CS 452 Operating Systems

Operating System Structures

Dr. Denton Bobeldyk

Operating System Services

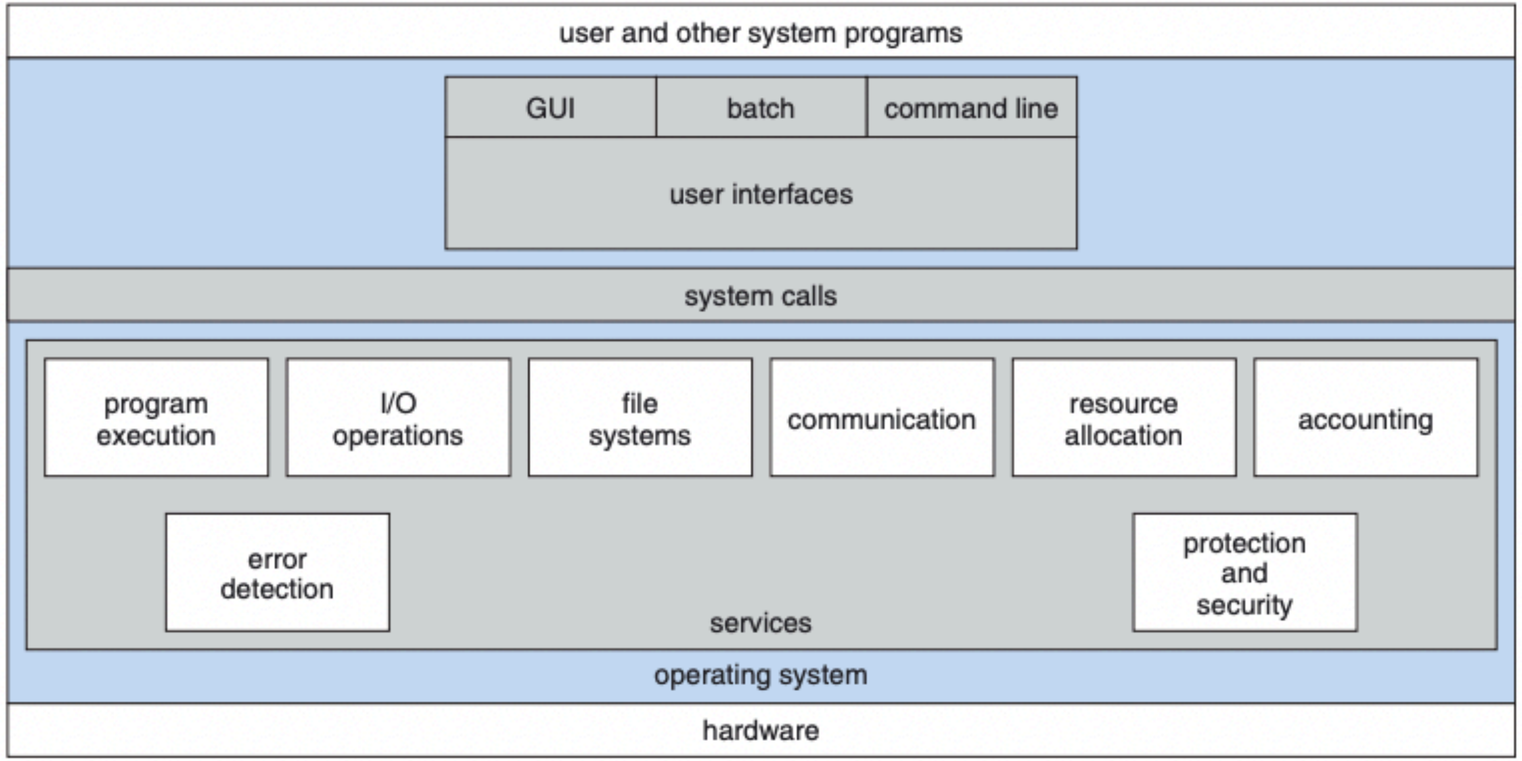


Figure 2.1 A view of operating system services.

