

# COS3023

# OPERATING SYSTEMS AND CONCURRENCY

---

Topic 1– Operating System – Introduction (Part2)

Lecturer : Ms Shafrah

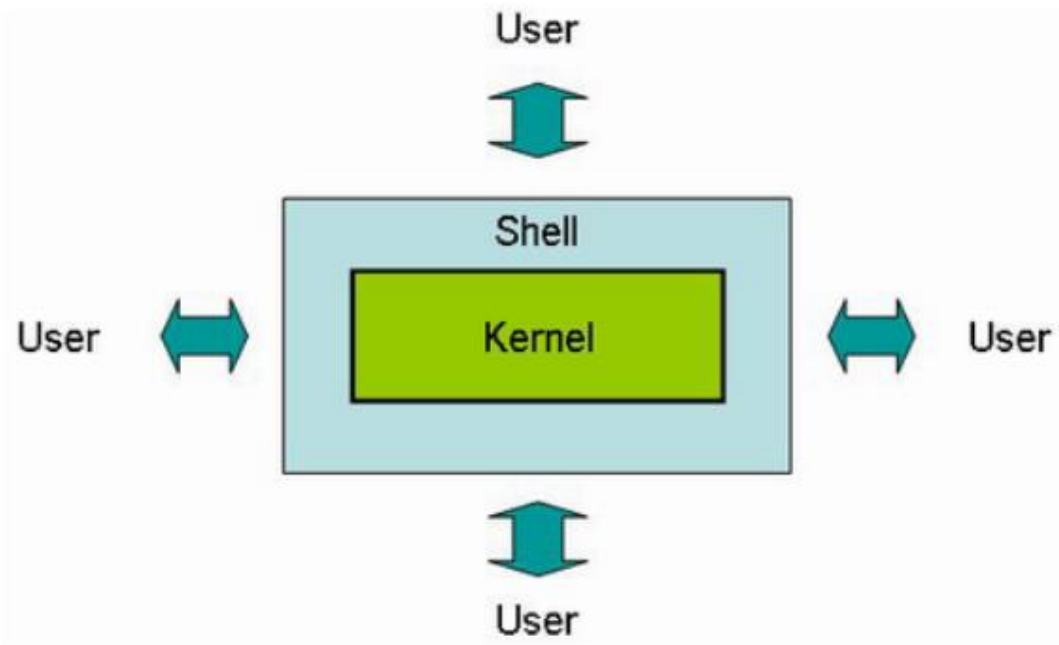
# PRESENTATION OUTLINE

---

- Component of Operating System
- User Operating System Interface
- System Calls

# COMPONENT OF OPERATING SYSTEM

---



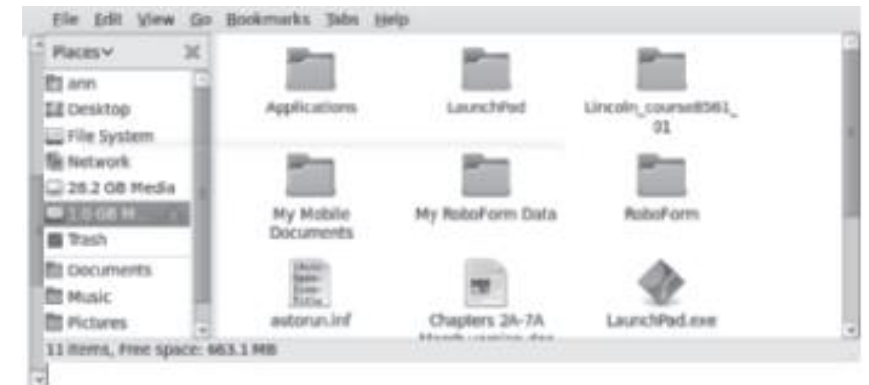
The shell as the interface between the kernel and the user.

