Unit-04

Virtual Memory

Unit-04/Lecture-01

Background

- Virtual memory is a technique that allows execution of processes that may not be completely in the physical memory.
- Virtual Memory gives the illusion of more physical memory than there really is (via demand paging)
- Virtual memory provides many benefits:
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Use of virtual memory is also justified for many reasons such as:
  - There may be some code (e.g. error code) that may never be executed.
  - Arrays, list and tables are often allocated more memory than they actually use.
  - Certain options and features of a program may be used rarely.
- Virtual memory can be implemented via:
  - Demand paging
  - **Demand segmentation**

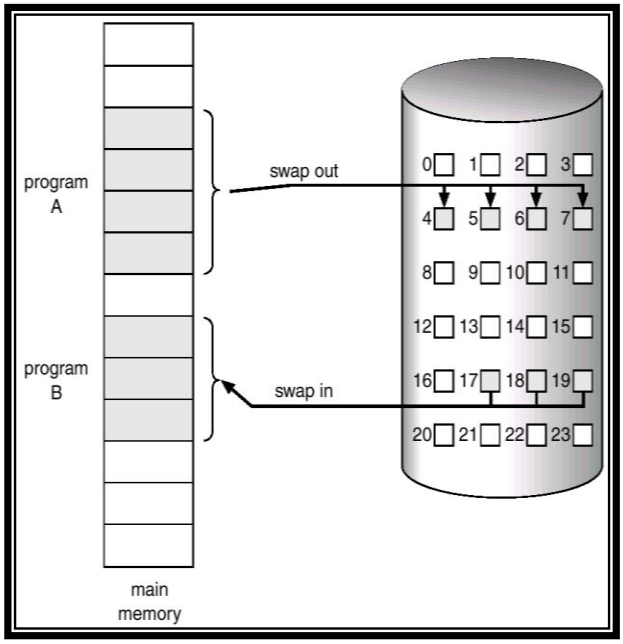
Program Execution in Virtual memory

- Operating system brings into main memory a few pieces of the program
  - Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state
- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Demand Paging

- When we want to execute a process, it is swapped into the memory. However, a pager (swapper) does not bring the whole process into the memory. Only those pages, which are needed, are brought into the memory. That is, bring a page into memory only when it is needed.
  - Often page 0 is loaded initially when a job is scheduled
- In Demand Paging a program's "working set" is kept in memory, reference outside WS causes corresponding code to be retrieved from disk ("page fault")
  - Provides the illusion of virtual memory
- Demand paging has the many benefits such as
  - Less I/O needed

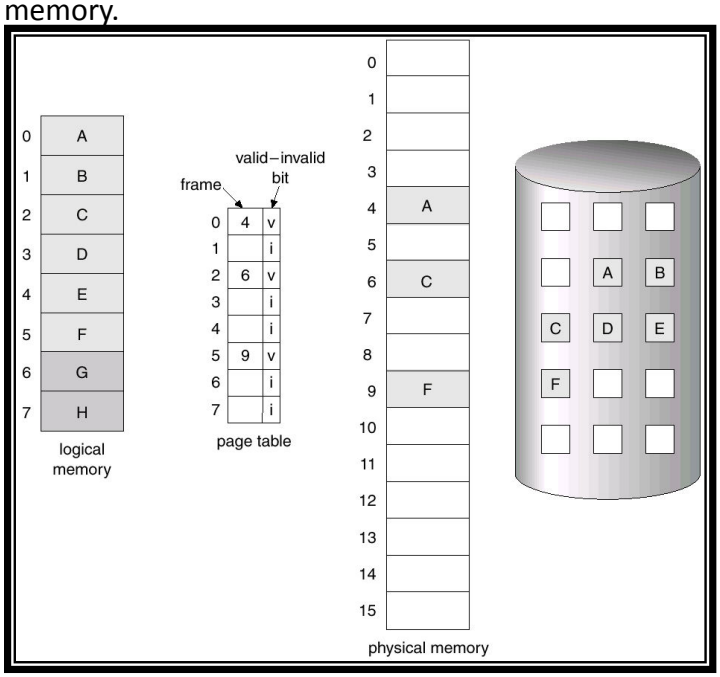
- Less memory needed
- Faster response
- More users



Transfer of a Paged Memory to Contiguous Disk Space

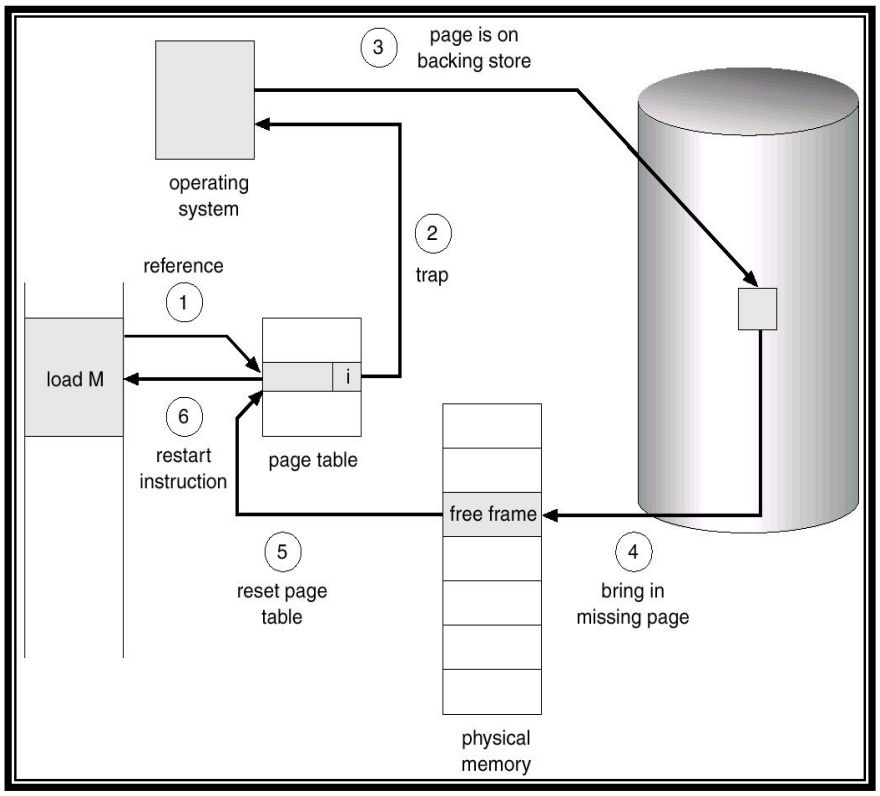
Valid-Invalid Bit

- For the above-mentioned scheme, we need some hardware support.
- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- During address translation, if valid–invalid bit in page table entry is 0 ⇒ **page fault**.
  - Page Fault - Interrupt that arises upon a reference to a page that is not in main memory.



