Memory Management

- There is a concept of **memory hierarchy**, in which, a computer consists of
  - a few megabytes of very fast, expensive, volatile **cache memory**
  - a few gigabytes of medium-speed, medium-priced, volatile **main memory**
  - a few terabytes of slow, cheap, nonvolatile **disk storage**
- The part of the OS that manages the memory hierarchy is called the **memory manager**. It manages memory efficiently;
  - keeps track of which parts of memory are in use
  - allocates memory to processes when they need it
  - deallocate the memory when the processes are done.

Memory Management

Swapping

- A memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes.
- It is used to improve main memory utilization.
- In secondary memory, the place where the swapped-out process is stored is called **swap space**.
- The thing to remember is that swapping is used only when **data is not present in RAM.**
- Idle processes are mostly stored on the disk, so they do not take up any memory when they are not running.

Memory Management

- **Swap-out** is removing a process from RAM and adding it to the hard disk.
- **Swap-in** is removing a program from a hard disk and putting it back into the main memory or RAM.

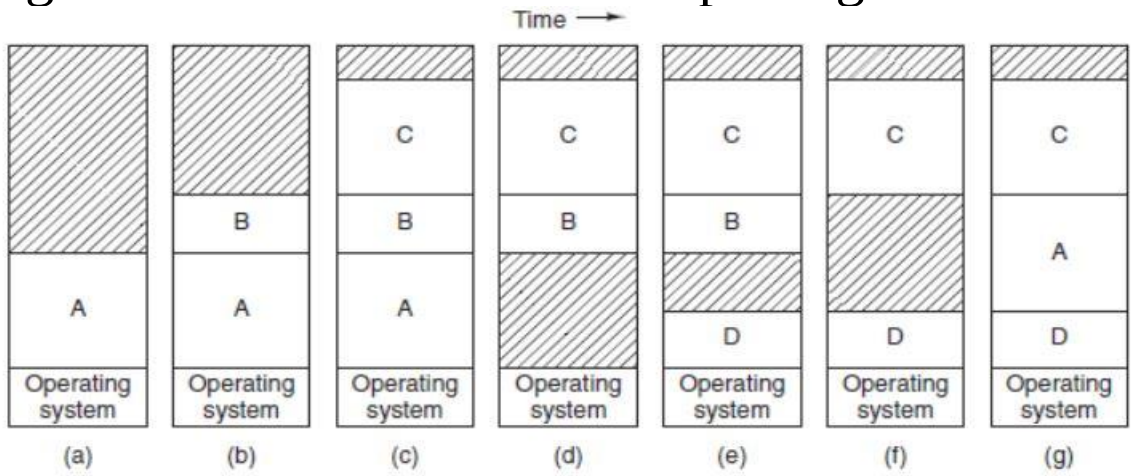


Figure 3-4. Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

- Since, A is now at different location, addresses contained in it must be relocated, either by software when it is swapped in or by hardware during program execution.

Memory Management

- When swapping creates multiple holes in memory, it is possible to combine them all into a big one by moving all the processes downward as far as possible. This is called **memory compaction**. But it requires a lot of CPU time.
- If the processes are created with a fixed size that never changes, then the OS allocates exactly what is needed, no more no less.
- If a process's data segments start growing, the process can be allocated to an adjacent hole to grow. But if there is no adjacent hole, the process have to move to a larger enough hole that can hold the entire process.
- Otherwise, one or more processes have to be swapped-out to create a larger enough hole. If the space is full, the process will have to be suspended and wait.

