

# Operating System Structure

**CSD2180 Operating Systems**

BSc in Computer Science (IMGD / RTIS)

Singapore Institute of Technology / DigiPen Institute of Technology  
September 2021

# Attendance Taking

<https://forms.office.com/r/9Cebkc19ef>

- Log in to your SIT account to submit the form
- You can only submit the form once
- The codeword is **pomelo**



# Submit Your Team Information for Assignment One

<https://forms.office.com/r/kK2ytTdhEU>

- Log in to your SIT account to submit the form
- **You can only submit the form once**
- Form will close at **5:00PM, September 17, 2021**





# Operating System Services

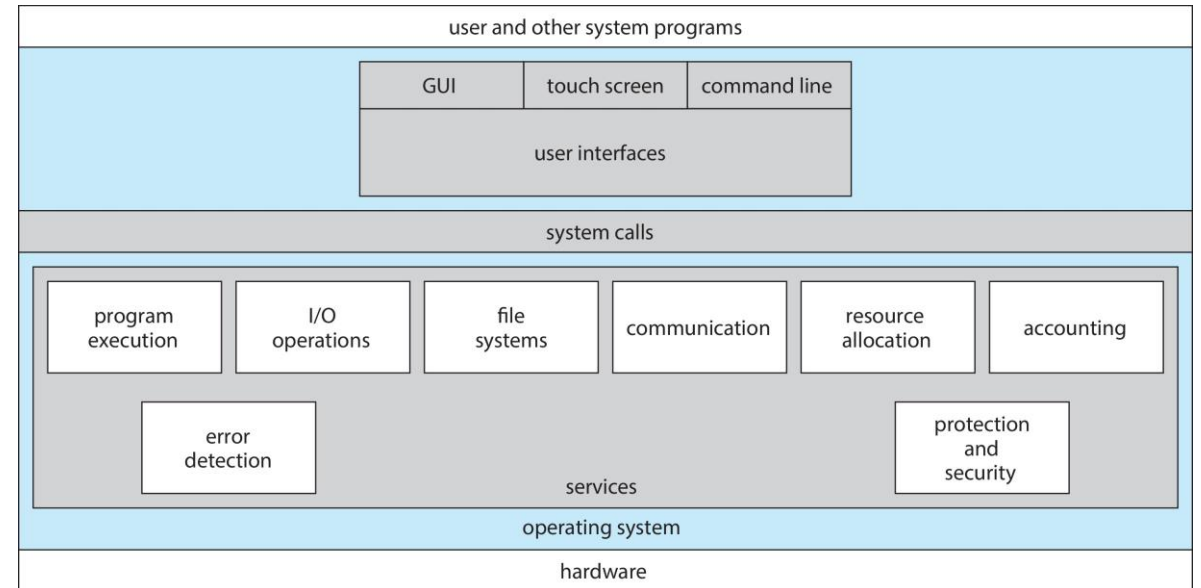


# Operating System Services in a Nutshell

Operating systems provide environment for execution of programs and services

Two general set of operating system services

- One set provide functions **helpful to the user**
- Another set exists for ensuring **efficient operation of the system** via resource sharing





# Functions Helpful to the User

## User interface

Most operating systems have some form of user interface, e.g., command-line (CLI), graphical user interface (GUI), touch-screen, etc.

## Program execution

The system must be able to load a program into memory and run that program, and later ending execution, either normally or abnormally (indicating error)

## Input / Output 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, etc.

# Functions Helpful to the User

## 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

Operating system needs to be constantly aware of possible errors

Errors may occur in the CPU and memory hardware, in I/O devices, in user program, etc.

For each type of error, OS should take the appropriate action to ensure correct and consistent computing

# Ensuring Efficient Operation of the System

## Resource allocation

When multiple users or multiple jobs running concurrently, resources must be allocated to each of them, e.g., CPU cycles, main memory, file storage, I/O devices

## Accounting and logging

To keep track of which users use how much and what kinds of computer resources

## Protection and security

The owners of information stored in a multiuser or networked computer system may want to control access to that information, concurrent processes should not interfere with each other (or with the operating system), etc.



Four decorative geometric shapes are scattered around the slide: a yellow circle in the upper left, a purple square in the upper center, a blue triangle in the upper right, and a yellow circle in the lower right. A blue triangle is also located in the lower left corner.

# User Interfaces

# User Interfaces

Several ways for users to interface with the operating system, such as (but not limited to)

Command-Line Interpreters

Graphical User Interfaces

Touch-screens

# Command-Line Interpreter (CLI)

## Allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors available to choose from (we call them **shells**)

**Main function:** Fetch a command from user and executes it

Sometimes the commands are built-in, sometimes they're just names of executable programs

```
Kali GNU/Linux Rolling kali tty2
kali login: kali
Password:
Linux kali 5.10.0-kali7-and64 #1 SMP Debian 5.10.28-1kali1 (2021-04-12) x86_64


The programs included with the Kali GNU/Linux system are free software:
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep 12 22:42:18 EDT 2021 on tty2
##(Message from Kali developers)
##
## We have kept /usr/bin/python pointing to Python 2 for backwards
## compatibility. Learn how to change this and avoid this message:
## https://www.kali.org/docs/general-use/python3-transition/
##
##(Run: #touch ~/.hushlogin# to hide this message)
(kali kali)~[~]
$ uname -a
Linux kali 5.10.0-kali7-and64 #1 SMP Debian 5.10.28-1kali1 (2021-04-12) x86_64 GNU/Linux

(kali kali)~[~]
$ cowsay "Hello, user"

  < Hello, user >
  -----
      \   ^__^
       (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

(kali kali)~[~]
$
```

How do you find out  
which commands are  
built-in, and which are  
executables? 