Page Fault Handling

- If there is ever a reference to a page, first reference will trap to
  OS ⇒ page fault
- OS looks at an internal table, usually kept in process's PCB, to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- If the page is not in the memory, then
  - Get empty frame.
  - Swap page into frame.
  - Reset tables, validation bit = 1.
  - Restart instruction
- Note that when a page fault happens, the hardware traps to the OS. The OS reads the desire page and restart the process from the point where it was interrupted.



S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Discuss hardware support required to support demand paging	Dec 2013 Dec 2011	7 8
Q.2	What do you mean by page fault in demand paging	Dec 2012	7
Q.3	What is virtual memory	Dec 2012 Dec 2012	7 4

Unit-04/Lecture-02

What Happens If There is no Free Frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
  - Algorithm
  - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- Effective Access Time (EAT)

EAT = (1 – p) x memory access
+ p (page fault overhead
+ [swap page out]
+ swap page in
+ restart overhead)

Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Swap Page Time = 10 msec = 10,000 msec

EAT = (1 – p) x 1 + p (15000)
1 + 15000P (in msec)

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

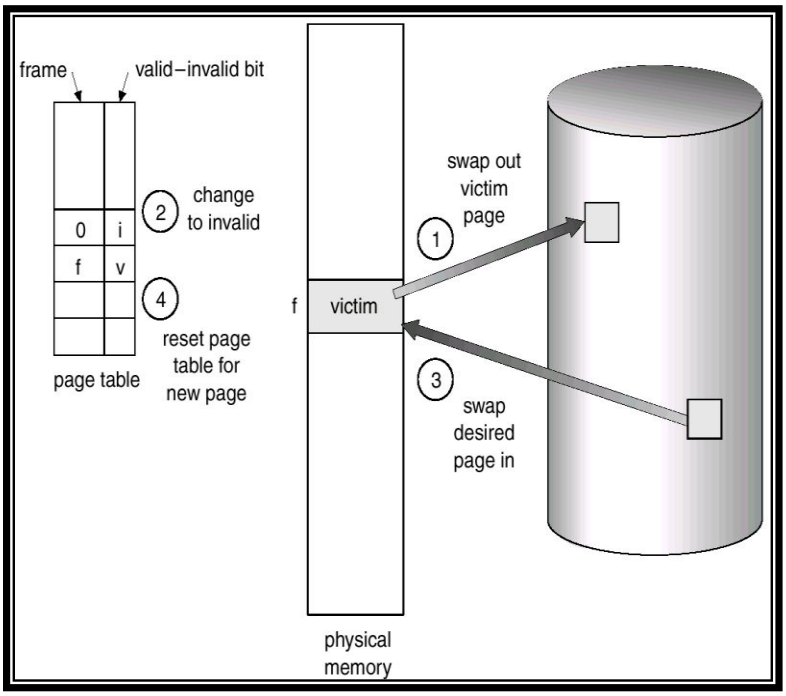
Basic Page Replacement Algorithm

Find the location of the desired page on disk.
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.

Read the desired page into the (newly) free frame. Update the page and frame tables.

Restart the process.



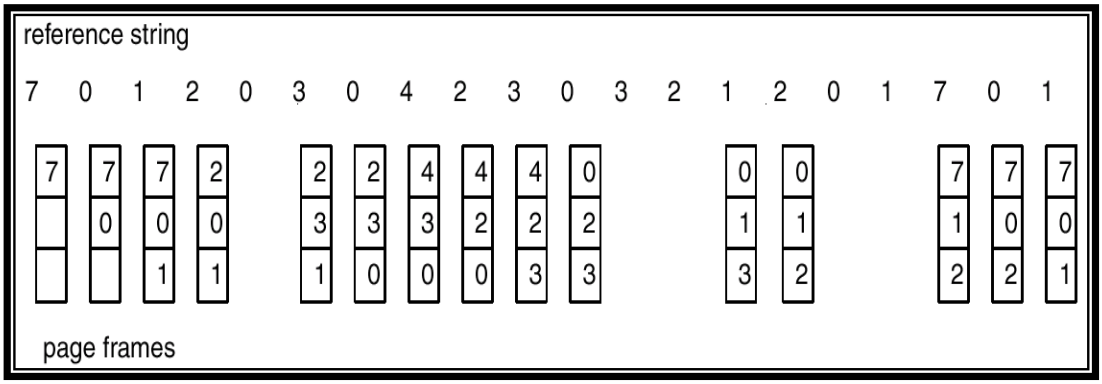
Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

FIFO Algorithm

- When a page must be replaced, the oldest page is chosen.
- FIFO page replacement algorithm associates with each page the time when the page was brought into the memory.
- A FIFO queue (instead of time) may also be used to implement this algorithm.

Example of FIFO Algorithm



Belady's Anomaly

- An interesting observation if FIFO algorithm is that increasing the page frames may not decrease the number of page faults.

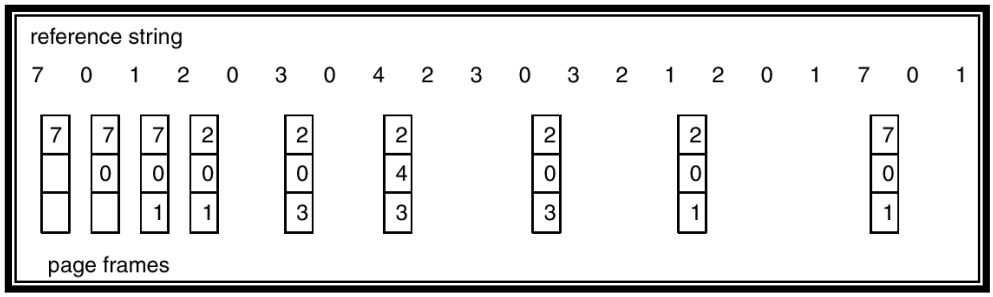
S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain Belady's anomaly	Dec 2012	4
Q.2	Consider following page reference string 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6 How many page faults occur for FIFO algorithm	Dec 2012	6

Unit-04/Lecture-03

Optimal Algorithm

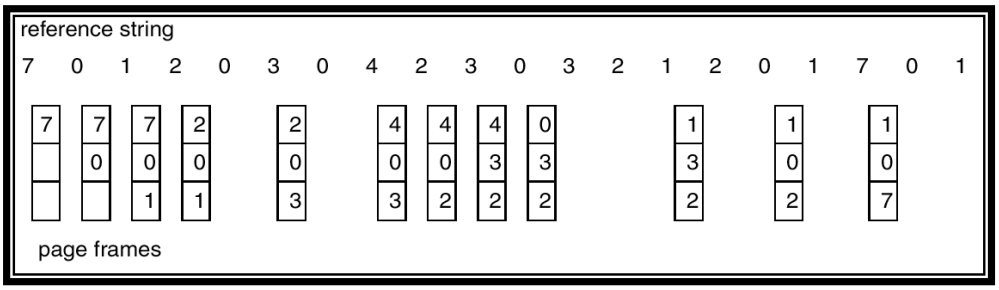
- According to optimal (OPT or MIN), the page that is to be replaced is the one that will not be used for the longest period of time.
- Difficult to implement because it requires future knowledge of the reference string.

