RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Operating System**

Subject Code: **IT-501**

Semester: **5th**

Department of Information Technology
Class Notes

Unit 4

Concept of virtual memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- 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.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- 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.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –

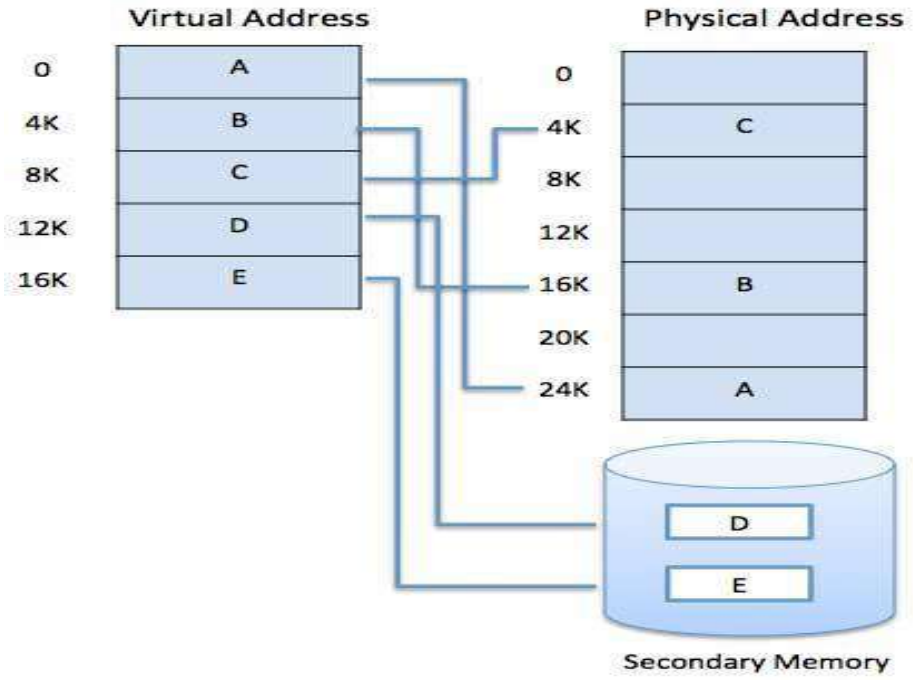


Figure 4.1 MMU

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Cache memory organization

A cache memory is a fast random access memory where the computer hardware stores copies of information currently used by programs (data and instructions), loaded from the main memory. The cache has a significantly shorter access time than the main memory due to the applied faster but more expensive

implementation technology. The cache has a limited volume that also results from the properties of the applied technology. If information fetched to the cache memory is used again, the access time to it will be much shorter than in the case if this information were stored in the main memory and the program will execute faster.

Time efficiency of using cache memories results from the locality of access to data that is observed during program execution.

Time locality: Time locality consists in a tendency to use many times the same instructions and data in programs during neighboring time intervals.

Space locality: Space locality is a tendency to store instructions and data used in a program in short distances of time under neighboring addresses in the main memory.

- Due to these localities, the information loaded to the cache memory is used several times and the execution time of programs is much reduced.
- Cache can be implemented as a multi-level memory. Contemporary computers usually have two levels of caches. In older computer models, a cache memory was installed outside a processor (in separate integrated circuits than the processor itself).
- The access to it was organized over the processor external system bus. In today's computers, the first level of the cache memory is installed in the same integrated circuit as the processor.
- It significantly speeds up processor's co-operation with the cache. Some microprocessors have the second level of cache memory placed also in the processor's integrated circuit.
- The volume of the first level cache memory is from several thousands to several tens of thousands of bytes. The second level cache memory has volume of several hundred thousand bytes.
- A cache memory is maintained by a special processor subsystem called cache controller.

Read implementation in cache memory on hit

If there is a cache memory in a computer system, then at each access to a main memory address in order to fetch data or instructions, processor hardware sends the address first to the cache memory. The cache control unit checks if the requested information resides in the cache. If so, we have a "hit" and the requested information is fetched from the cache. The actions concerned with a read with a hit are shown in the figure below.