# Identifying Shell Built-Ins

```

weihan@DESKTOP-H448E90: /n
(Message from Kali developers)
We have kept /usr/bin/python pointing to Python 2 for backwards
compatibility. Learn how to change this and avoid this message:
⇒ https://www.kali.org/docs/general-use/python3-transition/
(Run: "touch ~/.hushlogin" to hide this message)
(weihan@DESKTOP-H448E90)~/mnt/c/Users/Goh Weihan
$ compgen -b | column
.          cd          echo          getopt      mapfile      set          type
:          command     enable       hash         popd         shift        typeset
[          compgen      eval         help         printf       shopt        ulimit
alias      complete     exec         history      pushd        source       umask
bg         compopt      exit         jobs         pwd          suspend      unalias
bind       continue    export       kill          read         test         unset
break      declare     false        let           readarray    times        wait
builtin    dirs        fc           local         readonly     trap
caller     disown      fg           logout        return       true
(weihan@DESKTOP-H448E90)~/mnt/c/Users/Goh Weihan
$

```

# Identifying Executable Programs

```

weihan@DESKTOP-H448E90: /n
(Message from Kali developers)
We have kept /usr/bin/python pointing to Python 2 for backwards
compatibility. Learn how to change this and avoid this message:
⇒ https://www.kali.org/docs/general-use/python3-transition/
(Run: "touch ~/.hushlogin" to hide this message)
(weihan@DESKTOP-H448E90)-[/mnt/c/Users/Goh Weihan]
$ date
Mon 13 Sep 2021 02:47:43 AM +08

(weihan@DESKTOP-H448E90)-[/mnt/c/Users/Goh Weihan]
$ which date
/usr/bin/date

(weihan@DESKTOP-H448E90)-[/mnt/c/Users/Goh Weihan]
$ whoami
weihan

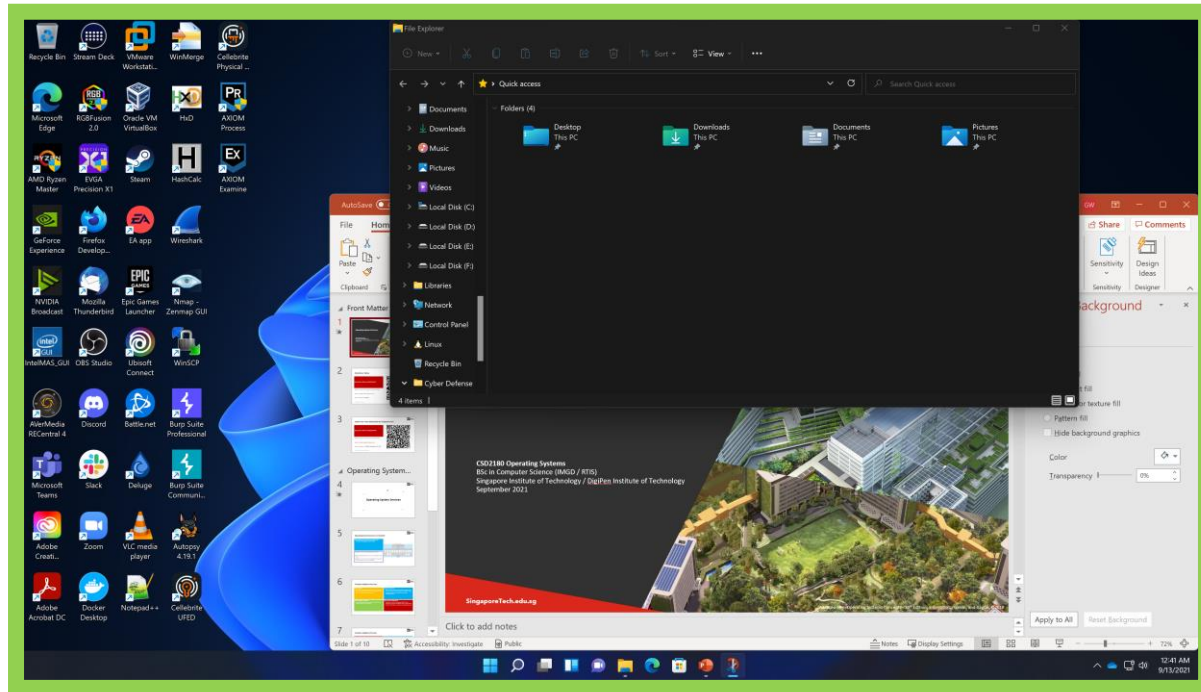
(weihan@DESKTOP-H448E90)-[/mnt/c/Users/Goh Weihan]
$ which whoami
/usr/bin/whoami

(weihan@DESKTOP-H448E90)-[/mnt/c/Users/Goh Weihan]
$

