CSE 4/521 Introduction to Operating Systems

Lecture 3 – Operating Systems Structures

(Operating-System Services, User and Operating-System Interface, System Calls, Types of System Calls, System Programs, Operating-System Structure, Operating-System Debugging)

Summer 2018

Overview

Objective:

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

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Recap

- Storage Management
  - Migration of Data from Disk -> Registers and associated tradeoffs.
- Protection and Security
  - Implemented through userIDs and groupIDs, privilege escalation
- Kernel Data Structures
  - Linked lists, binary (search) trees, hash maps
- Computing Environments
  - Traditional, Mobile, Distributed (Client-Server, Peer-to-peer), Virtualization, Cloud Computing
- Open-Source Operating Systems
  - GNU/Linux, BSD UNIX

Question

1. Distinguish between Client-Server and Peer-to-Peer models of distributed systems. (Easy)

2. Why is Cache Coherency important for Multi-processor environments? Why must multi-tasking environments be careful to use most recent value? (Medium)

3. In your analysis of Open-source OSes, did you find any other data structures apart from linked lists, binary trees or hash maps? (Hard and open-ended)

Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Operating Systems Services

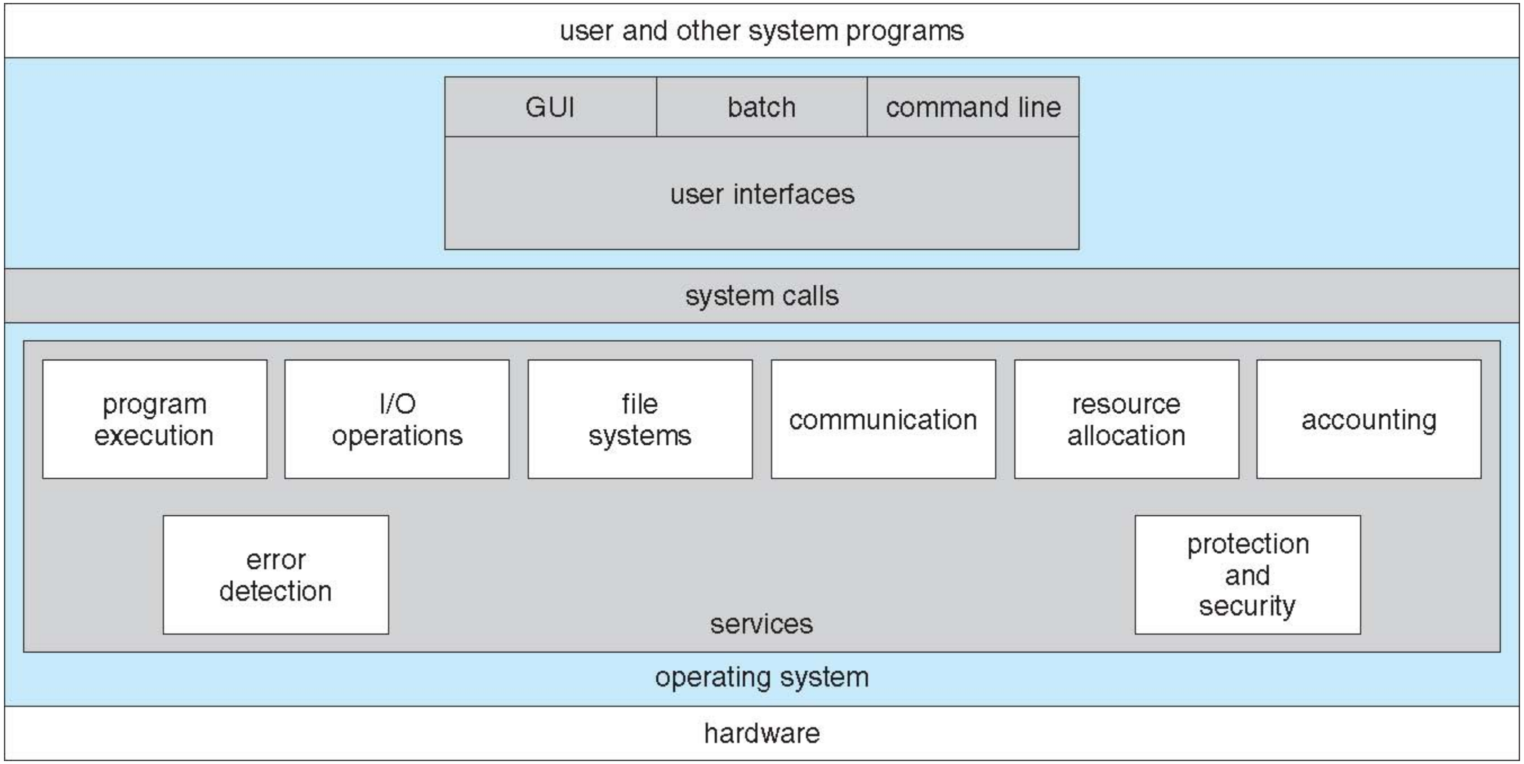


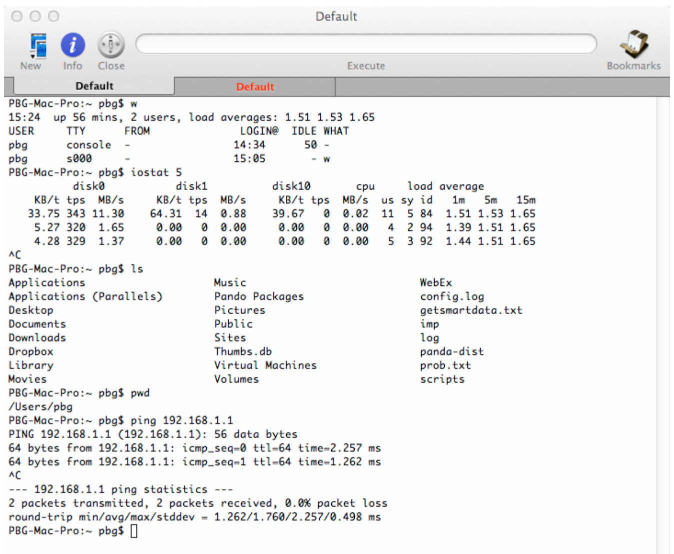
Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

User and Operating-System Interface

- Three predominant types of User and Operating-System Interface:

