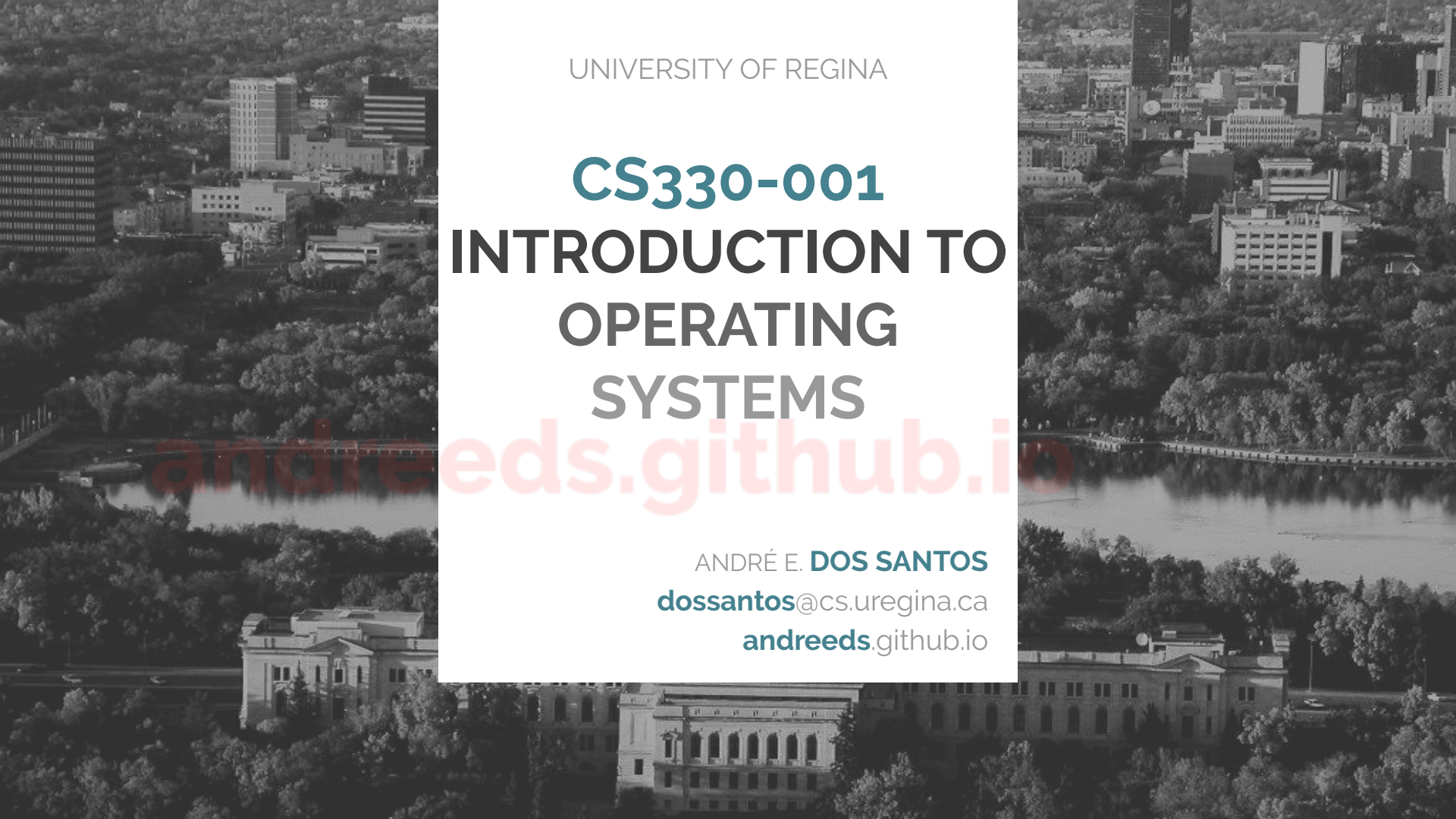


UNIVERSITY OF REGINA

# CS330-001
# INTRODUCTION TO OPERATING SYSTEMS



ANDRÉ E. **DOS SANTOS**
**dossantos**@cs.uregina.ca
**andreeds**.github.io

CS330-001
INTRODUCTION TO
OPERATING SYSTEMS

# OS STRUCTURES



ANDRÉ E. **DOS SANTOS**
**dossantos**@cs.uregina.ca
**andreeds**.github.io

# OS SERVICES

An OS provides an **environment** for the **execution of programs** by providing **services** to users and programs