```



# Graphical User Interface (GUI)



## User-friendly, graphical desktop interface

- Interaction using mouse, keyboard, and monitor
- Icons used to represent files, programs, actions, etc.
- Various mouse actions over objects in the interface cause various actions, e.g., provide information, display options, execute function, open directory, etc.
- Invented at Xerox PARC in the 1970s

# CLIs and GUIs

**Many systems today include both CLI and GUI interfaces**

Microsoft Windows is GUI-based with a CLI command shell

Apple macOS has a GUI interface with a UNIX kernel underneath

Unix and Linux have CLIs with optional GUI interfaces (CDE, KDE, GNOME)

# What about touch- screen interfaces?

What type of interface  
will you implement if  
you are developing an  
operating system? 🖱️

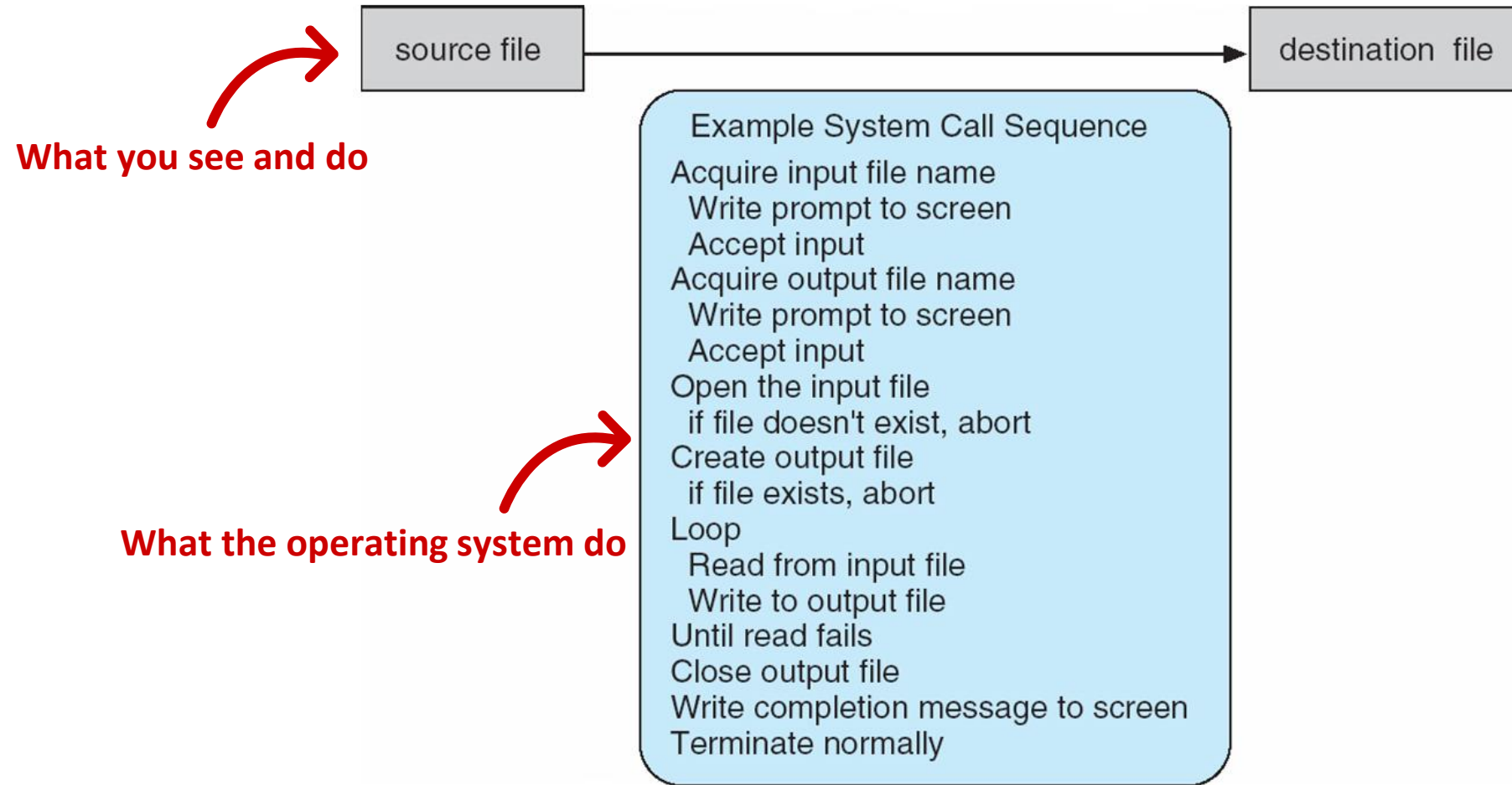


# System Calls

What happens when  
you run a program to  
copy a file to another  
file? 🍔



# An Example of a File Copy



# System Calls in a Nutshell

System calls provide an interface to the services made available by an operating system

