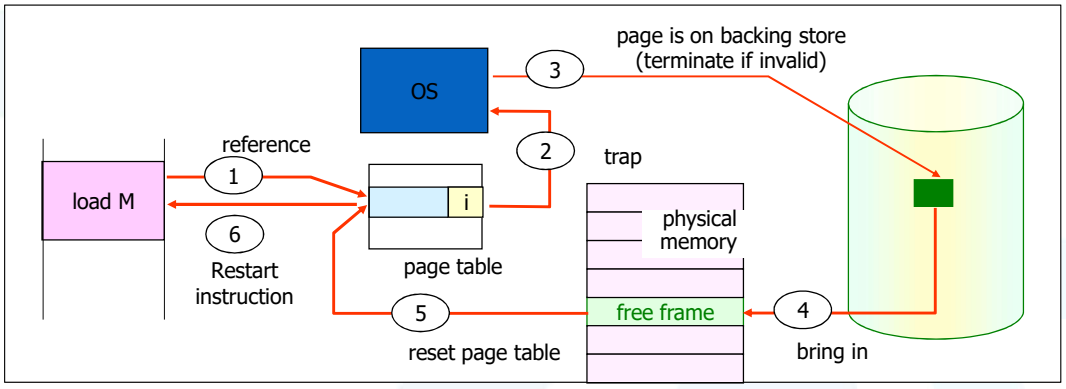
- We check an internal table, usually kept with the process control block (Is the reference a valid or invalid memory access.
- If the reference was invalid, we terminate the process.
- If it was valid, but we have not yet brought in that page, we now page in the latter.
- We find a free frame.
- We schedule a disk operation to read the desired page into the newly allocated frame.
- 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.
- 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 in memory.

Steps in handling a page fault



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 in memory.

Steps in handling a page fault



What happens if the process tries to use a page that was not brought into memory?

Access to a page marked invalid causes a page-fault trap.

This trap is the result of the operating system's failure to bring the desired page into memory.

The hardware to support demand paging

The hardware to support demand paging is (the same as the hardware for paging and swapping):

- **Page table**: This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.
- **Secondary memory:** This memory holds those pages that are not present in main memory." The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known *as swap, space* or *backing store.*

In addition to this hardware support, considerable software is needed

page fault on the instruction fetch

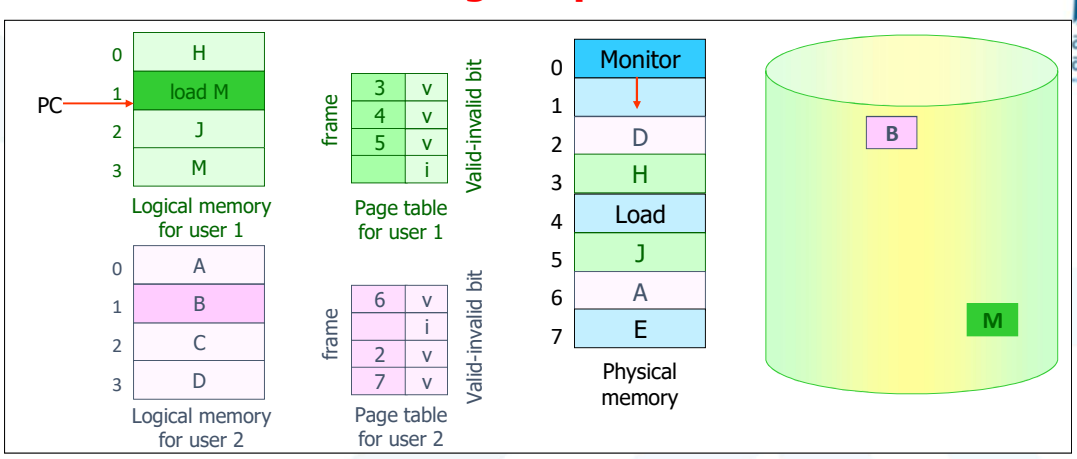
- If the page fault occurs on the instruction fetch, we can restart by fetching the instruction again.
- If a page fault occurs while we are fetching an operand, we must re-fetch the instruction, decode it again, and then fetch the operand.

An example: Consider a three-address instruction such as ADD the content of A to B placing the result in C. The steps to execute this instruction would be

1. Fetch and decode the instruction (ADD).
2. Fetch A.
3. Fetch B.
4. Add A and B.
5. Store the sum in C.

If we faulted when we tried to store in C (because C is in a page not currently in memory), we would have to get the desired page, bring it in, correct the page table, and restart the instruction.

Need For Page Replacement



What happens if there is no free frame?

- Page replacement is the solution: find some page in memory, but not really in use, swap it out.
- Same page may be brought into memory several times.

Page Replacement

Page replacement takes the following approach.

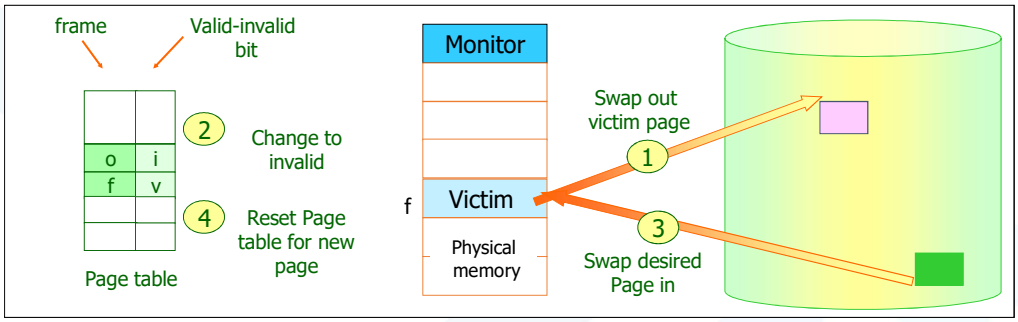
If no frame is free, We find one that is not currently being used and free it.