# OS SERVICES

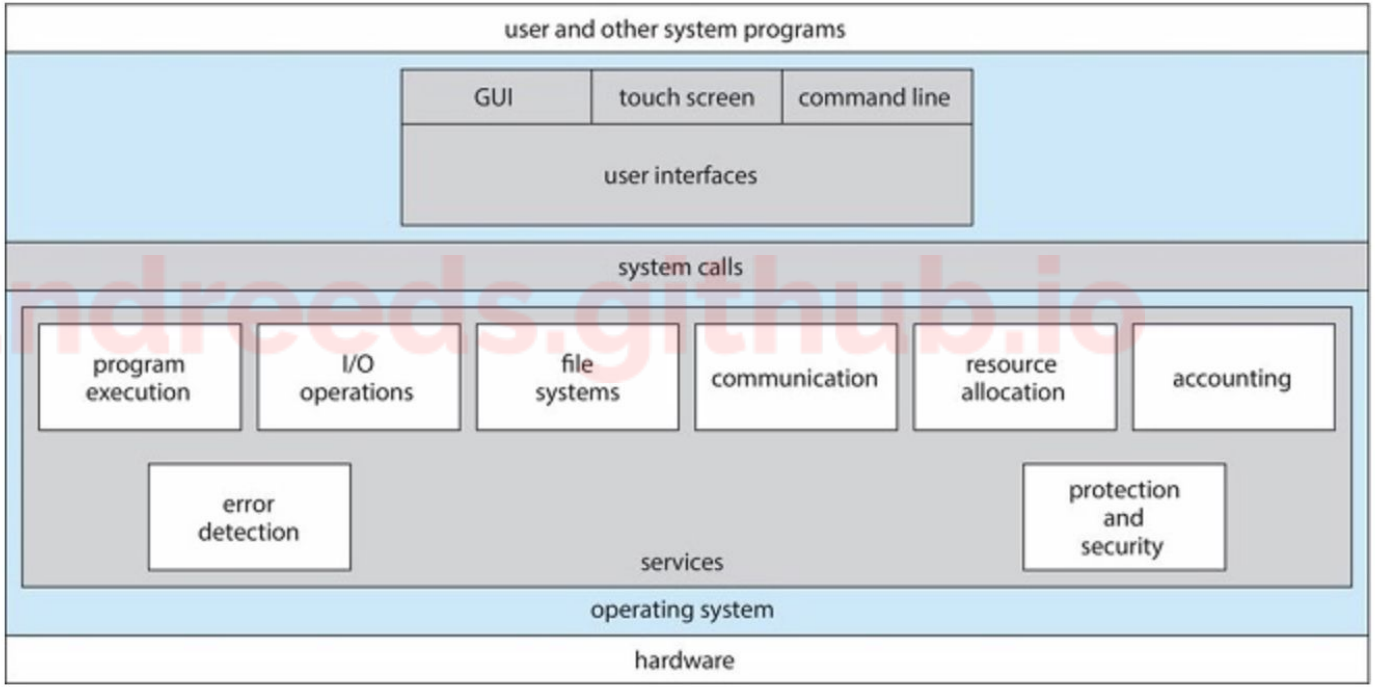
- One set of operating-system services provides functions that are helpful to the **user**:
  - **User interface**
  - **Program execution**
  - **I/O operations**
  - **File-system manipulation**
  - **Communications**
  - **Error detection**

