IA 313: Operating Systems Security

Lecture 2: Protecting System Resources

Basis of OS Protection

- An OS allows different users to access different resources in a shared way
- The OS needs to control
  - the sharing and
  - provide an interface to allow the access
- OS also protects users from each other
  - Attacks, mistakes, resource overconsumption
- Even for a single-user OS, protecting a user from him/herself is a good thing
  - Mistakes, malware

Basis of OS Protection …

- *OS Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

Basis of OS Protection …

- Following are the major activities of an operating system with respect to protection:
  - The OS ensures that all access to system resources is controlled.
  - The OS ensures that external I/O devices are protected from invalid access attempts.
  - The OS provides authentication features for each user by means of passwords.

Security methods in OSs

- Basis of security in OS: separation
  = keeping one user's objects secure from interference by other users
  = keeping one user's objects separate from other users
- Kinds of separation:
  1) Physical separation
     - Different processes use different physical objects
       - E.g., different printers for different 'confidentiality levels' of output
  2) Temporal separation
     - Processes having different security req's executed at different times
  3) Logical separation (process appears to be alone)
     - Illusion that OS executes processes only for single user
  4) Cryptographic separation
     - Processes conceal their data and computations from other processes
  5) Combinations of the above

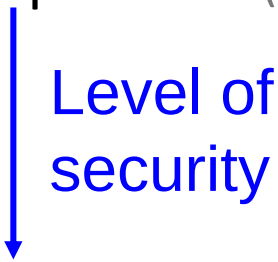
Security methods in OSs …

- Strength of security via separation (least to most secure):

  Logical separation

  Temporal separation

  Physical separation

- Complexity of implementation of separation (least to most complex):

  Physical separation

  Temporal separation

  Logical separation

  Cryptographic separation

- Resource utilization in different kinds of separation:
  - Poor: physical separation / temporal separation
  - Good: logical separation / cryptographic separation

Sharing

- Sometimes, users want to share resources without compromising security
  - Library routines (e.g., libc)
  - Files or database records
- OS should allow *flexible sharing*, not "all or nothing"
  - Which files or records? Which part of a file/record?
  - Which other users?
  - Can other users share objects further?
  - What uses are permitted?
    - Read but not write, view but not print (Feasibility?)
    - Aggregate information only
  - For how long?

Levels of protection in OSs

- Absolute separation reduces efficiency
  - need to share some resources for efficiency
- Full sharing-separation spectrum = levels of protection by OS:

1) No protection
   - *Caveat emptor*
   - User can still protect self by, e.g, temporal separation

2) Isolation
   - Concurrently running processes hidden from each other
     => unaware of each other
     - Own address space, files, other objects for each process

3) Full sharing or no sharing
   - Object/resource owner declares it as:
     - public (can be shared by all)

   or

     - private (not shared)

Levels of protection in OSs …

4) Sharing via access limitation

- Access to each object by each user determined by access rights

5) Sharing by *capabilities*

- Extension to „ Sharing via access limitation”

  — dynamic access rights

- Can be changed by owner, subject, computation context, object itself

6) Limited object use

- Limits not only object *access* — limit object *use*
  - E.g., can view a doc *but* can't copy it
  - E.g., can view statistical summary of data *but* can't view individual data records (e.g., can see average salary but not John Smith's salary)

Levels of protection in OSs …

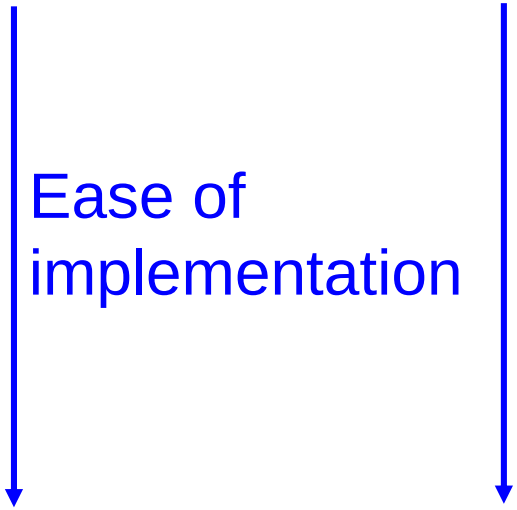
- OS can provide different levels of protection for different objects/resources

