# Introduction to OS Security

Bo Luo
bluo@ku.edu

# Introduction

- OS: still software
  - All software security vulnerabilities still apply
- OS must protect users from each other
  - memory protection 存储保护
  - file protection
  - general control and access to objects
  - user authentication

# Introduction

- The fundamental tradeoff of OS security
  - operating systems tradeoff between:
  - *Sharing*
  - *Protection*

- sharing is desirable
- protection is difficult

# Introduction

- Early History
  - no OS
  - programs entered directly in binary through switches
  - user's program only one on system
  - user responsible for:
    - loading dependent libraries, other tools
    - scheduling time to use computer
  - OS security?

每个软件用完一次，才能开下一个

# Introduction

- Later
  - machines very expensive
  - people less expensive
  - maximize use of machine
  - allow many users

# Introduction

- OS protection – separation.
- Physical separation, e.g. 1 user/printer
- Temporal separation 时间分离
- Logical separation: user thinks own machine
- Cryptographic separation

- Combinations of these

# Levels of Protection

- no protection
- isolation
- share all or nothing
- share via access limitation
- share by capabilities
- limit use of an object

# Levels of Protection

- No protection
  - *e.g.* early versions of windows
  - some embedded environments

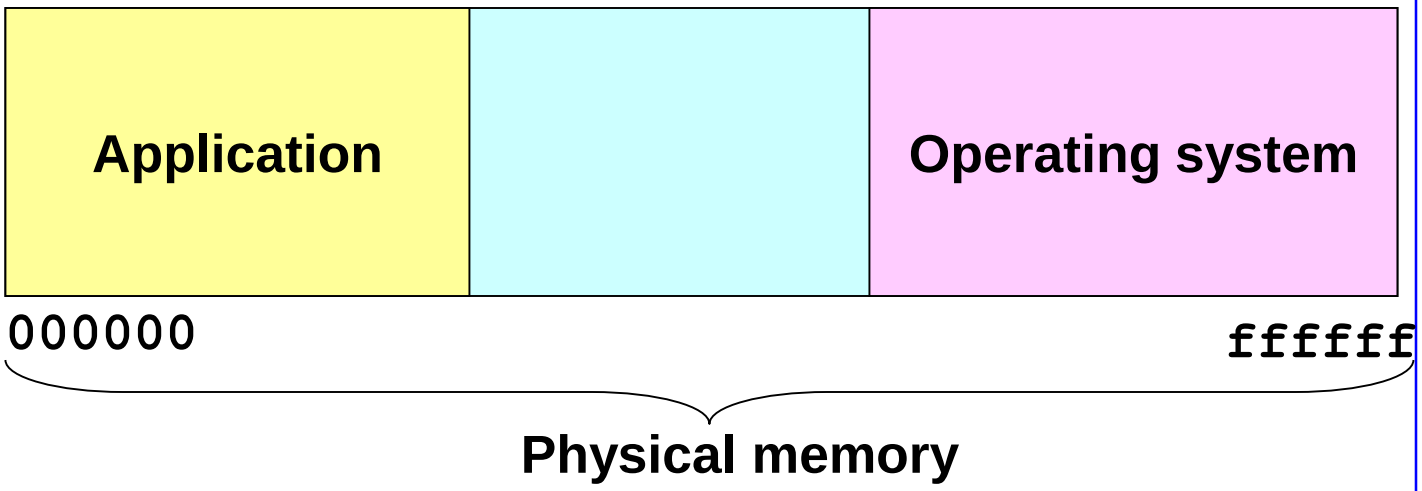
  - designed for one user
  - no need for isolation, access control, etc.

