CSC139 Operating System Principles

Fall 2019, Part 5-1

Instructor: Dr. Yuan Cheng

Outline: Protection

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights

Chapter Objectives

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

Protection

- Operating system consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem – ensure that each object is accessed correctly and only by those processes that are allowed to do so

Principles of Protection

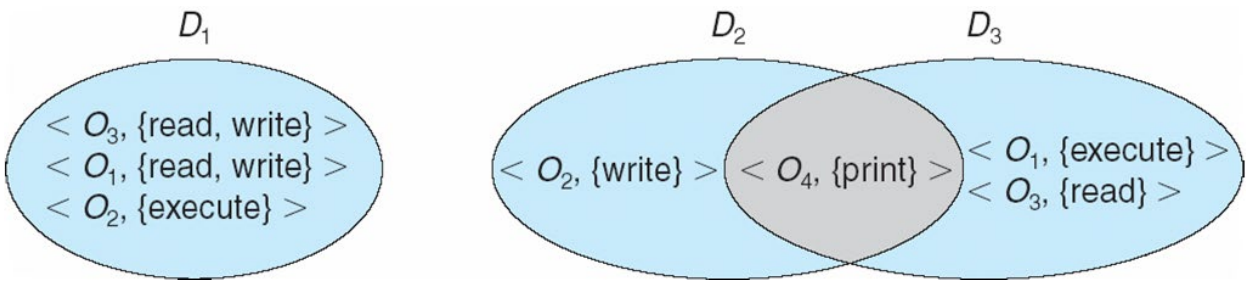
- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – domain switching, privilege escalation
  - “Need to know” a similar concept regarding access to data

Principles of Protection (cont.)

- Must consider “grain” aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC
- Domain can be user, process, procedure

Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights

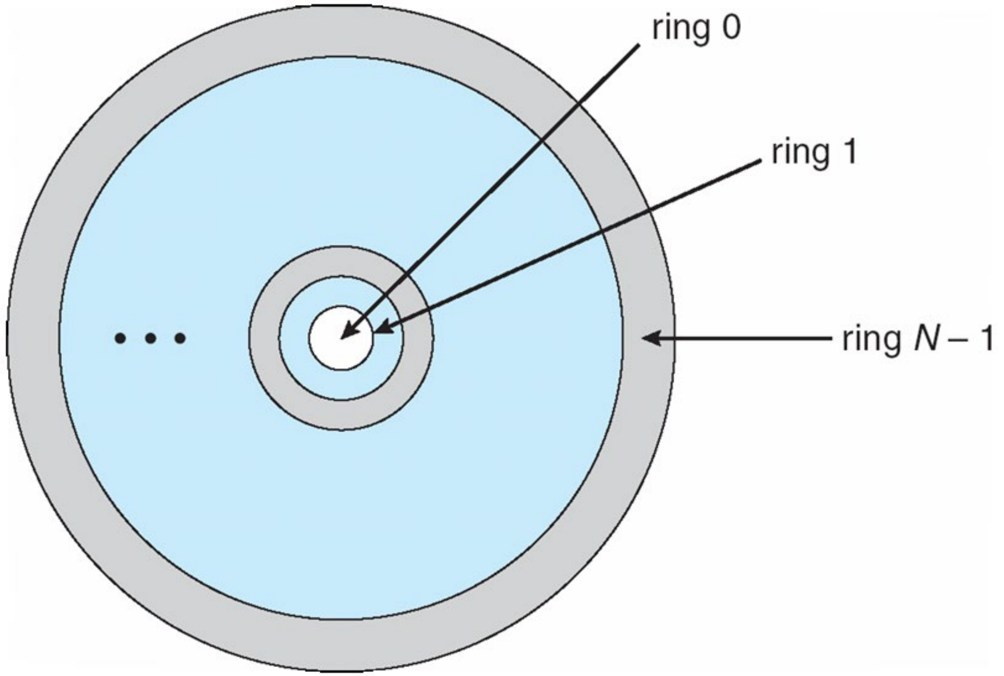


Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
    - Each file has associated with it a domain bit (setuid bit)
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed
    - When execution completes user-id is reset
- Domain switch accomplished via passwords
    - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
    - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < i \Rightarrow D_i \subseteq D_j$



Access Matrix

- View protection as a matrix (access matrix)
- Rows represent domains
- Columns represent objects
- `Access(i, j)` is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Use of Access Matrix (cont.)