Example of OPT Algorithm

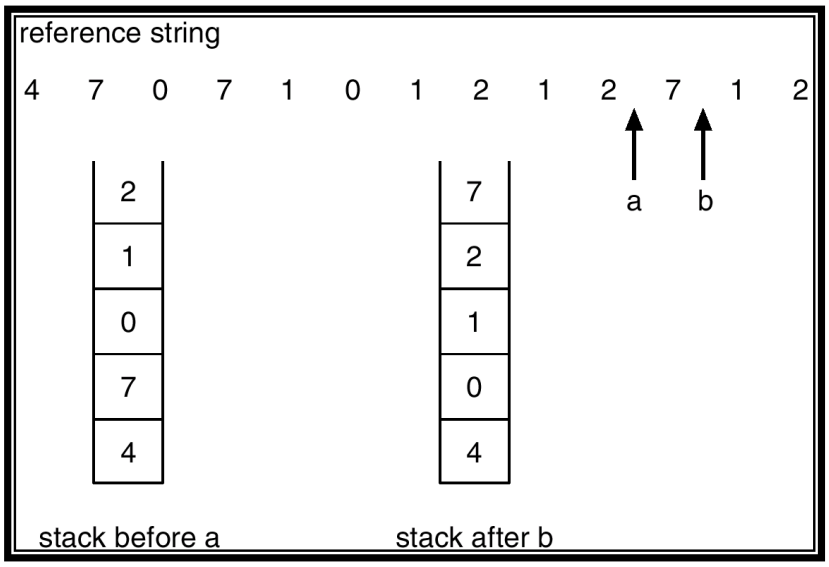


Least Recently Used (LRU) Algorithm

- LRU chooses a page for replacement that has not been used for the longest period of time.
- Uses the recent past as an approximation of the near future



- Two possible implantations of LRU:
  - Counter implementation
    - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
    - When a page needs to be changed, look at the counters to determine which are to change.
  - **Stack implementation**
    - keep a stack of page numbers in a double link form:
      - Page referenced:
        - move it to the top
        - requires 6 pointers to be changed
      - No search for replacement



LRU Approximation Algorithms

- **Use a reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- **Addition reference-bits algorithm**
  - Reference bits are recorded in regular intervals. For example, we can keep an 8-bit for each page in a table in the memory.
  - After regular intervals, a timer interrupt transfers the control to the OS which shifts the reference bit for each page into the high-order bit of 8-bit shift registers containing the history of page.
  - The page with the lowest number is the LRU page.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Consider following page reference string 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6 How many page faults occur for LRU, Optimal algorithm	Dec 2012	8
Q.2	How many page faults occur for an optimal page replacement algorithm for the following reference string, with four page frames? 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2	Dec 2013	7
Q.3	Explain LRU, Optimal page replacement algorithm	Dec 2013	7

Unit-04/Lecture-04

Counting Algorithms

- Keep a counter of the number of references that have been made to each page. The following two schemes can be used:
  - **LFU Algorithm**: Replaces page with smallest count is to be replaced.
  - **MFU Algorithm**: replace page with the largest value of count. It is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- Not very commonly used algorithm.

Allocation of Frames

- Each process needs **minimum** number of pages. Minimum number of frames is defined by the computer architecture.
- Two major allocation schemes:
  - Fixed allocation
  - Priority allocation

Fixed Allocation

- **Equal allocation** – e.g., if 100 frames and 5 processes, give each 20 pages.
- **Proportional allocation** – Allocate according to the size of process.

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault,
  - Select for replacement one of its frames.
  - Select for replacement a frame from a process with lower priority number.

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

Process Creation

- Virtual memory allows other benefits during process creation:
  - Copy-on-Write
  - Memory-Mapped Files

Copy on Write

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.
- If either process modifies a shared page, only then is the page copied.
- COW allows more efficient process creation as only modified pages are copied.

Memory Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.
- A file is initially read using demand paging. A page-sized portion of the file is read from

the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

- Simplifies file access by treating file I/O through memory rather than **read() write()** system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared

Pros/Cons of Demand Paging

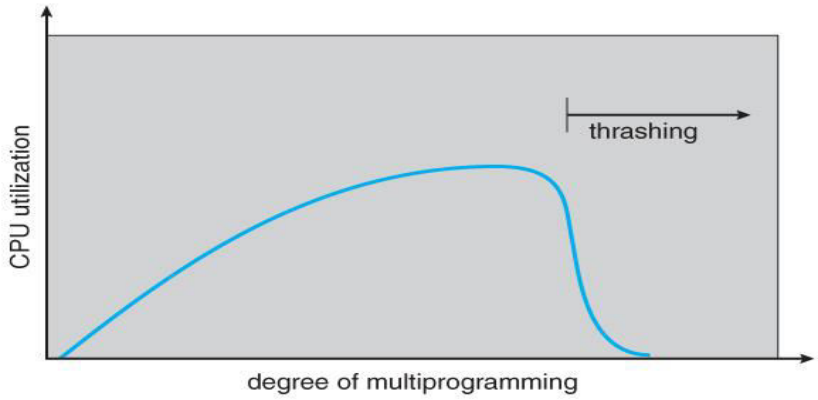
- Advantages:
  - Can run program larger than physical memory
  - Allows higher multiprogramming level than pure paging
  - Efficient memory usage
  - No compaction is required
  - Portions of process that are never called are never loaded
  - Simple partition management due to discontinuous loading and fixed partition size
  - Easy to share pages
- Disadvantages:
  - Internal fragmentation
  - Program turnaround time increases each time a page is replaced, then reloaded
  - Need special address translation hardware

Thrashing

- If a process cannot maintain its minimum required number of frames, then it must be swapped out, freeing up frames for other processes. This is an intermediate level of CPU scheduling.
- But what about a process that can keep its minimum, but cannot keep all of the frames that it is currently using on a regular basis? In this case it is forced to page out pages that it will need again in the very near future, leading to large numbers of page faults.
- A process that is spending more time paging than executing is said to be **thrashing.**

Cause of Thrashing

- Early process scheduling schemes would control the level of multiprogramming allowed based on CPU utilization, adding in more processes when CPU utilization was low.
- The problem is that when memory filled up and processes started spending lots of time waiting for their pages to page in, then CPU utilization would lower, causing the schedule to add in even more processes and exacerbating the problem! Eventually the system would essentially grind to a halt.
- Local page replacement policies can prevent one thrashing process from taking pages away from other processes, but it still tends to clog up the I/O queue, thereby slowing down any other process that needs to do even a little bit of paging (or any other I/O for that matter.)

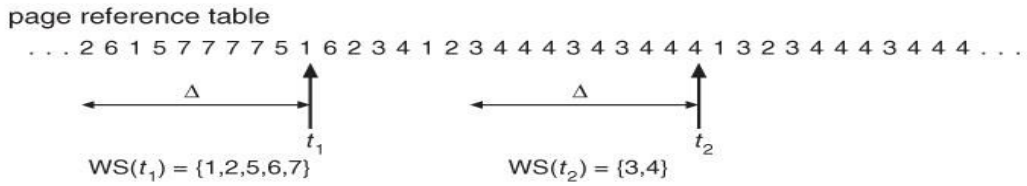


Thrashing

- To prevent thrashing we must provide processes with as many frames as they really need "right now", but how do we know what that is?