# Uniprogramming w/o memory protection

- Simplest.
  - Each application runs within a hardwired range of physical memory addresses
- One application runs at a time
  - Application can use the same physical addresses every time, across reboots

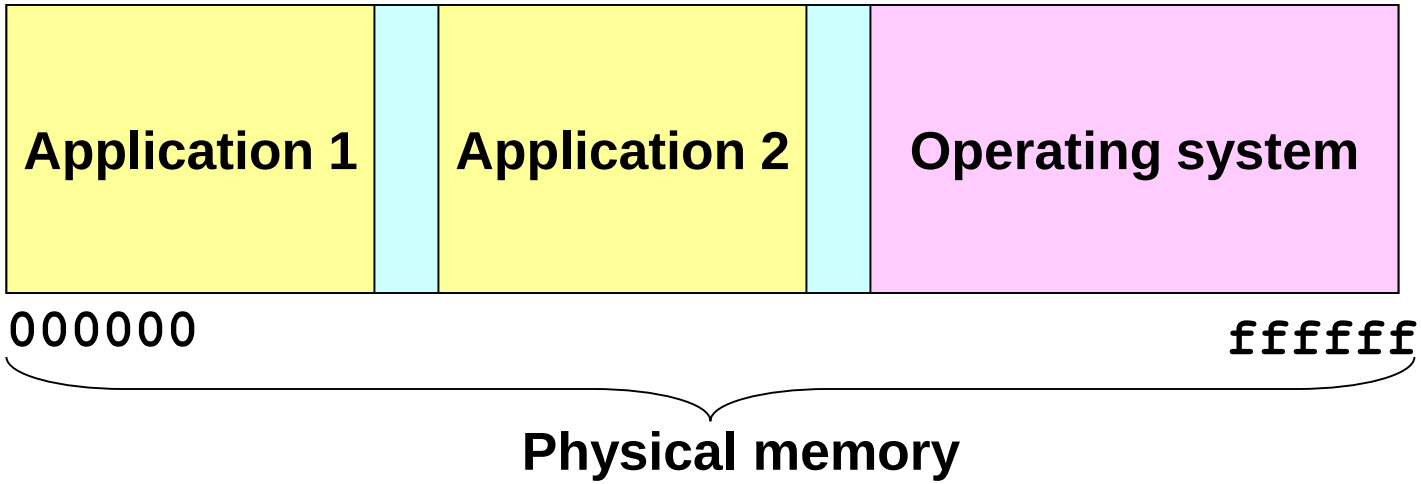
# Uniprogramming w/o memory protection

- Applications typically use the lower memory addresses
- An OS uses the higher memory addresses
- An application can address any physical memory location



# Levels of Protection

- Isolation
  - processes unaware of other processes
  - each process: own address space, files, *etc.*
  - OS provides confinement
  - Virtual machines



# Levels of Protection

- Share all or nothing
  - owner of object declares it: share 所有人
  - *Public:* available to all users
  - *Private:* not available

# Levels of Protection

- Share via access limitation
  - Resource/files are shared
  - Who can access what?
  - Access control lists
  - Access Control Matrices
  - Capabilities

# Levels of Protection

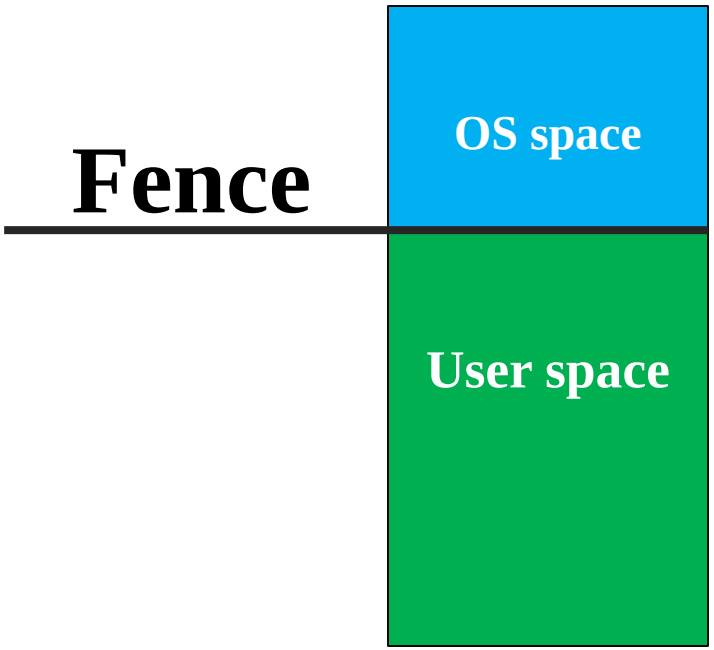
- Limit use of an object 权限限制更小
  - Sophisticated, fine-grained access control
  - Examples:
  - can view a file, but can't print
  - given aggregate info from database, but not individual records