Memory Management

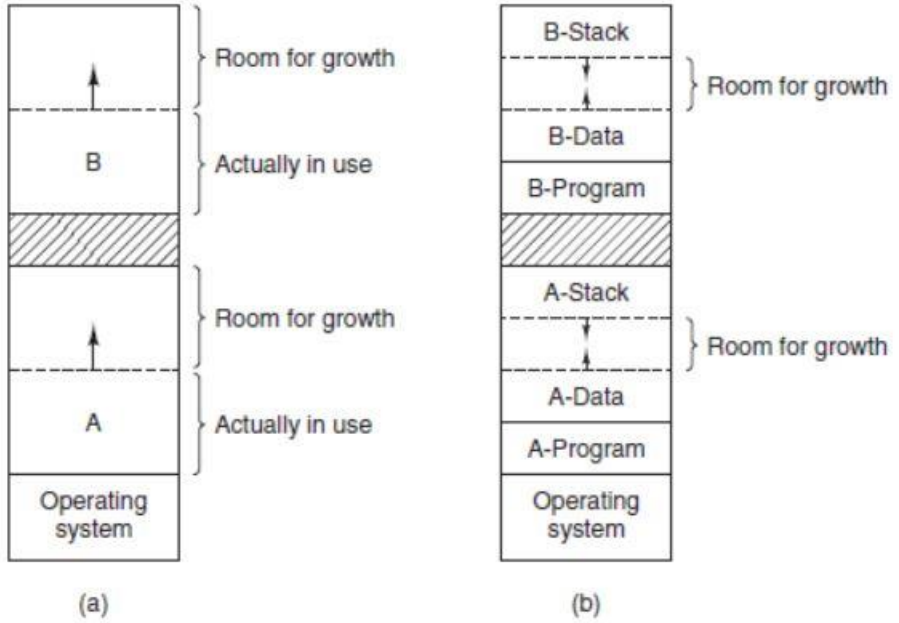


Figure 3-5. (a) Allocating space for a growing data segment. (b) Allocating space for a growing stack and a growing data segment.

Memory Management

Managing Free Memory

Using Bitmap

- Here memory is divided into allocation units; a few words size or several kilobytes.
- There is a bit in the bit map corresponding to each allocation unit. The bit is 0 if it is free, and 1 if it is occupied.
- It is a slow process.

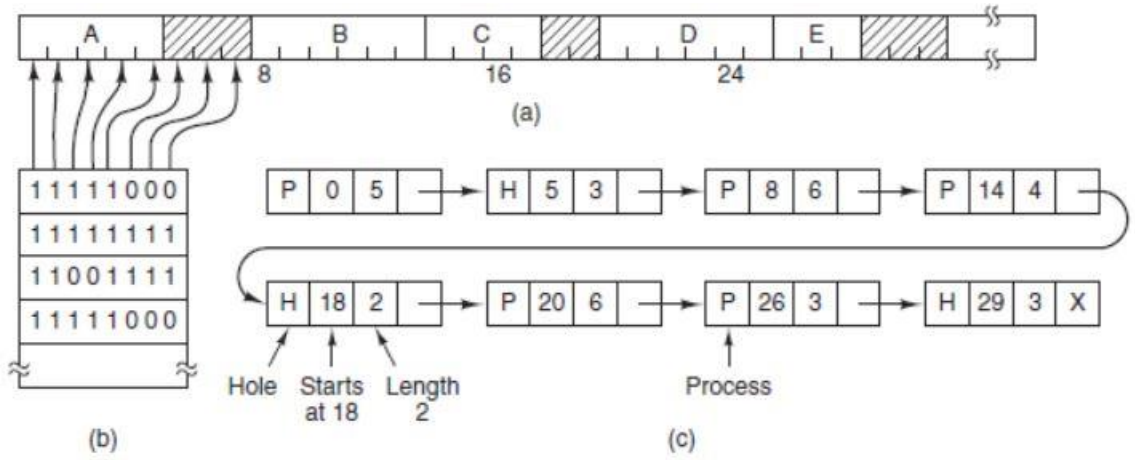


Figure 3-6. (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

Memory Management

Using Linked List

- When the processes and holes are kept in a single list sorted by address, several algorithms can be applied to allocate memory.
- **First Fit Algorithm**
  - The memory manager scans the list until it finds the **first hole that is big enough** to hold the process.
  - The hole is then broken up into two pieces; one for the process, and the other portion remains unused, except the exact fit.
  - It is a **fast algorithm** as it searches as little as possible.
  - **Disadvantage**: The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.

