OPERATING SYSTEMS · LECTURE 2

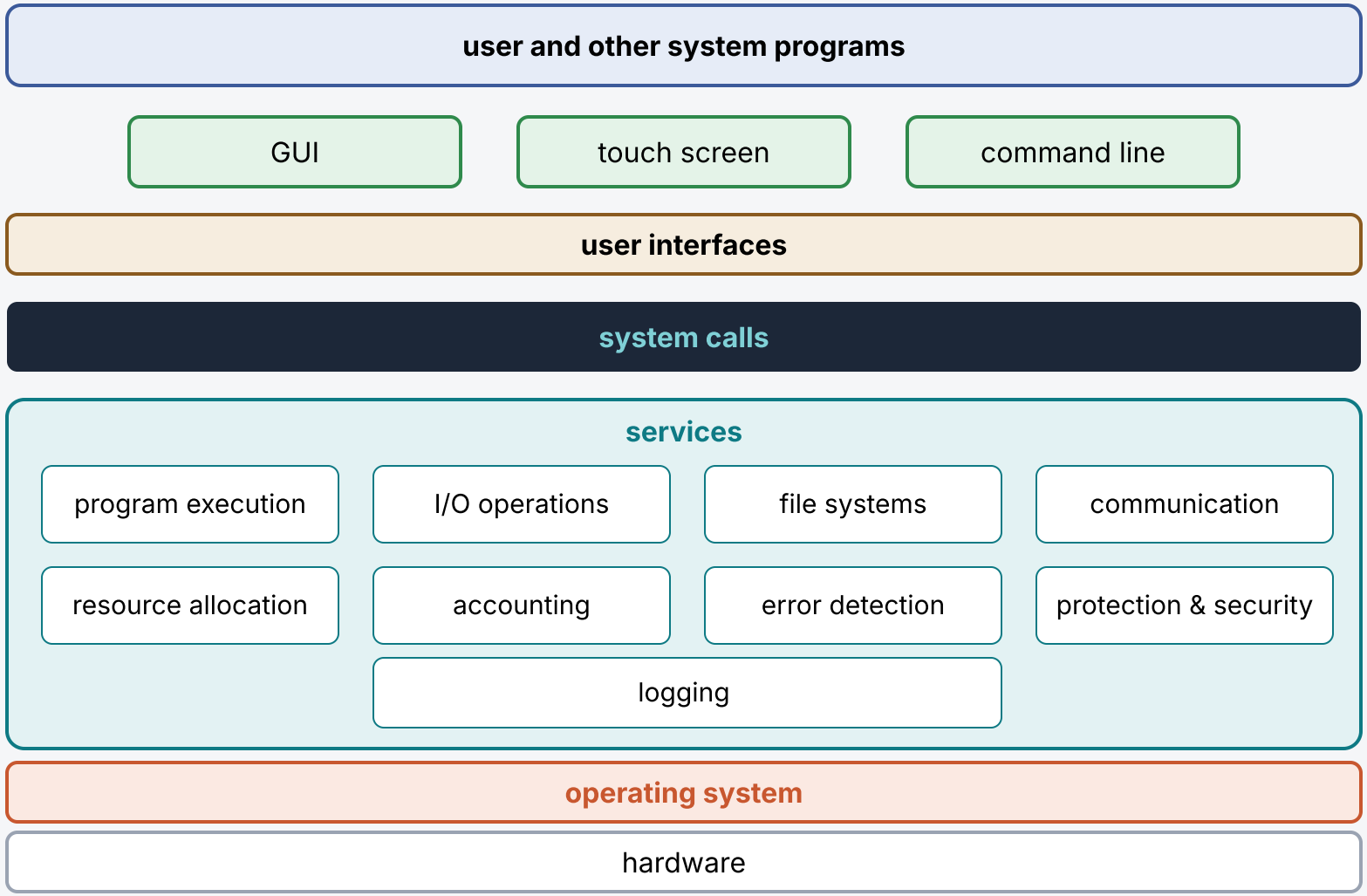
# Structure of Operating Systems

Services · interfaces · system calls · linkers and loaders




Lecturer: Aidana Karibayeva

# 01 A view of operating-system services



02

# Three ways to look at an OS

The OS provides the **environment** in which programs run. Internally, systems differ a lot — so we study them from several angles.