- Command Interpreter
- The Kernel
- File Manager
- Device Drivers
- Memory Managers
- Scheduler

# USER OPERATING SYSTEM INTERFACE

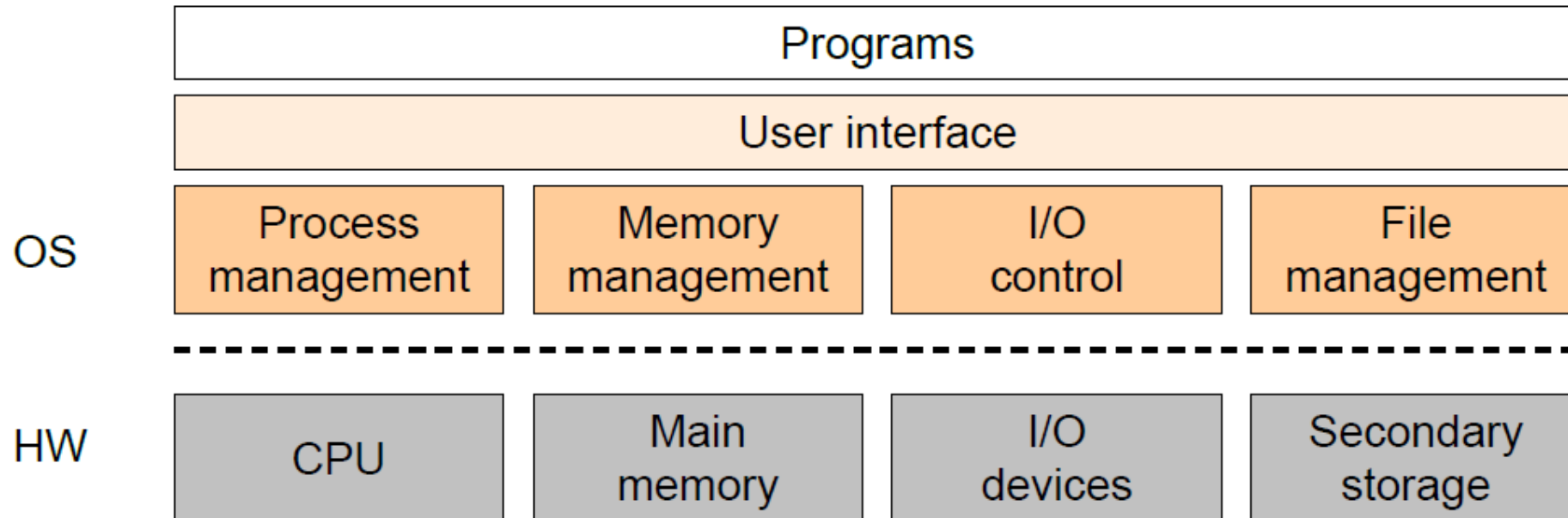
---

- is the portion of the operating system that users interact with directly.
  - the user interface consisted of :
    - commands typed on a keyboard and displayed on a monitor
- or
- a menu option from a list.

