BILKENT UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
CS342 OPERATING SYSTEMS

LECTURE 2
OPERATING SYSTEM STRUCTURES (CHAPTER 2)

Dr. İbrahim Körpeoğlu
http://www.cs.bilkent.edu.tr/~korpe

References

- *The slides here are adapted/modified from the textbook and its slides: Operating System Concepts, Silberschatz et al., 7th & 8th editions, Wiley.*

REFERENCES

- Operating System Concepts, 7th and 8th editions, Silberschatz et al. Wiley.
- Modern Operating Systems, Andrew S. Tanenbaum, 3rd edition, 2009.

Outline

- Operating System Services
- User Operating System Interface
- System Calls
- System Programs
- Operating System Structure

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system

Operating System Services

- Two major tasks of an operating system
  - 1) Providing a nice/convenient environment for users/applications
    - Provide some set of services for user applications
    - Visible to users
  - 2) Resource allocation and management
    - Not much visible to users

Operating System Services

- 1) One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
  - File-system manipulation - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

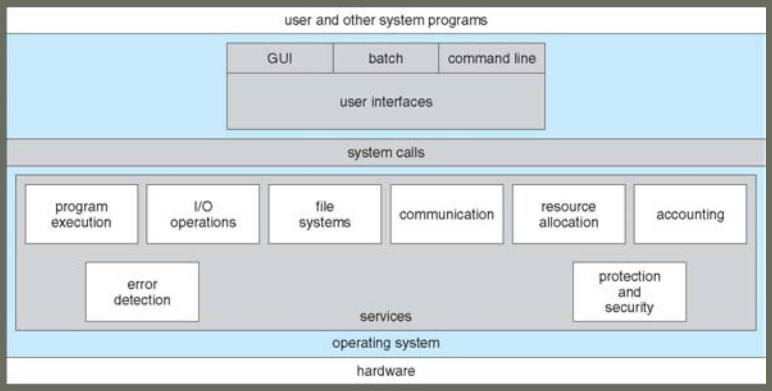
Operating System Services

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)
  - Error detection and handling – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing

A View of Operating System Services

- 2) Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - CPU, Memory, File Storage, I/O devices, etc.
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information should be able to control use of that information; concurrent processes should not interfere with each other
    - Access control to resources, authentication, …

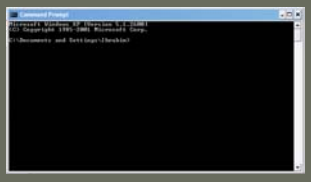
OS Services



[User - Operating System] Interface - CLI

Command Line Interface (CLI) or command interpreter allows direct command entry

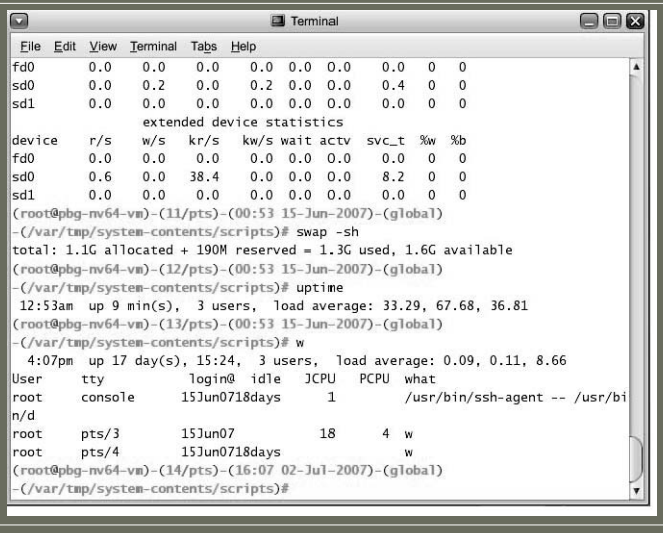
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
- Primarily fetches a command from user and executes it
  - Sometimes commands built-in, sometimes just names of programs
    - later case: add new programs easily



[User-Operating System] Interface - GUI

- User-friendly desktop metaphor interface
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory
- Many operating systems now include both CLI and GUI interfaces
  - Linux: command shells available (CLI); KDE as GUI