Memory Management

- **Next Fit Algorithm**
  - A variation of the first fit.
  - It works the same way as first fit does, except that, it keeps track of the place when it finds a suitable hole for a process.
  - The next time it looks for a hole, it starts searching from the place it left last time, instead of always at beginning.
  - Slightly worse performance than first fit.
- **Best fit Algorithm**
  - A well-known and widely used algorithm.
  - It searches the entire list, from beginning to end, and finds the smallest hole that is best fit to allocate the process.
  - It tries to find the best match that is close to the actual size needed.
  - It is slower than first fit algorithm.
  - It leaves the memory with tiny and useless holes.

Memory Management

- **Worst Fit Algorithm**
  - It always finds the largest available hole so that the new hole will be big enough to be useful.
  - This algorithm is not a good choice.
  - **Two separate lists can be maintained for processes and holes.** The additional complexity that slows down this procedure is deallocation of memory; since a free segment/block/portion of memory has to be removed from the process list and inserted into the hole list.
- **Quick Fit Algorithm**
  - It maintains separate lists for some of the more common sizes requested.
  - It might have a table of *n* entries. First entry is a pointer to the head of a list of 4 KB holes, second entry is a pointer to a list of 8 KB holes and so on.
  - So, to find a hole of the required size is extremely fast.
  - Disadvantage: When a process terminates or swapped out, finding its neighbor for merging quite expensive.

Memory Management

- **Virtual Memory**
  - Today, most personal computers (PCs) come with at least 8 GB (gigabytes) of RAM. But, sometimes, this is not enough to run several programs at one time. This is where virtual memory comes in.
  - Each program has its own address space, which is broken up into chunks called **pages**. Each page is contiguous range of addresses.
  - These pages are mapped onto physical memory, i.e., RAM. Not all pages have to be in physical memory at the same time to run the program.
  - When the RAM needs free space, virtual memory frees up RAM by swapping out data that has not been used recently to a storage device, such as a hard drive.
  - It improves performance in multiprogramming system.

Memory Management

- **Paging**
  - Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
  - The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.
  - One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
  - Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Memory Management

- A very **simple example of how virtual memory address mapping works**:
  - A computer that generates 16-bit addresses, from 0 up to 64K − 1. These are the virtual addresses. This computer, however, has only 32 KB, of physical memory. So although 64-KB programs can be written, they cannot be loaded into memory in their entirety and run. A complete copy of a program's core image, up to 64 KB, must be present on the disk, however, so that pieces can be brought in as needed.
  - The virtual address space consists of fixed-size units called **pages**. The corresponding units in the physical memory are called **page frames**. The pages and page frames are generally the same size.
  - With 64 KB of virtual address space and 32 KB of physical memory, there are 16 virtual pages and 8 page frames.

Memory Management

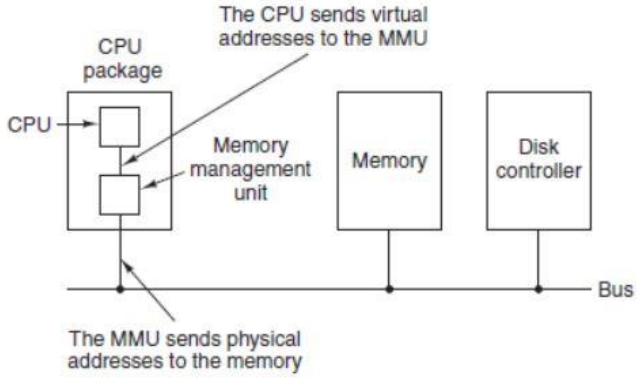


Figure 3-8. The position and function of the MMU. Here the MMU is shown as being a part of the CPU chip because it commonly is nowadays. However, logically it could be a separate chip and was years ago.