Working-Set Model

- The **working set model** is based on the concept of locality, and defines a **working set window**, of length **delta.** Whatever pages are included in the most recent delta page references are said to be in the processes working set window, and comprise its current working set, as illustrated in Figure:



Working-set model.

- The selection of delta is critical to the success of the working set model - If it is too small then it does not encompass all of the pages of the current locality, and if it is too large, then it encompasses pages that are no longer being frequently accessed.
- The total demand, D, is the sum of the sizes of the working sets for all processes. If D exceeds the total number of available frames, then at least one process is thrashing, because there are not enough frames available to satisfy its minimum working set. If D is significantly less than the currently available frames, then additional processes can be launched.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Distinguish between global & local allocation	Dec 2012	10
Q.2	Explain cause of Thrashing & by which thrashing can be controlled	Dec 2012 June 2011 Dec 2009	5 8 4

Unit-04/Lecture-05

Goals of Protection

- Obviously to prevent malicious misuse of the system by users or programs. See chapter 15 for a more thorough coverage of this goal.
- 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

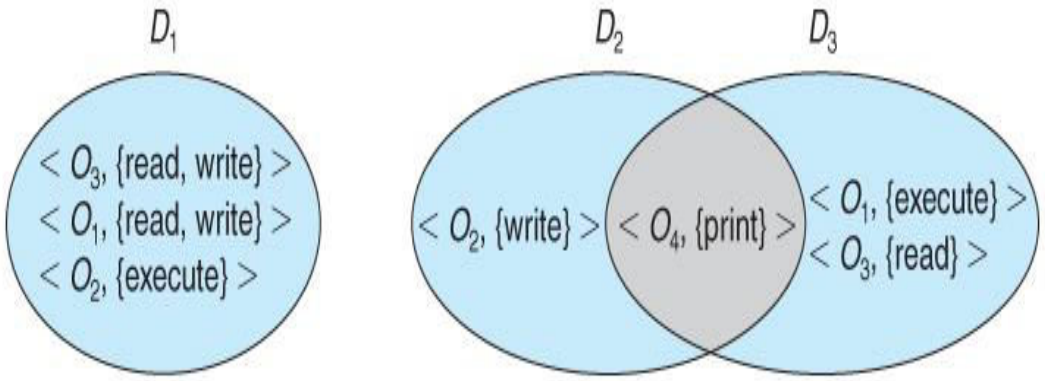
- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform 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.



System with three protection domains.

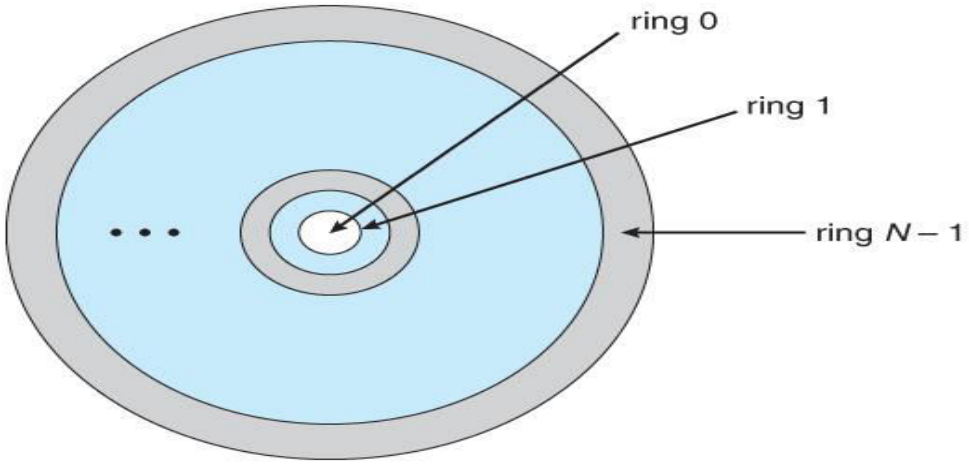
- The association between a process and a domain may be *static* or *dynamic.*
  - If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.
  - If the association is dynamic, then there needs to be a mechanism for ***domain switching.***
- 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.

An Example: UNIX

- UNIX associates domains with users.
- Certain programs operate with the SUID bit set, which effectively changes the user ID, and therefore the access domain, while the program is running. (and similarly for the SGID bit.) Unfortunately this has some potential for abuse.
- An alternative used on some systems is to place privileged programs in special directories, so that they attain the identity of the directory owner when they run. This prevents crackers from placing SUID programs in random directories around the system.
- Yet another alternative is to not allow the changing of ID at all. Instead, special privileged daemons are launched at boot time, and user processes send messages to these daemons when they need special tasks performed.

An Example: MULTICS

- The MULTICS system uses a complex system of rings, each corresponding to a different protection domain, as shown below:



MULTICS ring structure.

- Rings are numbered from 0 to 7, with outer rings having a subset of the privileges of the inner rings.
- Each file is a memory segment, and each segment description includes an entry that indicates the ring number associated with that segment, as well as read, write, and execute privileges.
- Each process runs in a ring, according to the *current-ring-number,* a counter associated with each process.
- A process operating in one ring can only access segments associated with higher (farther out) rings, and then only according to the access bits. Processes cannot access segments associated with lower rings.
- Domain switching is achieved by a process in one ring calling upon a process operating in a lower ring, which is controlled by several factors stored with each segment descriptor:
  - An ***access bracket***, defined by integers b1 <= b2.
  - A ***limit*** b3 > b2
  - A ***list of gates,*** identifying the entry points at which the segments may be called.
- If a process operating in ring i calls a segment whose bracket is such that b1 <= i <= b2, then the call succeeds and the process remains in ring i.
- Otherwise a trap to the OS occurs, and is handled as follows:
  - If i < b1, then the call is allowed, because we are transferring to a procedure with fewer privileges. However if any of the parameters being passed are of segments below b1, then they must be copied to an area accessible by the called procedure.
  - If i > b2, then the call is allowed only if i <= b3 and the call is directed to one of the entries on the list of gates.
- Overall this approach is more complex and less efficient than other protection schemes.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain different protection mechanism in operating system	Dec 2011 Dec 2012 Dec 2013	10 4 7

Unit-04/Lecture-06

Access Matrix

- The model of protection that we have been discussing can be viewed as an ***access matrix,*** in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access matrix.

- Domain switching can be easily supported under this model, simply by providing "switch" access to other domains:

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access matrix with domains as objects.

- The ability to ***copy*** rights is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e. for the same object. There are two important variations:
  - If the asterisk is removed from the original access right, then the right is ***transferred,*** rather than being copied. This may be termed a ***transfer*** right as opposed to a ***copy*** right.
  - If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propagated further. That is the new domain does not also receive the right to copy the access. This may be termed a ***limited copy*** right, as shown in Figure 14.5 below:

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access matrix with *copy* rights.