Bourne Shell Command Interpreter



The MacOS X GUI

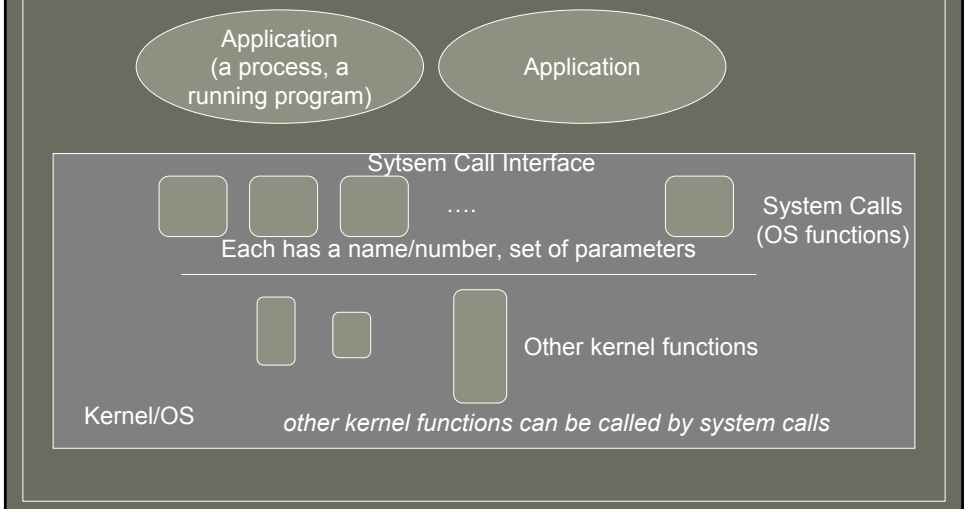
System Calls

- Programming interface to the services provided by the OS
  - Interface provided to applications
- Typically written in a high-level language (C or C++)

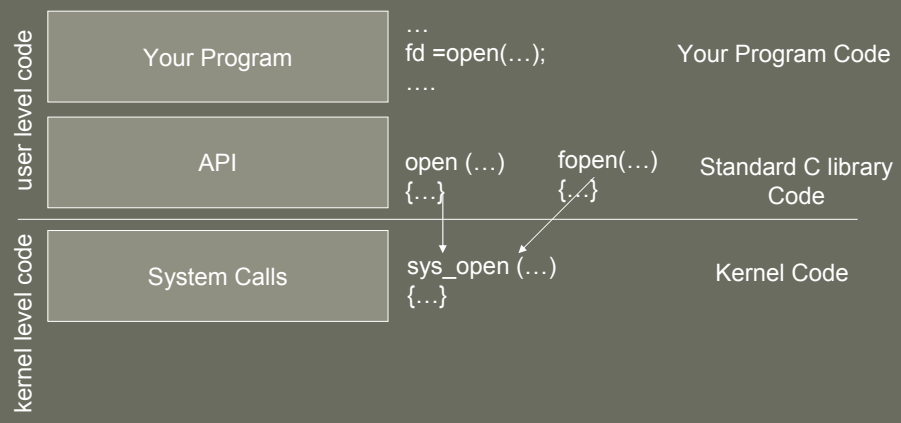
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are :
  - Win32 API for Windows,
    POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X),
  - Java API for the Java virtual machine (JVM)

- Why use APIs rather than system calls?