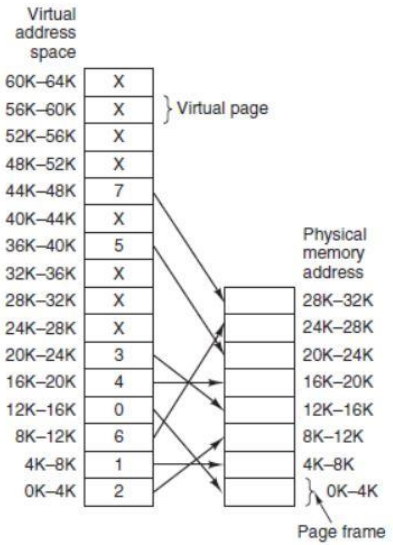


Figure 3-9. The relation between virtual addresses and physical memory addresses is given by the **page table**. Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K–8K really means 4096–8191 and 8K to 12K means 8192–12287.

- So to execute the instruction MOVE REG,100, the address 100 is sent to the MMU. The MMU checks and finds that the virtual address falls in the page 2.

Memory Management

Page Replacement Algorithm

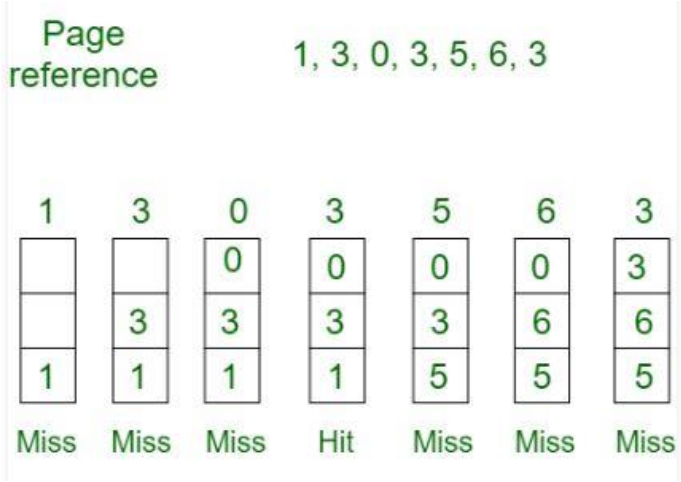
- A page replacement algorithm decides which page needs to be replaced when a new page comes in. It is used in an OS where paging is used in memory management.
- A **page fault** happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.
- Since actual physical memory is much smaller than virtual memory, page fault occurs.
- When page fault occurs, the OS might have to replace one of the existing pages with the newly needed page.
- Different page replacement algorithms suggest different ways to decide which page to replace. The main goal of all algorithms is **to reduce the number of page faults**.

Memory Management

- **First in First Out Page Replacement Algorithm**
  - ➢This is the simplest page replacement algorithm.
  - ➢In this algorithm, the OS keeps track of all pages currently in the memory in a queue, with the most recent arrival in the tail and the least recent arrival at the head of the queue.
  - ➢When a page needs to be replaced, the page in the front/head of the queue is selected for removal.
  - ➢The new page is added to the tail of the queue.
  - ➢For example, Consider page references

    1, 3, 0, 3, 5, 6, 3 with 3 page frames. So

    how many page faults occur here?



Memory Management

- **Second Chance Page Replacement Algorithm**
  - A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect.
  - There is a R bit associated with each page. R is set whenever the page is referenced (read or written).
  - If it is 0, the page is both old and unused, so it is replaced immediately.
  - If the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. The R bit is cleared also. Then the search continues.
  - If all the pages have been referenced, second chance degenerates into pure FIFO.
  - If all the pages have their R bits set, then the operating system moves the pages one by one to the end of the list, clearing the R bit each time it appends a page to the end of the list. Eventually, it comes back to page A, which now has its R bit cleared. At this point A is evicted.
  - Inefficient, because it is constantly moving pages around on the list.

