Operating Systems

Elvis C. Foster

Lecture 08: System Protection and Security

This lecture discusses how the operating system provides the critical services of protection and security. The lecture proceeds under the following captions:

- Introduction
- Resources Needing Protection and Security
- Address Checking — Protecting Processes From Each Other
- Protection Domains
- Access Matrix
- Dynamic Protection Structures
- Revocation
- Security via Encryption
- Levels of Security
- Security and Protection on IBM i
- Security and Protection on Windows
- Security and Protection on Linux

8.1 Introduction

Protection has to do with preventing unwanted interaction between processes due to any of the following:
- Mistakes in programs
- Deliberate user efforts
- OS failure

Security has to do with authorized access to the resources of a system. It also relates with confidence in the integrity of a system.

Both protection and security are closely inter-related. It is therefore virtually impossible to address one without speaking to the other. The goal is to have a reliable working environment.

Operating system protection/security measures include the following:
- Encryption of data
- Diversification of data transmission.
- Authority constraints
- Access constraints
- Privileged instructions
- Process base and offset limits (to protect processes from each other)

Some measures are implemented by hardware, others by software. Software implementation tends to be more flexible but slower than hardware implementations.

Note: The use of page (segment) table in memory management also supports easy protection of processes from each other, allowing access to only authorized process/memory locations. Also, the matter of privileged instructions has been covered in earlier lectures. This lecture will therefore de-emphasize these issues, and concentrate on the other issues not yet discussed.

8.2 Resources Needing Protection and security

An operating system typically provides three levels of security:
- Access to the system (typically by logon)
- Access to system resources (commands, files, directories, etc.)
- Access to data contained in system databases and files.

All objects on the system need protection in one form or another. These include files, folders (called directories or libraries on some systems), programs, commands, devices, user accounts (profiles), etc.

8.2 Resources Needing Protection and Security (continued)

The kind of protection needed depends on the type of object. For instance:
- Independent processes must be protected from each other.
- Programs and commands need execution accessibility.
- Source files have various access rights which can be granted to different users.
- Database files, like sources files, have access rights which may be granted to different users.

Typically, access rights to user created objects are controlled by the user that owns these objects (referred to as the creator). Access rights to system created objects are controlled by the OS or a special user endowed with superior authority.

Example: The superior user has the following names on the respective operation systems

On Unix, the superior user is the **Super User** (or Root); on OS/400, it is the **Security Officer**; on PICK, it is the **System Manager**; on Netware, it is the **System Supervisor**; on Windows NT and Windows, it is the **Administrator**.

As you are aware, privileged instructions are not accessible to users of the system. They are only executable in supervisory (or monitor) mode. Examples of privileged instructions include interrupts, I\O servicing loading fence registers etc.

8.3 Address Checking — Protecting Processes From Each Other

Inter-process protection can be done either logically or physically by fence registers. Alternatively there may be a base and limit strategy. Loading of base registers must be privileged.

Checking may be via hardware (machine code) or software; the former being preferred if speed is important.

Loading and updating of the page or segment table must be privileged. Within the segment or page table the following entries assist with protection:
- Segment Length (for segment length)
- Page/Segment Validity Flag
- Read/Write Flag
- Dirty bits etc

8.4 Protection Domains

A process operates within a *protection domain,* which specifies the resources that the process may access. Each domain defines a set of objects and the operations that may be invoked on each object. This ability to execute certain operations on an object is called *access right*.

More formally, a domain D is a set of access rights, each of which is an ordered pair [Object-name, Rights-set].

Example: A domain D may have the following access rights:

```
FileA, {Read, Update, Write}
...
PgmA, {Execute}
```