- The ***owner*** right adds the privilege of adding new rights or removing existing ones:

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access matrix with *owner* rights.

- Copy and owner rights only allow the modification of rights within a column. The addition of ***control rights***, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains. For example in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Modified access matrix

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain system protection in operating system	Dec 2012	4

Unit-04/Lecture-07

Implementation of Access Matrix

Global Table

- The simplest approach is one big global table with < domain, object, rights > entries.
- Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques.)
- There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

Access Lists for Objects

- Each column of the table can be kept as a list of the access rights for that particular object, discarding blank entries.
- For efficiency a separate list of default access rights can also be kept, and checked first.

Capability Lists for Domains

- In a similar fashion, each row of the table can be kept as a list of the capabilities of that domain.
- Capability lists are associated with each domain, but not directly accessible by the domain or any user process.
- Capability lists are themselves protected resources, distinguished from other data in one of two ways:
    - A ***tag,*** possibly hardware implemented, distinguishing this special type of data. (other types may be floats, pointers, booleans, etc.)
    - The address space for a program may be split into multiple segments, at least one of which is inaccessible by the program itself, and used by the operating system for maintaining the process's access right capability list.

A Lock-Key Mechanism

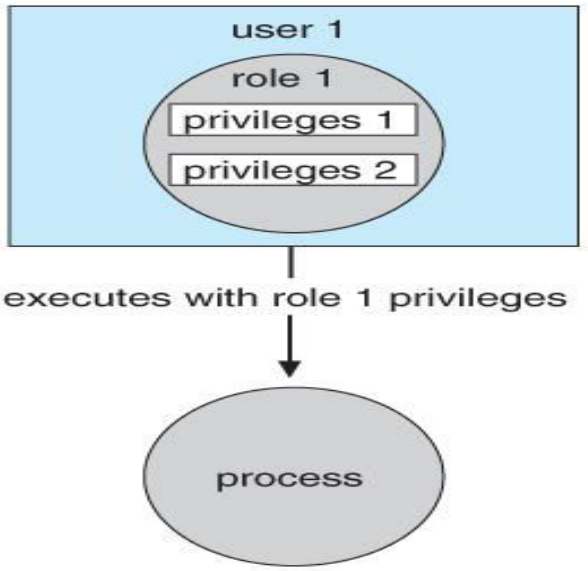
- Each resource has a list of unique bit patterns, termed locks.
- Each domain has its own list of unique bit patterns, termed keys.
- Access is granted if one of the domain's keys fits one of the resource's locks.
- Again, a process is not allowed to modify its own keys.

Comparison

- Each of the methods here has certain advantages or disadvantages, depending on the particular situation and task at hand.
- Many systems employ some combination of the listed methods.

Access Control

- ***Role-Based Access Control, RBAC,*** assigns privileges to users, programs, or roles as appropriate, where "privileges" refer to the right to call certain system calls, or to use certain parameters with those calls.
- RBAC supports the principle of least privilege, and reduces the susceptibility to abuse as opposed to SUID or SGID programs.



Role-based access control in Solaris 10.

Revocation of Access Rights

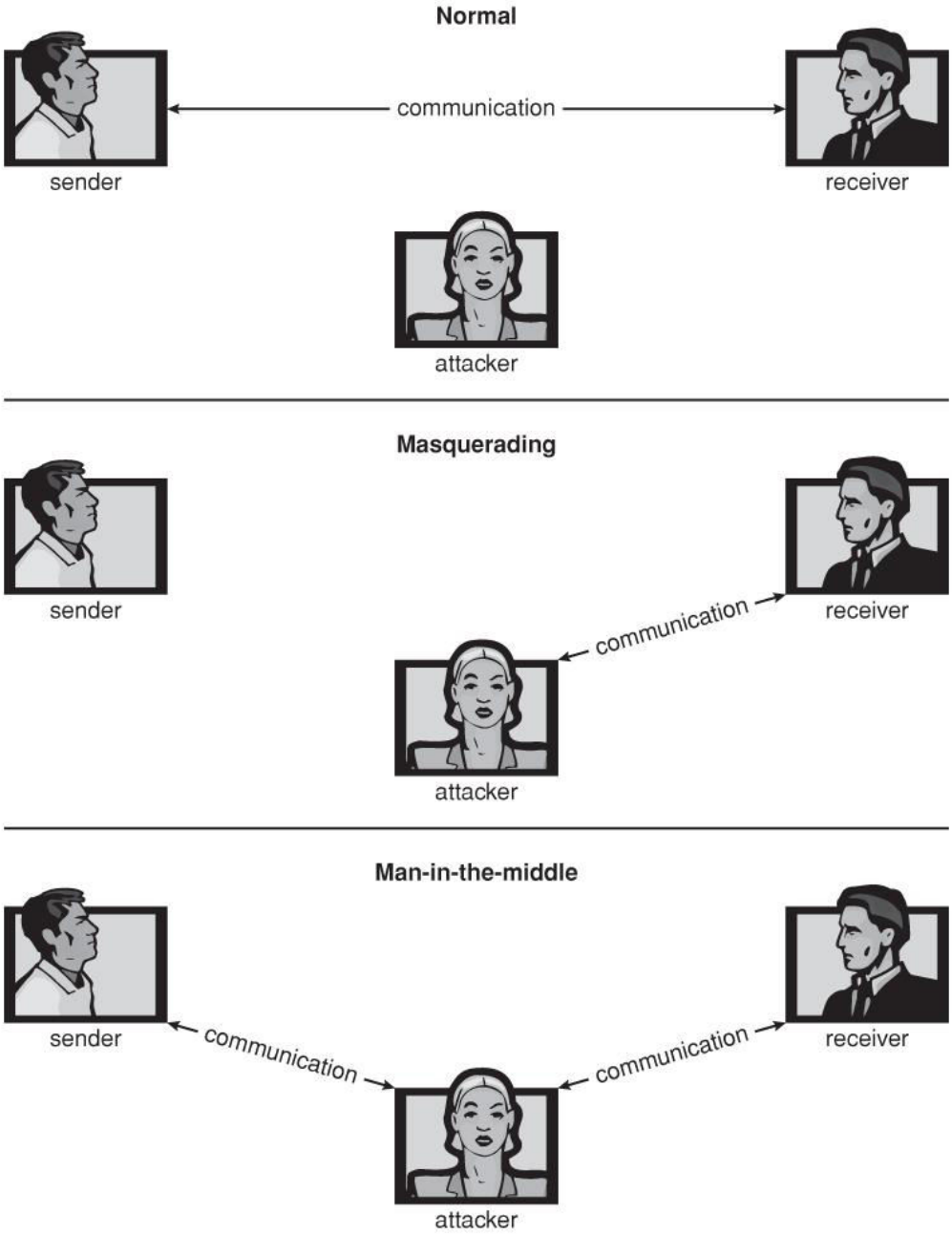
- The need to revoke access rights dynamically raises several questions:
  - Immediate versus delayed - If delayed, can we determine when the revocation will take place?
  - Selective versus general - Does revocation of an access right to an object affect *all* users who have that right, or only some users?
  - Partial versus total - Can a subset of rights for an object be revoked, or are all rights revoked at once?
  - Temporary versus permanent - If rights are revoked, is there a mechanism for processes to re-acquire some or all of the revoked rights?
- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.
- With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:
  - Reacquisition - Capabilities are periodically revoked from each domain, which must then re-acquire them.
  - Back-pointers - A list of pointers is maintained from each object to each capability which is held for that object.
  - Indirection - Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.