Memory Management

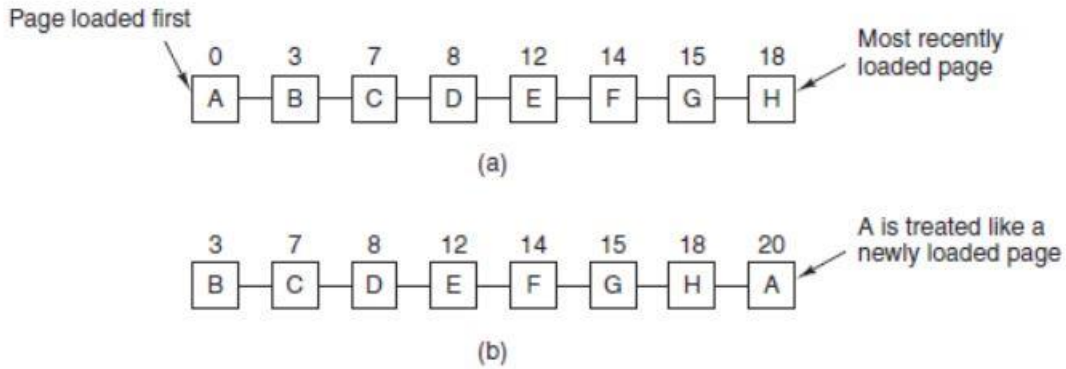


Figure 3-15. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and *A* has its *R* bit set. The numbers above the pages are their load times.

Memory Management

- **Clock Page Replacement Algorithm**
  - A better (than second chance algorithm) approach is to keep all the page frames on a circular list in the form of a clock. The hand points to the oldest page.
  - When a page fault occurs, the page being pointed to by the hand is inspected.
  - If its R bit is 0, the page is replaced/removed, the new page is inserted into the clock in its place, and the hand is advanced one position.
  - If R is 1, it is cleared and the hand is advanced to the next page.
  - This process is repeated until a page is found with R = 0.