System Calls

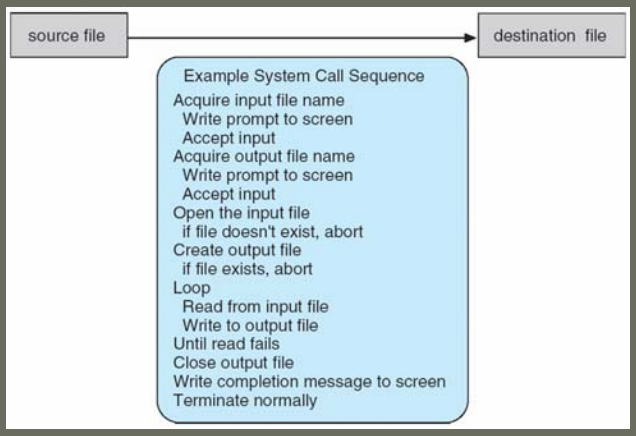


System Calls



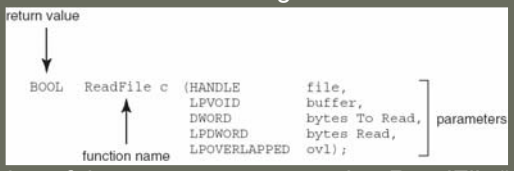
Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file

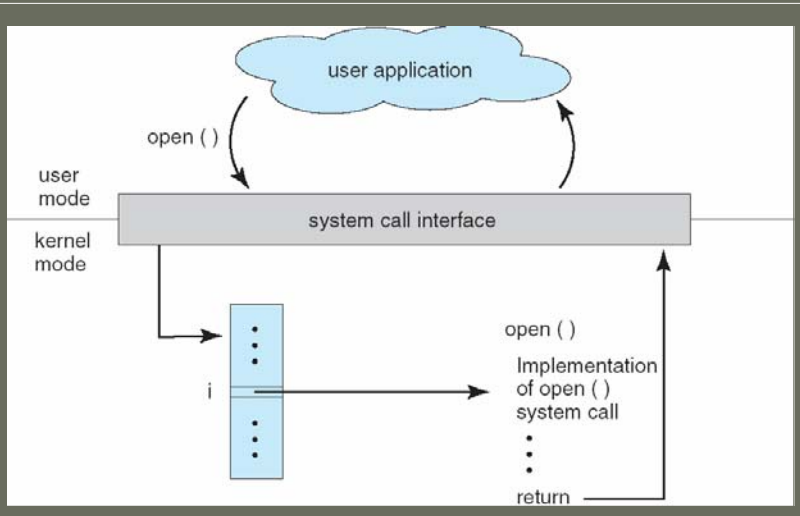


- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

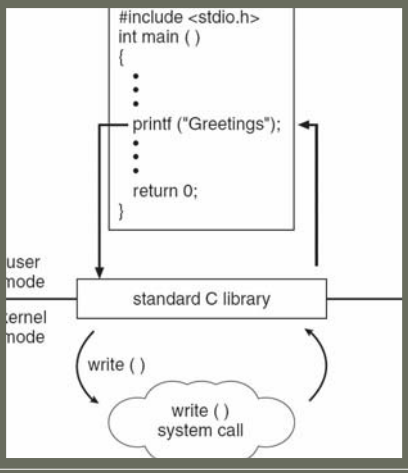
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Standard C Library Example

- C program invoking printf() library call, which calls write() system call

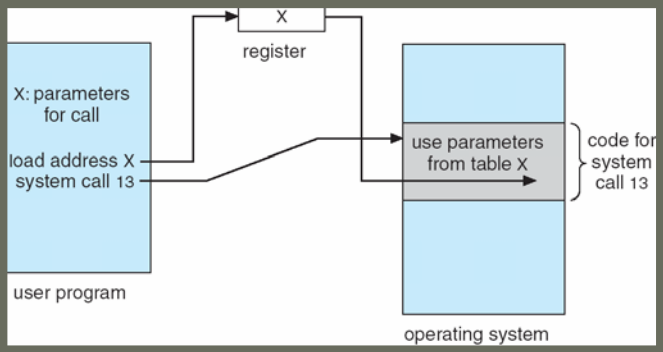


System Call Parameter Passing

- Often, more information is required than the identity of the desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - 1) Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - 2) Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
  - 3) Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system