- Keys - A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
  - A master key is associated with each object.
  - When a capability is created, its key is set to the object's master key.
  - As long as the capability's key matches the object's key, then the capabilities remain valid.
  - The object master key can be changed with the set-key command, thereby invalidating all current capabilities.
  - More flexibility can be added to this scheme by implementing a ***list*** of keys for each object, possibly in a global table.

Unit-04/Lecture-08

The Security Problem

- Protection dealt with protecting files and other resources from accidental misuse by cooperating users sharing a system, generally using the computer for normal purposes.
- This chapter (Security) deals with protecting systems from deliberate attacks, either internal or external, from individuals intentionally attempting to steal information, damage information, or otherwise deliberately wreak havoc in some manner.
- Some of the most common types of ***violations*** include:
    - ***Breach of Confidentiality -*** Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.
    - ***Breach of Integrity -*** Unauthorized ***modification*** of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.
    - ***Breach of Availability -*** Unauthorized ***destruction*** of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.
    - ***Theft of Service -*** Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.
    - ***Denial of Service, DOS -*** Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.
- One common attack is ***masquerading,*** in which the attacker pretends to be a trusted third party. A variation of this is the ***man-in-the-middle,*** in which the attacker masquerades as both ends of the conversation to two targets.
- A ***replay attack*** involves repeating a valid transmission. Sometimes this can be the entire attack, (such as repeating a request for a money transfer), or other times the content of the original message is replaced with malicious content.



Standard security attacks.

- There are four levels at which a system must be protected:
    1. **Physical -** The easiest way to steal data is to pocket the backup tapes. Also, access to the root console will often give the user special privileges, such as rebooting the system as root from removable media. Even general access to terminals in a computer room offers some opportunities for an attacker, although today's modern high-speed networking environment provides more and more opportunities for remote attacks.
    2. **Human -** There is some concern that the humans who are allowed access to a system be trustworthy, and that they cannot be coerced into breaching security. However more and more attacks today are made via ***social engineering,*** which basically means fooling trustworthy people into accidentally breaching security.
        - **Phishing** involves sending an innocent-looking e-mail or web site designed to fool people into revealing confidential information. E.g. spam e-mails pretending to be from e-Bay, PayPal, or any of a number of banks

or credit-card companies.
    - **Dumpster Diving** involves searching the trash or other locations for passwords that are written down.
    - **Password Cracking** involves divining users passwords, either by watching them type in their passwords, knowing something about them like their pet's names, or simply trying all words in common dictionaries.
3. **Operating System -** 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.
4. **Network -** As network communications become ever more important and pervasive in modern computing environments, it becomes ever more important to protect this area of the system. (Both protecting the network itself from attack, and protecting the local system from attacks coming in through the network.) This is a growing area of concern as wireless communications and portable devices become more and more prevalent.

Program Threats

- There are many common threats to modern systems.

Trojan Horse

- A ***Trojan Horse*** is a program that secretly performs some maliciousness in addition to its visible actions.
- Some Trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with ***viruses,*** (see below.)
- One dangerous opening for Trojan horses is long search paths, and in particular paths which include the current directory (".") as part of the path. If a dangerous program having the same name as a legitimate program (or a common mis-spelling, such as "sl" instead of "ls") is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake.
- Another classic Trojan Horse is a login emulator, which records a users account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again (with a proper login prompt), logs in successfully, and doesn't realize that their information has been stolen.
- Two solutions to Trojan Horses are to have the system print usage statistics on logouts, and to require the typing of non-trappable key sequences such as Control-Alt-Delete in order to log in. (This is why modern Windows systems require the Control-Alt-Delete sequence to commence logging in, which cannot be emulated or caught by ordinary programs. I.e. that key sequence always transfers control over to the operating system.)
- ***Spyware*** is a version of a Trojan Horse that is often included in "free" software downloaded off the Internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. (This is an example of ***covert channels,*** in which surreptitious communications occur.) Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user.

Trap Door

- A ***Trap Door*** is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system.
- Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door.
- A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

Logic Bomb

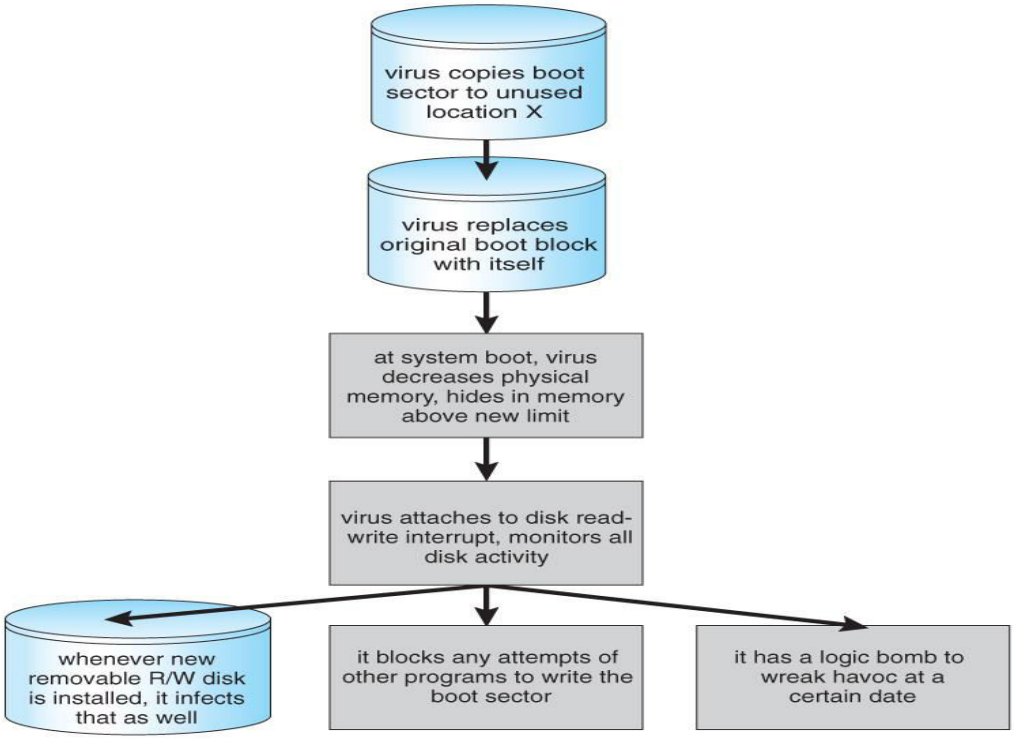
- A ***Logic Bomb*** is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.
- A classic example is the ***Dead-Man Switch***, which is designed to check whether a certain person (e.g. the author) is logging in every day, and if they don't log in for a long time (presumably because they've been fired), then the logic bomb goes off and either opens up security holes or causes other problems.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain security breaches	Dec 2012	5
Q.2	Explain role of OS in security	Dec 2012	10

Unit-04/Lecture-09

Viruses

- 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.



