Operating System Security

Ahmet Burak Can

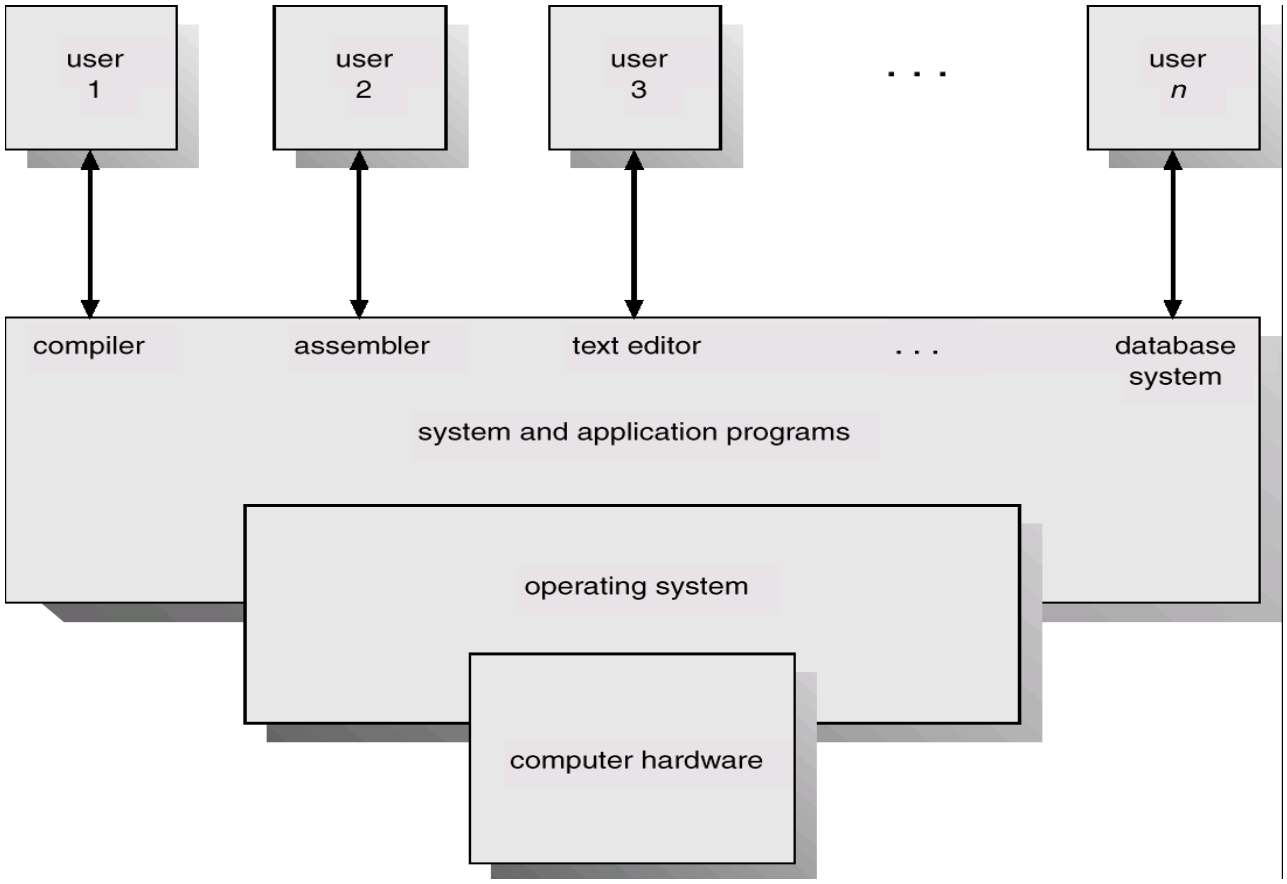
Hacettepe University

abc@hacettepe.edu.tr

Computer System Components

- Hardware
  - Provides basic computing resources (CPU, memory, I/O devices).
- Operating system
  - Controls and coordinates the use of the hardware among the various application programs.
- Applications programs
  - Define the ways in which the system resources are used to solve the computing problems of the users.
- Users
  - E.g., people, machines, other computers.

Abstract View of System Components



What Security Goals Does Operating System Provide?

- Goal 1: enabling multiple users securely share a computer
  - Separation and sharing of processes, memory, files, devices, etc.
- How to achieve it?
  - memory protection
  - processor modes
  - authentication
  - file access control

What Security Goals Does Operating System Provide?