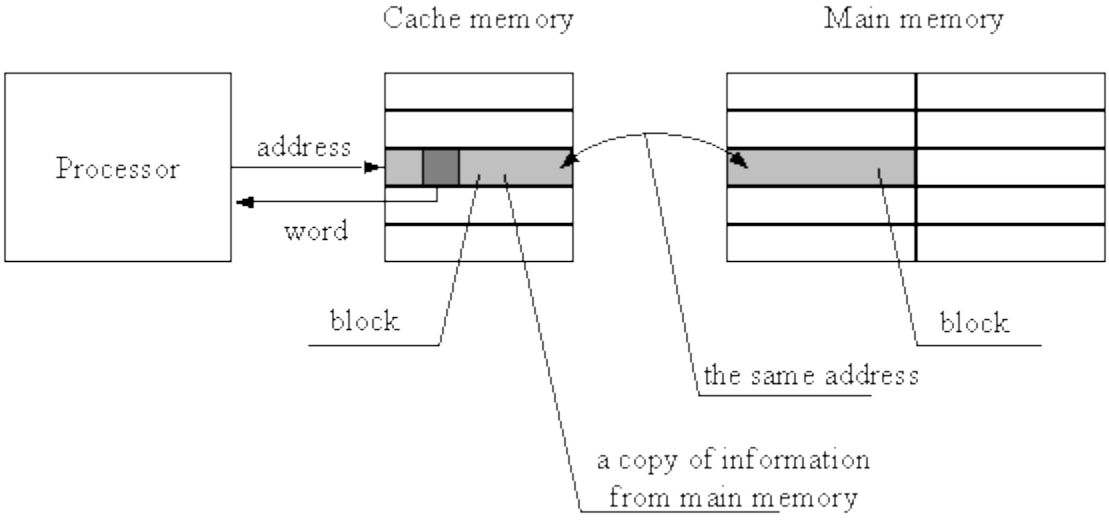


Figure 4.2 Read implementation in cache memory on hit

Read implementation in cache memory on miss

If the requested information does not reside in the cache, we have a "miss" and the necessary information is fetched from the main memory to the cache and to the requesting processor unit. The information is not copied in the cache as single words but as a larger block of a fixed volume. Together with information block, a part of the address of the beginning of the block is always copied into the cache. This part of the address is next used at readout during identification of the proper information block. The actions executed in a cache memory on "miss" are shown below.

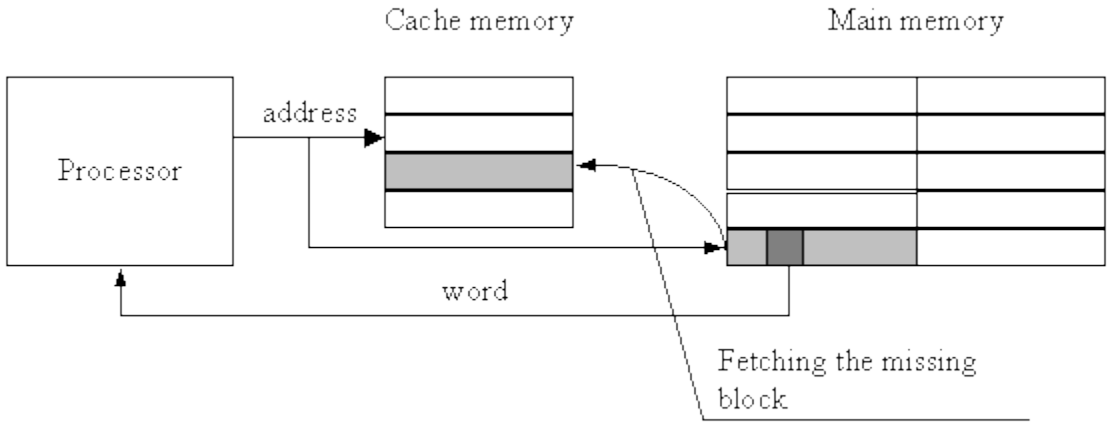


Figure 4.3 Read implementation in cache memory on miss

Memory updating methods after cache modification:

A cache memory contains copies of data stored in the main memory. When a change of data in a cache takes place (ex. a modification due to a processor write) the contents of the main memory and cache memory cells with the same address, are different. To eliminate this lack of data coherency two methods are applied:

- **Write through**, the new cache contents is written down to the main memory immediately after the write to the cache memory,
- **Write back,** the new cache contents are not written down to the main memory immediately after the change, but only when the given block of data is replaced by a new block fetched from the main memory or an upper level cache. After a data write to the cache, only state bits are changed in the modified block, indicating that the block has been modified (a dirty block).

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program’s pages as they are referenced.

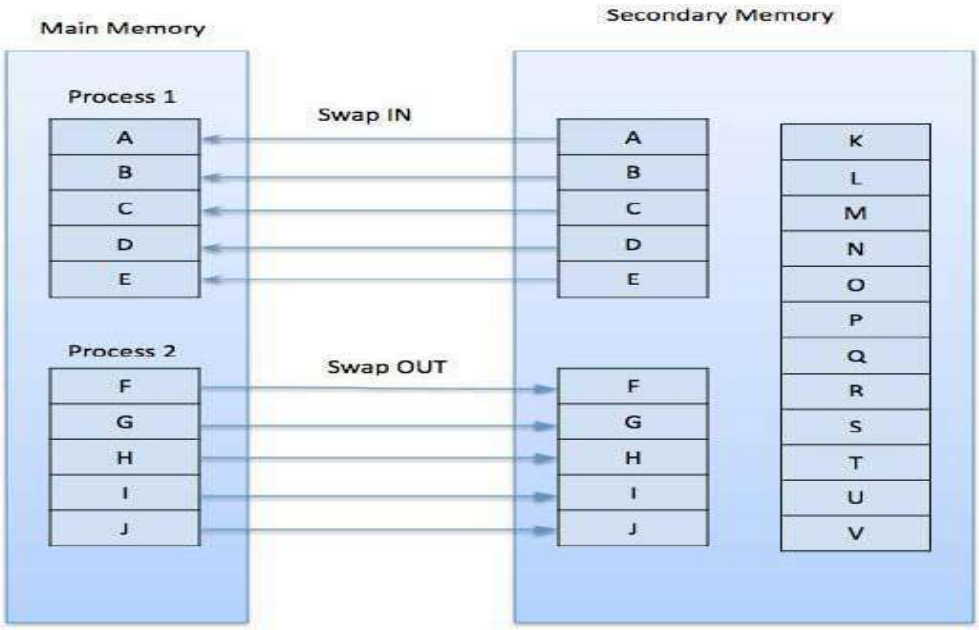


Figure 4.4 Demand Paging

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. 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,

Reference String

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, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.

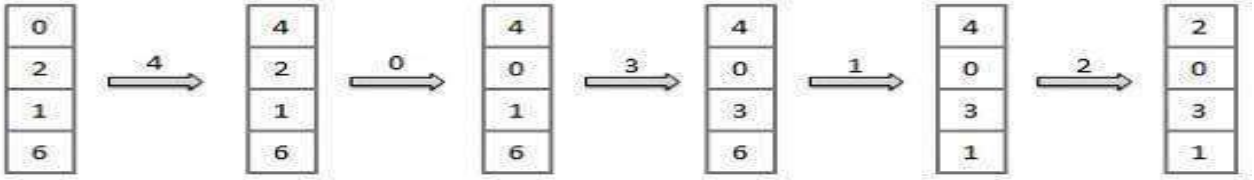
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First in First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