A boot-sector computer virus.

- Some of the forms of viruses include:
  - **File -** A file virus attaches itself to an executable file, causing it to run the virus code first and then jump to the start of the original program. These viruses are termed ***parasitic,*** because they do not leave any new files on the system, and the original program is still fully functional.
  - **Boot -** A boot virus occupies the boot sector, and runs before the OS is loaded. These are also known as ***memory viruses***, because in operation they reside in memory, and do not appear in the file system.
  - **Macro -** These viruses exist as a macro (script) that are run automatically by certain macro-capable programs such as MS Word or Excel. These viruses can exist in word processing documents or spreadsheet files.
  - **Source code** viruses look for source code and infect it in order to spread.
  - **Polymorphic** viruses change every time they spread - Not their underlying

functionality, but just their ***signature,*** by which virus checkers recognize them.
    - **Encrypted** viruses travel in encrypted form to escape detection. In practice they are self-decrypting, which then allows them to infect other files.
    - **Stealth** viruses try to avoid detection by modifying parts of the system that could be used to detect it. For example the read( ) system call could be modified so that if an infected file is read the infected part gets skipped and the reader would see the original unadulterated file.
    - **Tunneling** viruses attempt to avoid detection by inserting themselves into the interrupt handler chain, or into device drivers.
    - **Multipartite** viruses attack multiple parts of the system, such as files, boot sector, and memory.
    - **Armored** viruses are coded to make them hard for anti-virus researchers to decode and understand. In addition many files associated with viruses are hidden, protected, or given innocuous looking names such as "...".
- In 2004 a virus exploited three bugs in Microsoft products to infect hundreds of Windows servers (including many trusted sites) running Microsoft Internet Information Server, which in turn infected any Microsoft Internet Explorer web browser that visited any of the infected server sites. One of the back-door programs it installed was a ***keystroke logger,*** which records users keystrokes, including passwords and other sensitive information.
- There is some debate in the computing community as to whether a ***monoculture,*** in which nearly all systems run the same hardware, operating system, and applications, increases the threat of viruses and the potential for harm caused by them.

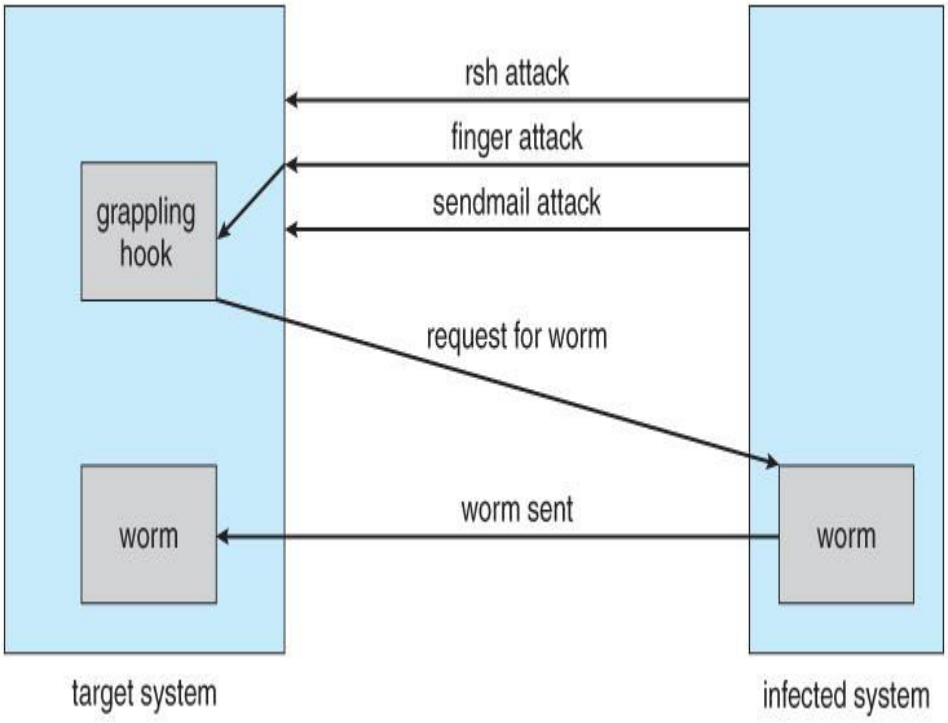
System and Network Threats

- Most of the threats described above are termed ***program threats***, because they attack specific programs or are carried and distributed in programs. The threats in this section attack the operating system or the network itself, or leverage those systems to launch their attacks.

Worms

- A ***worm*** is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system. Worms consume system resources, often blocking out other, legitimate processes. Worms that propagate over networks can be especially problematic, as they can tie up vast amounts of network resources and bring down large-scale systems.
- One of the most well-known worms was launched by Robert Morris, a graduate student at Cornell, in November 1988. Targeting Sun and VAX computers running BSD UNIX version 4, the worm spanned the Internet in a matter of a few hours, and consumed enough resources to bring down many systems.
- This worm consisted of two parts:
    1. A small program called a ***grappling hook,*** which was deposited on the target system through one of three vulnerabilities, and
    2. The main worm program, which was transferred onto the target system

and launched by the grappling hook program.



The Morris Internet worm.

- The three vulnerabilities exploited by the Morris Internet worm were as follows:
    1. **rsh (remote shell)** is a utility that was in common use at that time for accessing remote systems without having to provide a password. If a user had an account on two different computers (with the same account name on both systems), then the system could be configured to allow that user to remotely connect from one system to the other without having to provide a password. Many systems were configured so that ***any*** user (except root) on system A could access the same account on system B without providing a password.
    2. **finger** is a utility that allows one to remotely query a user database, to find the true name and other information for a given account name on a given system. For example "finger joeUser@somemachine.edu" would access the finger daemon at somemachine.edu and return information regarding joeUser. Unfortunately the finger daemon (which ran with system privileges) had the buffer overflow problem, so by sending a special 536-character user name the worm was able to fork a shell on the remote system running with root privileges.
    3. **sendmail** is a routine for sending and forwarding mail that also included a debugging option for verifying and testing the system. The debug feature was convenient for administrators, and was often left turned on. The Morris worm exploited the debugger to mail and execute a copy of the grappling hook program on the remote system.
- Once in place, the worm undertook systematic attacks to discover user passwords:
    1. First it would check for accounts for which the account name and the password were the same, such as "guest", "guest".

2. Then it would try an internal dictionary of 432 favorite password choices. (I'm sure "password", "pass", and blank passwords were all on the list.)
3. Finally it would try every word in the standard UNIX on-line dictionary to try and break into user accounts.