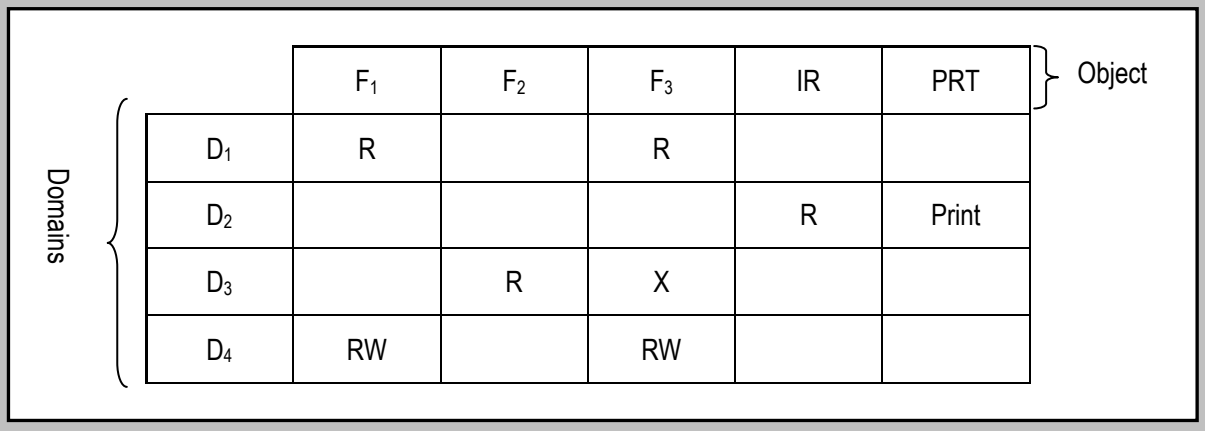
Usually access rights are granted by the owner of objects or the superior user, to other users of the system. When a user starts a process, those access rights given to the user, apply to the process. The domain is implemented as a user account or user group.

In a multiprogramming, multitasking, and nowadays multithreading systems, several domains exist, some of which may be overlapping. An *access matrix* stores access rights to system objects in various domains.

85. Access Matrix

The access matrix (also called a capability table), stores access rights to objects in various domains. In the illustration of figure 8.1, rows represent domains and columns represent objects; each entry is an access rights set (list). The entry Access ($_i$, $_j$) defines the set of operations allowed on object O_j in domain D_i.

Figure 8.1: Illustration of an Access Matrix



8.5 Access Matrix (continued)

Example: From the matrix the following may be deduced

In D_1, files F_1, F_3 may be read.
In D_2, the image reader (IR) may be read and the printer (PRT) used for printing.
In D_3, F_2 may be read and F_3 may be executed.
In D_4, F_1 and F_3 may be read and/or written.

The matrix may also be represented with objects along the rows and domains as columns.
In practice, it is not stored as shown as it would be too wasteful, being a sparse array.

8.6 Implementation of the Access Matrix

As mentioned in the previous section, the access matrix is typically a sparse array. Therefore implementing it using arrays would be extremely wasteful. From your data structures course, recall that sparse arrays may be implemented as circular queues. That implementation also has its problems, with the introduction of multiple pointers. Writing an operating system is by definition, a difficult undertaking, so added complexity is not welcome, unless it brings advantages that cannot be excluded.

We will look at three implementation alternatives:
- Global table
- Access list
- Capability list

8.6.1 Global Table

The simplest implementation is to use a global table consisting of ordered triples [Domain, Object, Right-set], as illustrated in figure 8.2. Using this configuration, if an operation, X is to be preformed on an object O_j within domain D_i the global table is searched for the triple [D_i O_j R_k] where $X \in R_k$, as shown in figure 8.2. If the sought triplet is found, the operation is allowed, otherwise an exception condition is raised. The system could then return a message to the user, indicating that he/she does not have the appropriate authority.

The main advantage of this approach is that for each domain (user), it can be easily determined what access rights apply to objects to which that domain has authority.

8.6.1 Global Table (continued)

Two drawbacks with this approach are:
- The table is potentially large, thus necessitating virtual memory techniques.
- Special groupings are not facilitated. If an object can be read by everyone, it must have a separate entry in each domain.

Figure 8.2: Global Table

Domain	Object	Rights Set
D_1	F_1	R
D_1	F_3	R
D_2	IR	R
D_2	PRT	P
D_3	F_2	R
D_3	F_3	X
D_4	F_1	R, W

8.6.2 Access Lists

In this approach, each column of the access matrix is stored as an access list for one object. The empty entries are simply discarded. The list for each object consists of ordered pairs [Domain, Rights-set], as illustrated in figure 8.3.

When an operation, X on object O_j is attempted in domain D_i, the access list is searched for O_j, looking for entry $[D_j\ R_k]$ where $X \in R_k$.