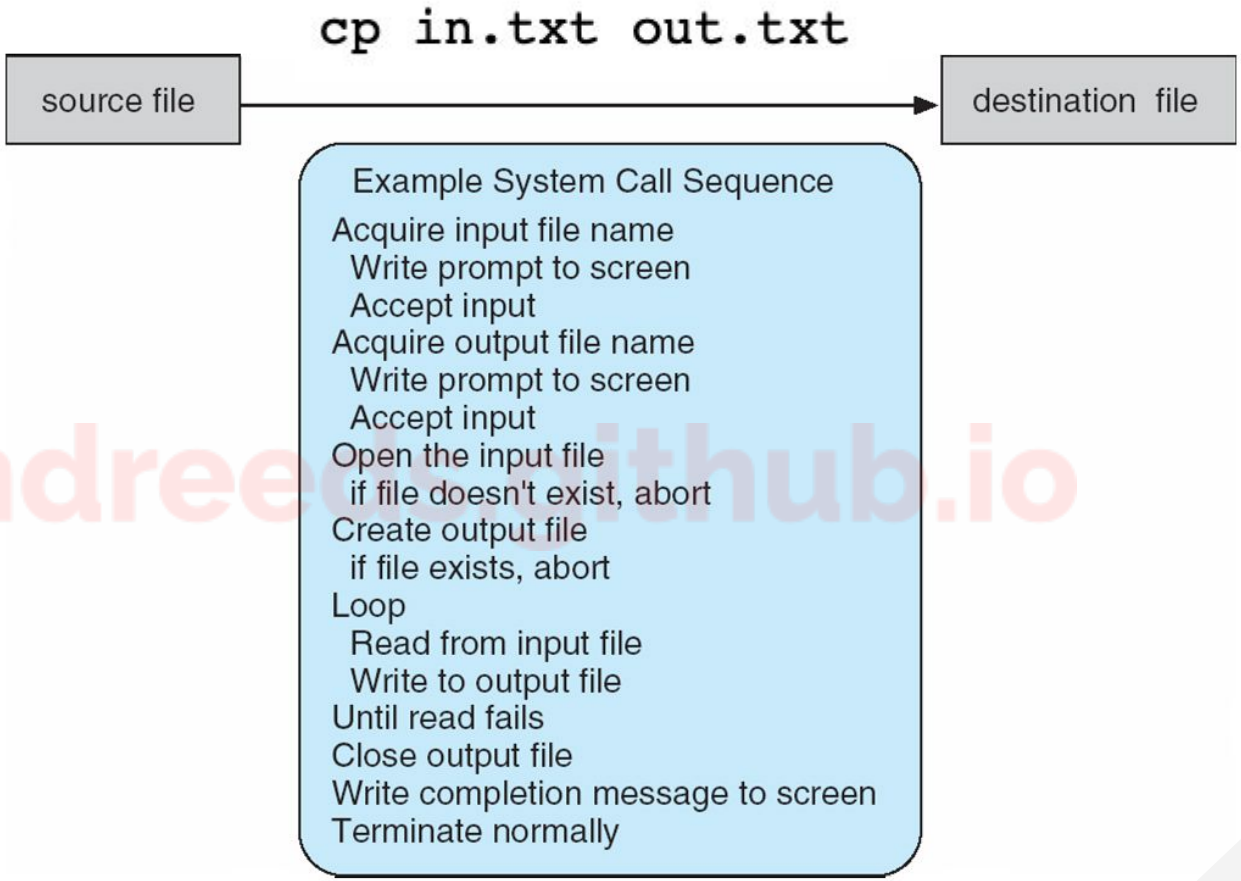
# OS SERVICES

- Another set of OS functions exists for ensuring the efficient operation of the **system itself** via resource sharing
  - **Resource allocation**
  - **Accounting**
  - **Protection and security**

# A VIEW OF OS SERVICES







# 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**) 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)

### *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



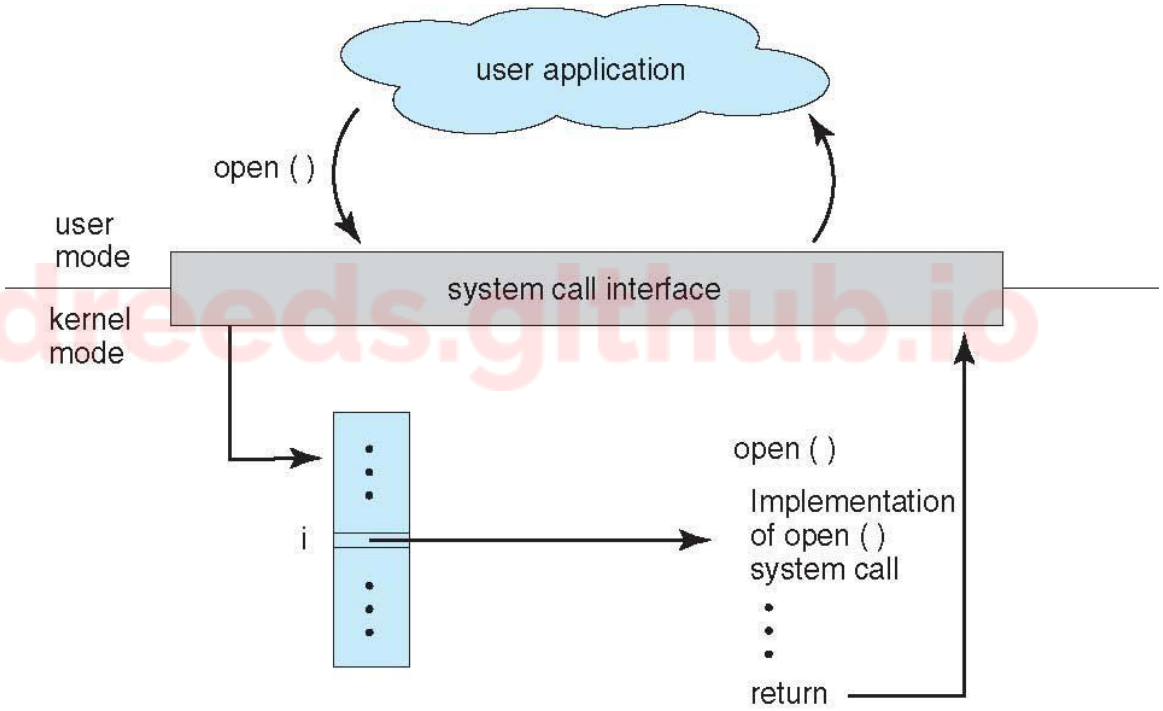
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `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.