### Services

what the system provides



### Interface

what users and programmers see



### Components

its parts and how they connect

# 03 Services that help the user

### User interface

GUI, touch screen or command line

### Program execution

load a program into memory, run it, end it — normally or with an error

### I/O operations

users cannot drive devices directly — the OS does I/O for them

### File-system manipulation

create, delete, read, write, search, list; permissions

### Communications

between processes — via **shared memory** or **message passing**, locally or over a network

### Error detection

CPU, memory, devices, user programs → halt, kill the process, or return an error code

# 04 Services that keep the system efficient

## Resource allocation

share resources among many running processes

`CPU cycles` `main memory` `file storage` `I/O devices`

e.g. CPU-scheduling routines consider CPU speed, cores, the process

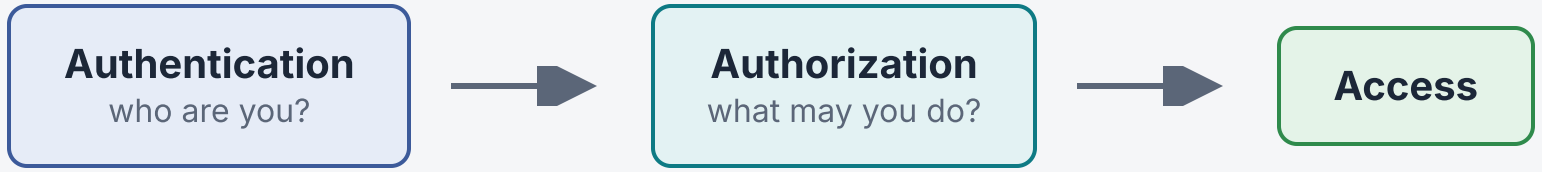
## Logging

record which programs use how much of which resources — for **billing** or **usage statistics** to tune the system

## Protection & security

**Protection**: all access to resources is controlled; processes cannot interfere.
**Security**: users authenticate (password); defend against outsiders

# 05 Authentication vs authorization



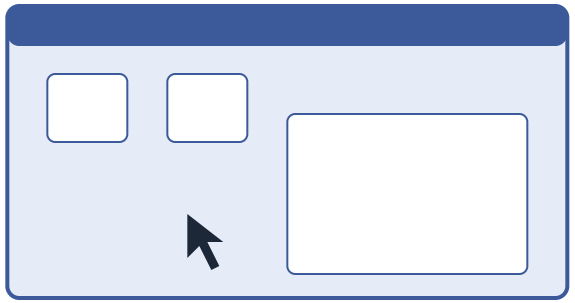

| | AUTHENTICATION | AUTHORIZATION |
|---|---|---|
| **Checks** | whether users are who they claim to be | what users can and cannot access |
| **How** | credentials: password, OTP, biometrics | permissions set by policies and rules |
| **Order** | first step | after successful authentication |

# 06 User interfaces

```
$ ls -l
$ cp in.txt out.txt
$ rm file.txt
$
```

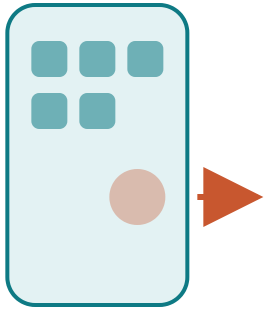
## Command line (CLI)

text commands. Shells: Bourne-Again (bash), C shell, Korn shell...



## Graphical (GUI)

windows, icons, menus, mouse — the **desktop metaphor**



## Touch screen

gestures — press and swipe. iPhone / iPad Springboard, Android

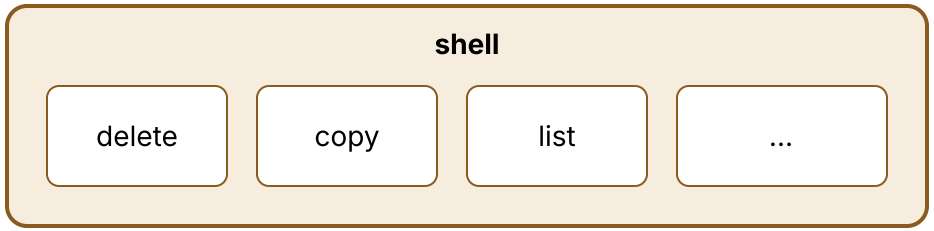
> The choice of CLI or GUI is mostly **personal preference**. Windows users stay in the GUI; macOS has both Aqua GUI and a UNIX shell; mobile users almost only use touch.

07

# The command interpreter

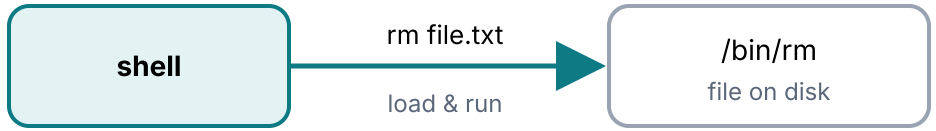
Main job: **get and execute** the next user command. Two ways to implement a command:

## ① Code inside the interpreter




More commands → **bigger** interpreter.