# Memory and Access Protection

- In a multi-user multi-task environment, what's in the memory?
  - OS, processes from different users
  - Data (keys!) in plaintext!
- Memory management
  - Fences
  - Relocation
  - Base/Bounds Registers
  - Tagged Architecture
  - Segmentation
  - Paging
  - Combined Paging with Segmentation

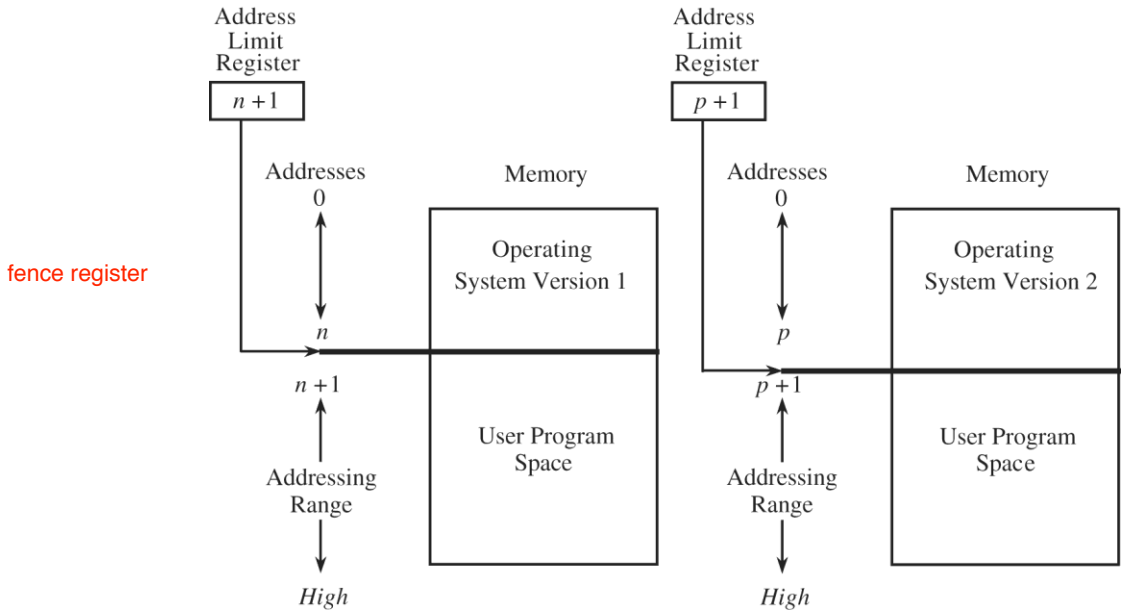
# Memory and Access Protection

- Fences: protect OS from user program
  - confine users to one side of a boundary
  - predefined memory address: user code on one side, OS on the other




# Memory and Access Protection

- Fences: protect OS from user program
  - Problem?
    - fixed boundary too restrictive
    - doesn't protect users from each other
  - moveable fence:
    - store fence location in fence register



# Memory and Access Protection

- Relocation address 不连续性
  - programs written to run starting at address 0
  - can be run at any address
  - addresses in source are symbolic:
    - e.g., numStudents
  - compiler binds these to relocatable addresses.
    - e.g. 20 bytes from beginning of module func
  - then linker or loader binds to absolute addresses
    - e.g. 20114
  - logical addresses mapped to physical by MMU
  - program never sees real addresses

# Memory and Access Protection

- We skip the details of segmentation and paging.
- They have been covered in your OS class.