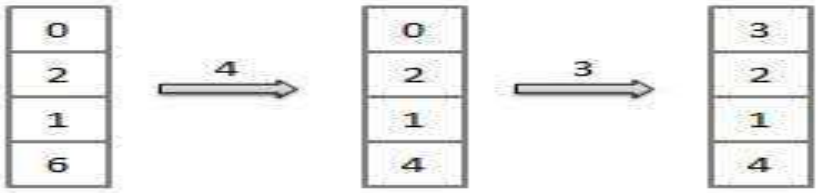
Fault Rate = 9 / 12 = 0.75

Optimal Page 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.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



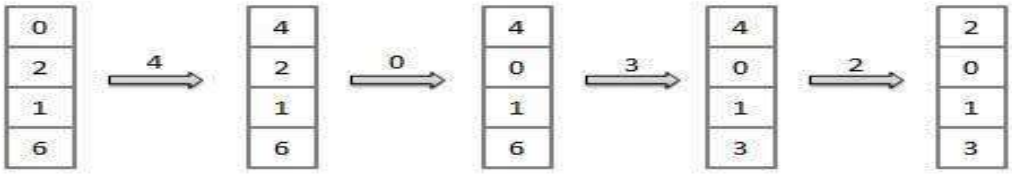
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used (LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- 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.

Most frequently Used (MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Maintain 3 free frames at all times
- Consider a machine where all memory-reference instructions have only 1 memory address → need 2 frames of memory
    - Now consider indirect modes of addressing
    - Potentially every page in virtual memory could be touched, and the entire virtual memory must be in physical memory
    - Place a limit the levels of indirection
- The minimum number of frames per process is defined by the computer architecture
- The maximum number of frames is defined by the amount of available physical memory

Allocation Algorithms

- m frames, n processes
- Equal allocation: give each process m/n frames
- Alternative is to recognize that various processes will need different amounts of memory
    - Consider
        - 1k frame size
        - 62 free frames
        - student process requiring: 10k
        - interactive database requiring: 127k

It makes no sense to give each process 31 frames the student process needs no more than 10 so the additional 21 frames are wasted

- proportional allocation:
    - m frames available
    - Size of virtual memory for process pi is si

- S = Sigma si
- ai = (si/S) * m

Student process gets 4 frames = (10/137)*62
Database gets 57 frames = (127/137)*62

Global vs. Local Allocation

- **Global**: one process can select a replacement frame from the set of all frames (i.e., one process can take a frame from another or itself) (e.g., high priority processes can take the frames of low priority processes)
- **Local**: each process can only select from its own set of allocated frames
- local page replacement is more predictable; depends on no external factors
- A process which uses global page replacement cannot predict the page fault rate; may execute in 0.5 seconds once and 10.3 on another run
- Overall, global replacement results in greater system throughput

Thrashing

- **simple thrashing**: 1 process of 2 pages only allocated 1 frame
- High page activity is called thrashing
- A process is thrashing if it spends more time paging than executing

Scenario:

- The process scheduler sees that CPU utilization is low.
- So we increase the degree of multiprogramming by introducing a new process into the system.
- One process now needs more frames.
- It starts faulting and takes away frames from other processes. (i.e., global-page replacement).
- These processes need those pages and thus they start to fault, taking frames from other processes.
- These faulting processes must use the paging device to swap pages in and out.
- As they queue up on the paging device, the ready-queue empties.
- However, as processes wait on the paging device, CPU utilization decreases.
- The process scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming, ad infinitum.
- The result is thrashing, page-fault rates increase tremendously, effective memory access time increases, no work is getting done, because all the processes are spending their time paging.
- We can limit the effects of thrashing by using a local replacement algorithm (or priority replacement algorithm)
    - But this only partially solves the problem
      If one process starts thrashing, it cannot steal frames of another process and cause the later to thrash.
    - However, the thrashing processes will be in the paging device queue which will increase the time for a page fault to be serviced and, therefore, the effective access time will increase even for those processes not thrashing.
- To really prevent thrashing, we must provide processes with as many frames as they need.
        - But how do we know how many frames a process "needs"?
        - Look at how many frames a process actually "uses".