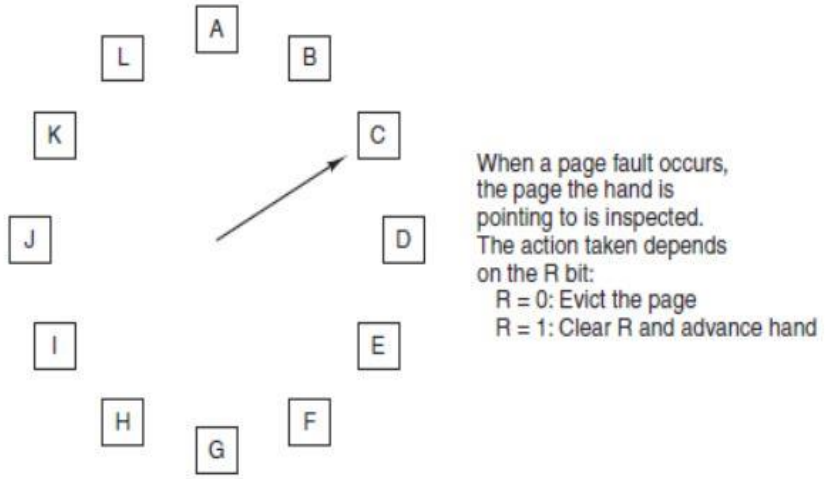
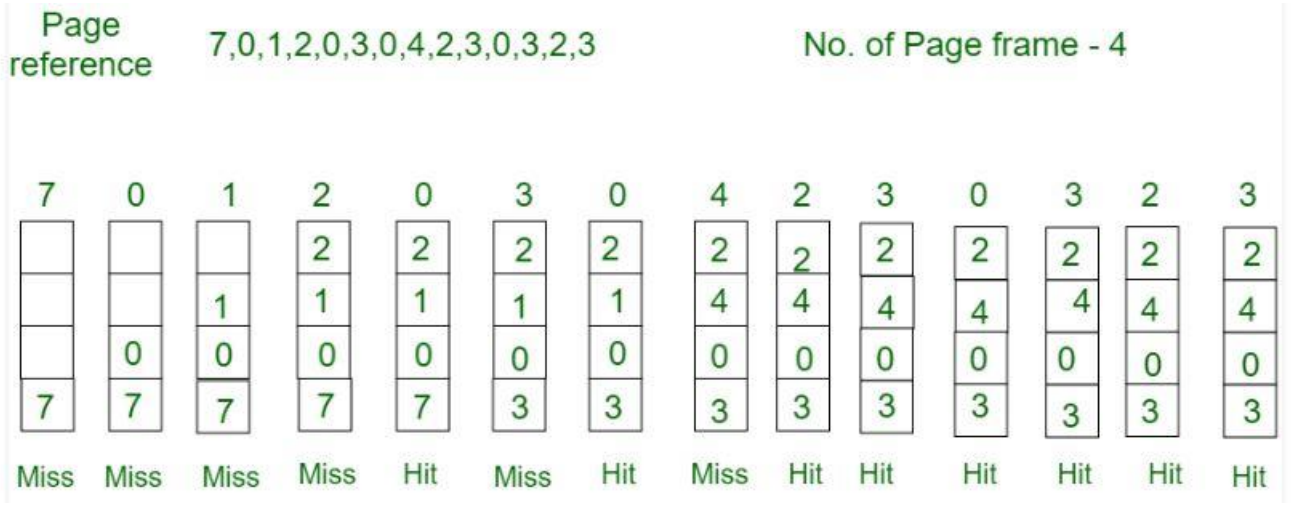


Figure 3-16. The clock page replacement algorithm.

Memory Management

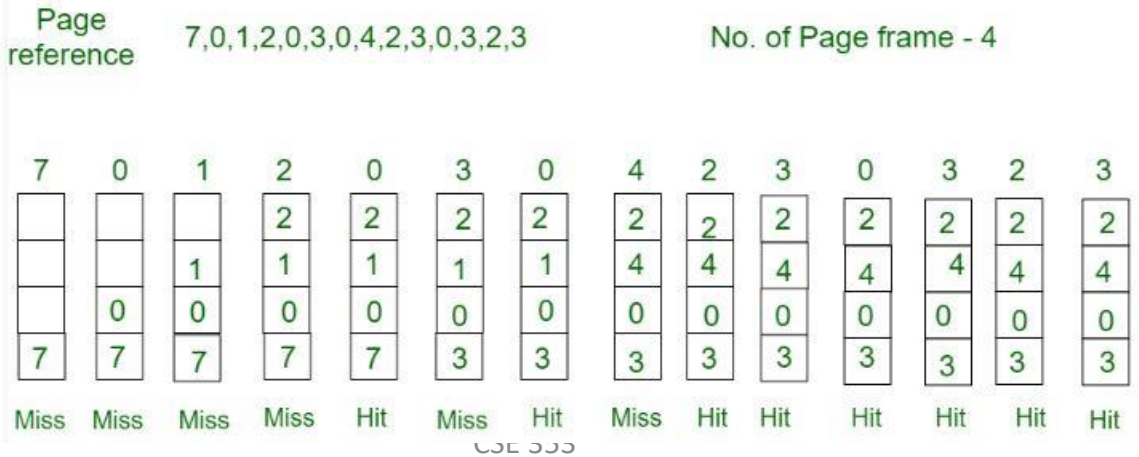
- **Optimal Page Replacement Algorithm**
  - In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
  - This is the best page replacement algorithm but impossible to implement actually.
  - At the time of the page fault, the OS has no way of knowing when each of the pages will be referenced next.
  - Still by running a program on a simulator and keeping track of all page references, it is possible to implement this algorithm on the second run by using the page reference information collected during the first run.
  - Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. So how many page faults occur here?