# Memory and Access Protection

- Wrap-up
  - Each process
    - has its own address space
    - thinks it's the only process on machine
  - MMU provides translation between process's address space and physical space
  - process cannot generate address not in its own space

# Memory and Access Protection

- Relocation
  - Offer protection if the translation tables cannot be altered by applications
  - An application can only touch its address space under the user mode
  - Hardware requires the CPU to be in the kernel mode to modify the address translation tables

# Memory and Access Protection

- Switching from Kernel mode to User mode
  - To run a user program, the kernel:
  - Creates a process and initialize the address space
  - Loads the program into the memory
  - Initializes translation tables
  - Sets the hardware pointer to the translation table
  - Sets the CPU to user mode
  - Jumps to the entry point of the program

# Memory and Access Protection

- Switching from User mode to Kernel mode
- Voluntary
  - System calls: a user process asks the OS to do something on the process's behalf
- Involuntary
  - Hardware interrupts (e.g., I/O)
  - Program exceptions (e.g., segmentation fault)

# Memory and Access Protection

- For all cases, hardware atomically performs the following steps
  - Sets the CPU to kernel mode
  - Saves the current program counter
  - Jumps to the handler in the kernel
    - The handler saves old register values

# Memory and Access Protection

- Context switching between processes
  - Need to save and restore pointers to translation tables
- To resume process execution
  - Kernel reloads old register values
  - Sets CPU to user mode
  - Jumps to the old program counter

# Memory Attacks

- Motivation
  - You can encrypt HD, USB drives, network traffic, etc.
  - You cannot encrypt memory!
  - A lot of sensitive information: keys, passwords, etc.
- Two categories of memory attacks
  - Software attacks
  - Hardware attacks

# Memory Attacks

- Software attacks
  - System bugs: allow a process to read any address
  - Swap and dump: memory contents are written to hard drives: swap, core dump, hibernation, crash reports, etc
    - Attackers: trigger a core dump and examine the dump file, looking for keys.
    - It has been reported that core dumps of FTP servers and email servers contained passwords.
    - Hypervisors: suspend the current state of a VM to a check point file.
  - Uncleared buffers

交换和储存

# Memory Attacks

- Physical attacks
  - Bypass OS, bypass CPU
  - Directly read from RAM
  - Cold boot attacks
    - The remanence effect of RAM: the contents in RAM fade away gradually after power off in several minutes or hours (low temperature)
    - Attacker: reboot the computer with OS from USB drive; or move the RAM chips to another machine
    - Read from RAM – reduce the temperature so that data stays longer
    - Completely bypass access control, encryption, task isolation, authentication, etc
  - DMA: direct memory access

# Control of Access to General Objects

- control of any kind of object
- examples:
  - memory
  - secondary storage
  - hardware devices
  - some data structure
  - instructions
  - passwords and user-authentication mechanism
  - the protection mechanism itself