1. Command line interface



Goals: Batch processing

2. Graphic user interface

Goals: Intuitive user experience

3. Touchscreen interface

Goals: Maximizing efficiency of touch

Table of Content

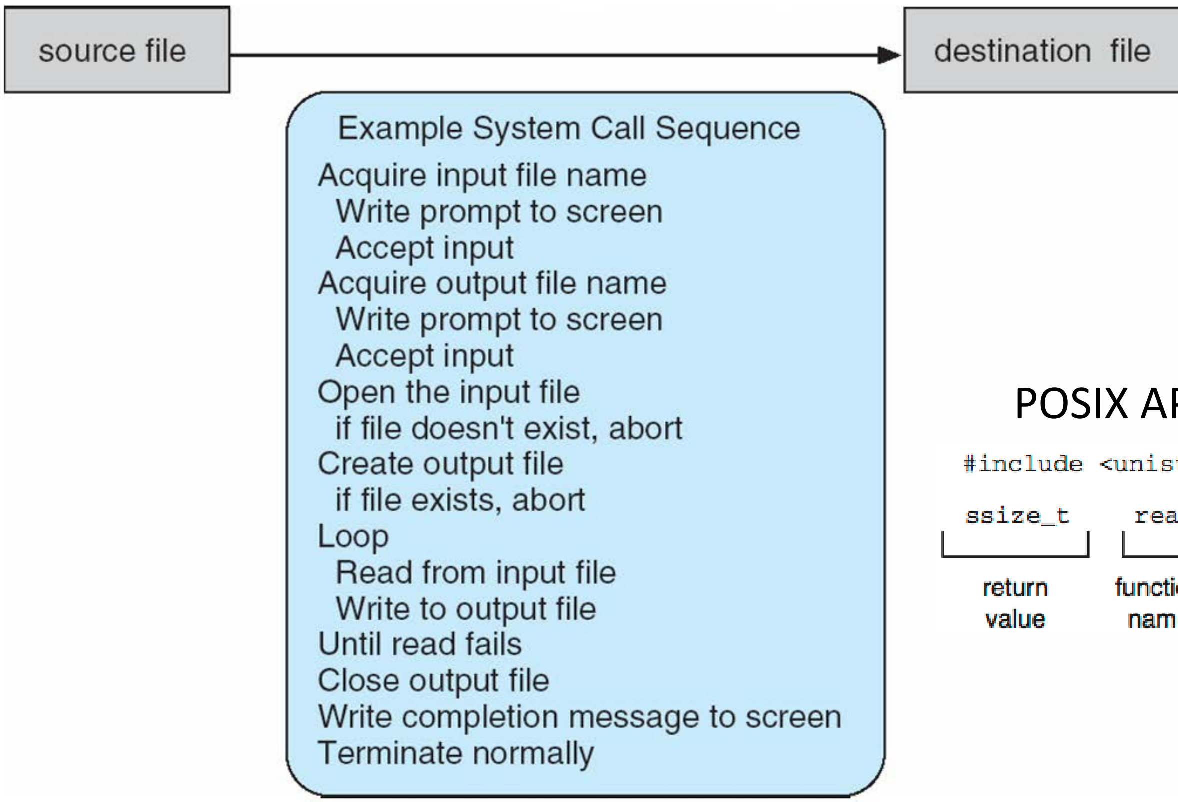
- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

System Calls

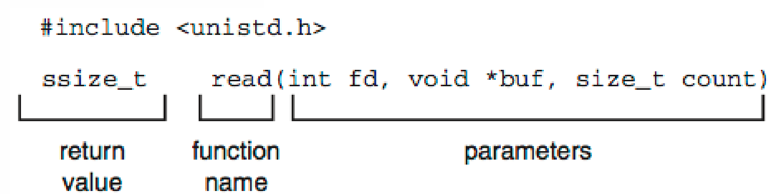
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Programming Interface (API).
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for UNIX, Linux and Mac OS X systems
  - Java API for the Java virtual machine (JVM)