## ② Separate system programs (UNIX)




New commands = new files; the shell stays **small**.

# 08 System calls

The **interface to OS services**. Usually C/C++ functions; low-level tasks may need assembly.

Example: **`cp in.txt out.txt`** — every step is a system call

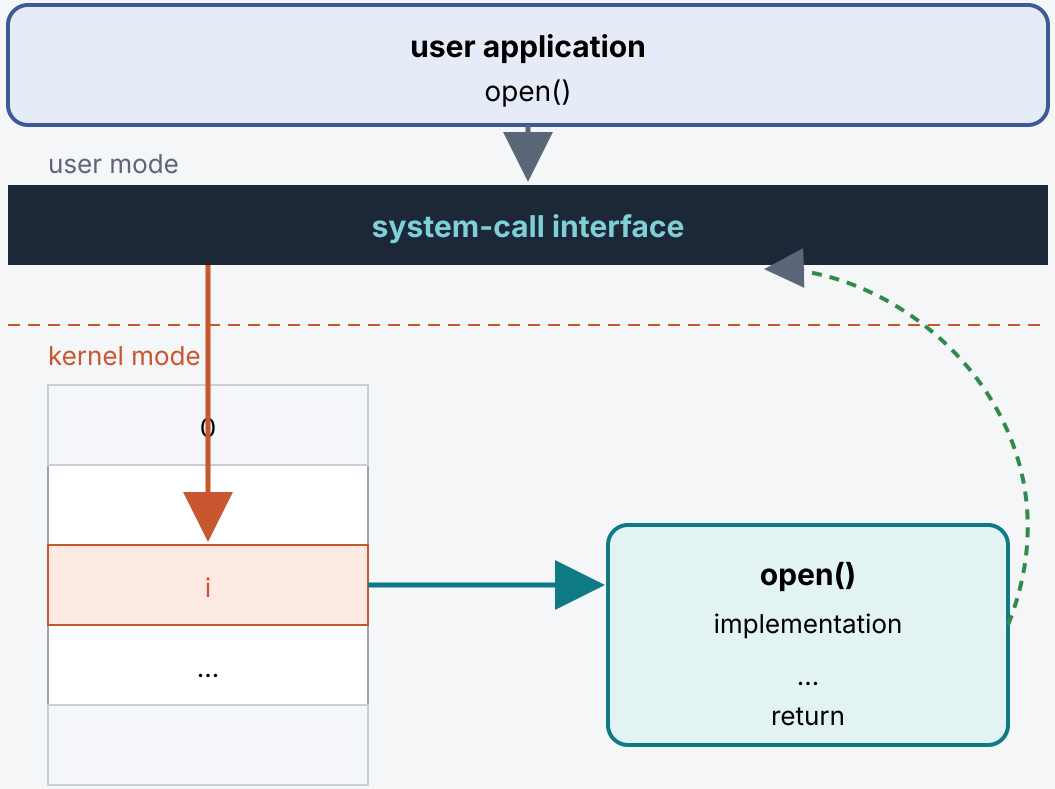
- acquire input file name · write prompt · accept input
- acquire output file name · write prompt · accept input
- open input file — abort if missing
- create output file — abort if it exists
- loop: **read** from input → **write** to output, until read fails
- close output · write completion message · terminate normally

**1000s**

system calls per second — even simple programs use the OS heavily

> Programmers rarely see this level — they use an **API**.

# 09 API → system-call interface → kernel




## Common APIs

`Windows API` `POSIX API` `Java API (JVM)`

> Each system call has a **number**; the interface keeps a **table** indexed by it.

> **RTE** (run-time environment) — compilers, interpreters, libraries, loaders; provides the system-call interface.