# API – SYSTEM CALL
# OS RELATIONSHIP





## TYPES OF SYSTEM CALLS

- System calls can be divided into six major categories:
  1. PROCESS CONTROL
  2. FILE MANAGEMENT
  3. DEVICE MANAGEMENT
  4. INFORMATION MAINTENANCE
  5. COMMUNICATIONS
  6. PROTECTION

# SYSTEM CALLS

## 1 PROCESS CONTROL

- Provides facilities for creating, executing, suspending, and terminating processes, and for allocating and de-allocating memory
- A few examples:
  - Create a child process identical to the parent
    - `pid = fork ();`
  - Wait for a child to terminate
    - `w = waitpid (pid, &status, options);`
  - Terminate a process
    - `exit (status);`
  - Replace a process' core image
    - `status = execve (name, argv, argp);`

# SYSTEM CALLS

## 2 FILE MANAGEMENT

- Provides facilities for creating, opening, reading, writing, closing, and deleting files
- A few examples:
  - Open a file for reading, writing, or both reading and writing
    - `fd = open (filename, access);`
  - Close an open file
    - `status = close (fd);`
  - Read data from a file into a buffer
    - `n = read (fd, buffer, nbytes);`

## 3 DEVICE MANAGEMENT

- Provides facilities for managing main memory, disk drives, and other resources, and for obtaining a device's status information
- A couple of examples:
    - Make a directory
        - `status = mkdir (path, mode);`
    - Get the size of the blocks that a device driver reads from an audio device.
        - `status = ioctl (audio_fd, retrieve, &buffer);`

## 4 INFORMATION MAINTENANCE

- Provides facilities for transferring information between an application program and the operating system
- A few examples:
  - Get the elapsed time since January 1, 1970
    - `seconds = time (NULL);`
  - Get a file's status information
    - `status = stat (filename, &buffer);`

## 5 COMMUNICATION

- Provides facilities for inter-process communication
- A couple of examples:
  - Send a message to another process using the message passing model (UDP)
    - `nbytes = sendto (socket_fd, *buffer, length, flags, address, address-size);`
  - Set the values of locations in shared memory
    - `status = shmctl (segment, set, *buffer);`