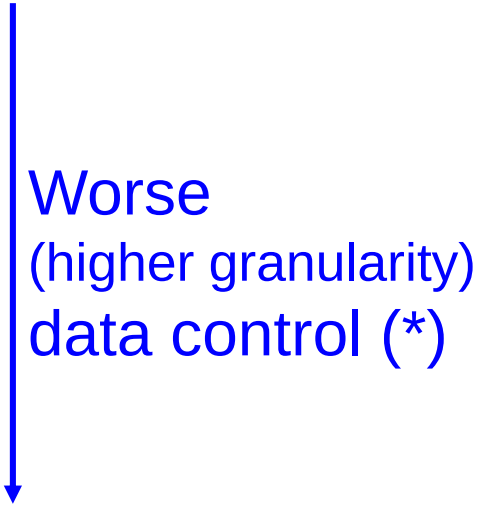
- Complexity of implementation and *fineness* of protection:
  1) No protection
  2) Isolation
  3) Full sharing or no sharing
  4) Sharing via access limitation
  5) Sharing by capabilities
  6) Limited object use



Granularity of *data* protection

- Granularity of data protection
  - Aplicable only to data
  - Protect by:
    - Bit
    - Byte
    - Element/word
    - Field
    - Record
    - File
    - Volume





(*) If no control at proper granularity level, OS must grant access to more data than necessary

E.g., if no field-level data control, user must be given whole record

Memory and Address Protection

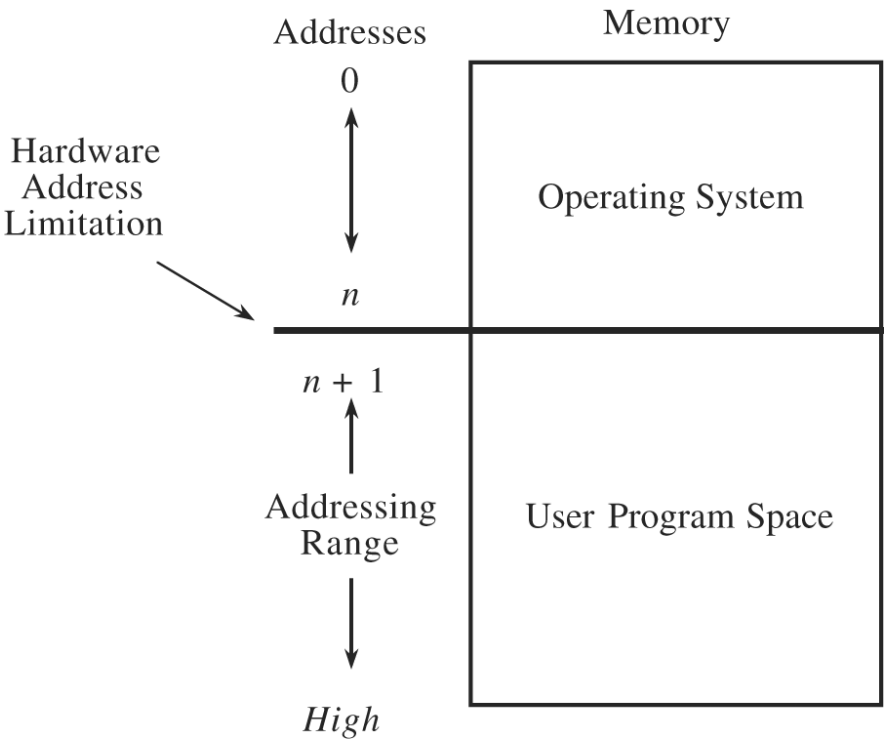
- Prevent program from corrupting other programs or data, operating system and may be itself
  - Often, the OS can exploit hardware support for this protection, so it's cheap
- Memory protection is part of translation from virtual to physical addresses
  - Memory management unit (MMU) generates exception if something is wrong with virtual address or associated request
  - OS maintains mapping tables used by MMU and deals with raised exceptions