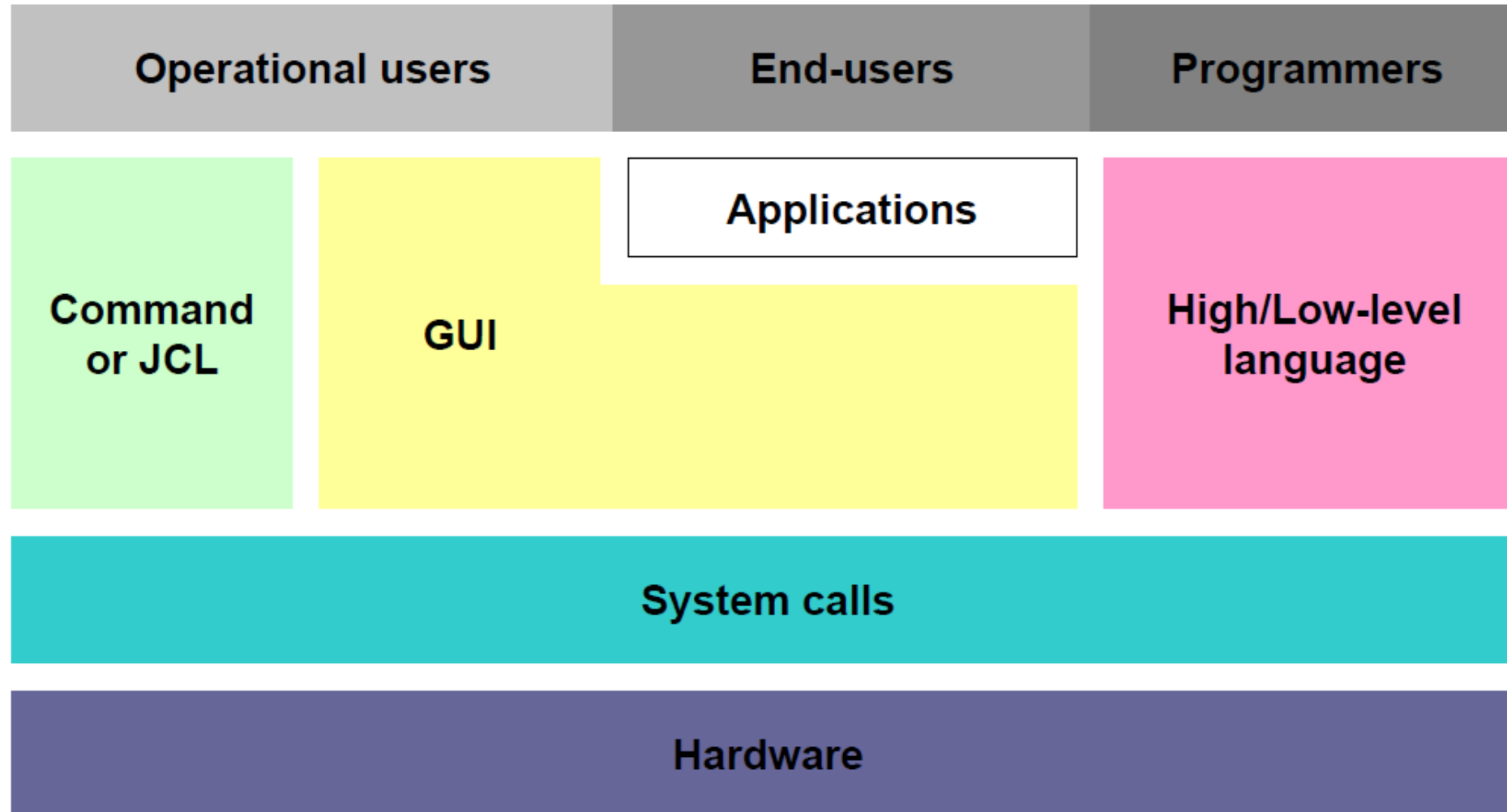


# Overview

- The user interface is the necessary link for human and computer interaction (HCI)




# User Interface Model



# Classes of Users

- Computer users can generally grouped into three different categories:

- *Programmer*

- *Operational* 

- *End-user*

- These groups define *roles*, not individuals, therefore a user can belong to more than category (e.g. both programmer and operational)

# Types of Interfaces

- There are 4 different types of interfaces
  - *System call*
  - *Command language*
  - *Job Control Language (JCL)*
  - *Graphical User Interface (GUI)*





# System Calls

- System calls are conceptually similar to the procedures / functions of a program
- The main difference is that the object code of procedures / functions is part of the calling program, while **the system call code is part of the OS**
- **High-level APIs** (Application Programming Interface) can be **provided with the OS to use system calls**

# SYSTEM CALLS

---

- System calls provide the interface between a process and the OS.



- A system call is used to complete a specific task:

- Reading input file name
- Opening a file
- Deletion of file
- Closing a file
- Abnormal termination

## HOW SYSTEM CALLS WORK

— mechanism → to allow processes (many programs/applications) to request → services to perform → privileged operations.  
— interface ← user-level processes + OS kernel.

- **User Process Request:** When a user process needs to perform a task that requires <sup>\*</sup>privileged access or involvement of the operating system, it makes a system call request. This request is typically initiated through a special instruction or a software interrupt.
- **Transition to Kernel Mode:** In response to the system call request, the processor switches from user mode to kernel mode. In kernel mode, the process gains access to the protected resources and functionalities of the operating system.
- **System Call Execution:** The kernel, upon receiving the system call request, identifies the requested service or operation. It verifies the parameters provided by the process, checks for permission and validity, and performs the requested action.
- **Kernel-User Transition:** After the system call is executed, the processor switches back to user mode, and the process resumes its execution. The result or output of the system call may be returned to the process, allowing it to continue with its execution or take further actions based on the outcome.