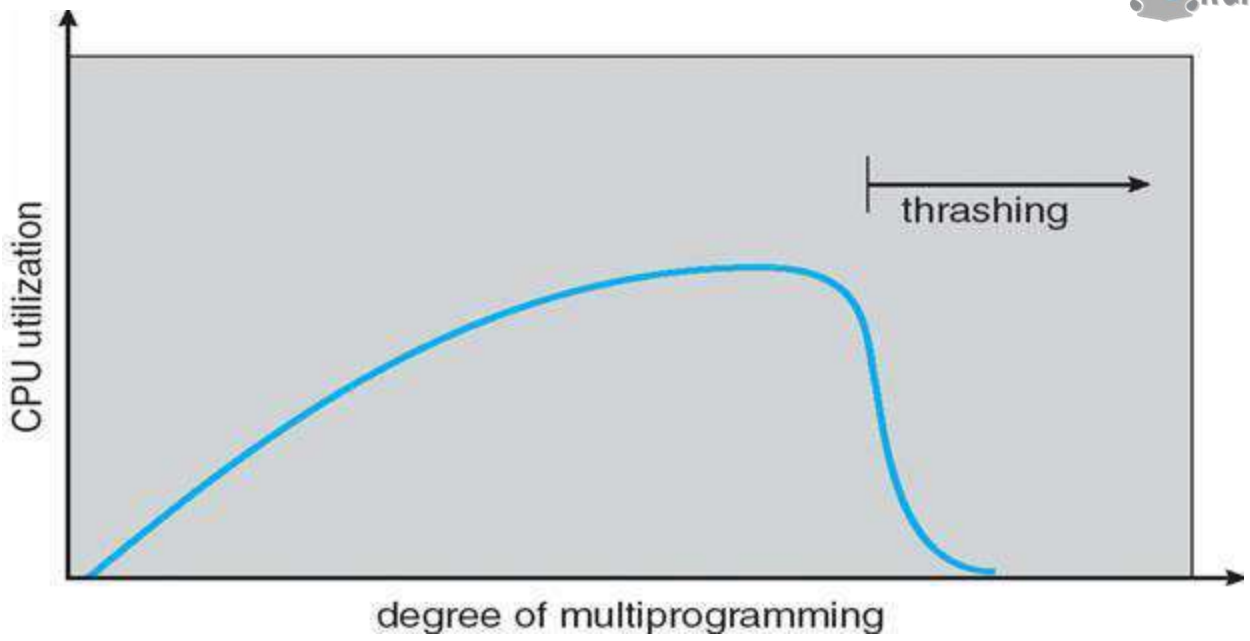


Figure 4.5 Thrashing

Demand Segmentation: Same idea as demand paging applied to segments.

- If a segment is loaded, base and limit are stored in the STE and the valid bit is set in the PTE.
- The PTE is accessed for each memory reference (not really, TLB).
- If the segment is not loaded, the valid bit is unset. The base and limit as well as the disk address of the segment is stored in the OS table.
- A reference to a non-loaded segment generates a segment fault (analogous to page fault).
- To load a segment, we must solve both the placement question and the replacement question (for demand paging, there is no placement question).

The following table compares demand paging with demand segmentation.

Consideration	Demand Paging	Demand Segmentation
Programmer aware	No	Yes
How many addr spaces	1	Many
VA size > PA size	Yes	Yes
Protect individual procedures separately	No	Yes
Accommodate elements with changing sizes	No	Yes
Ease user sharing	No	Yes
Why invented	let the VA size exceed the PA size	Sharing, Protection, independent addr spaces
Internal fragmentation	Yes	No, in principle
External fragmentation	No	Yes
Placement question	No	Yes

Replacement question	Yes	Yes

Role of OS in Security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If an unauthorized user runs a computer program, then he/she may cause severe damage to computer or data stored in it. So a computer system protected against unauthorized access, malicious access to system memory, viruses, worms etc. We are going to discuss following topics.