## 6 PROTECTION

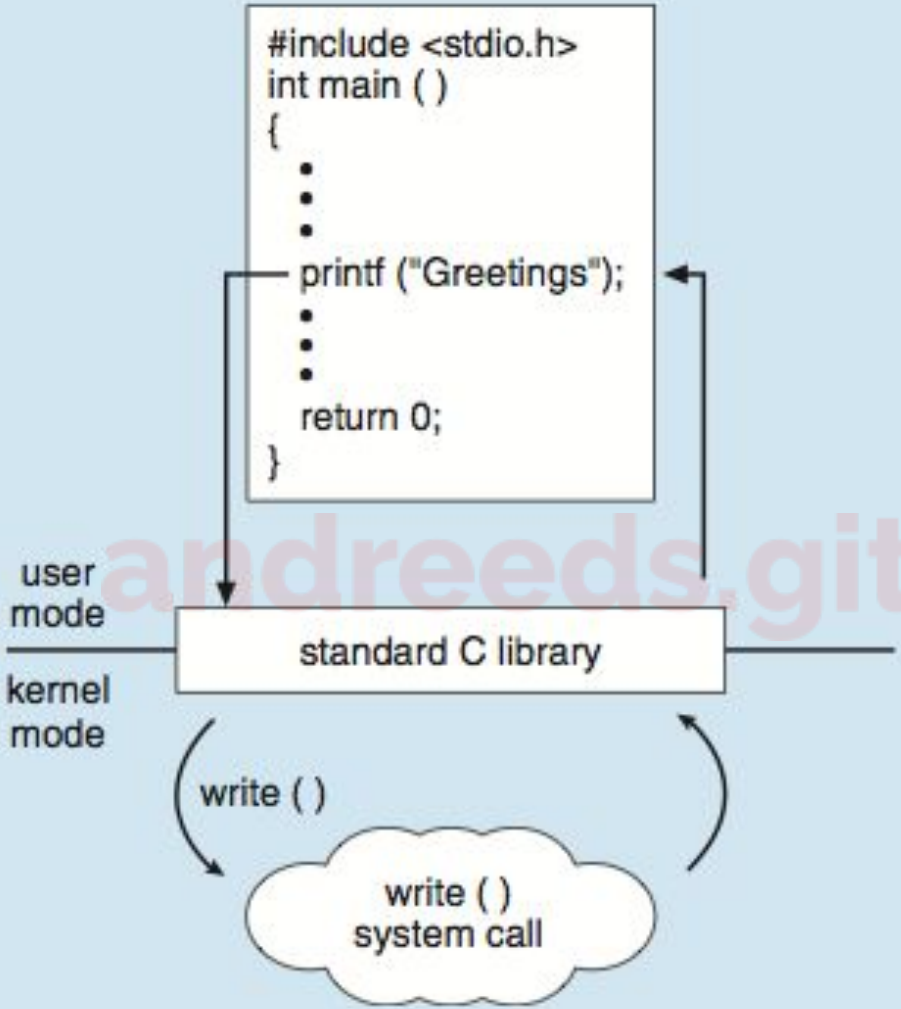
- Provides facilities for controlling access to computer resources
- A couple of examples:
  - Change a file's permissions
    - `status = chmod (filename, mode);`
  - Change the working directory
    - `status = chdir (directoryName);`

# SYSTEM CALLS

## EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS



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




# STANDARD C LIBRARY EXAMPLE

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

# OS STRUCTURE

- An OS is designed with specific goals in mind
  - These goals ultimately determine the OS's policies
- An OS implements these policies through specific mechanisms.
- Various ways to structure ones
  1. **Monolithic**
  2. **Layered**
  3. **Microkernel**
  4. **Modular**

# OS STRUCTURE

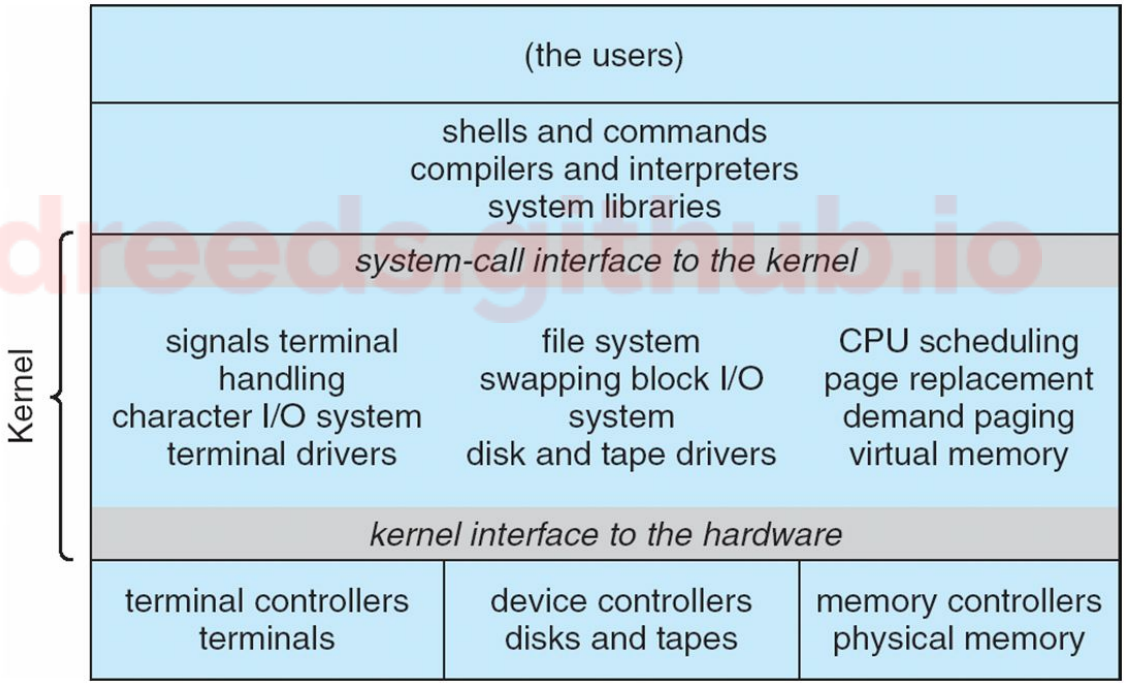
## 1 MONOLITHIC STRUCTURE

- A monolithic operating system has **no structure**; all functionality is provided in a **single, static binary file** that runs in a **single address space**
- Although such systems are **difficult to modify**, their primary benefit is **efficiency**