**Example APIs:** Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and macOS)

Typically written in a **high-level language** (C / C++); some may be written in **assembly**

Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use

- API specifies a **set of functions** that are available to an application programmer
- Programmers access an API via a **library of code** provided by the operating system, e.g., *libc* for programs written in C for UNIX or Linux

Why use APIs rather  
than system calls  
directly? 🍔

# Why APIs?

**An application programmer using an API can expect the program to compile and run on any system that supports the same API**

**Actual system calls can often be more detailed and difficult to work with than the API**

# System Call Implementation

Typically, a number is associated with each system call

System call interface maintains a table indexed according to these numbers

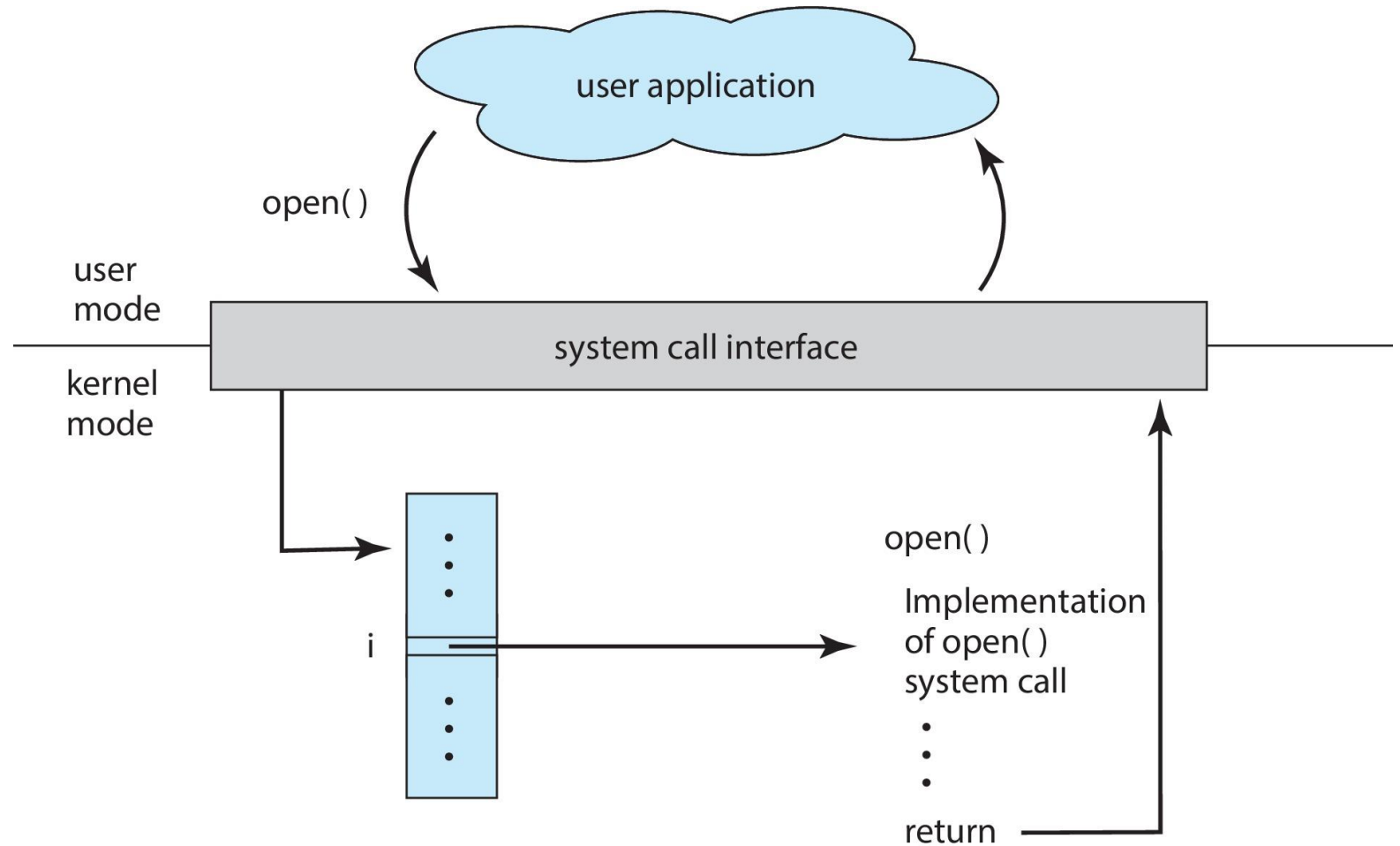
The system call interface invokes intended system call in operating system kernel and returns status of the system call and any return values

The caller does not need to know how the system call is implemented

- Just needs to obey API and understand what the operating system will do as a result of the call
- Most details of operating system interface hidden from programmer by API and managed by the runtime environment\*

**\*Runtime Environment:** The full suite of software needed to execute applications written in a given programming language, including its compilers or interpreters as well as other software, such as libraries and loaders