Memory and Address Protection …

- Ensures that one user's process cannot access others' memory
- Protection Techniques
  - Fence
  - Relocation
  - Base/bounds register
  - Segmentation
  - Paging
  - …
- Operating system and user processes need to have different privileges

Fence

- Introduced in single-user operating systems
- Goal
  - Prevent a faulty user program from destroying part of the resident portion of the OS
- Methods
  - Predefined memory address
  - Fence register



Fixed fence

Fence …

- Fence register
  - Exception if memory access below address in fence register
  - Protects operating system from user programs
  - Single user only

Relocation

- Assuming a fixed size OS is inconvenient
- Relocation: The process of address translation
  - All programs begin at address 0
  - When loading a program into memory, change all the addresses to reflect the actual address at which the program is located
- Can be done by adding a constant relocation factor to each address of the program
- Fence register can be used for relocation

Base/Bounds Registers

- A relocation register is also called a base register, since it provides a base (or starting address) for programs
  - All addresses inside a program are offsets from the base address
  - Provides a lower bound of the program address
- Bounds register
  - Provides an upper address limit
  - Helpful for checking overflows into “forbidden” areas
- Base/bounds registers must be changed during context switch (execution changes from one program to another)

Base/bounds register pair

- Base/bounds register pair
  - Exception if memory access below/above address in base/bounds register
  - Different values for each user program
  - Maintained by operating system during context switch
  - Limited flexibility

Tagged Architecture

- Every word of machine memory has one or more extra bits to identify the access rights to that word.
- These bits can be set by privileged instructions
- The bits are tested every time an instruction accesses that location
- Has been used in a few systems
- Expensive

Tagged Architecture …

- A variation
  - One tag applies to a group of consecutive locations (e.g., 128 bytes)
  - Much reduced cost
  - Allow more bits in a tag