# TRADITIONAL UNIX SYSTEM STRUCTURE

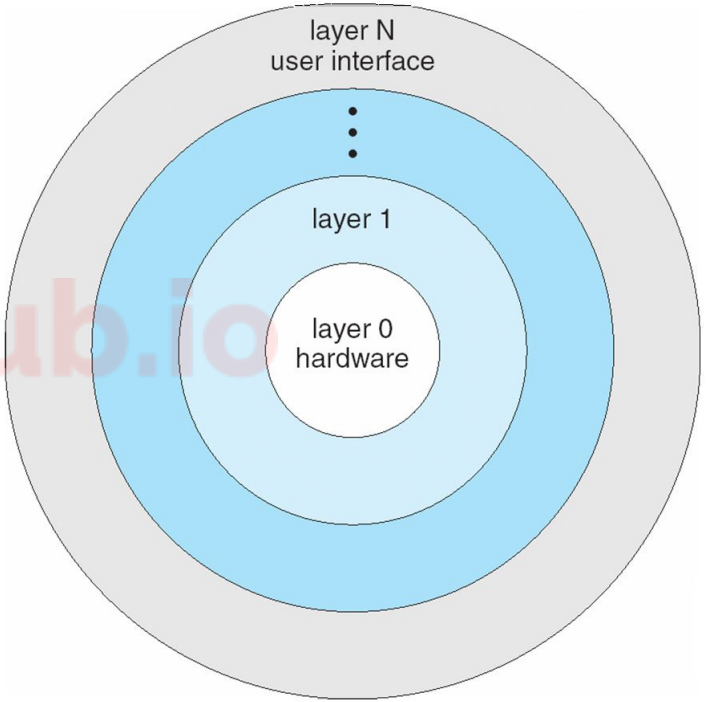


Beyond simple but not fully layered



## 2 LAYERED APPROACH

- A layered operating system is divided into a number of **discrete layers**, where the **bottom layer is the hardware interface** and the **highest layer is the user interface**
- Although layered software systems have had some success, this approach is generally not ideal for designing operating systems due to performance problems.