# 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()

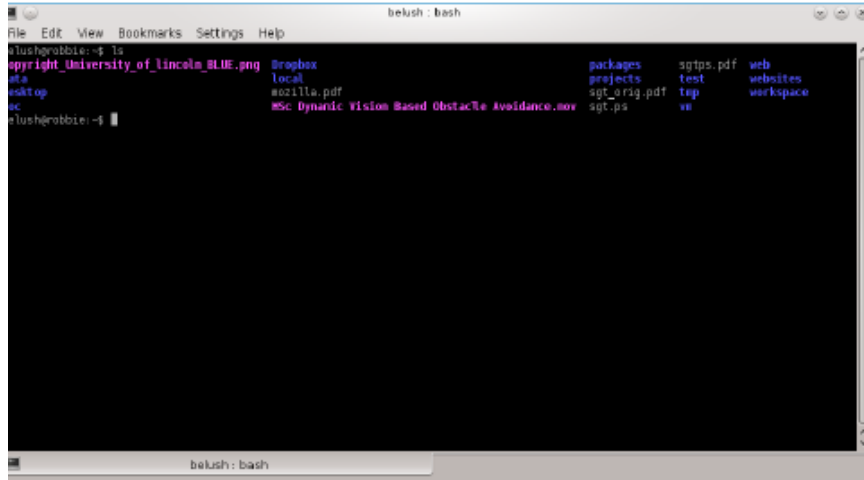
# COMMAND LINE INTERFACE (CLI)

---

- Command line interface (CLI) is a text-based interface that is used to operate software and operating systems while allowing the user to respond to visual prompts by typing single commands into the interface and receiving a reply in the same way.
- CLI is an older method for interacting with applications and operating systems and is used to perform specific tasks required by users.
- CLI allows a user to perform tasks by entering commands. Its working mechanism is very easy, but it is not user friendly.
- MS-DOS is the best example of CLI.

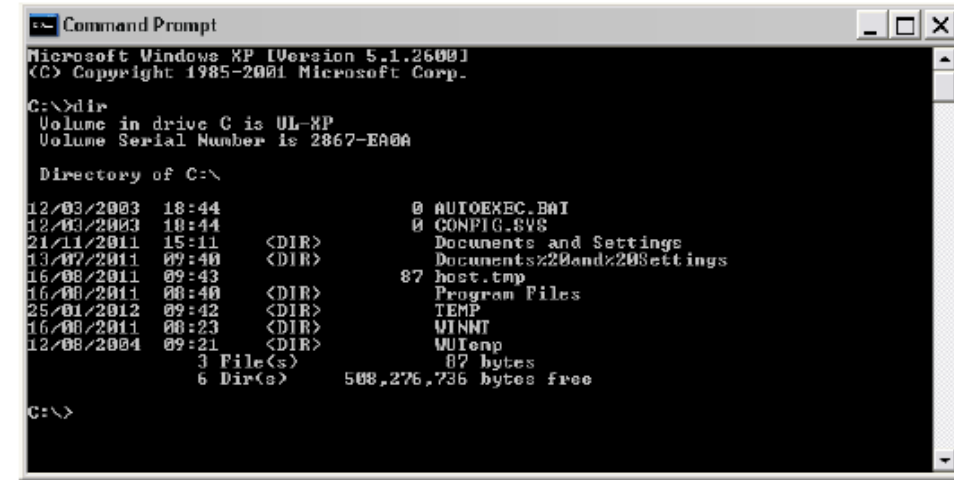
# Command Languages

- Designed to interact with the OS from a terminal
- Mainly used by operational users



A screenshot of a Linux shell terminal window titled 'belush: bash'. The window shows a file manager interface with a list of files and directories. The files include 'copyright\_University\_of\_lincoln\_BLUE.png', 'local', 'mozilla.pdf', 'MSc Dynamic Vision Based Obstacle Avoidance.mov', 'packages', 'projects', 'sgt\_orig.pdf', 'sgt.ps', 'sgtps.pdf', 'test', 'tmp', 'web', 'websites', and 'workspace'. The terminal prompt is 'belush@robber:~\$'.

Linux shell



A screenshot of a DOS terminal window titled 'Command Prompt'. The window shows the output of the 'dir' command in the C:\ directory. The output lists files and directories with their dates, times, and sizes. The files include 'AUTOEXEC.BAT', 'CONFIG.SYS', 'Documents and Settings', 'Documents\20and20Settings', 'host.tmp', 'Program Files', 'TEMP', 'WINNT', 'WINtsp', and 'File(s)'. The total size of the files is 87 bytes, and the total size of the directory is 508,276,736 bytes free. The terminal prompt is 'C:\>'.