- Once it had gotten access to one or more user accounts, then it would attempt to use those accounts to rsh to other systems, and continue the process.
- With each new access the worm would check for already running copies of itself, and 6 out of 7 times if it found one it would stop. (The seventh was to prevent the worm from being stopped by fake copies.)
- Fortunately the same rapid network connectivity that allowed the worm to propagate so quickly also quickly led to its demise - Within 24 hours remedies for stopping the worm propagated through the Internet from administrator to administrator, and the worm was quickly shut down.
- There is some debate about whether Mr. Morris's actions were a harmless prank or research project that got out of hand or a deliberate and malicious attack on the Internet. However the court system convicted him, and penalized him heavy fines and court costs.
- There have since been many other worm attacks, including the W32.Sobig.F@mm attack which infected hundreds of thousands of computers and an estimated 1 in 17 e-mails in August 2003. This worm made detection difficult by varying the subject line of the infection-carrying mail message, including "Thank You!", "Your details", and "Re: Approved".

Port Scanning

- ***Port Scanning*** is technically not an attack, but rather a search for vulnerabilities to attack. The basic idea is to systematically attempt to connect to every known (or common or possible) network port on some remote machine, and to attempt to make contact. Once it is determined that a particular computer is listening to a particular port, then the next step is to determine what daemon is listening, and whether or not it is a version containing a known security flaw that can be exploited.
- Because port scanning is easily detected and traced, it is usually launched from ***zombie systems,*** i.e. previously hacked systems that are being used without the knowledge or permission of their rightful owner. For this reason it is important to protect "innocuous" systems and accounts as well as those that contain sensitive information or special privileges.

Denial of Service

- ***Denial of Service (DOS)*** attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work. Tight loops that repeatedly request system services are an obvious form of this attack.
- DOS attacks can also involve social engineering, such as the Internet chain letters that say "send this immediately to 10 of your friends, and then go to a certain URL", which clogs up not only the Internet mail system but also the web server to which everyone is directed. (Note: Sending a "reply all" to such a message notifying everyone that it was just a hoax also clogs up the Internet mail service, just as effectively as if you had forwarded the thing.)

- Security systems that lock accounts after a certain number of failed login attempts are subject to DOS attacks which repeatedly attempt logins to all accounts with invalid passwords strictly in order to lock up all accounts.

Passwords

- Passwords are the most common form of user authentication. If the user is in possession of the correct password, then they are considered to have identified themselves.
- In theory separate passwords could be implemented for separate activities, such as reading this file, writing that file, etc. In practice most systems use one password to confirm user identity, and then authorization is based upon that identification. This is a result of the classic trade-off between security and convenience.

Password Vulnerabilities

- Passwords can be guessed.
    - Intelligent guessing requires knowing something about the intended target in specific, or about people and commonly used passwords in general.
    - Brute-force guessing involves trying every word in the dictionary, or every valid combination of characters. For this reason good passwords should not be in any dictionary (in any language), should be reasonably lengthy, and should use the full range of allowable characters by including upper and lower case characters, numbers, and special symbols.
- "Shoulder surfing" involves looking over people's shoulders while they are typing in their password.
    - Even if the lurker does not get the entire password, they may get enough clues to narrow it down, especially if they watch on repeated occasions.
    - Common courtesy dictates that you look away from the keyboard while someone is typing their password.
    - Passwords echoed as stars or dots still give clues, because an observer can determine how many characters are in the password. :-(
- "Packet sniffing" involves putting a monitor on a network connection and reading data contained in those packets.
    - SSH encrypts all packets, reducing the effectiveness of packet sniffing.
    - However you should still never e-mail a password, particularly not with the word "password" in the same message or worse yet the subject header.
    - Beware of any system that transmits passwords in clear text. ("Thank you for signing up for XYZ. Your new account and password information are shown below".) You probably want to have a spare throw-away password to give these entities, instead of using the same high-security password that you use for banking or other confidential uses.
- Long hard to remember passwords are often written down, particularly if they are used seldomly or must be changed frequently. Hence a security trade-off of passwords that are easily divined versus those that get written down. :-(
- Passwords can be given away to friends or co-workers, destroying the integrity of the entire user-identification system.

- Most systems have configurable parameters controlling password generation and what constitutes acceptable passwords.
    - They may be user chosen or machine generated.
    - They may have minimum and/or maximum length requirements.
    - They may need to be changed with a given frequency. (In extreme cases for every session.)
    - A variable length history can prevent repeating passwords.
    - More or less stringent checks can be made against password dictionaries.

Encrypted Passwords

- Modern systems do not store passwords in clear-text form, and hence there is no mechanism to look up an existing password.
- Rather they are encrypted and stored in that form. When a user enters their password, that too is encrypted, and if the encrypted version match, then user authentication passes.
- The encryption scheme was once considered safe enough that the encrypted versions were stored in the publicly readable file "/etc/passwd".
    - They always encrypted to a 13 character string, so an account could be disabled by putting a string of any other length into the password field.
    - Modern computers can try every possible password combination in a reasonably short time, so now the encrypted passwords are stored in files that are only readable by the super user. Any password-related programs run as setuid root to get access to these files. (/etc/shadow)
    - A random seed is included as part of the password generation process, and stored as part of the encrypted password. This ensures that if two accounts have the same plain-text password that they will not have the same encrypted password. However cutting and pasting encrypted passwords from one account to another will give them the same plain-text passwords.

One-Time Passwords

- One-time passwords resist shoulder surfing and other attacks where an observer is able to capture a password typed in by a user.
    - These are often based on a **challenge** and a **response.** Because the challenge is different each time, the old response will not be valid for future challenges.
        - For example, The user may be in possession of a secret function f(x). The system challenges with some given value for x, and the user responds with f(x), which the system can then verify. Since the challenger gives a different (random) x each time, the answer is constantly changing.
        - A variation uses a map (e.g. a road map) as the key. Today's question might be "On what corner is SEO located?", and tomorrow's question might be "How far is it from Navy Pier to Wrigley Field?" Obviously "Taylor and Morgan" would not be accepted as a valid answer for the second question!
    - Another option is to have some sort of electronic card with a series of constantly changing numbers, based on the current time. The user enters

the current number on the card, which will only be valid for a few seconds. A ***two-factor authorization*** also requires a traditional password in addition to the number on the card, so others may not use it if it were ever lost or stolen.
    - A third variation is a ***code book,*** or ***one-time pad.*** In this scheme a long list of passwords is generated, and each one is crossed off and cancelled as it is used. Obviously it is important to keep the pad secure.

Biometrics

- Biometrics involve a physical characteristic of the user that is not easily forged or duplicated and not likely to be identical between multiple users.
    - Fingerprint scanners are getting faster, more accurate, and more economical.
    - Palm readers can check thermal properties, finger length, etc.
    - Retinal scanners examine the back of the users' eyes.
    - Voiceprint analyzers distinguish particular voices.
    - Difficulties may arise in the event of colds, injuries, or other physiological changes.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain Security breaches	Dec 2012	5
Q.2	Explain password management in operating system	Dec 2012	4