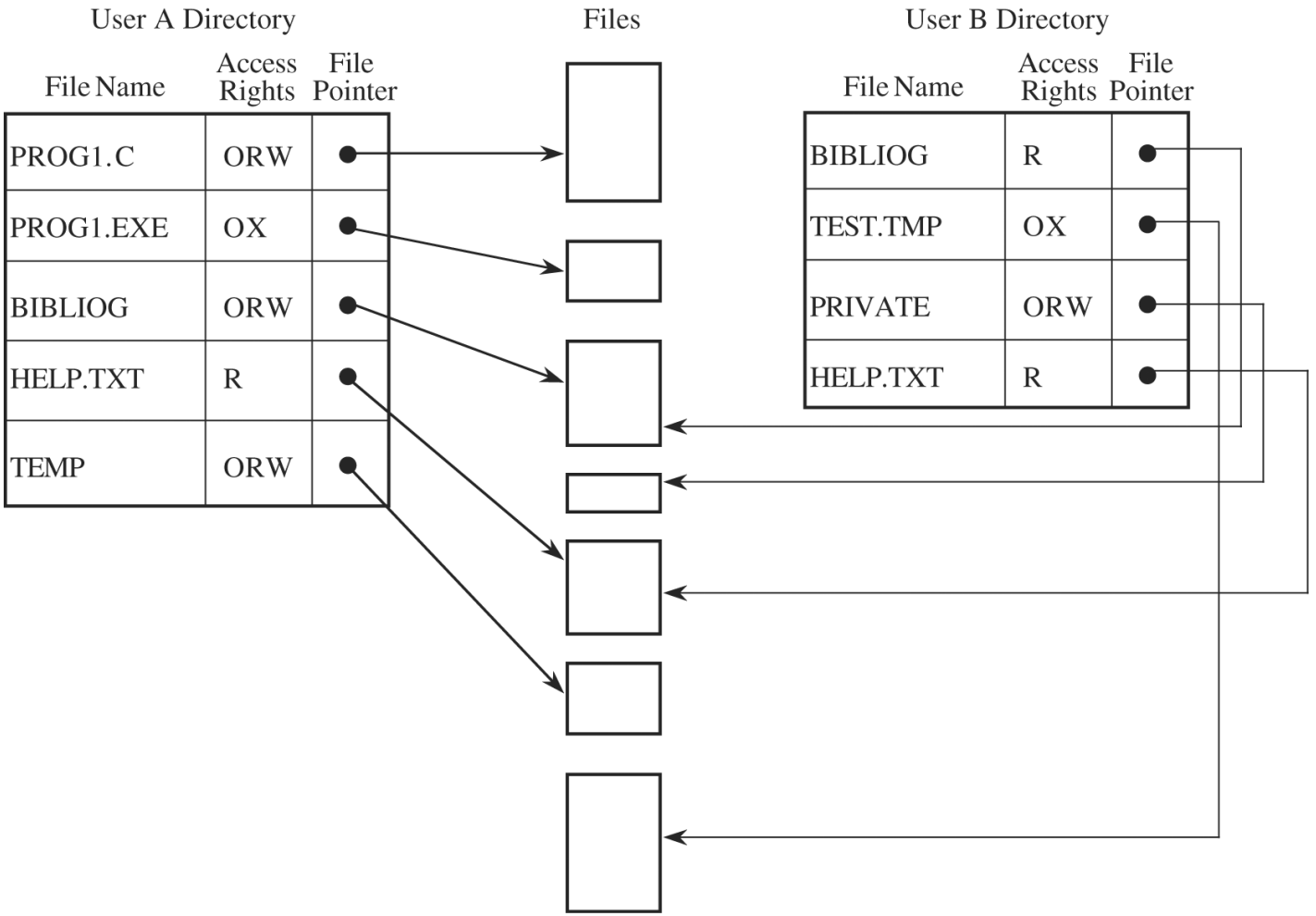
# Control of Access to General Objects

- Goals in protecting objects
  - check every access
    - user permitted doesn't mean always permitted
  - enforce least privilege
    - grant access to minimum set of objects required to complete a task
  - verify acceptable usage
    - stack: push(), pop(), …
    - Shouldn't be able to do anything else to stack

# Control of Access to General Objects

- Directory
  - not directory as in FS directory
  - each user has a list (directory) of objects the user owns or has access to
  - no user should be able to write to the directory
  - for each file, directory contains list of permissions, e.g. R, W, X, and owner