DOS terminal

# UNIX Commands

- UNIX commands have the following structure:  
`command options arguments #comment`
  - `options` are “switches” modifying the behaviour of the command (e.g. `-h`, `-v`, ...)
  - `arguments` are “inputs” to the command
  - `comment` is just a string without effects
- Commands can be combined in *script* files

# MS-DOS Commands

- • Commands have usually the following structure: —

**command**   **options**   **arguments**

- **options** (e.g. /A, /W) and **arguments** have the same functions as in the UNIX shell

- MS-DOS is a single user command language (i.e. no commands for login, password, file permissions, etc. like in UNIX shell)
- The equivalent of the shell script in UNIX, but much simpler, is the **batch file**



# JOB CONTROL LANGUAGE

---

- Job control language (JCL) is a scripting language executed on an IBM mainframe operating system. It consists of control statements that designate a specific job for the operating system.
- Job Control Language (JCL) is the command/scripting language of Multiple Virtual Storage (MVS), which is the commonly used Operating System in the IBM Mainframe computers.
- JCL identifies the program to be executed, the inputs that are required and location of the input/output and informs the Operating System through Job control Statements.
- In mainframe environment, programs can be executed in batch and online modes. JCL is used for submitting a program for execution in batch mode.



# EXAMPLE OF JCL JOB

---

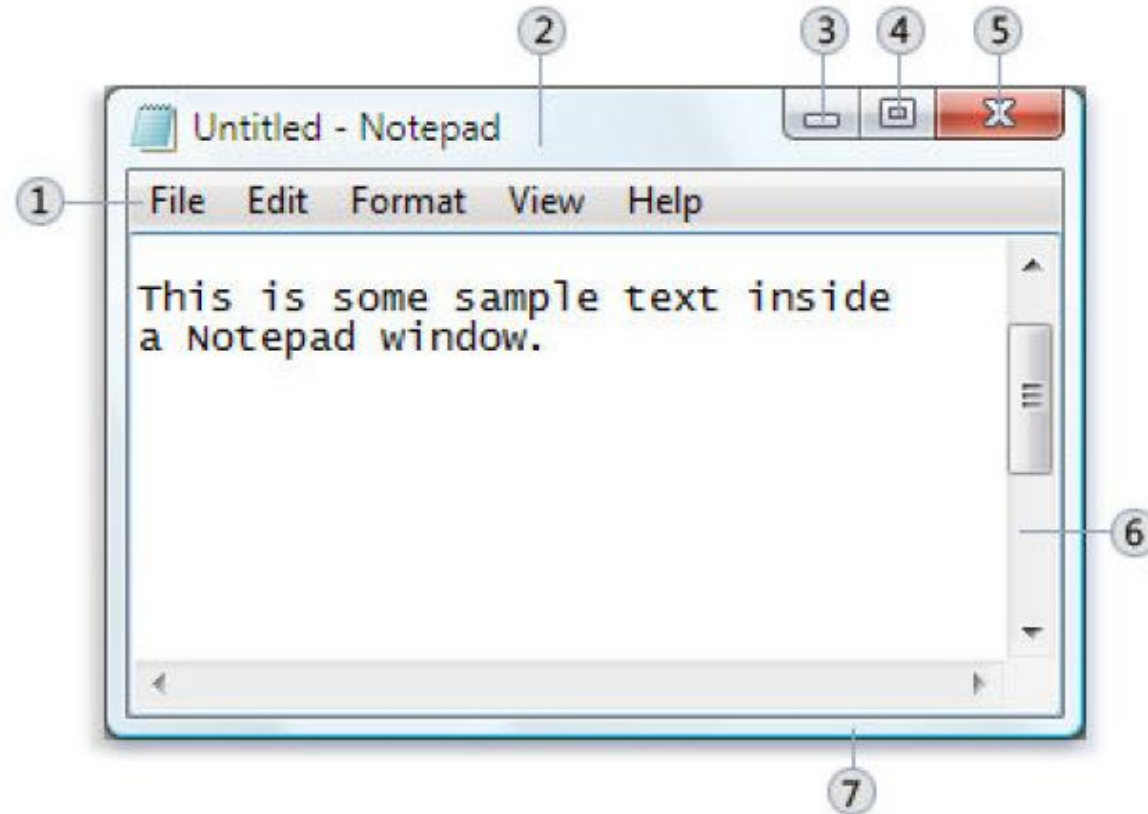
```
//MYJOB JOB (ACCT), 'JOB DESCRIPTION', CLASS=A, MSGCLASS=X, NOTIFY=USER1  
//STEP1 EXEC PGM=MYPROGRAM  
//INPUT DD DSN=INPUT.FILE,DISP=SHR  
//OUTPUT DD DSN=OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,SPACE=(CYL,(5,5))  
//SYSOUT DD SYSOUT=*
```