  Last two methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

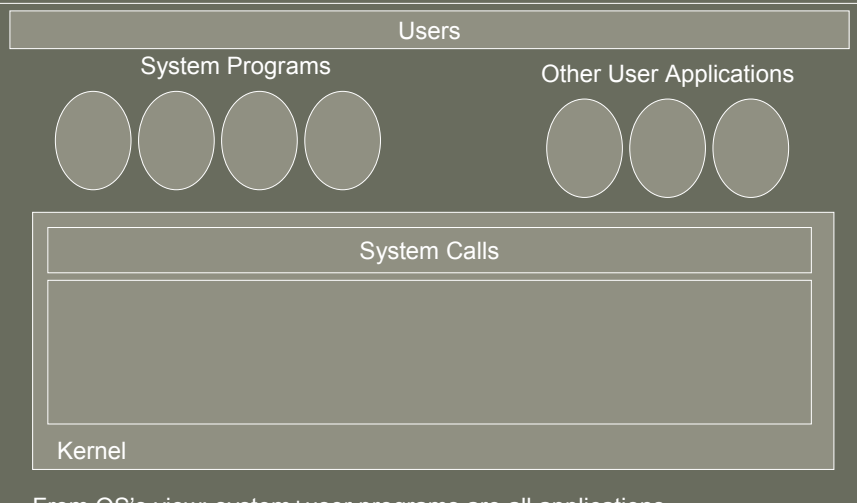
System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation (create, delete, copy, rename, print, list, …)
  - Status information (date, time, amount of available memory, disk space, who is logged on, …)
  - File modification (text editors, grep, …)
  - Programming language support (compiler, debuggers, …)
  - Program loading and execution (loaders, linkers)
  - Communications (ftp, browsers, ssh, …)
  - Other System Utilities/Applications may come with OS CD (games, math solvers, plotting tools, database systems, spreadsheets, word processors, …)

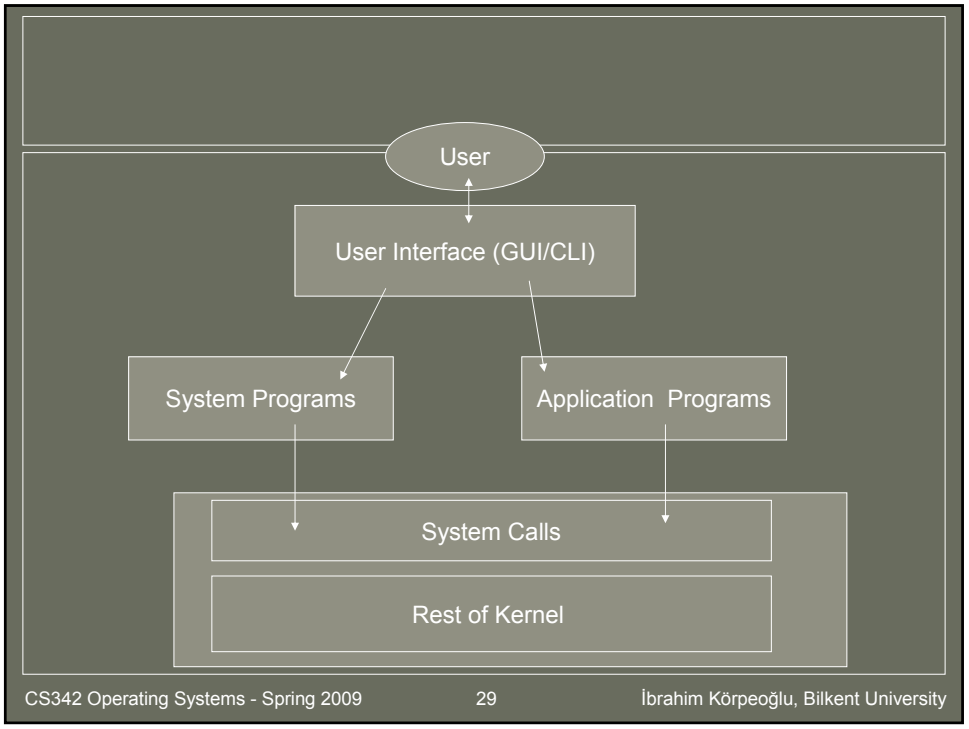
System Programs

- Most users' view of the operation system is defined by system programs, not the actual system calls
- System programs provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
    - create file: simple system program that can just call "create" system call or something similar
    - compiler: complex system program