# Relationship Between API, System Call, and the OS

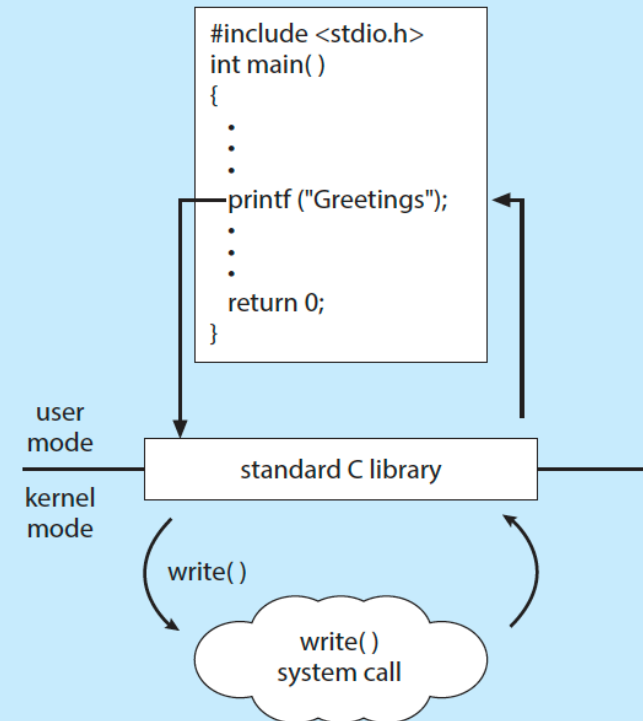




# Standard C Library Example

## THE 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:



# System Call Parameter Passing

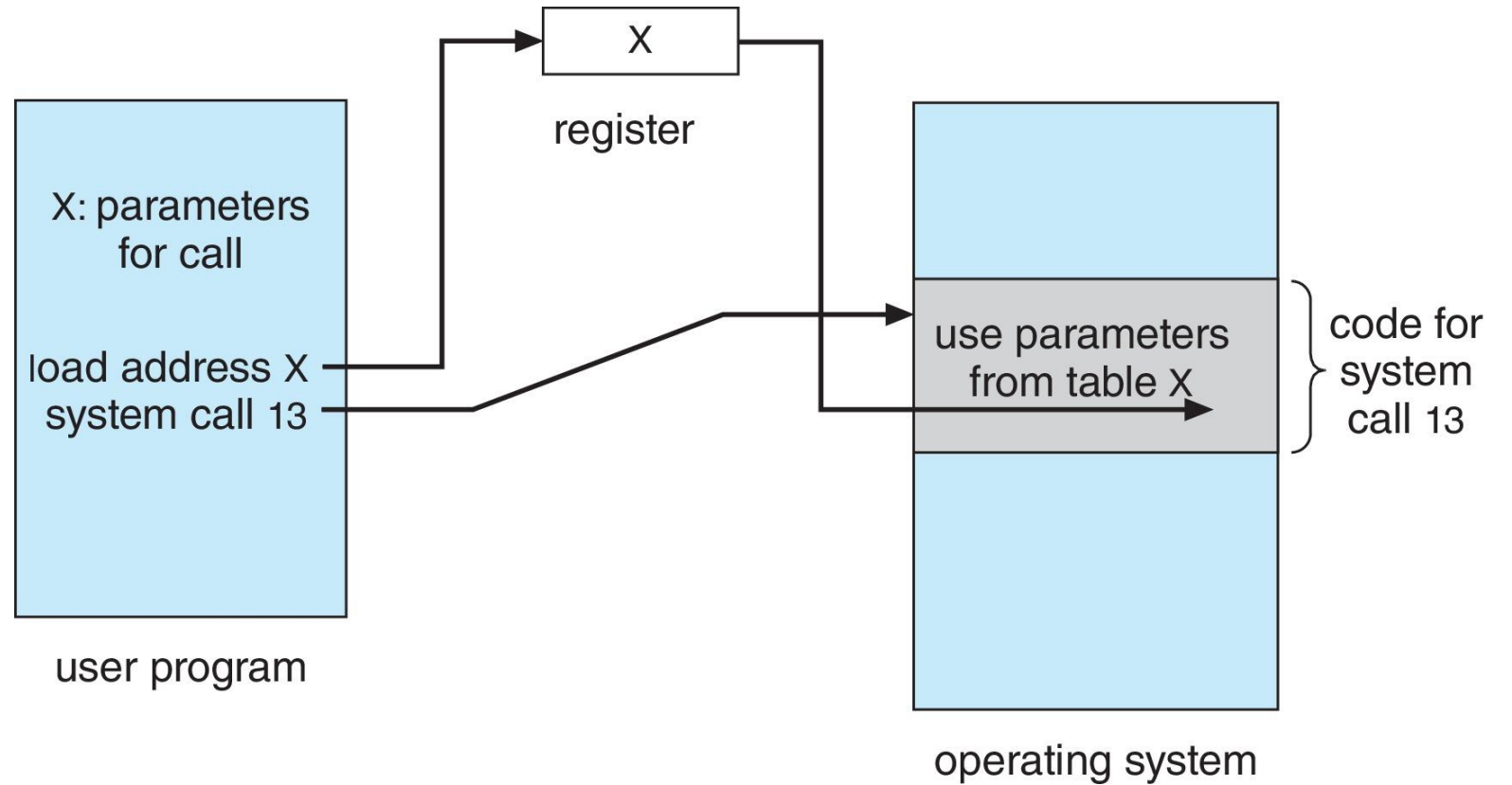
Often, more information is required than simply the identity of desired system call

Pass the parameters in **registers**

Parameters **stored in a block, or table, in memory**, and address of block passed as a parameter in a register

Parameters **placed, or pushed, onto the stack** by the program and popped off the stack by the operating system

## Passing Parameters via Table



What are the  
advantages of each  
approach to parameter  
passing? 