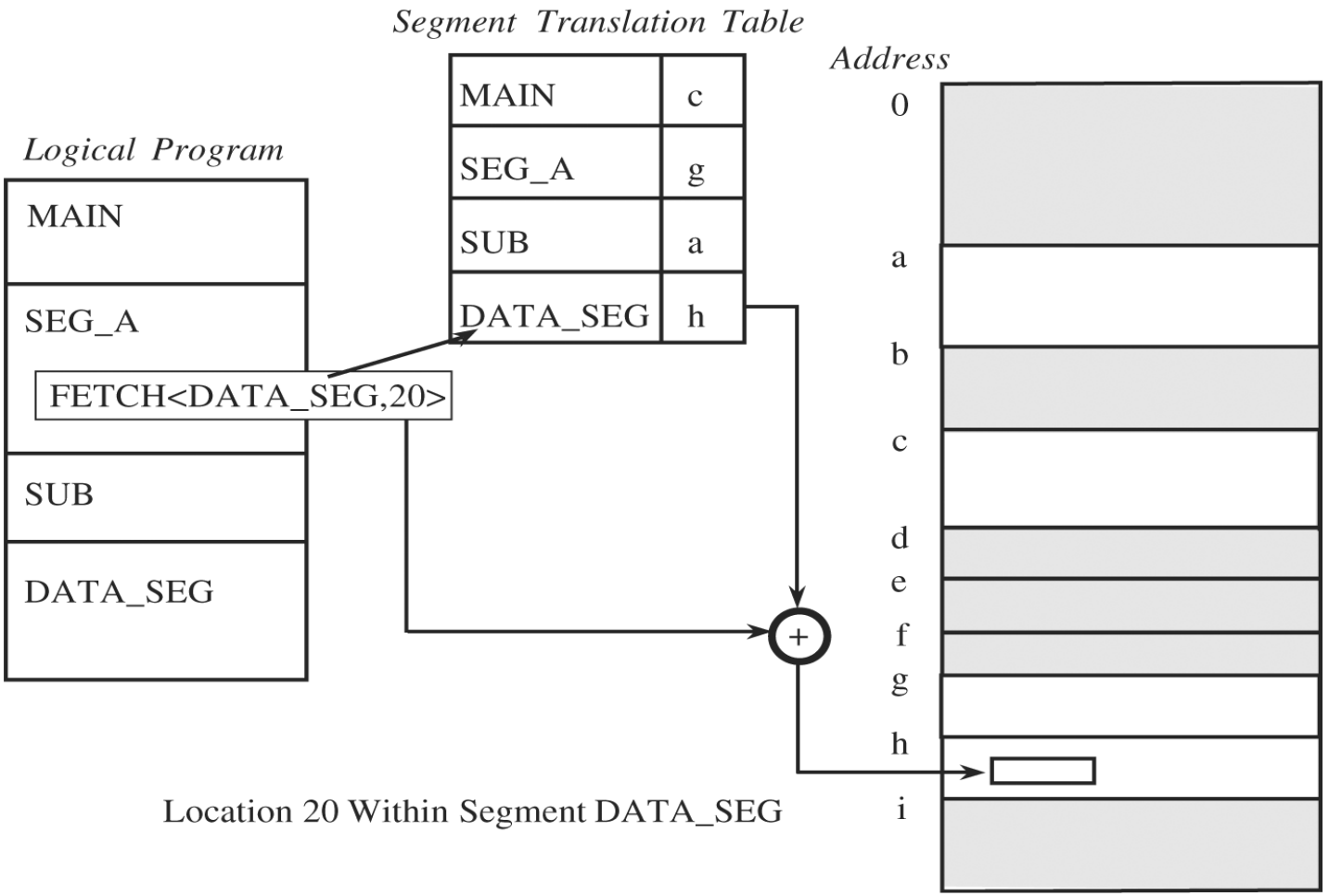
Tagged Architecture …

- Benefit of tagged architecture:
  - Low (good!) granularity of memory access control – at memory word level
- Problems with tagged architecture:
  - Requires special h/w
  - Incompatible with code of most OSs
- OS compatible with it must:
  - Accommodate tags in each memory word
  - Test each memory word accessed
  - Rewriting OS would be costly
  - Higher memory costs (extra bits per word)

Segmentation

- Divide a program into several pieces
  - Example:
    - Code segment, Data segment, …
  - A feasible means to have the effect of an unbounded number of base/bounds registers
  - Allows a program to be divided into many pieces having different access rights
  - A memory location is addressed as <name, offset>
    - Name: name of the segment
    - Offset: location within the segment

Translation of Segment Address



Segment Table also contains memory protection attributes

Segmentation …

- Benefits for the OS
  - The OS can place any segment at any location, or move any segment to any location
  - A segment can be removed from main memory if it's not being used currently
  - Every address reference passes through the OS. So there is an opportunity to check each one for protection

Segmentation …

- Benefits for protection
  - Each address is checked for protection
  - Many different classes of data items can be assigned different levels of protection
  - Two or more users can share access to a segment, with different access rights
  - A user cannot generate an address or access to an unpermitted segment

Segmentation …

- Limitations
  - Overhead to deal with segmentation size
    - Segments need to be dynamic
    - Segment length must be maintained, and compared with every address generated
  - Segmentation leads to fragmentation of memory
    - Poor memory utilization
    - Compacting and updating appropriate tables take time