System Programs



From OS's view: system+user programs are all applications

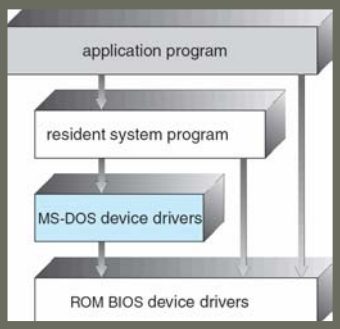


OS Structure

- Simple Structure (MSDOS)
- Layered Approach
- Microkernel Approach
- Modules Approach

Simple Structure

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



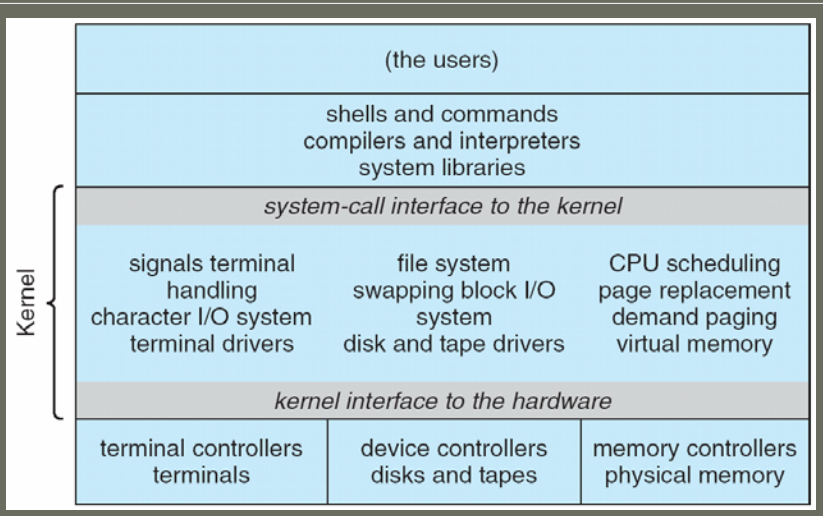
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

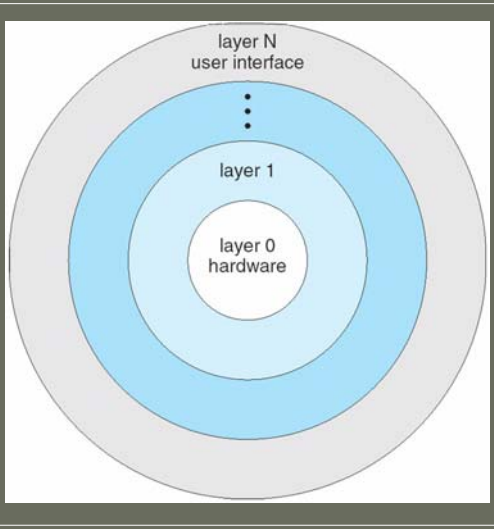
Unix

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Traditional UNIX System Structure



Layered Operating System



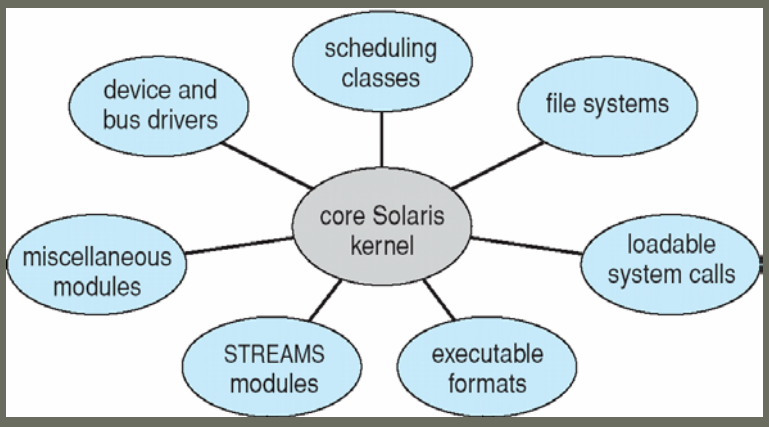
Microkernel System Structure

- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Solaris Modular Approach



End of Lecture