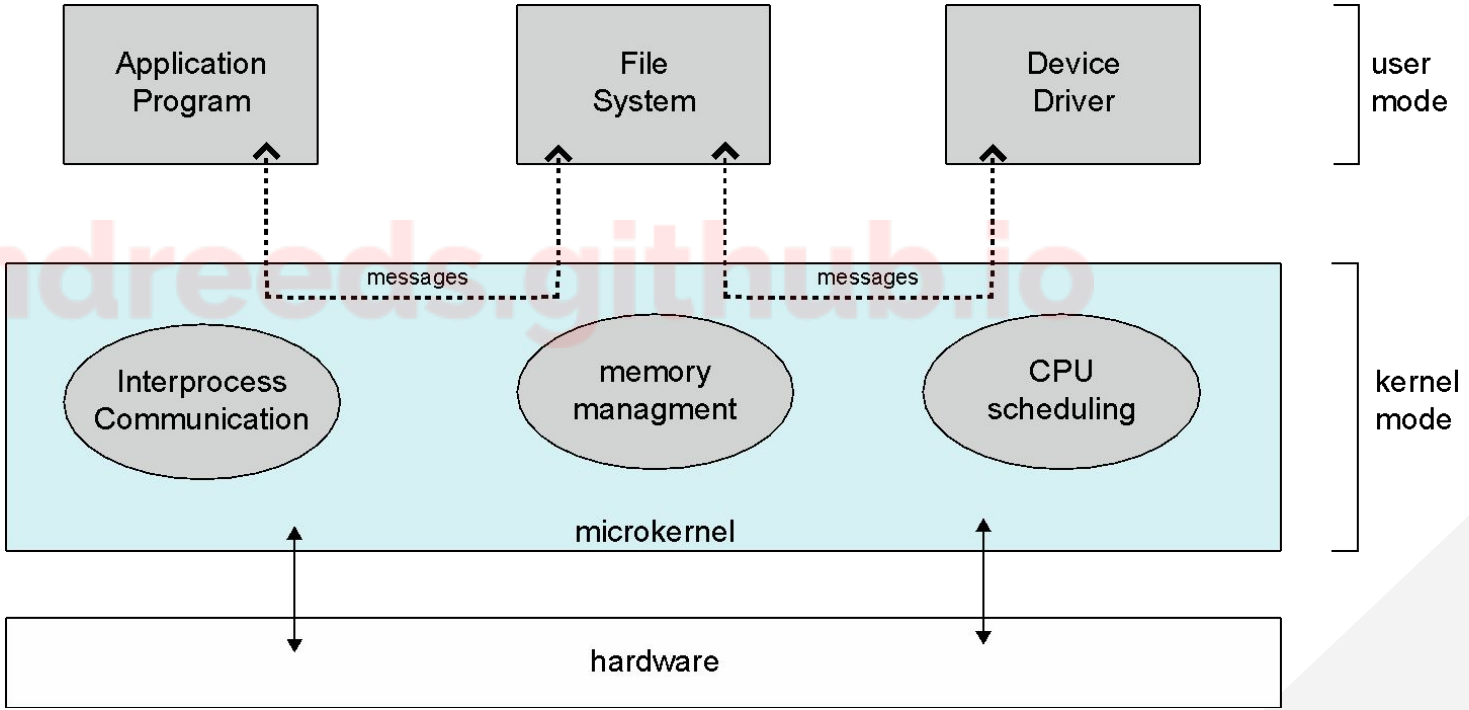


## 3 MICROKERNEL SYSTEM STRUCTURE

- 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**

# MICROKERNEL SYSTEM STRUCTURE

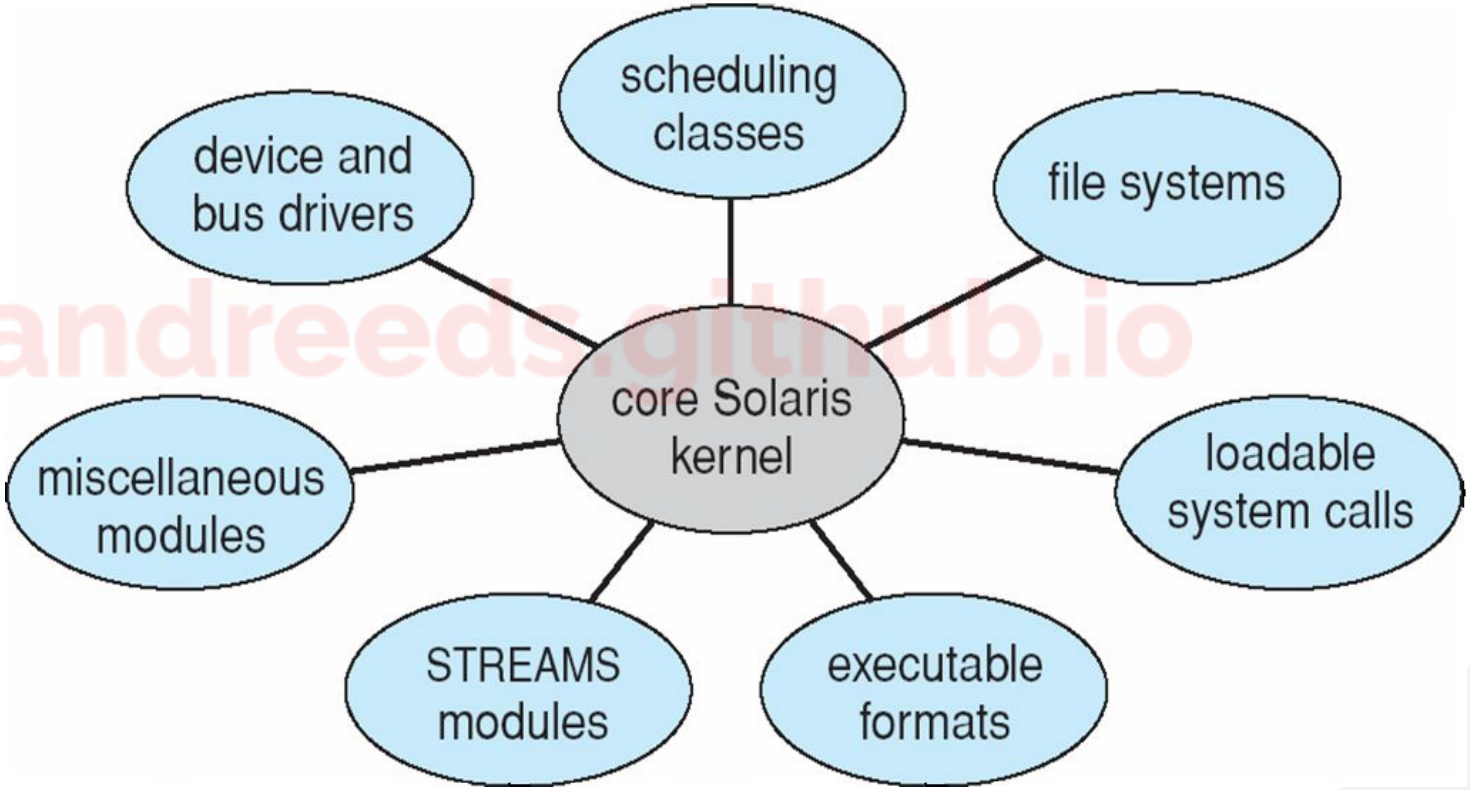




## 4 MODULES

- Many modern operating systems implement **loadable 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
- Modules vs. layered system
- Modules vs. microkernel approach

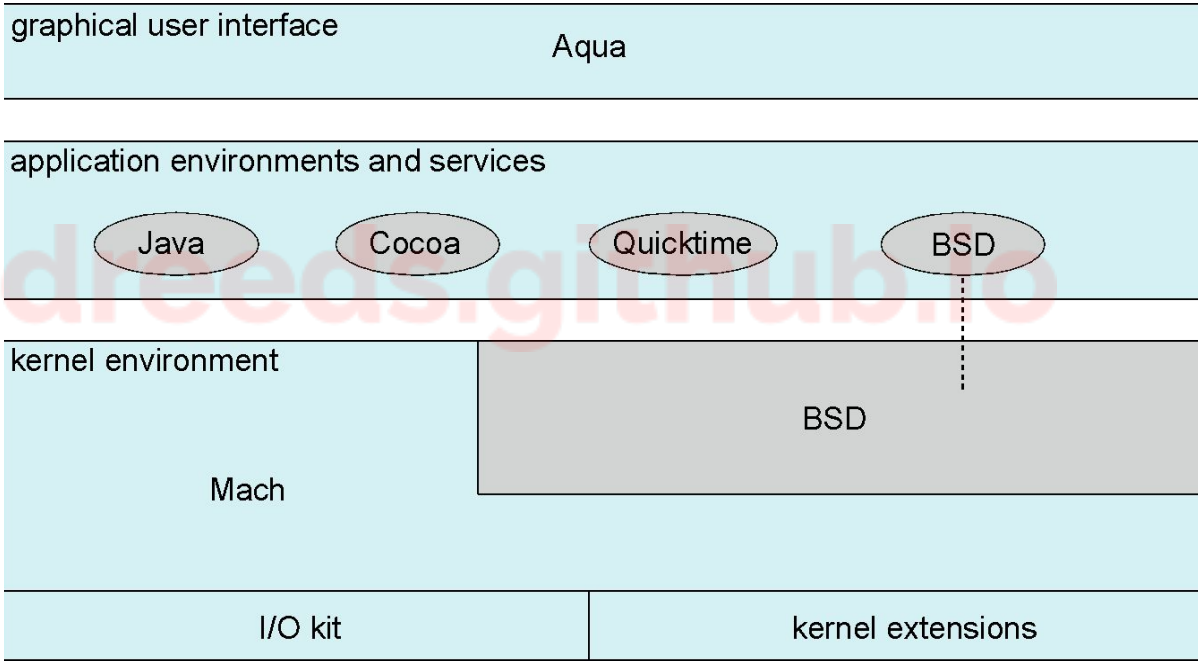
# SOLARIS
# MODULAR APPROACH



## HYBRID SYSTEMS

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

# MAC OS X STRUCTURE



# REVIEW QUESTIONS
## OS STRUCTURES



- What is the purpose of system calls?
- What is the purpose of the command interpreter? Why is it usually separate from the kernel?
- List five services provided by an operating system, and explain how each creates convenience for users
- Give an UNIX example of a system call for each of the six major categories

A _____ is an example of a systems program.

a) command interpreter
b) Web browser
c) text formatter
d) database system

_____ allows operating system services to be loaded dynamically.

a) Virtual machines
b) Modules
c) File systems
d) Graphical user interfaces