Memory Management



Memory Management

- **Least Recently Used Page Replacement Algorithm**
  - In this algorithm, page will be replaced which is least recently used.
  - The algorithm is based on the observation that the pages that have not been used for ages will probably remain unused for a long time.
  - So, when a page fault occurs, the page is replaced which has been unused for the longest time.
  - Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. So how many page faults occur here?



Memory Management

Working Set Page Replacement Algorithm

- Processes are started up with none of their pages in memory.
- As soon as the CPU tries to fetch the first instruction, a page fault occurs.
- Then the OS brings the page containing that instruction in the memory.
- After a while, the process has most of the pages it needs and settles down to run with relatively few page faults.
- This strategy is called **demand paging**, as the pages are loaded on demand only, not in advance.
- The set of pages that a process is currently using is called its **working set**.
- If the memory is too small to hold the entire working set, the process will cause many page faults and run slowly. A program causing page faults every few instructions is said to be **thrashing**.

Memory Management

- The basic idea of the working set page replacement algorithm is to find a page that is not in the working set and remove it.
- Here, pages that are in the memory are considered for removal.
- Two information are mostly required here; the time the page was last used and The R, reference bit.
- When a page fault occurs, the following happens.
  - the R bit is examined. If it is 1, the current virtual time is written into the Time of last use field in the page table, indicating that the page was in use at the time the fault occurred.
  - Since the page has been referenced during the current clock tick, it is clearly in the working set and is not a candidate for removal.
  - If R is 0, the page has not been referenced during the current clock tick and may be a candidate for removal.
  - To see whether or not it should be removed, its age (the current virtual time minus its Time of last use) is computed and compared to τ .
  - If the age is greater than τ , the page is no longer in the working set and the new page replaces it. The scan continues updating the remaining entries.
  - However, if R is 0 but the age is less than or equal to τ , the page is still in the working set. The page is temporarily spared, but the page with the greatest age (smallest value of Time of last use) is noted.
  - If the entire table is scanned without
  - finding a candidate to evict, that means that all pages are in the working set. In that case, if one or more pages with R = 0 were found, the one with the greatest age is removed.