- Access matrix design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- But doesn’t solve the general confinement problem

Access Matrix of Figure A with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with *Copy* Rights

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix with *Owner* Rights

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Modified Access Matrix of Figure B

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples `<domain, object, rights-set>` in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

Implementation of Access Matrix (cont.)

- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs `<domain, rights-set>` defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access

Implementation of Access Matrix (cont.)

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (cont.)

- Option 3 – Capability list for domains
  - Instead of object-based, list is domain based
  - Capability list for domain is list of objects together with operations allows on them
  - Object represented by its name or address, called a capability
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

Implementation of Access Matrix (cont.)

- Option 4 – Lock-key
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called locks
  - Each domain as list of unique bit patterns called keys
  - Process in a domain can only access object if domain has key that matches one of the locks

Comparison of Implementations

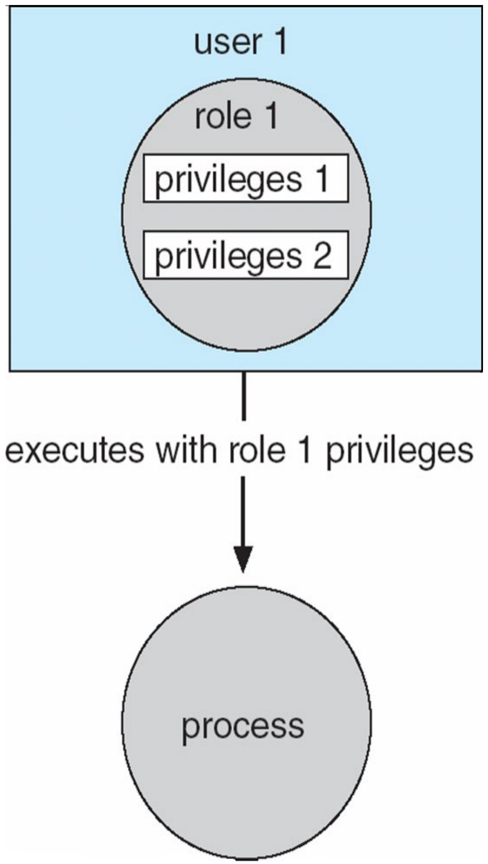
- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file

Access Control

- Protection can be applied to non-file resources
- Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege
  - ***Privilege*** is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned ***roles*** granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix



Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent
- Access List – Delete access rights from access list
  - Simple – search access list and remove entry
  - Immediate, general or selective, total or partial, permanent or temporary

Revocation of Access Rights (cont.)

- Capability List – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

Summary

- Protection aims to enforce safety in the OS by limiting what processes can do
- Policy vs mechanism
- Protection schemes often broken!

Outline: Security

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Implementing Security Defenses
- Firewalling to Protect Systems and Networks
- Computer-Security Classifications
- An Example: Windows 7

Chapter Objectives

- To discuss security threats and attacks
- To explain the fundamentals of encryption, authentication, and hashing
- To examine the uses of cryptography in computing
- To describe the various countermeasures to security attacks

The Security Problem

- System secure if resources used and accessed as intended under all circumstances
  - Unachievable
- Intruders (crackers) attempt to breach security
- Threat is potential security violation
- Attack is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