The freed frame can now be used to hold the page for which the process faulted.

The page-fault service routine is now modified to include page replacement

Notice that, if no frames are free, ***two page transfers*** (one out and one in) are required. This situation effectively doubles the page-fault service time and will increase the effective access time accordingly.

Page Replacement



1. Find the location of the desired page on disk.
2. 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.
3. Read the desired page into the (newly) free frame. Update the page and frame tables.
4. Restart the process.

Page Replacement

1. Page replacement is basic to demand paging. ***It completes the separation between logical memory and physical memory***.
2. With this mechanism, a very large virtual memory can be provided for programmers on a smaller physical memory. All of the pages of a process still must be in physical memory, however. With demand paging, the size of the logical address space is no longer constrained by physical memory.
3. If a page that has been modified is to be replaced, its contents are copied to the disk. A later reference to that page will cause a page fault. At that time, the page will be brought back into memory, perhaps replacing some other page in the process.
4. If we have multiple processes in memory, we must decide how many frames to allocate to each process. Further, when page replacement is required, we must select the frames that are to be replaced. Designing appropriate algorithms to solve these problems is an important task, because disk I/0 is so expensive.

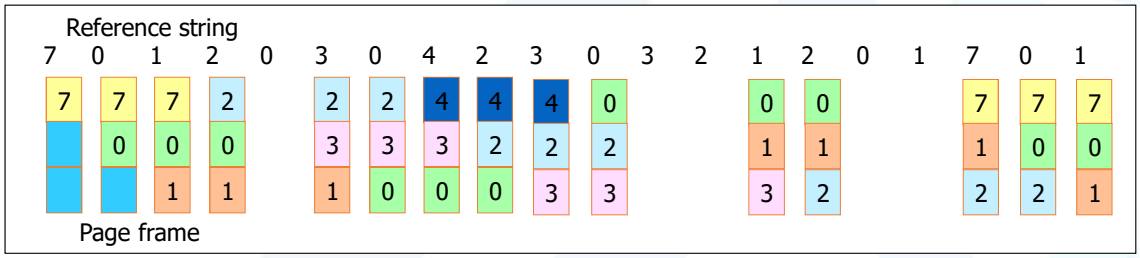
Page Replacement Algorithms

- FIFO algorithm
- Optimal algorithm
- LRU algorithm

The FIFO Algorithm

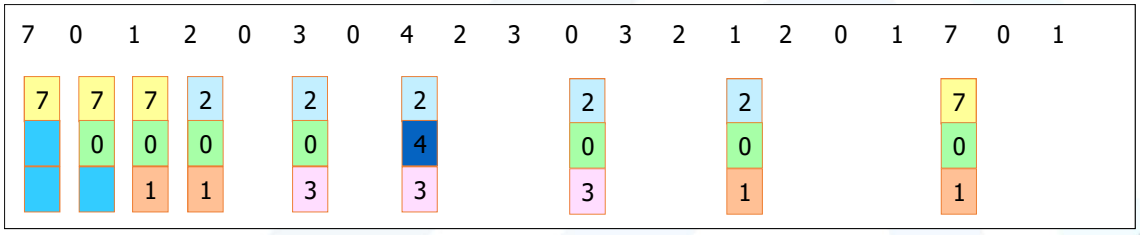
1. Associates with each page the time when that page was brought into memory.
2. When a page must be replaced, the oldest page is chosen.

Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.



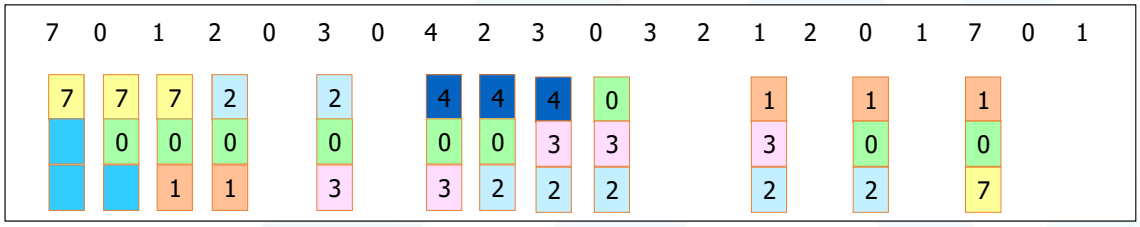
Optimal Algorithm

- Has the lowest page-fault rate of all algorithms
- It replaces the page that *will not be used* for the longest period of time.
- difficult to implement, because it requires future knowledge
- used mainly for comparison studies



LRU Algorithm (Least Recently Used)

- An approximation of optimal algorithm: looking backward, rather than forward.
- It replaces the page that has not been used for the longest period of time.
- It is often used, and is considered as quite good.



Exercise

4 3 1 6 7 3 2 1 0 4 3 5 7 6 3 1 2 4 5 4

4 3 1 6 7 3 2 1 0 4 3 5 7 6 3 1 2 4 5 4

- For the series of pages shown, compare the three algorithms you studied for a three-frame memory and then for a four-frame memory.