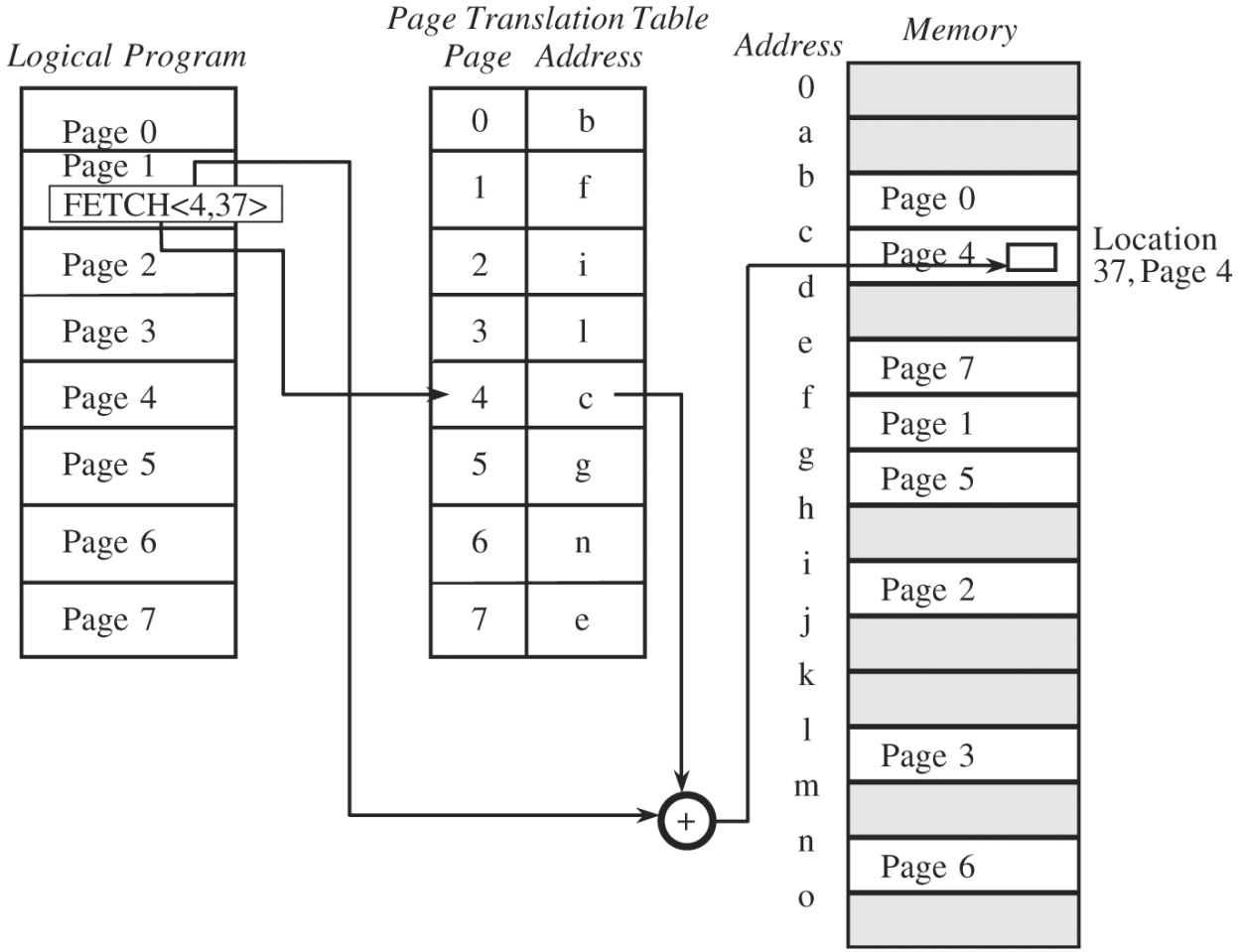
Paging

- Program (i.e., virtual address space) is divided into equal-sized chunks
  - *Pages*
- Physical memory is divided into equal-sized chunks
  - *Frames*
- Frame size equals page size

Paging ...

- Virtual addresses consist of two parts:
  - **`<page #, offset within page>`**
  - # bits for offset = log2(page size),
  - no out-of-bounds possible for offset
- OS keeps mapping from page # to its base physical address in *Page Table*
- Each page has its own memory protection attributes

Page Address Translation



Paging ...

- Advantages:
  - Each address reference is checked for protection by hardware
  - Users can share access to a page, with potentially different access rights
  - Users cannot access an unpermitted page
- Disadvantages:
  - Internal fragmentation
  - Assigning different levels of protection to different classes of data items not feasible

Combined Paging with Segmentation

- Segmentation followed by paging
  - Segments offer logical units for protection rights
  - Each segment is accessed through paging

x86 Architecture

- x86 architecture provides both segmentation and paging
  - Linux uses a combination of segmentation and paging
    - Only simple form of segmentation to avoid portability issues
    - Segmentation cannot be turned off on x86
  - Same for Windows

x86 Architecture ...