# Types of System Calls

## Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

## File management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

## Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

# Types of System Calls

## Information maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

## Protection

- control access to resources
- get and set permissions
- allow and deny user access

## Communications

- create, delete communication connection
- send, receive messages to host name or process name (if message passing model)
- create and gain access to memory regions (shared-memory model)
- transfer status information
- attach and detach remote devices



# Examples of Windows and UNIX System Calls

## EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
<b>Process control</b>	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
<b>File management</b>	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
<b>Device management</b>	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
<b>Information maintenance</b>	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
<b>Communications</b>	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
<b>Protection</b>	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



# System Services

# System Services

**System services provide a convenient environment for program development and execution**

Most users' view of the operation system is defined by system services, not the actual system calls

System services provide a convenient environment for program development and execution

Some of them are simply user interfaces to system calls; others are considerably more complex

## Different types of system services

- File manipulation
- Status information
- Programming language support
- Program loading and execution
- Communications
- Background services
- Application programs

# Types of System Services

## File management

- Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

## File modification

- Text editors to create and modify files
- Special commands to search contents of files or perform transformations of the text

## Programming-language support

- Compilers, assemblers, debuggers and interpreters sometimes provided

## Status information

- Some ask the system for info - date, time, amount of available memory, disk space, number of users
- Others provide detailed performance, logging, and debugging information
- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a registry - used to store and retrieve configuration information

# Types of System Services

## Program loading and execution

- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

## Communications

- Provide the mechanism for creating virtual connections among processes, users, and computer systems
- Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

## Background services

- Launched at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as *services*, *subsystems*, *daemons*



# Operating System Implementation

# Operating System Implementation

## Much variation in language used

- Early operating systems written in assembly language
- Then system programming languages like Algol, PL/1
- Now, C / C++

## In reality, usually a mix of languages

- Lowest levels in assembly
- Main body in C
- Systems programs in C, C++, or scripting languages like PERL, Python, shell scripts, etc.

**More high-level language easier to port to other hardware, but slower**

**Emulation can allow an OS to run on non-native hardware**



# Operating System Structure



# Operating System Structure

General-purpose OS is very large program

## Different approaches for operating system structure

- Monolithic structure
- Layered approach
- Microkernels
- Modules

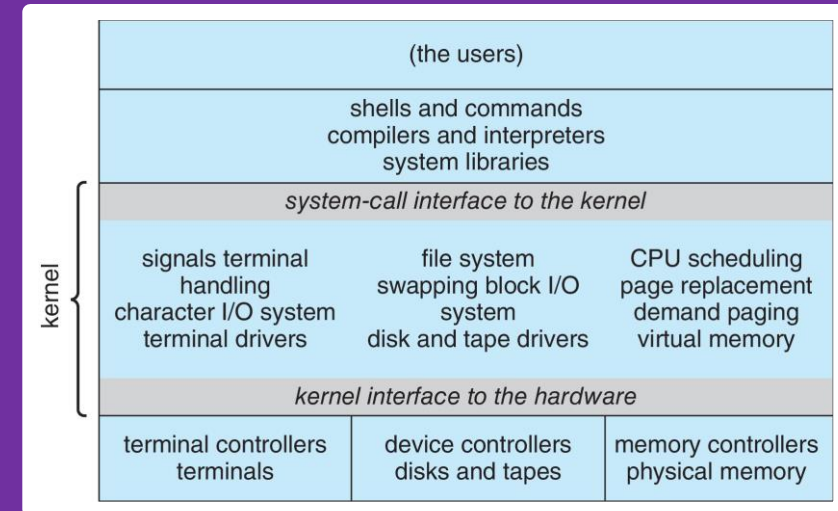
*The simplest structure for organizing an operating system is no structure at all. That is, place all of the functionality of the kernel into a single, static binary file that runs in a single address space. This approach - known as a **monolithic structure** - is a common technique for designing operating systems.*

# Monolithic Structure (e.g., Original UNIX)

Limited by hardware functionality, the original UNIX operating system had limited structuring