Operating System Services

- User Interface
  - Command-line interface - text commands via keyboard or typing in commands
  - Batch interface - commands/directives entered into files, and the files are executed
  - Graphical user interface (GUI) - window system with pointing/touch device

Operating System Services

- Program Execution
  - Load the program into memory and run the program
  - Program ends normally or abnormally (indicating an error)

Operating System Services

- I/O operations
  - For efficiency and protection users usually cannot control I/O devices directly
  - Operating system therefore handles these functions

Operating System Services

- File-system manipulation
  - Read/Write files
  - Create/Delete files
  - Provide permissions management to allow/deny access

Operating System Services

- Communications
  - Process needs to exchange information with another process
  - Processes could be on same computer, or different computers
  - Implemented via shared memory
    - Two processes read/write to shared section of memory
  - Implemented via message passing
    - Packets with predefined formats are moved between processes

Operating System Services

- Error detection
  - Memory error
  - Power failure
  - Parity error on disk
  - Network connection failure
  - Printer out of paper
  - Illegal memory location
  - Too much CPU time used

Operating System Services

- Resource allocation
  - CPU cycles
  - Main Memory
  - File storage

Operating System Services

- Accounting
  - What resources and how how long are users using them for

Why would this be useful??

Operating System Services

- Accounting
  - What resources and how how long are users using them for
  - Could be used for billing purposes or simply accumulating usage statistics

Operating System Services

- Protection and Security
  - Processes from one user should not affect those from other users
  - Require each user to authenticate when logging in

System Calls

- System calls
  - Provide interface to services provided by the operating system

System Calls

- Windows - Windows API
- Unix, Linux, Mac OSX - POSIX API
- Java Virtual Machine - Java API

