1. Protection system controls access to *objects* by *subjects*. These are defined as:
- *object* - some entity needing protection, e.g., a memory region, file, service translating names to addresses, etc.
- *subject* - active entity attempting to make use of an object (e.g., a process)
- *principal* - a subject acts on behalf of a principal (e.g., human-user or organization)

a) Why would controlling access to an object's operations depend on the object's type?

b) If a process is a subject, what implications does that have on threads of a multi-thread process?

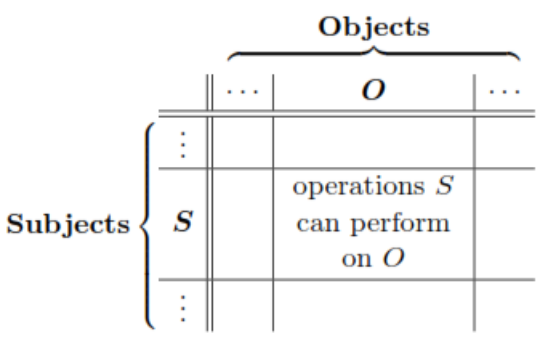
2. The distinguishing feature of a principal is that its rights are completely a question of policy, not of technical mechanism. In the following scenarios determine the principal:

a) UNI policy directs its web server(s) to grant access rights to files in "uni_web" folders to client web browsers with on-campus addresses.

b) UNI policy directs its web server(s) to grant access to Dr. Schafer's Electronic Submission system only after identifying the human sitting at the web browser.

Common OS protection design patterns:
- One common design is for operating systems to give some subjects (e.g., web-server) the union of all access rights it needs for all the principals. The subject is responsible for enforcing more specific protections.
- Another design of the OS protection mechanism would be to give the subject a limited set of access rights (its *protection domain*), but allow it to switch domains to obtain the extra access rights of its principal.
- Allow a subject to switch domains to gain extra access rights that would not normally be available to the principal. For example, Linux/UNIX systems allow a program file to have a special *set user ID (setuid)* bit set. When a principal executing a program with the setuid bit set, the program acquires the protection domain of the program owner and not the principal. (The setuid program can check information about the principal.)

We can conceptualize the dynamic state the protection system as an *access matrix* with one row for each subject (*S*), one column for each object (*O*), and the entry at [*S*, *O*] specifying the *S*'s access rights for *O*.



	F	P_1	P_2	...
P_1	change accessibility		transfer rights	
P_2	change accessibility			
⋮				

Figure 7.11: An access matrix can contain rights that control changes to the matrix itself. In this example, the processes P_1 and P_2 have the right to change the accessibility of file F, that is, to change entries in F's column of the access matrix. Process P_1 also has the right to transfer rights to process P_2, that is, to copy any access right from the P_1 row of the matrix to the corresponding entry in the P_2 row. Notice that the representation of the right to transfer rights relies upon the fact that each subject is also an object.

Since principals (users) have access rights regardless of whether they are running any processes (subjects), principals are added as subjects.

(a)

	F_1	F_2	*JDoe*	P_1	···
JDoe	read	write			
P_1	read	write			
⋮					

(b)

	F_1	F_2	*JDoe*	P_1	···
JDoe	read	write			
P_1			use the rights of		
⋮					

Figure 7.12: If access rights are initially granted to a principal, such as *JDoe*, then there are two options for how those rights can be conveyed to a process, such as P_1, operating on behalf of that principal. In option (a), when the process P_1 is created, all of *JDoe*'s rights are copied to P_1's row of the matrix; in this example, the rights are to read file F_1 and write file F_2. In option (b), P_1 is given just a special right to indirectly use the rights of *JDoe*.

Access matrices can represent very differenct security policies depending on their contents. Two broad categories of policies depending on the allowable modifications of the matrix:

1. *Discretionary Access Control (DAC)* - (used by common OSs: Linux, Microsoft Windows, etc.)
   - each object is considered to be owned by a principal
   - when your process creates an object (e.g., a file), you become the owner of the object
   - the owner of an object has broad rights to control an object's accessibility. For example, the owner of a file decides whether to let other users read or write the file.
2. *Mandatory Access Control (MAC)* -
   - an object's creator does not obtain control over access rights to the object
   - access rights are determined by an explicit security policy and changed within the parameters of that policy (For example, a principal that creates a "classified document" cannot share it with unclassified users)

a) Which approach is used by common OSs: Linux, Microsoft Windows, etc. ?

Two techniques used to keep track of access rights (independent of policy):

- *Capability* - an indirect reference to an *object* that includes its location ("like a pointer") and a set of access rights. (*handle* used by Microsoft, and *descriptor* used by POSIX systems)

a) How does this notition of a capability translate to an access matrix? A process P_1 has the capability to read file F_1 and process P_2 has the capability to read and write file F_1.

- *Access Control Lists (ACL) and Credentials* - (a column-wise slice of the access matrix) for each object maintain, a list of principals (users or groups of users) and their access rights.