Figure 8.3: Access List

F_1:	[D_1 { R }],	[D_4, { R, W }]	
F2:	[D_3, { R }]		
F3:	[D_1, { R }],	[D_3, { X}],	[D_4, { R, W }]
IR:	[D_2, { R }]		
PRT:	[D_2, { P }]		

8.6.2 Access Lists (continued)

To condense the length of the access list, many systems recognize three popular user classifications: owner, group, public. Access would therefore be defined with respect to these three classifications (*instead of* domains). With this approach, only three fields are needed to define protection for a system.

Example:

> Unix defines three fields of three bits each: rwx for (read, write, execute respectively). Thus, for each file a 9-bit code defines file protection information. Windows NT uses a similar approach.

The advantage of this approach is that by examining the access list of a given object, one can easily determine what domain (user or user's group) rights apply for that object.

The main drawback is that given a domain, it cannot be easily determined, what access rights to objects apply to that domain.

8.6.3 Capability lists

For this approach, each row of the matrix is stored as a set of objects and allowed operations — a capability list for the domain. Empty entries are discarded. The list for each domain consists of ordered pairs [Object, Right-set], as illustrated in figure 8.4.

When an operation X is attempted for object O_j in domain D_i, the capability list is searched for D_i, looking for the entry $[O_j\ R_k]$ where $X \in R_k$.

Figure 8.4: Capability List

D_1:	[F_1, (R)],	[F_3, (R)]
D_2:	[CR, (R)],	[PRT, (P)]
D_3:	[F_2, (R)],	[F_3, (X)]
D_4:	[F_1, (R, W)],	[F_3, (R, W)]

The advantage of the capability list is that given a domain (user or user's group), it can be easily determined what access rights to objects apply to that domain.

The disadvantage is that for a given object, it cannot be easily determined what domain (user) rights apply for that object. There are also problems associated with revocation of access rights, as you will see later.

8.6.4 Comparison: Access List Vs Capability List

For the access list, access rights information for a particular domain is not localized. It is therefore difficult to determine access rights for a domain (the access list must be searched or re-ordered). Additionally, every access to the object must be checked, against the access list.

Capability lists do not correspond to the need of users in the sense that access lists do. They are, however, useful in localizing information for a particular domain and process. Every access to the object does not warrant a search of the capability list.

Most systems use the strengths of both approaches, while minimizing the observed weaknesses. Since in either approach, the same data is stored, logical views facilitate which interpretation is applied and utilized.

8.7 Dynamic Protection Structures

The association between a process and a domain may be static (if the set of resources available to the process is fixed throughout its lifetime) or dynamic.

For static association, it may be necessary to have a mechanism whereby the contents of a domain can be modified. This will lend flexibility to processes in terms of what operations they can execute on objects, while ensuring that at any point in time the access matrix reflects the minimum necessary access rights r required for the process.

For dynamic association, a mechanism is available to allow process to switch between (*among*) domains.

Switching among domains requires that the domains themselves be treated as objects, and an access right (*which we will call*) *switch* be introduced. Figure 8.5 illustrates.

Figure 8.5 Access Matrix with Domains as Objects

	Object								
Domain	F_1	F_2	F_3	IR	PRT	D_1	D_2	D_3	D_4
D_1	Read		Read				Switch		
D_2				Read	Print			Switch	Switch
D_3		Read	Execute						
D_4	Read Write		Read Write			Switch			

Note:
1. A process executing D_1 can switch to D_2
2. A process executing D_2 can switch to D_3 and D_4
3. A process executing D_4 can switch to D_1

8.8 Revocation

It is sometimes necessary to revoke access rights from certain domains (users or user groups). Typically access rights are given to users by the owner of the object in question, or the superior user of the system. When an object is created, certain default access rights are given to the owner, user group (if it exists), public. Those defaults may be changed or revoked by the owner or the superior user.

Various questions about revocation may arise:

- **Immediate Versus Delayed:** Does the revocation take effect immediately, or is it delayed. If it is delayed, when does it take effect?
- **Selective Versus General:** Is the revoked right selective to a specific user or group of users, or is it general, applying to all users?
- **Partial Versus Total:** Can a subset of the rights associated with the object be revoked, or must all the associated rights be revoked?
- **Temporary Versus Permanent:** Is the revocation permanent (never to be reinstated) or is it temporary (can be reinstated).