# Control of Access to General Objects



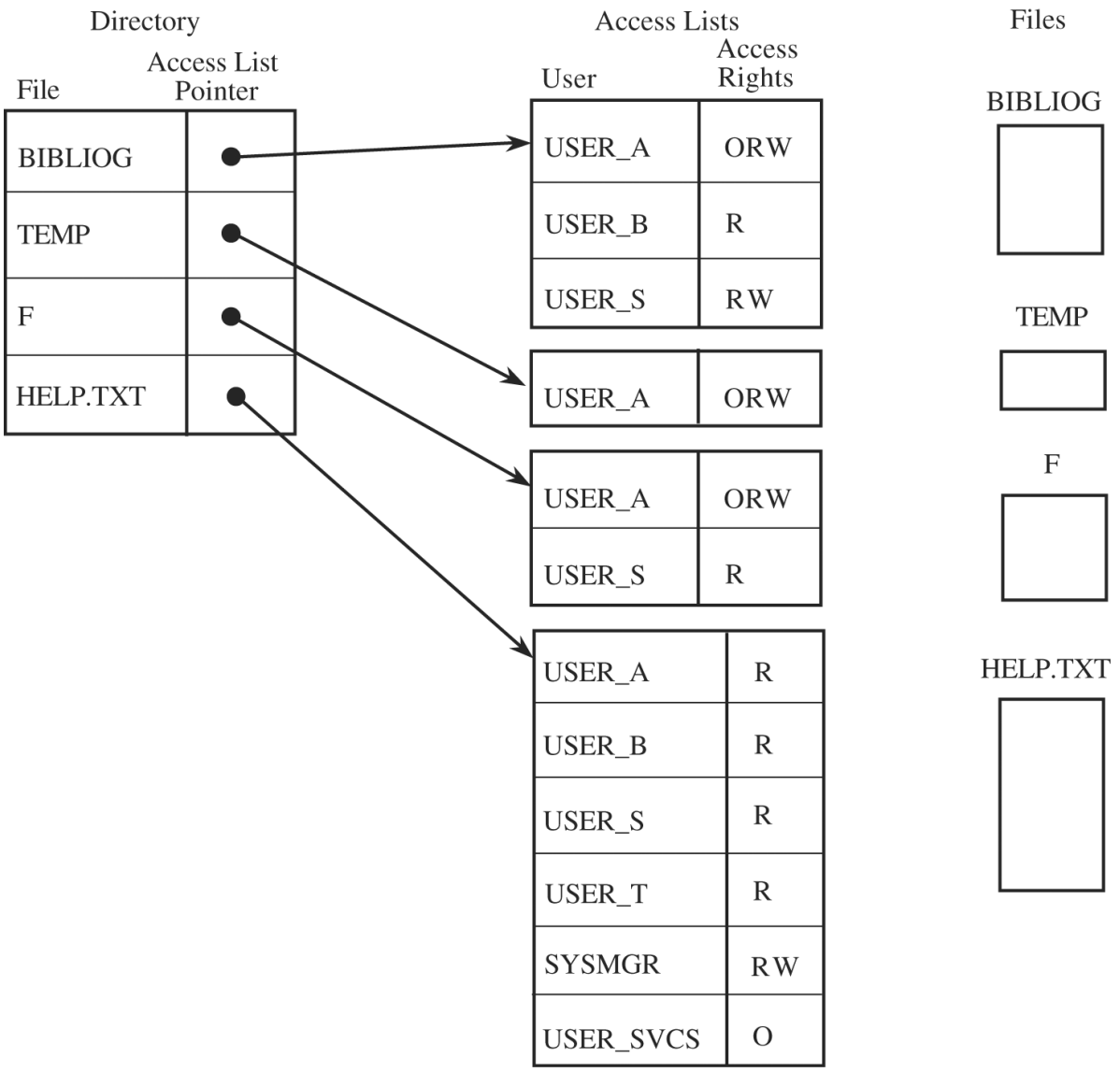
# Control of Access to General Objects

- Directory
  - simple *but*
  - lists can get very long
    - what about shared libraries, programs?
  - same item in many lists
  - revoking permissions?
    - have to go through everyone's lists

# Control of Access to General Objects

- Access control lists
  - maintain a list per object, not user
  - use wildcards (*) to grant permission to a group
    - e.g. administrator-*

# Control of Access to General Objects



# Control of Access to General Objects

- Access control matrix
  - row for each user
  - column for each protected object

  - simple lookups
  - but probably lots of empty spaces

| | BIBLIOG | TEMP | F | HELP.TXT | C_COMP | LINKER | SYS_CLOCK | PRINTER |
|---|---|---|---|---|---|---|---|---|
| USER A | ORW | ORW | ORW | R | X | X | R | W |
| USER B | R | - | - | R | X | X | R | W |
| USER S | RW | - | R | R | X | X | R | W |
| USER T | - | - | - | R | X | X | R | W |
| SYS_MGR | - | - | - | RW | OX | OX | ORW | O |
| USER_SVCS | - | - | - | O | X | X | R | W |