Memory Management

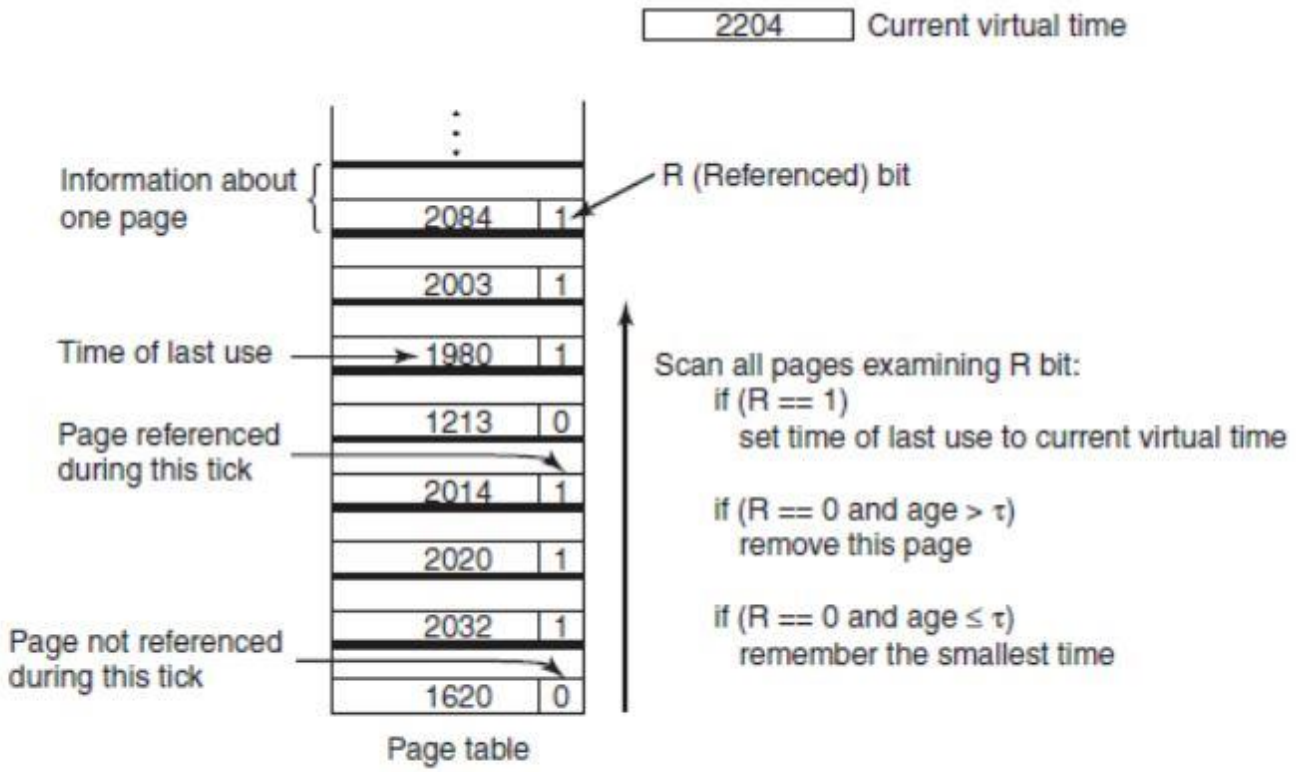


Figure 3-19. The working set algorithm.

Memory Management

Segmentation

- Another memory management scheme.
- The division of a program or application into segments as part of a virtual memory scheme.
- Once the process of segmentation occurs, the entire process can be loaded into different areas in memory instead of one contiguous space. Loading smaller segments of the process into memory allows the physical memory to be used more efficiently.
- Operating system doesn't care about the user's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.
- It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.
- A segment can contain a procedure, a stack, array, or collection variables. But it does not contain a mixture of different type.
- Segmentation allows the programmer to view memory as consisting of multiple address space or segments.
- Each segment consists of a linear sequence of addresses, starting at 0 and going up to some maximum level.
- The length of each segment can be different.
- Different segments can expand or shrink independently without affecting each other.

Memory Management

- Each segment has a segment table that contains information about segment; base address of the segment and limit, i.e., lenth of the segment.
- A logical address is generated by the CPU which consists of two things: Segment number and Offset.
- The segment number which logical address contains is mapped into the segment table. Now the limit of the segment is compared with the offset. If the respective limit is more than the offset, then there will be valid address otherwise the address will be invalid. When the address in invalid, then it will show an error.
- It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast.
- Segmentation also facilitates sharing procedures or data between several processes.

Memory Management

Paging	Segmentation
A page is always of **fixed size units.**	A segment is of **variable size**.
Each page has a **single address**.	A segment has a **segment number and offset**.
For the paging **operating system is responsible**.	For segmentation **compiler is responsible**.
The size of the page is decided or specified by the **hardware**.	The size of the segment is specified by the **user**.
Paging leads to **internal fragmentation.**	The segmentation leads to **external fragmentation**.
In paging, **both main memory and secondary memory** are divided into **equal fixed-size partitions**.	In segmentation, **secondary and main memory** are **not** divided into partitions of **equal size**.
Hard to allow sharing of procedures between processes.	**Facilitates sharing** of procedures between the processes.

Memory Management

Segmentation with Paging

- Segmentation can be combined with Paging to get the best features out of both the techniques.
- In this approach, the main memory is divided into variable size segments which are further divided into fixed size pages.
- Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.
- It reduces memory usage.