System Calls

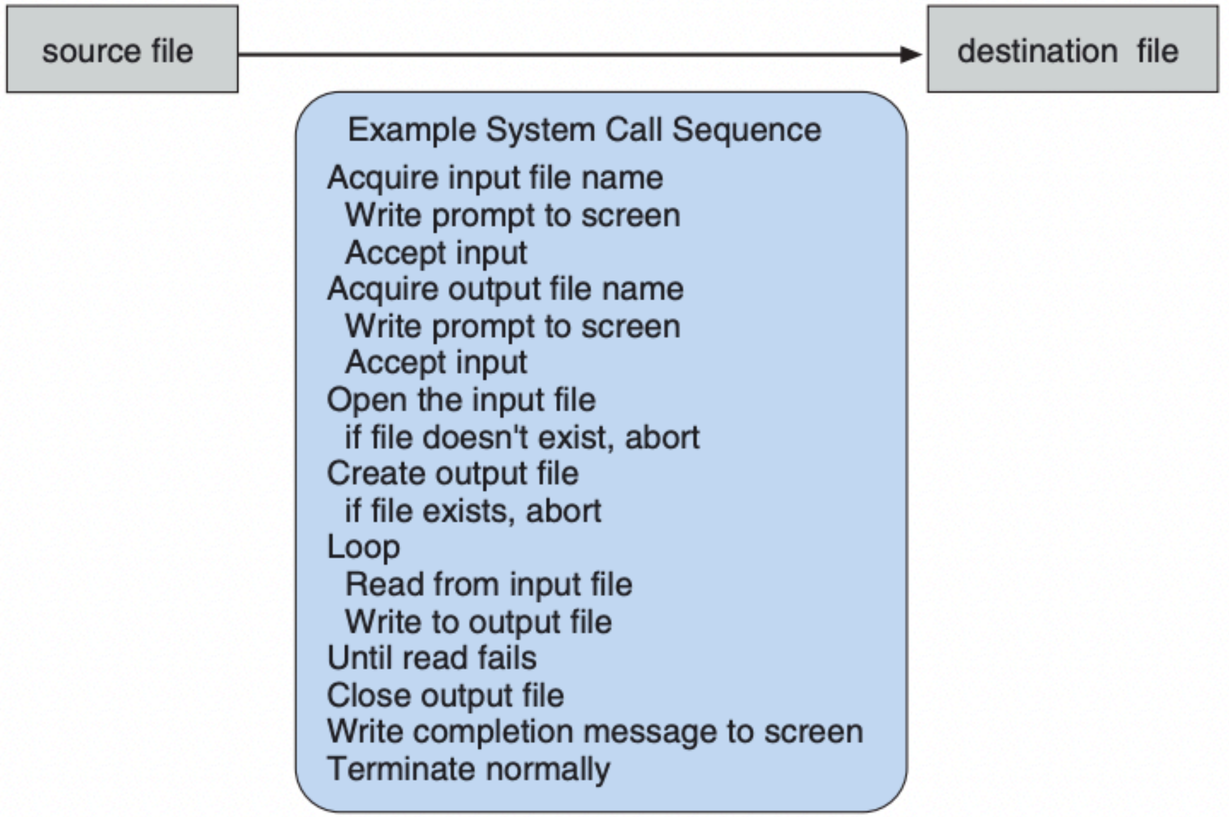


Figure 2.5 Example of how system calls are used.