With the access-list scheme, revocation is easy. The access list is searched for the given object; the rights to be revoked are the deleted from the list. The revocation is immediate; it can be partial or total, general or selective, permanent or temporary.

With capabilities, revocation is a more difficult problem. Since for any given object, capabilities are distributed through the system, they must be searched for, found, and then removed. Several revocation schemas have been proposed for capability list; the prominent ones are summarized below:

- **Reacquisition:** A process can attempt to reacquire a capability after it has been deleted from each domain.
- **Back-pointers:** For each object, a list of pointers, pointing to all capabilities of the object, is maintained. When revocation is acquired, the appropriate pointers are followed and changed as necessary. This system was adopted on the MULTICS System.
- **Indirection:** Capabilities point indirectly (not directly) to objects. Each capability points to a unique entry in a global table, which in turn, point to the object. Revocation is implemented by searching the global table for the desired entry and deleting it. When access is attempted, the capability would be found pointing to an illegal global table entry. This scheme was adopted in the CAL system. It does not allow selective revocation.
- **Keys:** A unique bit pattern — called a *key* — is associated with each capability; this key can neither be modified nor inspected by the process. A *master key* is associated with each object, which can be defined or replaced by the **set-key** operation. When a capability is created its key is set to the value of the object's master key. When the capability is exercised, the capability key and the master key are compared. If they match, the operation is allowed; otherwise the operation is denied. Revocation invokes the **set-key** operation to reset the value of the master key for that object. Thus, all subsequent access attempts will be denied for that capability. This schema does not allow selective revocation. However, with some enhancements (involving the use of a global table), it can be improved to be selective.

8.8 Revocation (continued)

From the forgoing discussion, it should be evident why an access list implementation (with the flexibility of using logical views to access information in the order of capability list(s)) is preferred to using capability list(s) only.

8.9 Security via Encryption

As part of their security mechanism, most modern operating systems implement some form of encryption to further enhance the security of the system. Encryption is the act of converting data and storing it in a disguised form, so that it intercepted by an intruder (for example a hacker), the data will be meaningless. When the data is read by an authorized user, it is decrypted back to a meaningful form.

Some systems automatically encrypt all data stored so that a hacker is guaranteed to access gibberish, unless he/she has access to the encryption-decryption algorithm or key. IBM i provides an example of this approach.

On systems such as Windows and Linux, encryption is selective. Windows provides more flexibility than RedHat Linux: On Windows, the user decides what is to be encrypted. On RedHat Linux Linux-7, data transmitted over a network is encrypted, but for regular files, third party software is required for encryption. On both systems, user passwords are encrypted by default.

Encryption algorithms typically use highly complex mathematical formulas. The reason for this is to ensure secrecy and discourage the likelihood of cracking the algorithm. In fact, some encryption algorithms are virtually crack-proof to manual deciphering efforts; they can only be cracked through automation, and this is not easy. Many algorithms involve the issue and use of *access keys*: When the data is encrypted, it is assigned an access key. When the encrypted data is accessed, the correct access key must be supplied in order to invoke the decryption algorithm.

Among the popularly used encryption techniques are the following:

- Data Encryption Standard (DES) — an encryption block cipher developed by IBM and endorsed by the US government (in 1977)
- RSA — a public key encryption system developed by Ron Rivest, Adi Shamir and Leonard Adleman (in 1977)
- Digital Signature Algorithm (DSA) — an authentication algorithm
- Diffe-Hellman Key Agreement Protocol — a secret key system
- ElGamal System —a public key system
- Elliptic Curve Cryptosystems — public key systems
- Knapsack Cryptosystems — also public key systems
- Secure Hash Algorithm (SHA)
- MD2, MD4, MD5 — Message Digest algorithms, developed by Ron Rivest

A full discussion of encryption is beyond the scope of this course. However, it must be stated that it is quite a fascinating field.

8.10 Levels of Security

The operating system typically provides the following three levels of security:

- Access to the system
- Access to system resources
- Access to system data

8.10.1 Access to the System

Access to the system typically involves a login process. Each legitimate user is provided with an account, without which they cannot access the system. Figure 8.6 provides some details that are stored in the user account.