System Calls - Example

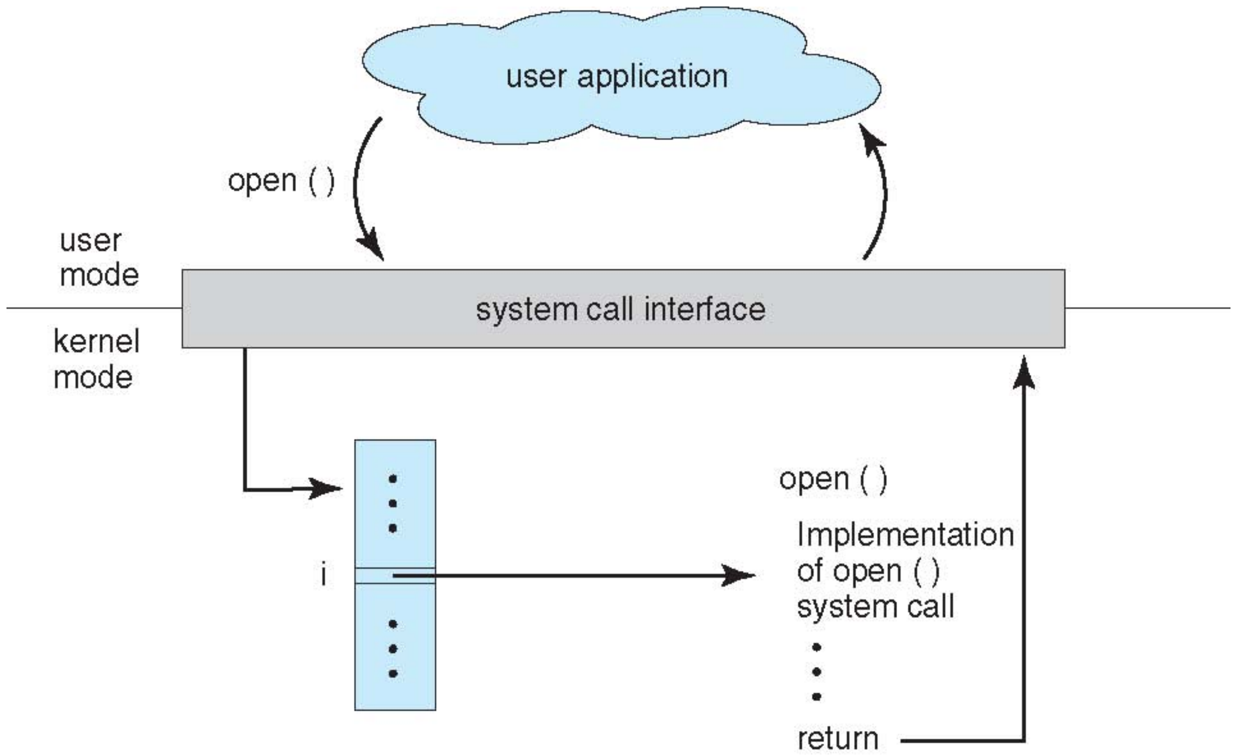
Collection of Systems Call APIs to copy source file to destination file



POSIX API for read system call



System Calls - Invocation



System Calls – Passing Parameters

- Three general methods used to pass parameters to the OS
  - Pass the parameters in registers
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system

Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Types of System Calls

1. Process control
   - Create process, terminate process, wait for time
2. File Management
   - Open file, read file, write file, reposition file, close file
3. Device Management
   - Request device, read device, write device, release device
4. Information Maintenance
   - Get and set process, get and set time
5. Communications
   - Create and delete communication connection, shared memory model create and gain access to memory regions
6. Protection
   - Control access, get and set permissions, allow and deny users

Types of System Calls - Examples

	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()

Total LINUX system calls: 362

Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

System Programs

- Most users' view of the operation system is defined by system programs, not the actual system calls
- Mostly divided into:
  - File management – (create, delete and generally manipulate files and directories)
  - Status information – (date, time, logging, amount of memory)
  - Programming language support – (compilers, assemblers, debuggers)
  - Program loading and execution – (absolute loaders, relocatable loaders, linkage editors)
  - Communications – (provide mechanism for creating virtual connections among processes, users and computer systems)
  - Background services – (disk checking, error logging)
  - Application programs – (launched by command line, click)

Table of Content

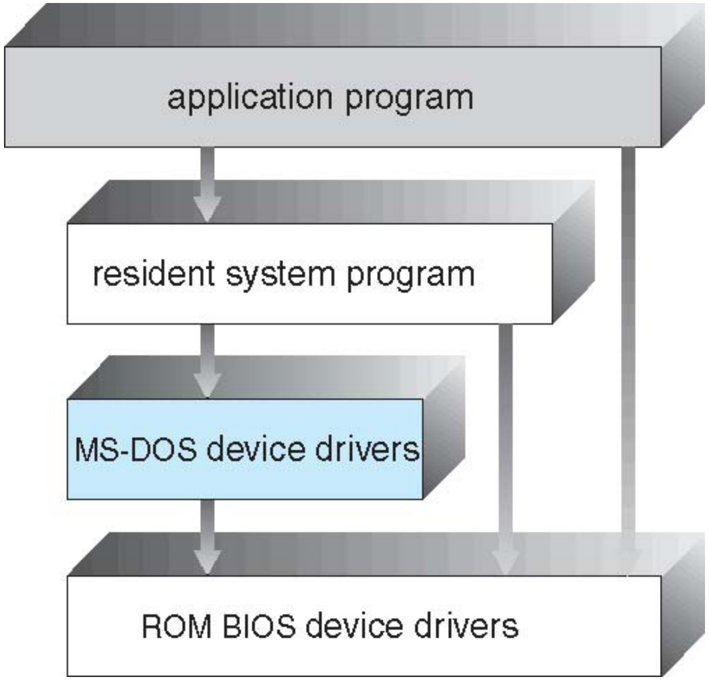
- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Operating-System Structure