- Memory protection bits indicate no access, read/write access or read-only access
- Recent x86 processors also include *NX (No eXecute)* bit, forbidding execution of instructions stored in page
  - Enabled in Windows XP SP 2 and some Linux distros
  - Helps against some buffer overflows

File Protection Mechanisms

- Basic forms of protection
- Single file permissions
- Per-object and per-user protection

Basic forms of protection

- Basic forms of protection
  1) All-none protection
  2) Group protection

1) All-none protection

- Public files (all) or files protected by passwords (none)
  - Access to public files required knowing their names
  - Ignorance (not knowing file name) was an extra barrier
- Problems w/ this approach
  - Lack of trust for public files in large systems
    - Difficult to limit access to trusted users only
  - Complexity – for password-protected files, human response (password) required for each file access
  - File names easy to find
    - File listings eliminate ignorance barrier

Basic forms of protection …

2) Group protection

- Groups with common relationship:
  - i.e., group – if has need to share something
  - User belongs to one group
    - Otherwise can leak info objects groups
- Example — In Unix: user, (trusted) group, others
  - E.g., u+r+w+x,g+r+w-x,o+r-w-x
- Advantage: Ease of implementation
  - OS recognizes user by user ID and group ID (upon login)
  - File directory stores for each file:

    File owner's *user ID* and file owner's *group ID*

Basic forms of protection …

- Problems with group protection

  a) User can't belong to > 1 group

  Solution: Single user gets multiple accounts
  - E.g., Tom gets accounts Tom1 and Tom2
  - Tom1 in Group1, Tom2 in Group2
  - Problem: Files owned by Tom1 can't be accessed by Tom2 (unless they are public – available to 'others')

  Problems: account proliferation, inconvenience, redundancy (e.g., if admin copies Tom1 files to Tom2 acct)

  b) User might become responsible for file sharing

  E.g., admin makes files from all groups visible to a user (e.g., by copying them into one of user's accts and making them private user's files)
  => User becomes responsible for 'manually' preventing unauthorized sharing of his files between his different 'groups'

  c) Limited file sharing choices

  Only 3 choices for any file: private, group, public

Single file permissions

- Single permissions – associating permission with single file
- Types of single file permissions:
  1) Password or other token
  2) Temporary acquired permission

1) Password or other token
   - Provide a password for each file
     - File pwd for W only
     - File pwd for any access
   - Finer degree of protection
     - Like having a different group for each file
       - file X group = all those who know file X pwd

Single file permissions …

- Problems with file pwds
  - Loss of pwd
    - Requires admin unprotecting file, then assigning new
    - Requires notifying all legitimate users
  - Using them inconvenient, takes time
  - Pwd disclosure allows unauthorized file accesses
    - Change of pwd requires notifying all legitimate users
  - Revocation of (just) a single user requires pwd change
    - Then, must notify all legitimate users

Single file permissions …

2) Temporary acquired permission

- Used in UNIX – the approach:
  - Based on user-group-others access hierarchy
  - Permission called *set userid* (suid)
    - If "user" (owner) of executable file X sets *suid* for X for his group, any group member executing X has "user" access rights (ARs) for X
      - Rather than having just "regular" group ARs for X
- Allows users to share data files
  - Access only via procedures that access them
    - Procedures encapsulate files
  - E.g., convenient for OS pwd file
    - Pwd change program with suid - any user can access own pwd record
    - OS owns this program (only OS, as "user" can access whole pwd file)

Per-object and per-user protection

- Per-object and per-user protection
  - Approach:
    - File owner specifies access rights (ARs) for *each file* he owns for *each user*
    - Can implement with ACL (access control list) or ACM (access ctrl matrix)
  - Advantages:
    - Fine granularity of file access
    - Allows to create groups of users with similar ARs
  - Problem: Complex to create and maintain groups
    - File owner's overhead to specify ARs for *each file* for *each user* he owns