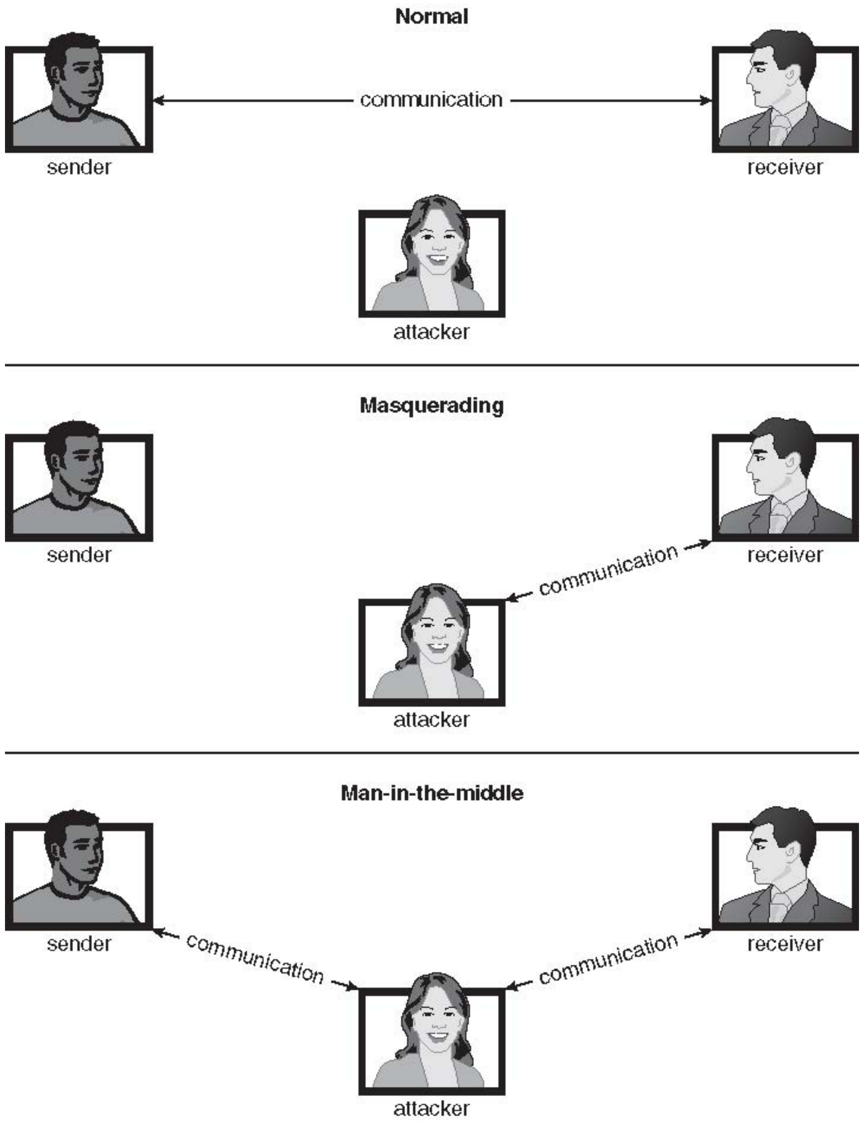
Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DoS)**
  - Prevention of legitimate use

Security Violation Methods

- Masquerading (breach authentication)
  - Pretending to be an authorized user to escalate privileges
- Replay attack
  - As is or with message modification
- Man-in-the-middle attack
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- Session hijacking
  - Intercept an already-established session to bypass authentication

Standard Security Attacks



Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid social engineering, phishing, dumpster diving
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

Program Threats

- Many variations, many names
- Trojan Horse
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - Spyware, pop-up browser windows, covert channels
  - Up to 80% of spam delivered by spyware-infected systems
- Trap Door
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

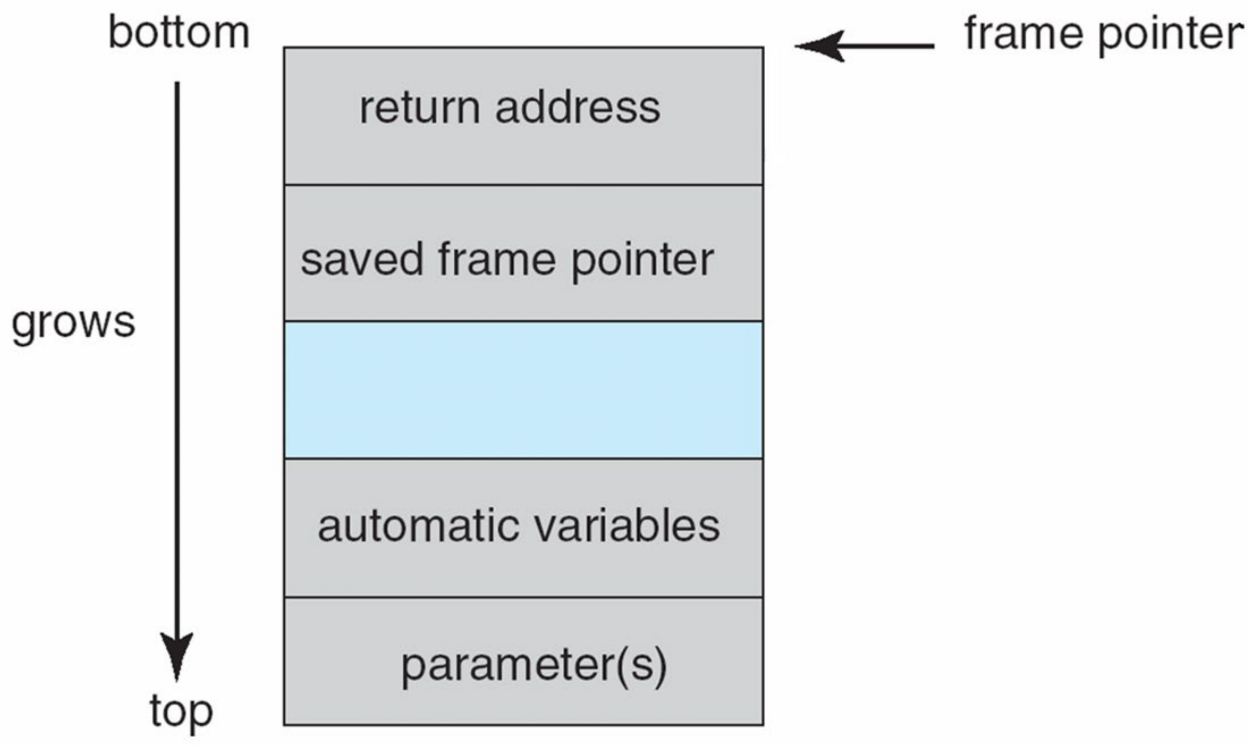
Program Threats (cont.)