- General-purpose OS is very large program
- 4 ways to structure an OS
  - Simple structure – MS-DOS
  - Complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mach

Operating System Structure - Simple

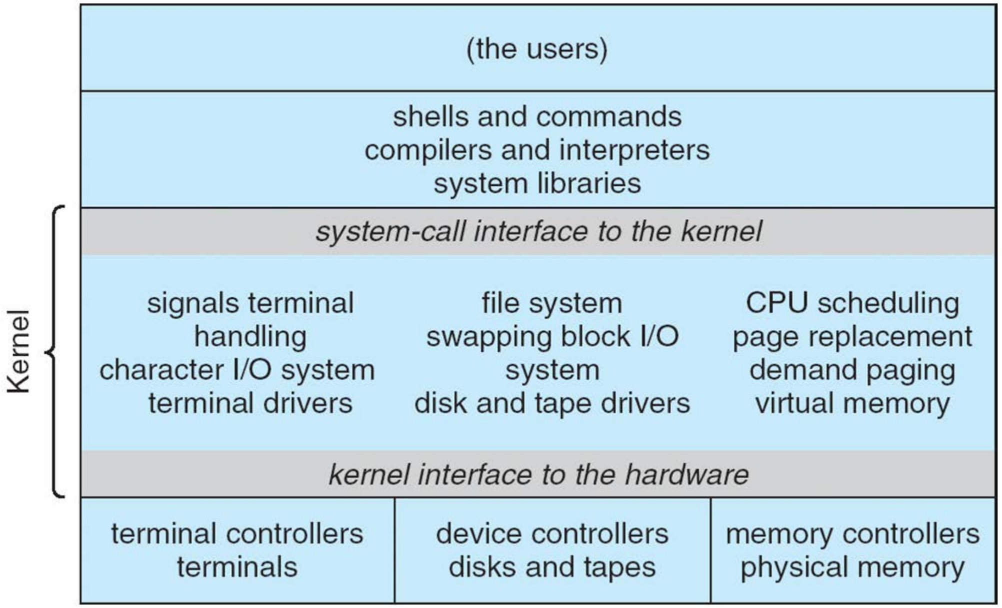
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



Operating System Structure - Complex

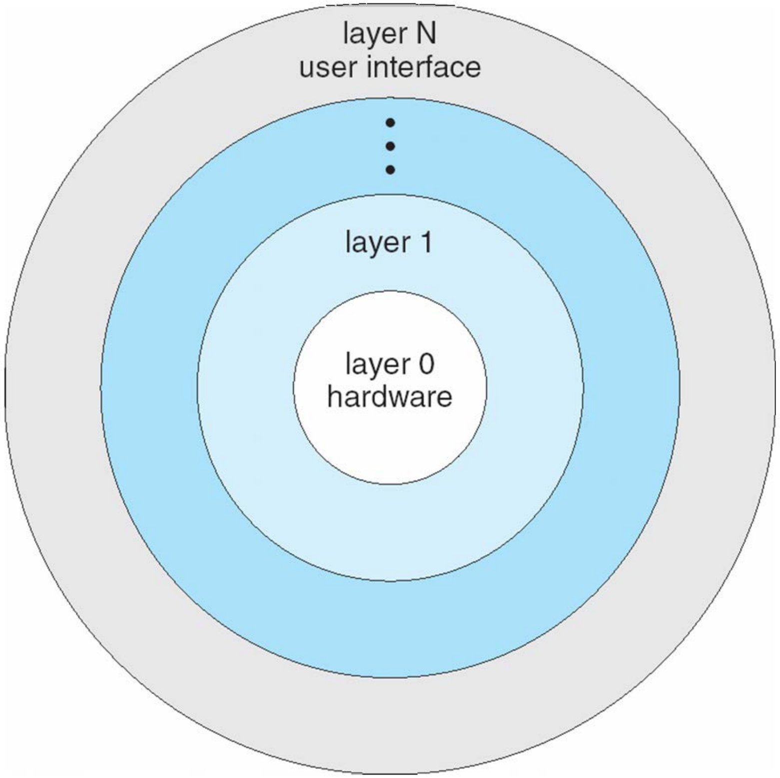
The UNIX OS consists of 2 separate parts:

- 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



Operating System Structure - Layered

- 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



Operating System Structure - Microkernel

- 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
- Drawbacks:
  - Performance overhead of user space to kernel space communication

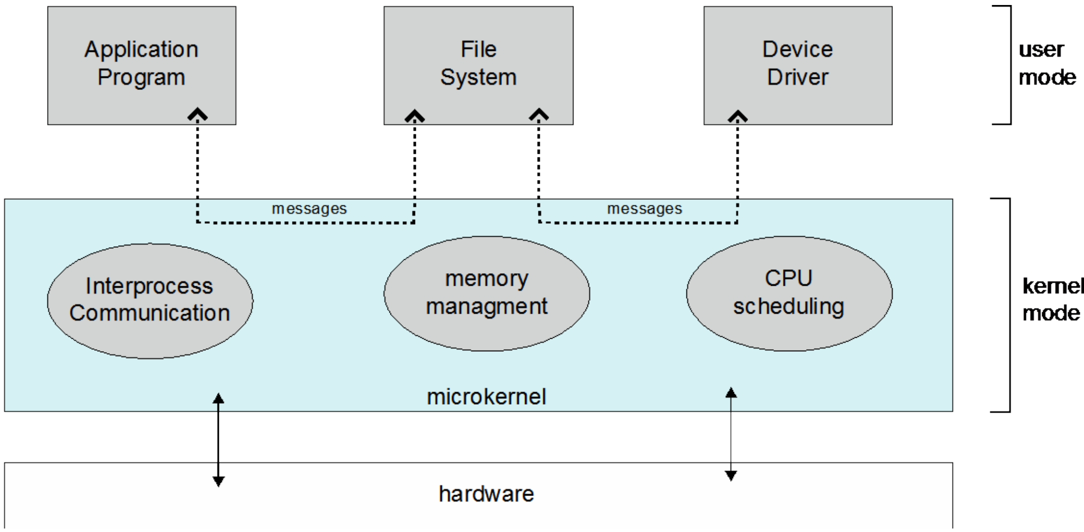


Table of Content

- Operating-System Services
- User and Operating-System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating-System Structure
- Operating-System Debugging

Operating-System Debugging

- Debugging is finding and fixing errors, or bugs
- OS generate log files containing error information
- Failure of an application can generate core dump file capturing memory of the process
- Operating system failure can generate crash dump file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Profiling is periodic sampling of instruction pointer to look for statistical trends

Operating System Debugging - DTrace

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- Probes fire when code is executed within a provider, capturing state data and sending it to consumers of those probes

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
   gnome-settings-d          142354
   gnome-vfs-daemon          158243
   dsdm                      189804
   wnck-applet               200030
   gnome-panel               277864
   clock-applet              374916
   mapping-daemon            385475
   xscreensaver              514177
   metacity                  539281
   Xorg                     2579646
   gnome-terminal           5007269
   mixer_applet2            7388447
   java                    10769137
```

Figure 2.21 Output of the D code.

Credential for slides

Silberschatz, Galvin and Gagne