- Goal 2: ensure secure operation in networked environment
- How to achieve it?
  - Authentication
  - Access Control
  - Secure Communication (using cryptography)
  - Logging & Auditing
  - Intrusion Prevention and Detection
  - Recovery

CPU (Processor) Modes or Privileges

- Generally, two basic CPU modes are available: System mode, User mode
- System mode (privileged mode, master mode, supervisor mode, kernel mode)
  - can execute any instruction and access any memory locations, e.g., accessing hardware devices, enabling and disabling interrupts, changing privileged processor state, accessing memory management units, modifying registers for various descriptor tables .

Reading: http://en.wikipedia.org/wiki/CPU_modes

CPU (Processor) Modes or Privileges

- User mode
  - access to memory is limited, cannot execute some instructions
  - cannot, e.g., disable interrupts, change arbitrary processor state, access momory management units
- Transition from user mode to system mode must be done through well defined call gates (system calls)

System Calls

- Guarded gates from user mode (space, land) into kernel mode (space, land)
  - use a special CPU instruction (often an interruption), transfers control to predefined entry point in more privileged code; allows the more privileged code to specify where it will be entered as well as important processor state at the time of entry.
  - the higher privileged code, by examining processor state set by the less privileged code and/or its stack, determines what is being requested and whether to allow it.

Reference: http://en.wikipedia.org/wiki/System_call

Kernel Space vs User Space

- Part of the OS runs in the kernel model
  - known as the OS kernel
- Other parts of the OS run in the user mode, including service programs (daemon programs), user applications, etc.
  - they run as processes
  - they form the user space (or the user land)

User Authentication

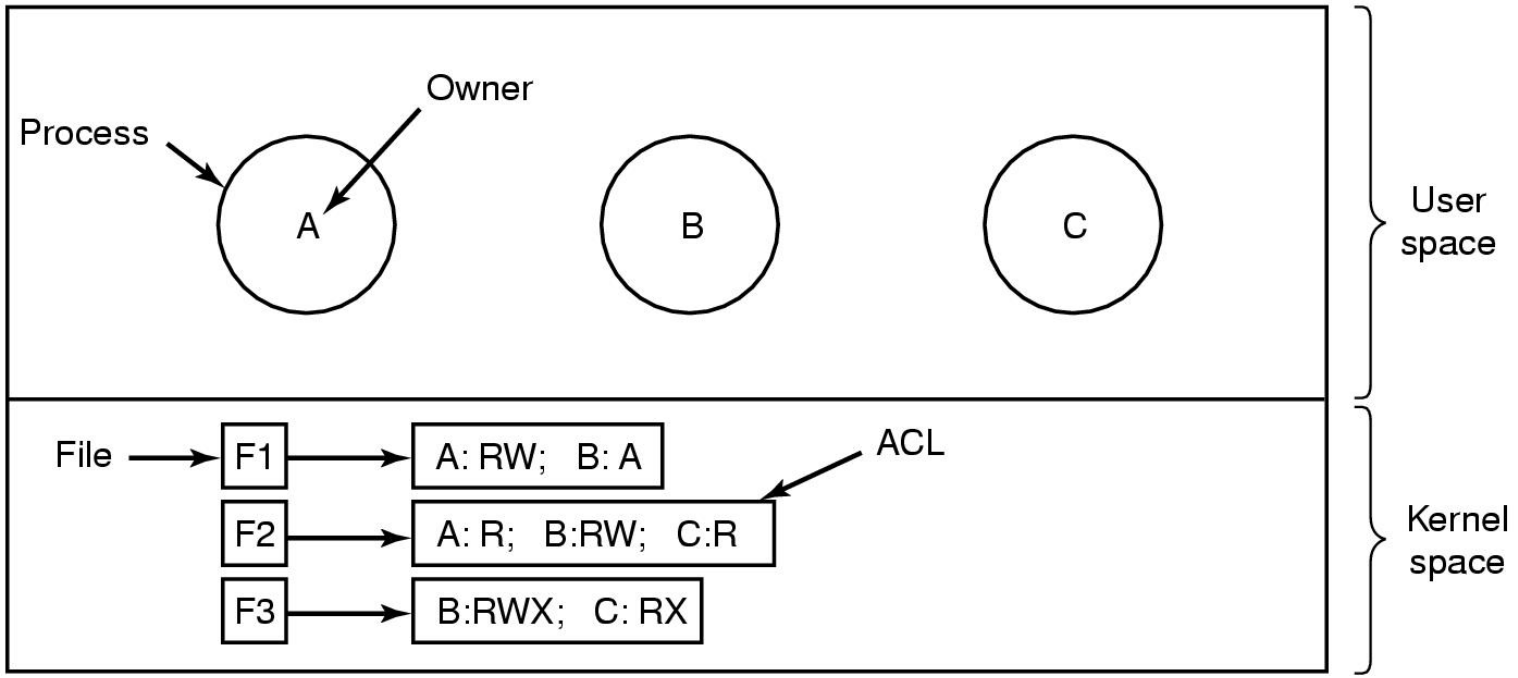
- Using a method to validate users who attempt to access a computer system or resources, to ensure they are authorized
- Examples
  - User accounts with passwords
    - something you known
  - Smart cards or other security tokens
    - something you have
  - Biometrics
    - something you are