# Graphical User Interfaces



- The first GUI was designed by Xerox Corporation in the 80's, but exploited commercially by Apple
- GUI systems share the following features:
  - on-screen **desktop** with overlapping **windows**
  - **pointer** (or cursor, usually controlled with a mouse device)
  - several graphical features such as **icons, buttons, menus, scroll bars, selection lists, dialog boxes**, etc.
  - a.k.a. **WIMP** – **Windows, Icons, Menus, Pointer**

# Graphical User Interfaces



① Menu bar

② Title bar

③ Minimize button

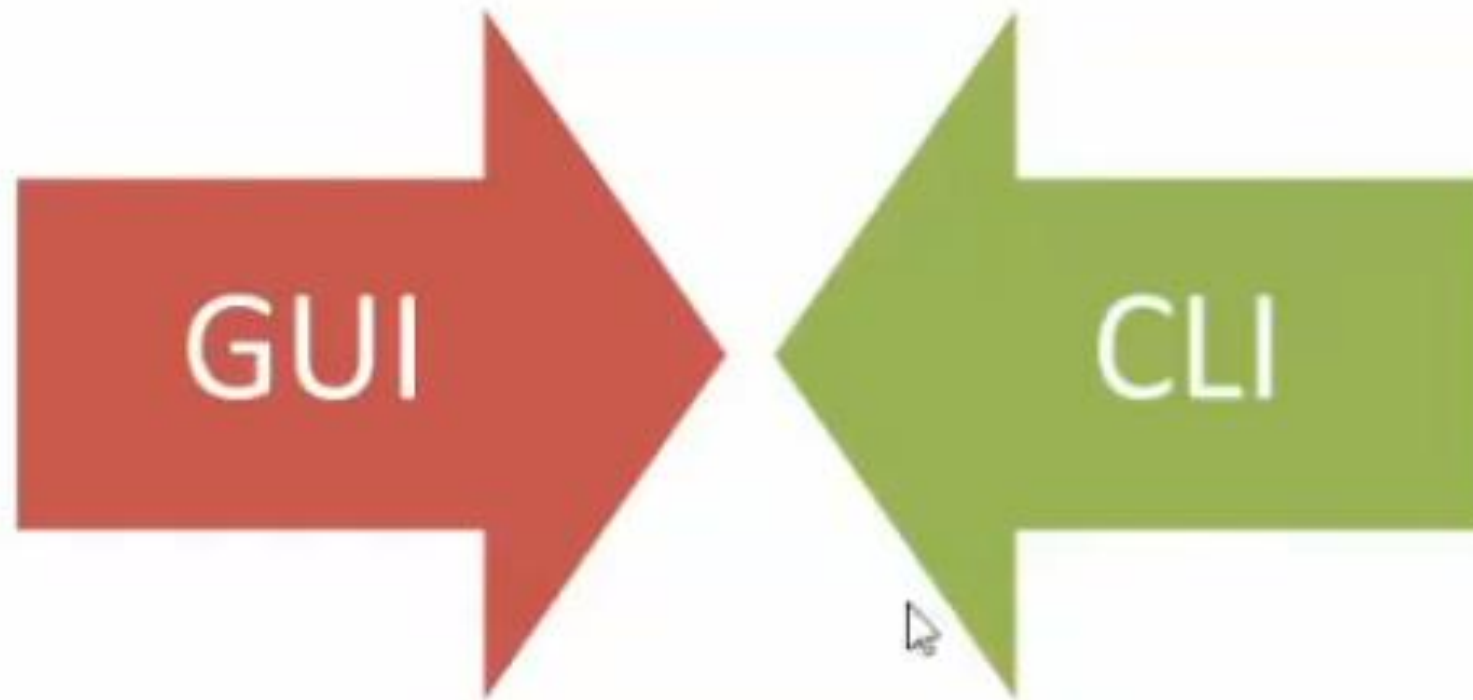
④ Maximize button

⑤ Close button

⑥ Scroll bar

⑦ Border

# Compare



## GUI

- Easier to learn
- Better for multi-tasking

## CLI

- Have to learn commands
- More system control
- Can be faster
- Less system resources

# QUESTION:

- Although graphical user interface (GUI) is the predominant user interface in today's operating systems, there are still users that prefer a command line interface (CLI). What are the advantages in the use of a GUI over CLI and vice versa? What is the target audience for each type of interface?



THANK YOU