Where an OS Storages of a process capabilities:

- In a special storage area (C-list) independent of normal virtual address space of the process (*handle tables* in Windows and *descriptor tables* in POSIX)
- In the process virtual address space (IBM iSeries)

a) Why are system calls needed to put an entry into the C-list or manipulate an entry?

b) Entries in the C-list are referenced by integer indexes. For example, an operation to open a file for reading adds an entry to the C-list for the process and returns the integer index of that entry. The integer handle or descriptor can be used to access the file. When a program opens, uses, and closes a file, how is the C-list used?

c) If capabilities are stored in the virtual address space of the process, how can the process be prevented from forging capabilities?

Most real OSs:

- use a combination of capability-based protection with access control lists.
- Capabilities are selectively granted, but not selectively revoked. (e.g., a process with an open file, cannot have its access revoked by the owner of the file)

Access Control Lists (ACL) and Credentials - (a column-wise slice of the access matrix) for each object maintain, a list of principals (users or groups of users) and their access rights.

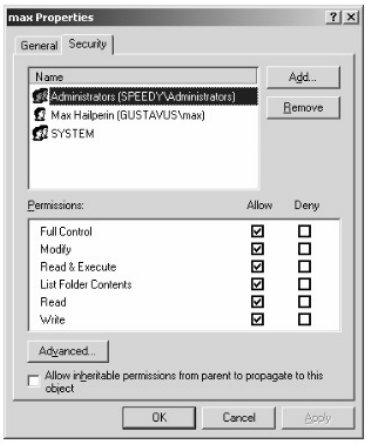


Figure 7.13: This is the initial dialog box summarizing a Microsoft Windows ACL, found in the Security tab of a Properties dialog box.

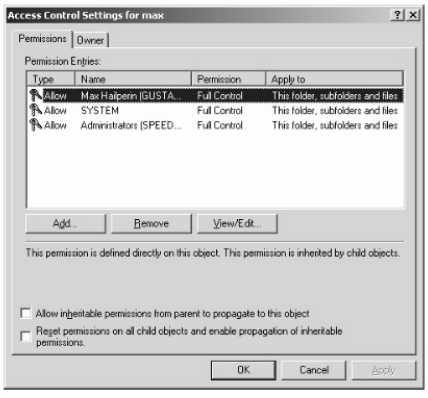


Figure 7.14: Clicking the Advanced button on the dialog box shown in Figure 7.13 produces this dialog box, which in turn gives you the opportunity to click the View/Edit button to obtain the detailed view shown in Figure 7.15.

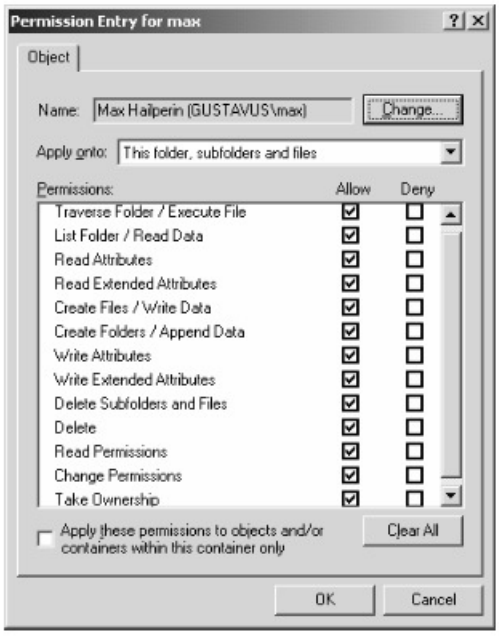


Figure 7.15: This detailed view of a Microsoft Windows ACL entry allows you to see that Full Control really is a summary name for thirteen different permissions.

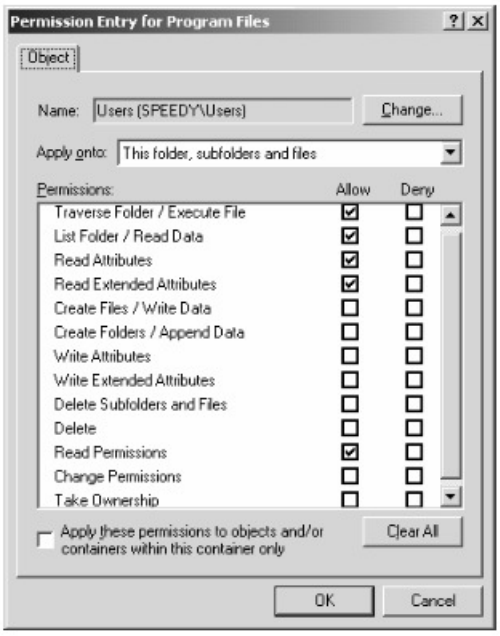


Figure 7.16: In the Microsoft Windows ACL entry shown in this detailed view, some permissions are neither allowed nor denied. In this circumstance, other ACL entries are allowed to control access.