Access Control

- Reference Monitor Concept
  - Mediates all security-sensitive operations
    - Process Creation, File Access, …
- Subjects
- Objects
- Access

Access Control Lists

- Each file is associated a set rules defining the access permissions of users on the file.



Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

- Each user has a unique UID
- Users belong to multiple groups
- Processes are subjects
  - associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid)
- Objects are files: each file has the following information
  - owner
  - group
  - 12 permission bits
    - read/write/execute for user, group, and others,
    - suid, sgid

Basic Permissions Bits on Files (Non-directories)

- Read bit controls reading the content of a file
  - i.e., the read system call
- Write bit controls changing the content of a file
  - i.e., the write system call
- Execute controls loading the file in memory and execute
  - i.e., the execv system call

The Three Sets of Permission Bits

- UNIX classifies three sets of permission bits for files:
  - user, group, other
- When a user wants to access a file:
  - if the user is the owner of a file, then the r/w/x bits for owner apply
  - otherwise, if the user belongs to the group the file belongs to, then the r/w/x bits for group apply
  - otherwise, the r/w/x bits for others apply

UNIX Permission Bits for Files

- Example:

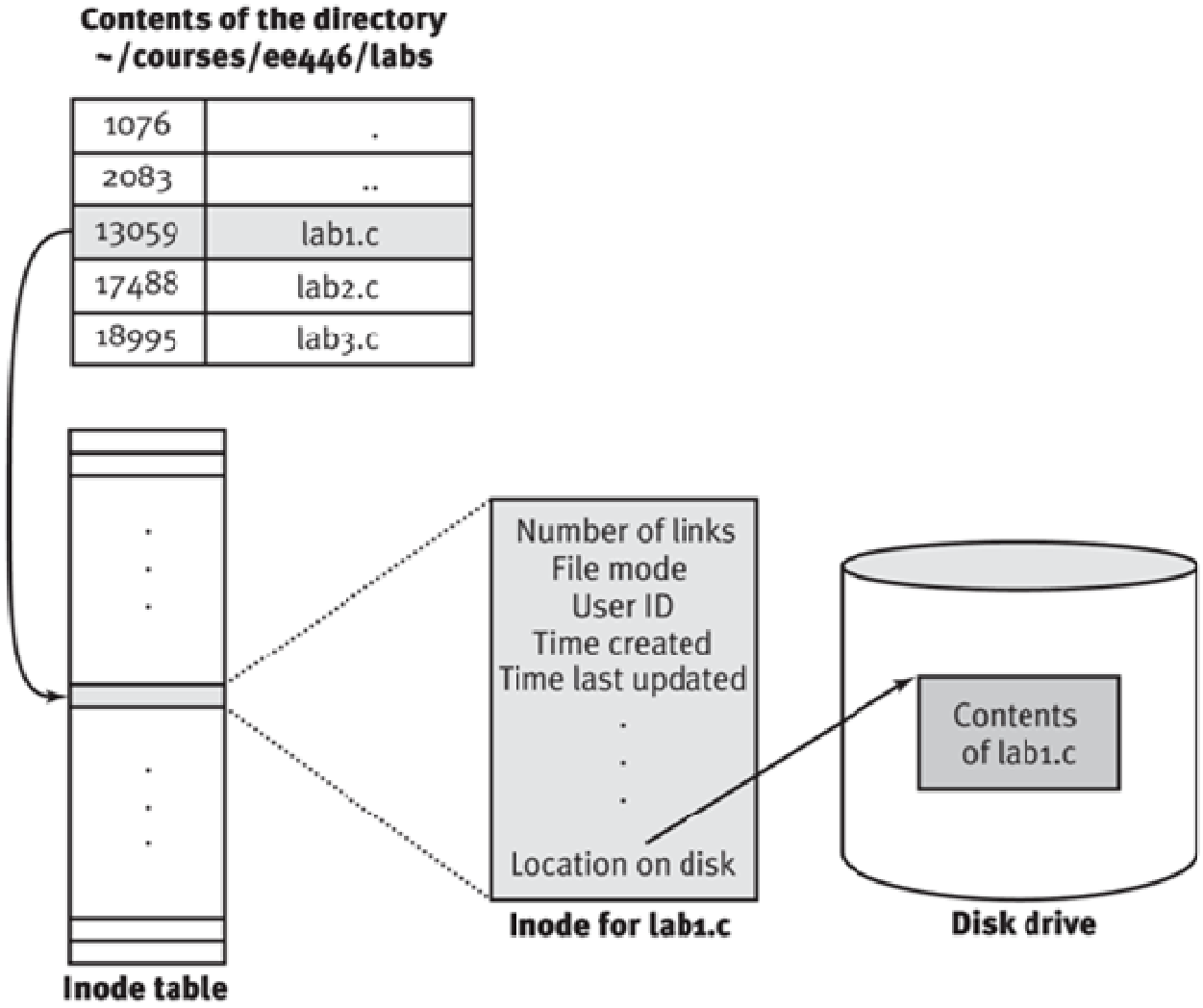
```
$ ls -l
-rwxr-xr--+  2 abc  akd  4096 May  3 11:54 a.txt
```