The UNIX operating system 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



# What are the advantages of a monolithic structure?



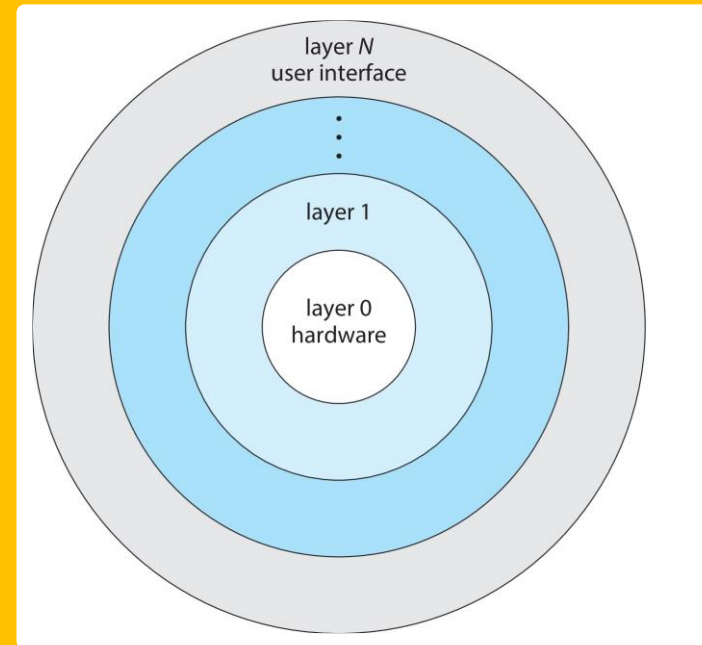
# What are the disadvantages of a monolithic structure?



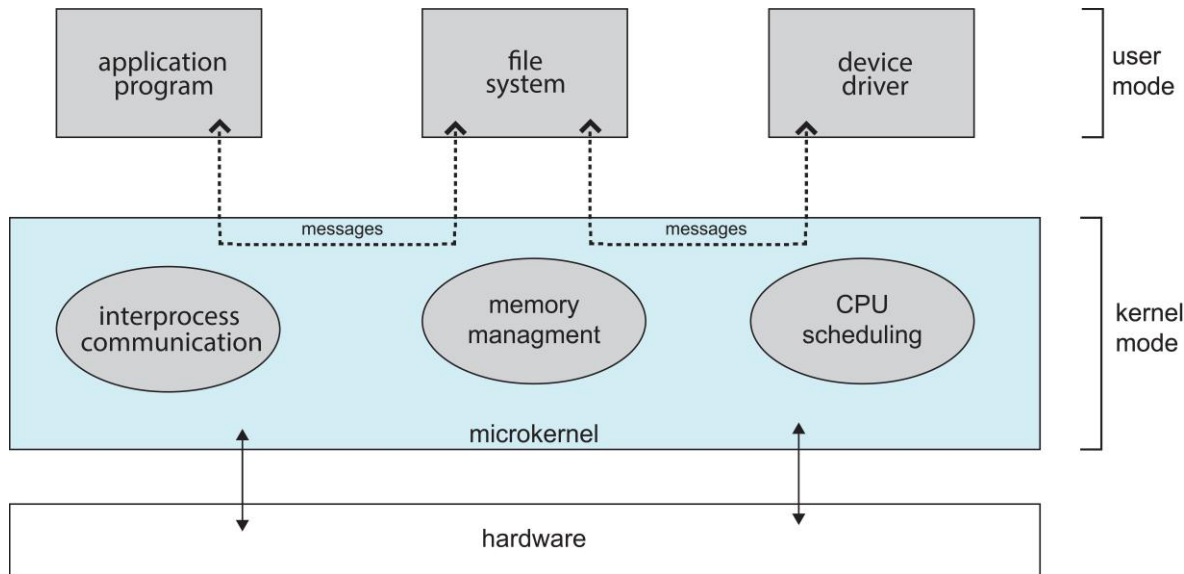
# Layered Approach

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



# Microkernel Structure



**Moves as much from the kernel into user space**

Communication takes place between user modules using **message passing**

Performance overhead due to user space to kernel space communication

What then are the  
benefits of a  
microkernel structure?