Standard API Example

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

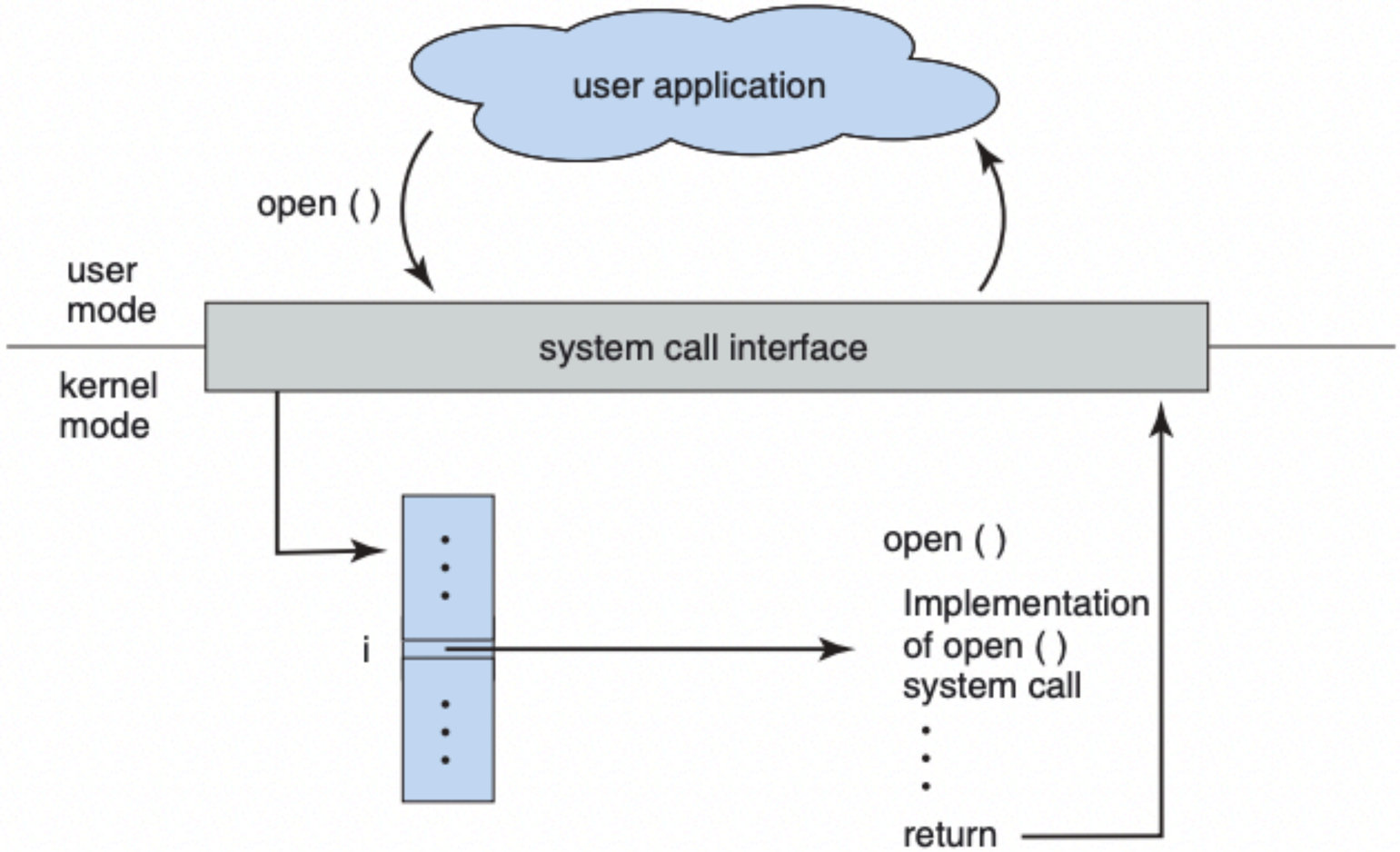


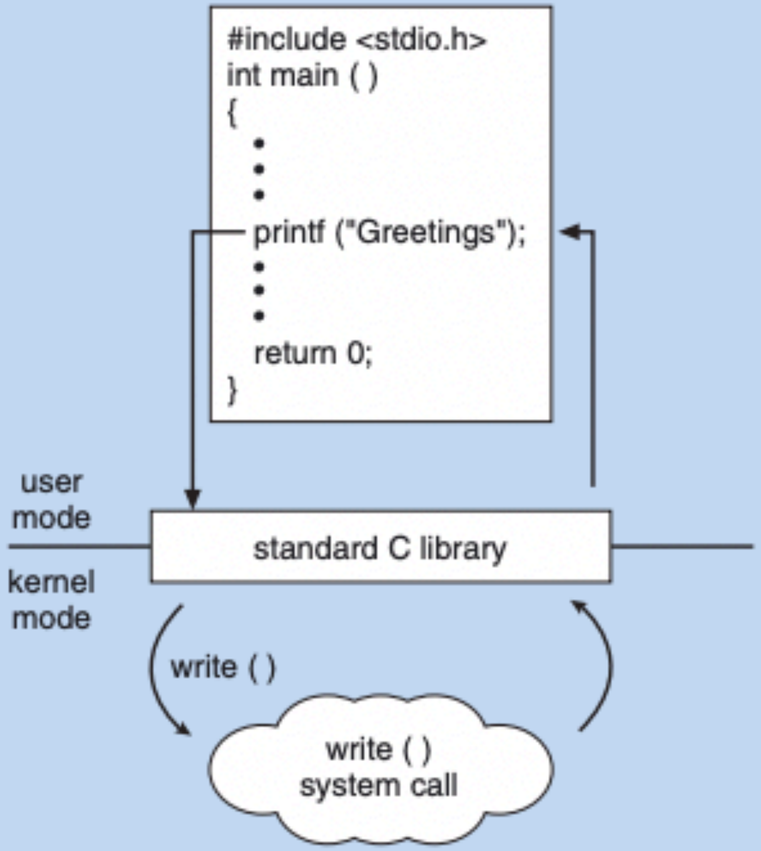
Figure 2.6 The handling of a user application invoking the `open()` system call.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below:



Policy vs Mechanism

- Mechanisms - How to do something
- Policies - What will be done

- Example: Timer construct is a mechanism for ensuring CPU protection. Deciding how long the timer is set is a policy decision

Policy vs Mechanism

- Policies may change from implementation to implementation

Modules

- Loadable kernel modules - modules that can be loaded at boot time OR during run time
- Saves memory by only loading kernel modules that are needed (e.g., printing)

Operating-System Debugging

- Failure Analysis
  - Write to log files
  - Core dump - capture of the memory of the process and store it for later analysis

DTrace

- Utility that dynamically adds probes to a running system
- Uses the D programming language

Strace vs Dtrace

- Google Exercise (5 minutes):
- What is the difference between strace and dtrace

In-Class Exercise

- Create a “Hello World” program in c
- Create a variable named ‘myIntVariable’ and set it’s value to 1
- Use the printf command to output the value of myIntVariable
- Use the printf command to put the address of myIntVariable
- Create a variable named ‘myPointerToIntVariable’ and set it’s value to the memory address of myIntVariable
- Output each of the following via printf:
  - myPointerToIntVariable
  - &myPointerToIntVariable
  - *myPointerToIntVariable