  - Permissions for a.txt:
    - User has r/w/x permissions
    - Group has r/x permissions
    - Others has r permission

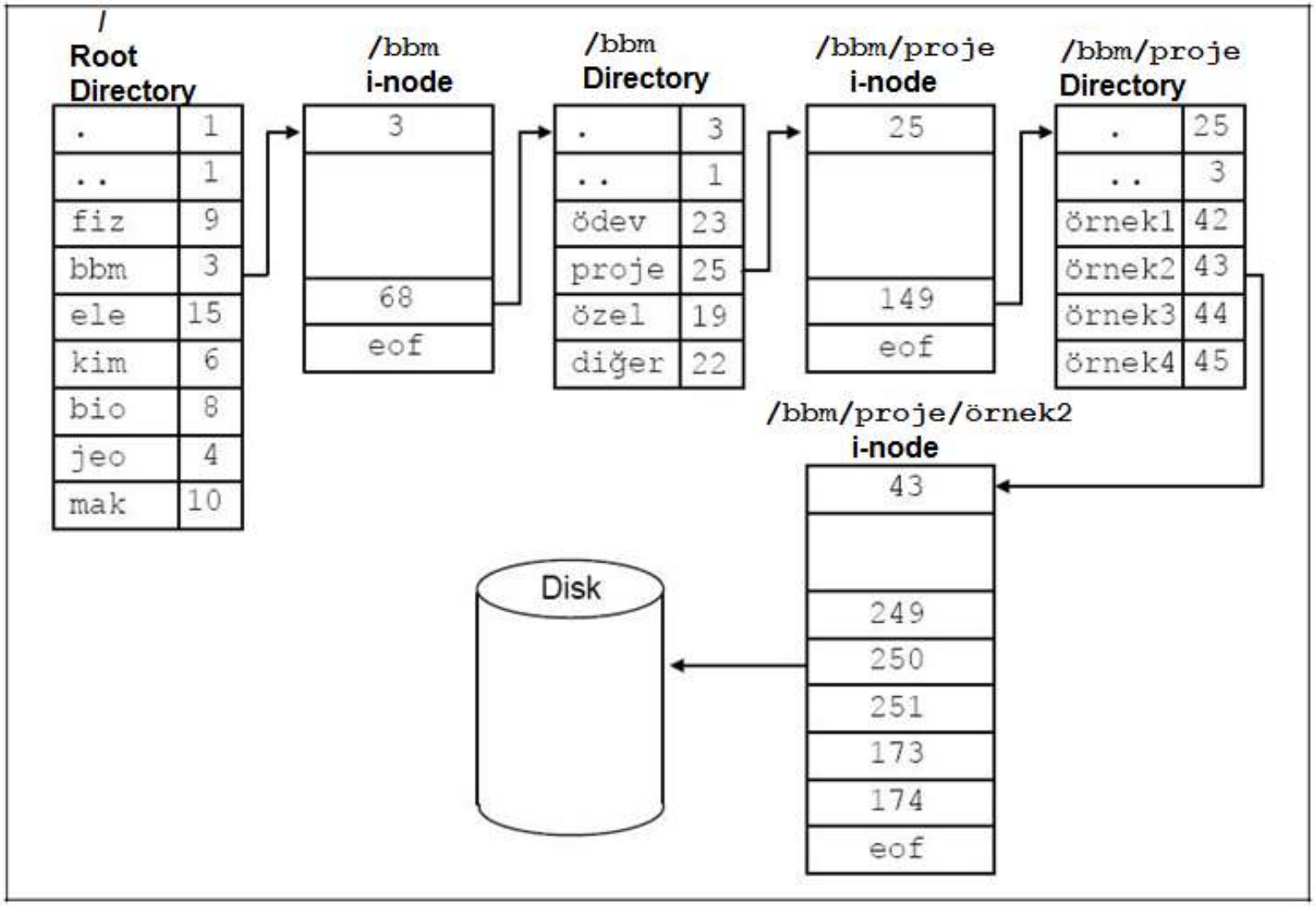
Execution of a File on UNIX

- Binary file vs. script file
- Having execute but not read, can one run a file?
  - Yes
- Having execute but not read, can one run a script file?
  - No
- Having read but not execute, can one run a script file?
  - No

UNIX Directories and *i-node*



UNIX Directory Traversal



UNIX Permission Bits on Directories - I

- Read bit allows one to show file names in a directory

```
$ ls -l
dr--------   2 abc  akd  4096 May  2 23:33 dir

$ ls dir
file1  file2

$ ls -l dir
total 0
?---------  ? ? ? ?            ? file1
?---------  ? ? ? ?            ? File2

$ ls -l dir/file1
ls: dir/file1: Permission denied

$ cd dir
-bash: cd: dir: Permission denied
```

UNIX Permission Bits on Directories - 2

- The execution bit controls traversing a directory
  - does a lookup, allows one to find inode number from file name
  - chdir to a directory requires execution
  - Accessing a file identified by a path name requires execution to all directories along the path

UNIX Permission Bits on Directories - 3

- Example: Usage of execution bit on a directory

```
$ ls -l
d--x------   2 abc  akd  4096 May  2 23:33 dir

$ ls dir
ls: dir: Permission denied

$ ls -l dir
ls: dir: Permission denied

$ ls dir/file1
dir/file1

$ ls -l dir/file1
-rw-r--r--  1 abc akd 0 May  2 23:33 dir/file1

$ cd dir
$ ls
ls: .: Permission denied
```

UNIX Permission Bits on Directories - 4