Figure 8.6: Details Stored About a User Account

Element	Categorization
Account (login) name	Essential
User name	Essential
User password (usually encrypted and/or disguised)	Essential
User group(s) or class(es)	Optional
User logo or picture	Optional

Some systems use a different name to refer to the user account. One such example is IBM i, where the term *user profile* is preferred.

The user accounts are stored in an underlying database file. When the user attempts to log on, this file is accessed to determine whether this is a legitimate user. If the test is successful, the user is admitted in; otherwise, an error message is returned to the user. Of course, the database file must be appropriately designed to store all required details about the user account.

8.10.2 Access to System Resources

Once a user gains access to the system, the next level of security to be addressed is access to system resources. By system resources, we mean files, commands, utility programs, services, etc. Depending on how complex the operating system is, this could be quite extensive, involving the storage and (often transparent) management the *access matrix*. When the user attempts to access a resource, the access matrix is checked to determine whether the attempted access is legitimate. If it is, the operation is allowed; otherwise, an error message is returned to the user.

One way to implement the resource access matrix is to incorporate its design into the underlying database for the operating system. Every time a new resource or user is added to the system, the access matrix is automatically updated. The access matrix is also automatically updated whenever resources and/or users are removed from the system. Obviously, maintenance of the matrix involves a set of privileged instructions that must be carefully guarded by the operating system.

8.10.3 Access to System Data

To a large extent, access to system resources is an extension of access to system resources. Access to system data therefore involves manipulation of operating system security mechanisms to allow or deny users access to data contained in certain files. For instance, users may be given read-only access to certain data files, but no write-access. In other instances, users may be denied any form of access to certain data files.

Operating system designers have explored and implemented various strategies and techniques for providing data-level security through encryption (discussed earlier) and access control. The following case studies should provide you with some insight.

8.11 Summary of Protection and Security in IBM i

By way of illustration, a brief summary of how security is managed in the operating system IBM i is provided below (figure 8.7):

Figure 8.7: Summary of IBM i Security

- Users are identified by user profiles. User profiles may be created by any user with appropriate authority. User profile names are confined to a maximum of 10 bytes.
- The *user-class* parameter of the CRTUSRPRF command facilitates the assignment of the user profile to a user class. Included among possible user classes are:
  - ✓ QSECOFR — Security Officer
  - ✓ QPGMR — Programmers
  - ✓ QSYSOPR — System Operator
  - ✓ QUSR — User

  Other classes can be created. The user-class defines an authority set, which includes access rights to certain system utility commands, files etc. – system objects.
- The special authority parameter facilitates further assignment to specific OS features not covered in user-class parameter.
- User profiles are stored in a system library, QSYSUSR. All objects (e.g. programs, files, commands, etc) are all stored in libraries (which are themselves objects), with assigned access rights.
- Access rights to objects are referred to as *authority*.
  Authority is classified as follows:
  - ✓ Owner
  - ✓ Group (typically, a user belong to a group-user-profile)
  - ✓ Public

Figure 8.7: Summary of IBM i Security (continued)

- Authority includes the right to certain operation on the object in question.
  Example: For a database file or a source file, the following rights apply:
  - ✓ Existence rights – All rights including deletion of the file
  - ✓ Add records
  - ✓ Update records
  - ✓ Delete records
  - ✓ Read records

  For a program the following rights apply:
  - ✓ Existence rights
  - ✓ Execute rights

- When an object is created, a set of default authorities are assigned.
  - ✓ The Owner has *existence* rights
  - ✓ The Public is given **NORMAL* rights – meaning basic ADD, UPDATE, and DELETE records, READ, EXECUTE (the object type determines which ones are applicable).
  - ✓ If the object belongs to a group profile, then the group profile gets *existence* rights.

- An access matrix is created by the OS for each object created. The owner of an object or a user with user-class *QSECOFR may adjust the access matrix of an object by any of the following means:
  - ✓ GRTOBJAUT Command
  - ✓ GRTUSRAUT Command
  - ✓ CHGOBJAUT Command
  - ✓ RVKOBJAUT Command
  - ✓ RVKUSRAUT Command

- Because of the relational, object oriented nature of the operating system, it is conceptually easy to appreciate that these access matrices may be conveniently ordered by object or user profile etc. so that control is very efficient.