- Logic Bomb
  - Program that initiates a security incident under certain circumstances
- Stack and Buffer Overflow
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

C Program with Buffer-overflow Condition

```
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
 char buffer[BUFFER SIZE];
 if (argc < 2)
     return -1;
 else {
     strcpy(buffer,argv[1]);
     return 0;
 }
}
```

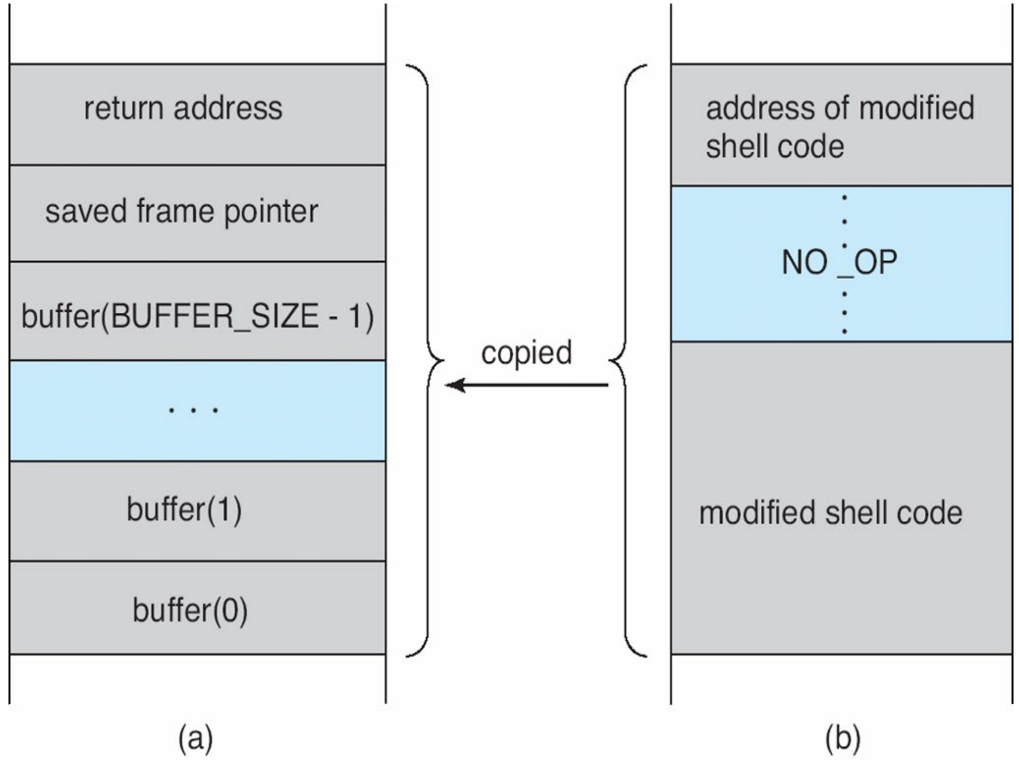
Layout of Typical Stack Frame



Modified Shell Code

```
#include <stdio.h>
int main(int argc, char *argv[])
{
 execvp("\bin\sh","\bin \sh", NULL);
 return 0;
}
```

Hypothetical Stack Frame



Before attack

After attack

Exit Slips

- Take 1-2 minutes to reflect on this lecture
- On a sheet of paper write:
  - One thing you learned in this lecture
  - One thing you didn’t understand

Next class

- Final Exam Review

Acknowledgment

- The slides are partially based on the ones from
  - The book site of *Operating System Concepts (Tenth Edition)*: http://os-book.com/