- Authentication
- One Time passwords
- Program Threats
- System Threats
- Computer Security Classifications

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system, which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username/Password** −User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it used again. One-time password implemented in various ways

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding few alphabets randomly chosen.
- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id, which generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it known as Program Threats. One of the common examples of program threat is a program installed in a computer, which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded

in a program. As user accesses the program, the virus starts embedded in other files/ programs and can make system unusable for user

System Threats

System threats refer to misuse of system services and network connections to put user in trouble. System threats used to launch program threats on a complete network called as program attack. System threats create such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

- **Worm** – Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms' processes can even shut down an entire network.
- **Port Scanning** – Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** – Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

Computer Security Classifications

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria, there are four security classifications in computer systems: A, B, C, and D. this is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level uses formal design specifications and verification techniques. Grants a high degree of assurance of process security
2	**Type B** Provides mandatory protection system have all the properties of a class C2 system. Attaches a sensitivity label to each object It is of three types. **B1** – Maintains the security label of each object in the system. Label is used for making decisions to access control. **B2** – Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. **B3** – Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.
3	**Type C** Provides protection and user accountability using audit capabilities It is of two types. **C1** – Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly Cl class. **C2** – Adds an individual-level access control to the capabilities of a Cl level system.
4	**Type D**

	Lowest level Minimum- protection MS-DOS, Window 3.1 fall in this category

Virus in details:

- A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc.
- Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures (such as the boot block.)
- Viruses are delivered to systems in a virus dropper, usually some form of a Trojan horse, and usually via e-mail or unsafe downloads.

Figure 4.6 shows typical operation of a boot sector virus:

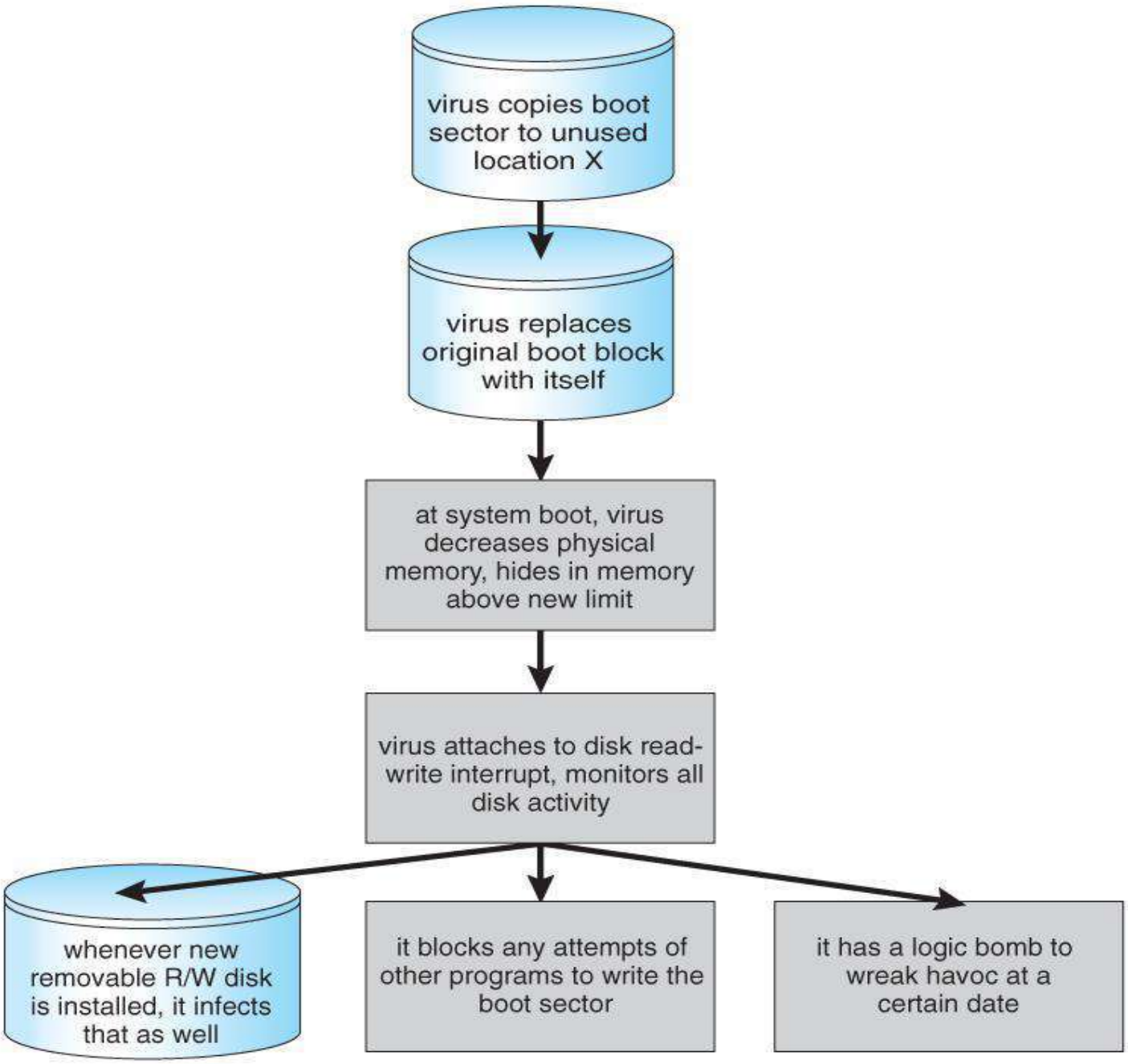


Figure 4.6 a boot-sector computer virus.

Security Techniques

Design Principles Saltzer and Schroeder (1975) identified a core set of principles to operating system security design:

Least privilege: Every object (users and their processes) should work within a minimal set of privileges; access rights should be obtained by explicit request, and the default level of access should be "none".

Economy of mechanisms: Security mechanisms should be as small and simple as possible, aiding in their verification. This implies that they should be integral to an operating system's design, and not an afterthought.

Acceptability: Security mechanisms must at the same time be robust yet non-intrusive. An intrusive mechanism is likely to be counter-productive and avoided by users, if possible.

Complete: Mechanisms must be pervasive and access control checked during all operations — including the tasks of backup and maintenance.

Open design: An operating system's security should not remain secret, nor be provided by stealth. Open mechanisms are subject to scrutiny, review, and continued refinement.

Security breaches: The OS must protect itself from security breaches, such as runaway processes (denial of service), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.

Stack and Buffer Overflow: This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow.

C program with buffer-overflow condition

Consider what happens in the following code, for example, if argv[1] exceeds 256 characters:

- The strcpy command will overflow the buffer, overwriting adjacent areas of memory.
- (The problem could be avoided using strncpy, with a limit of 255 characters copied plus room for the null byte.)

```
#include
#define BUFFER_SIZE 256
 int main( int argc, char * argv[ ] )
{
      char buffer[ BUFFER_SIZE ];
      if ( argc < 2 )
         return -1;
      else {
         strcpy( buffer, argv[ 1 ] );
         return 0;
      }
   }
```

System Protection:

Goals of Protection

- Obviously to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies,* which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

Principles of Protection

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Domain of Protection

- A computer can be viewed as a collection of *processes* and *objects* (both HW & SW).
- The ***need to know principle*** states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure

- A **protection domain** specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An **access right** is the ability to execute an operation on an object.
- A domain is defined as a set of < object, {access right set} > pairs, as shown below. Note that some domains may be disjoint while others overlap.