# Microkernel Structure Benefits

**Easier to extend a microkernel**

**Easier to port the operating system to new architectures**

**More reliable**

**More secure**

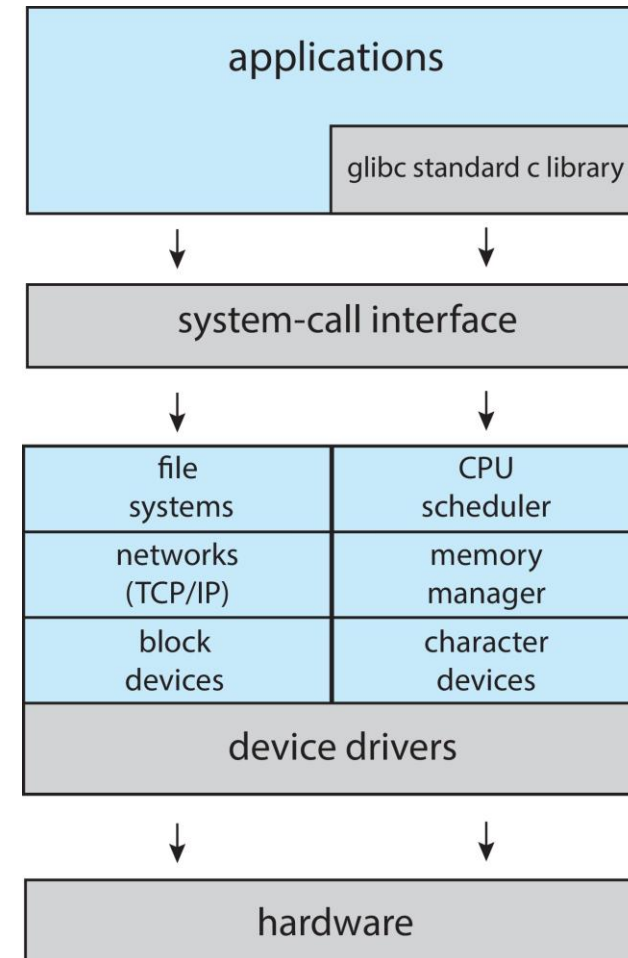
# Modular Design

Many modern operating systems implement **loadable kernel modules (LKMs)**

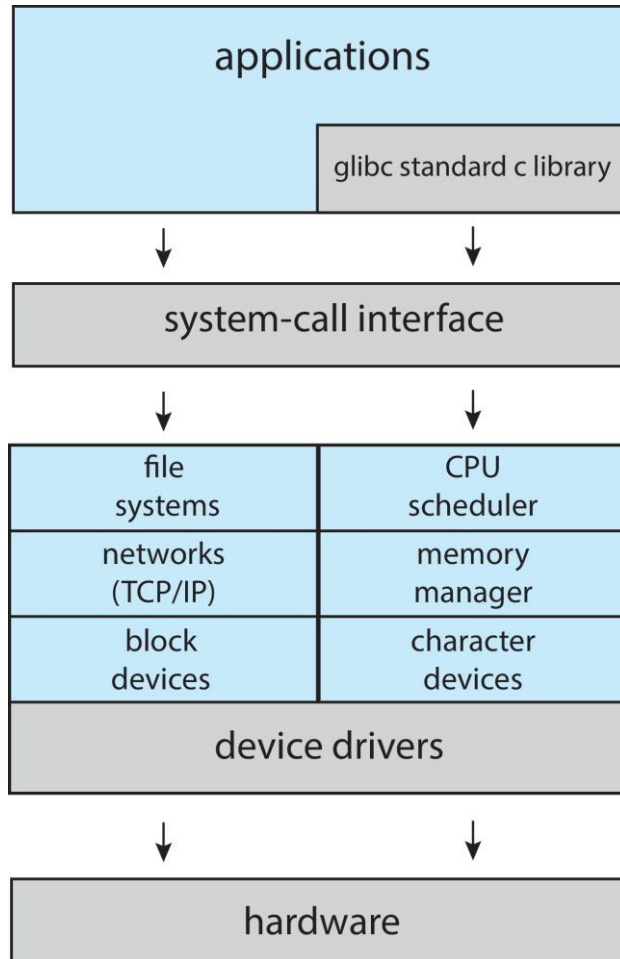
- Uses object-oriented approach
- Each core component is separate
- Each talks to the others over known interfaces
- Each is loadable as needed within the kernel

Similar to layers but with more flexibility

- Linux, Solaris, etc. employ modules
- The Linux kernel is monolithic but have a modular design that allows the kernel to be modified during run time



# Hybrid Systems



Most modern operating systems do not belong to **one pure model**

Hybrid systems combines **multiple approaches** to address e.g., performance, security, usability needs, etc.

## Examples

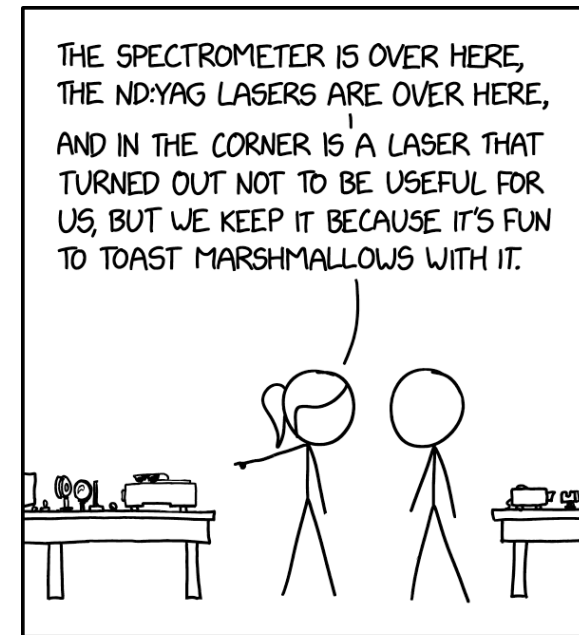
- Linux and Solaris kernels run in kernel address space (hence **monolithic**), but have a **modular design** for dynamic loading of functionality
- Windows mostly **monolithic**, plus **microkernel** for different subsystems

# Questions? Thank You!

 [Weihan.Goh {at} Singaporetech.edu.sg](mailto:Weihan.Goh@Singaporetech.edu.sg)

 <https://www.singaporetech.edu.sg/directory/faculty/weihan-goh>

 <https://sg.linkedin.com/in/weihan-goh>



EVERY LAB IN EVERY FIELD HAS SOME PIECE OF EQUIPMENT LIKE THIS.