- Additionally, access lists may be set up for various objects, these access lists determine how the object owner wishes other users to access his/her object. Access lists simplify the management of object and resource security significantly.

- The MOVOBJ command allows the user to move objects from one library to another. Objects retain their security settings when moved.

8.12 Summary of Security in Windows 2000

We now provide a brief summary of system security on Windows 2000 (figure 8.8), the base for various current versions of the Windows platform:

Figure 8.8: Summary of Windows 2000 Security

- A Windows 2000 domain is a group of computers that are part of a network, and share a common directory database; it is a span of control which may consist of one or more servers. A domain controller is a server that controls a set of computers in a network; the domain usually takes the name of its domain controller.

- The domain is the basis for security management: every object is linked to a domain. At the local level, objects reside on local machines (which have names and are linked to some domain) and are managed by Windows 2000 Professional — the client operating system. At the server level, objects reside on the domain controller and are managed by Windows 2000 Server.

- User accounts are classified into user groups; when a new account is created, it is placed into a user group. The user account inherits all the authority of its group. The user account could belong to more than one groups, in which case, it inherits from multiple groups. There is no restriction of the length of the user account name up to 256 bytes.

- The system is shipped with the following default user groups:
  - ✓ Administrator
  - ✓ Backup Operator
  - ✓ Power User
  - ✓ User
  - ✓ Guest
  - ✓ Replicator
  - ✓ Other Special Group (which includes Interactive and Network)

  Of course, other groups can be created.

- The superior user account is a default account, Administrator, which belongs to the user group of the same name.

- When a user creates an object, he/she is given **Full Control** rights to that object. By default, everyone has **read, write and execute** access to the object. The user may grant or deny additional access permission(s) of that object to some users, while restricting other users. Object permissions that can be granted or revoked include:
  - ✓ Full Control
  - ✓ Modify
  - ✓ Read
  - ✓ Execute
  - ✓ Write

- When user groups and/or user accounts are created on a server they are stored in the **Active Directory**. Local accounts are stored in a special folder on the local machine (under the **Documents and Settings** folder). Active Directory user accounts can log on to any machine on the network. The converse does not hold: local users cannot log on to any machine in the network; a local user can only log on to the machine on which the account has been created.

Figure 8.8: Summary of Windows 2000 Security (continued)

- Objects can be readily moved from one folder to another. Every object takes on the security constraints of the folder that it resides in. (As an exercise, you are encouraged to discuss the advantages and disadvantages of this policy.)

- For each work-station in a Windows 2000 network, the user has the option of logging on to the work-station (in which case the network is inaccessible), or a network domain.

- In addition to user accounts, Windows 2000 allows for the creation and management of **user profiles**. A Windows 2000 user profile is essentially a working environment for a user account. Three types of profiles are allowed:
  - ✓ Local profiles
  - ✓ Roaming profiles (stored on the domain controller)
  - ✓ Customized (mandatory) profiles (stored on the domain controller)

- Group policies can be set up for local machines as well as servers in a network. These policies can be made to apply to users, computers and domains in the network.

8.13 Summary of Security in RedHat Linux

As a final case, let us briefly examine how system security is structured and administered in RedHat Linux 7 (figure 8.9):

Figure 8.9: Summary of RedHat Linux Security

- Unlike Windows 2000, and similar to IBM i, there is one version of the operating system for both client machines and server machines.

- Users are identified by user accounts. However (unlike windows 2000 and IBM i), only the Super User can create user accounts. The command to be used is **adduser**. User names were traditionally confined to a maximum of 8 bytes; nowadays, long name are supported.

- When a user account is created, a private group automatically created, and the user assigned to that group. The user can be made to belong to other group(s) via the **usermod** command. Each user must belong to at least 1 group. The user group can be changed via the **usermod** command.

- The system is shipped with several (over 20) default user groups; the more common ones are:
  - ✓ **root** — for the super user
  - ✓ **adm** — for system administrators
  - ✓ **rpcuser** — for remote procedure calls
  - ✓ **floppy** — for use of floppy drive
  - ✓ **machines** — for networking machines

  Other groups can be added via the command **groupadd**, and removed via the **groupdel** command. All groups are stored in the file **/etc/group**.