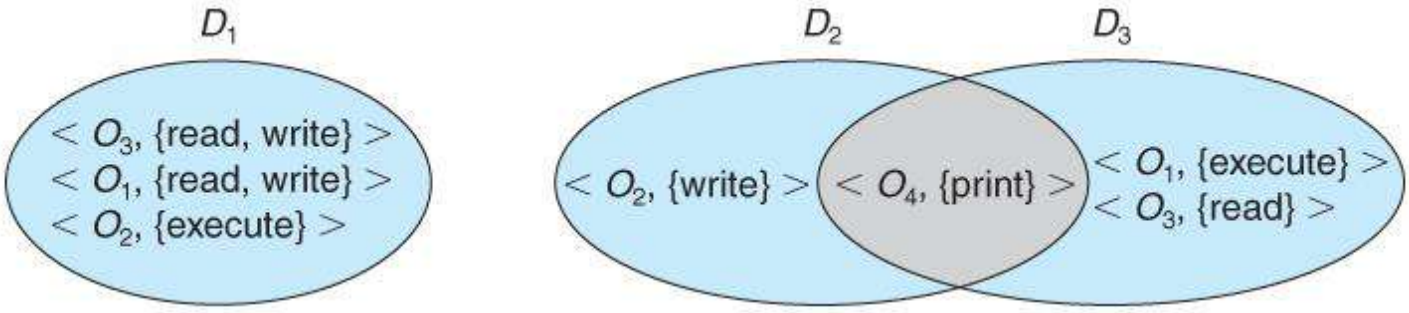


Figure 4.7 systems with three protection domains.

- The association between a process and a domain may be static or dynamic.
- If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.
- If the association is dynamic, then there needs to be a mechanism for domain switching.
- Domains may be realized in different fashions - as users, or as processes, or as procedures. E.g. if each user corresponds to a domain, then that domain defines the access of that user, and changing domains involves changing user ID.

Password Management

There are several forms of software used to help users or organizations better manage passwords:

- Intended for use by a single user:
  - Password manager software is used by individuals to organize and encrypt many personal passwords using a single login. This often involves the use of an encryption keyas well. Password managers are also referred to as **password wallets**.
- Intended for use by a multiple users/groups of users:

- Password synchronization software is used by organizations to arrange for different passwords, on different systems, to have the same value when they belong to the same person.
- Self-service password reset software enables users who forgot their password or triggered an intruder lockout to authenticate using another mechanism and resolve their own problem, without calling an IT help desk.
- Enterprise Single sign on software monitors applications launched by a user and automatically populates login IDs and passwords.
- Web single sign on software intercepts user access to web applications and either inserts authentication information into the HTTP(S) stream or redirects the user to a separate page, where the user is authenticated and directed back to the original URL.
- Privileged password management (used to secure access to shared, privileged accounts).

Privileged password management

Privileged password management is a type of password management used to secure the passwords for login IDs that have elevated security privileges. This is most often done by periodically changing every such password to a new, random value. Since users and automated software processes need these passwords to function, privileged password management systems must also store these passwords and provide various mechanisms to disclose these passwords in a secure and appropriate manner. Privileged password management is related to privileged identity management.

Examples of privileged passwords

There are three main types of privileged passwords. They are used to authenticate:

- **Local administrator accounts**
  On UNIX and Linux systems, the root user is a privileged login account. On Windows, the equivalent is administrator. In general, most operating systems, databases, applications and network devices include an administrative login, used to install software, configure the system, manage users, apply patches, etc. On some systems, different privileged functions are assigned to different users, which mean that there are more privileged login accounts, but each of them is less powerful.
- **Service accounts**
  On the Windows operating system, service programs execute in the context of either system (very privileged but has no password) or of a user account. When services run as a non-system user, the service control manager must provide a login ID and password to run the service program so service accounts have passwords. On UNIX and Linux systems, init and inetd can launch service programs as non-privileged users without knowing their passwords so services do not normally have passwords.
- **Connections by one application to another**
  Often, one application needs to be able to connect to another, to access a service. A common example of this pattern is when a web application must log into a database to retrieve some information. These inter-application connections normally require a login ID and password and this password.

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in