- Write + execution control creating/deleting files in the directory
  - Deleting/creating a file under a directory requires no permission on the file.
  - To update a file under a directory, write permission on the directory is not required, only execution permission on the directory is enough.

UNIX Permission Bits on Directories - 5

- Example: Usage of execution and write bits on a directory

```
$ ls -l
d--x------+  2 abc  akd  4096 May  2 23:33 dir

$ ls -l dir/file1
----------+ 1 abc akd 0 May  2 23:33 dir/file1

$ rm dir/file1
rm: cannot remove `dir/file1': Permission denied

$ chmod 300 dir
$ ls -l
d-wx------+  2 abc  akd  4096 May  2 23:33 dir

$ rm dir/file1
$ ls -l dir/file1
ls: dir/file1: No such file or directory
```

Some Examples

- What permissions are needed to access a file/directory?
  - read a file: /d1/d2/f3
  - write a file: /d1/d2/f3
  - delete a file: /d1/d2/f3
  - rename a file: from /d1/d2/f3 to /d1/d2/f4

Process User ID Model in Modern UNIX Systems

- Each process has two user IDs
  - real user ID (ruid): owner of the process
  - effective user ID (euid): user ID which affects the most access control decisions
- and two group IDs
  - real group ID: original group of the process
  - effective group ID: group ID which affects the most access control decisions

The Need for suid/sgid Bits

- System integrity requires more than controlling who can write, but also how it is written
- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on "privileged ports" (TCP/UDP ports below 1024)
  - changing password