Figure 8.9: Summary of RedHat Linux Security (continued)

- The superior user is called the Super User (or more commonly **root**).
- User accounts are stored in the file **/etc/passwd**. This file in accessible to everyone (including a hacker) — a traditional security loophole. To plug this security hole, the file **/etc/shadow** contains the password for each user name. Only the super user has access to this file.
- When an object (file) is created, the owner has existence rights to that object. He /she can then grant rights to read (R), write (W), or execute (X) the object (file) a user, a group or others. The command used is **chmod** command. This command allows the user to specify such rights in a 9-byte field:
  B1-B3: User specification
  B4-B6: Group specification
  B7-B9: Others specification
- The super user can also change the ownership of an object via the **chown** command. The group that a file belongs to can also be changed via the **chgrp** command.
- The **mv** command allows you to move objects from one directory to another. Objects retain their security settings when moved.
- Linux has an additional security programs called **Tripwire** and **Logwatch** which alert the Super User when files and/or directories have been modified in the system.
- Also RedHat Linux has various shells that the user can choose to work in. The more common ones and their corresponding accession commands are indicated below:
  - ✓ Basic Shell : **sh**
  - ✓ Born Again Shell : **bash**
  - ✓ Bourn Shell : **bsh**
  - ✓ C Shell: **csh**
  - ✓ TC Shell: **tcsh**

8.14 Summary and Concluding Remarks

Here is a summary of what has been covered in this lecture:

- Protection relates to the preventing unwanted interaction between processes. Security has to do with authorized access to the resources of a system.
- Operating system protection/security measures include (but are not confined to) encryption of data, authority constraints, access constraints, privileged instructions, and inter-process protection
- All objects on the system need protection in one form or another. These include files, folders (called directories or libraries on some systems), programs, commands, devices, user accounts (profiles), etc.
- Each process operates within a *protection domain,* which specifies the resources that the process may access. Each domain defines a set of objects and the operations that may be invoked on each object. This ability to execute certain operations on an object is called *access right*.
- The access matrix (also called a capability table), stores access rights to objects in various domains. I may be implemented via the global table approach, the access-list approach, or the capability-list approach.
- Global Table Approach: The access matrix is stored as a global table consisting of the triplets {Domain, Object, Rights-set}.
- Access-list Approach: A list of ordered pairs {Domain, Rights-set} is stored for each object.
- Capability List Approach: A list of ordered pairs {Object, Rights-set} is stored for each domain.
- Traditionally, the access-list approach is the most popular. However, modern database theory allows for benefits of the other approaches to be achieved via logical views.
- The Os security should also include a coherent mechanism for revocation of privileges. The access-list approach is most ideal for this.
- A modern operating system should also include a comprehensive encryption strategy. This is quite expensive to implement; hence, many systems make this optional for the user.

As a case study, we also examined how security is handled in IBM i, Windows 2000, and Redhat Linux 7. Typically, a modern operating system will mirror aspects from these three models. Our next lecture will examine I/O mechanisms for the operating system.

8.15 References and/or Recommended Readings

[Bacon & Harris 2003] Bacon, Jean & Tim Harris. 2003. *Operating Systems: Concurrent and Distributed Software Design*. Addison-Wesley. See Chapter 8.

[LDP 2016] LDP Worldwide. *The Linux Documentation Project*. Accessed June 2016. http://tldp.org

[Nutt 2004] Nutt, Gary. 2004. *Operating Systems: A Modern Perspective* 3rd ed. Boston: Addison-Wesley. See Chapter 14.

[Russel & Crawford 2000] Russel, Charlie & Sharon Crawford. 2000. *Microsoft Windows 2000 Server Administrator's Companion*. Redmond, Washington: Microsoft Press. See Chapter 9.

[Silberschatz 2012] Silberschatz, Abraham, Peter B. Galvin, & Greg Gagne. 2012. *Operating Systems Concepts*, 9th Ed. Update. New York: John Wiley & Sons. See Chapters 14 & 15.

[Stallings 2005] Stallings, William. 2005. *Operating Systems* 5th ed. Upper Saddle River, New Jersey: Prentice Hall. See Chapter 16.