10

# Six types of system calls

## Process control

create / terminate process · load, execute · get/set attributes · wait / signal event · allocate / free memory

## File management

create / delete file · open, close · read, write, reposition · get/set file attributes

## Device management

request / release device · read, write, reposition · get/set attributes · attach / detach

## Information maintenance

get / set time or date · get / set system data · process, file, device attributes

## Communications

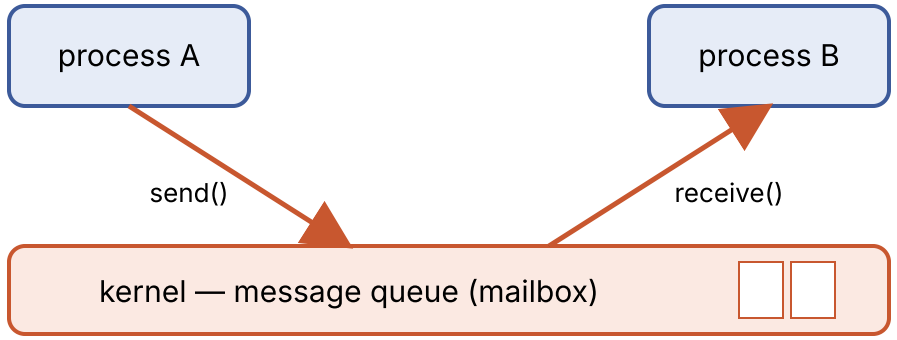
create / delete connection · send / receive messages · transfer status · attach remote devices

## Protection

get / set file permissions · `allow_user()`, `deny_user()`

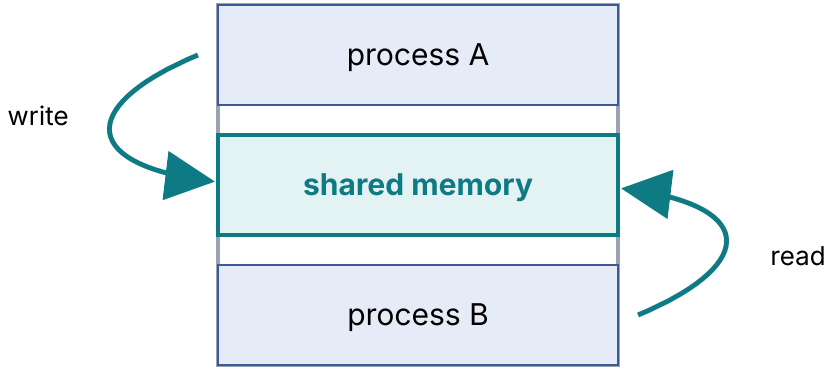
# 11 Two models of interprocess communication

## Message passing



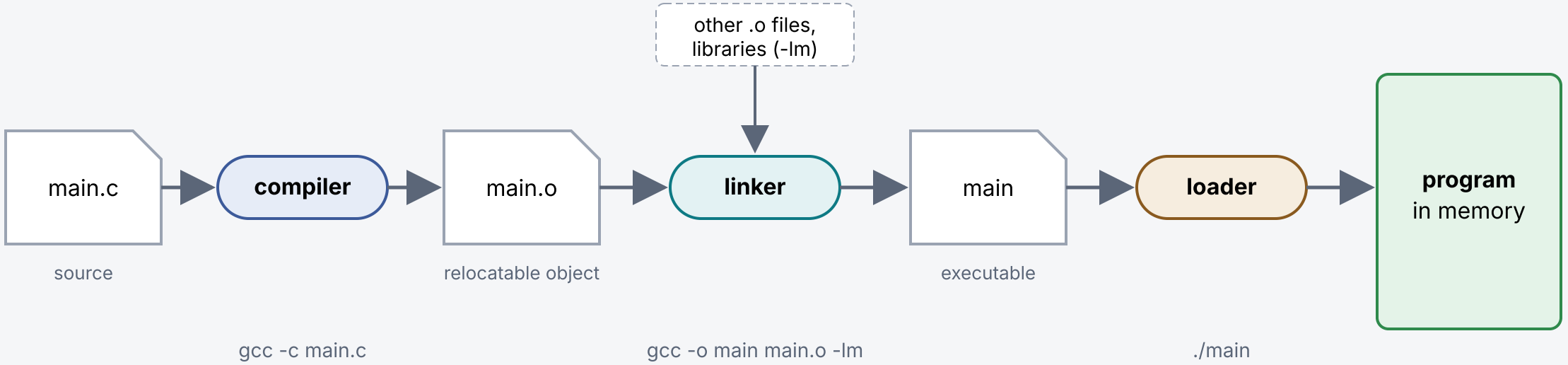

open connection → exchange messages (direct or via a **mailbox**) → close. Good for small data and networks.

## Shared memory




`shared_memory_create()`, `attach()`. Fast; processes must avoid writing the same place at once.

# 12 Linkers and loaders




**Relocation** — assigns final addresses to program parts and adjusts code and data to match them.

# 13 Summary

| TOPIC | KEY IDEA |
| --- | --- |
| **OS services** | help the user (UI, execution, I/O, files, communication, errors) and the system (resources, logging, protection) |
| **Interfaces** | CLI, GUI, touch screen; the shell runs commands |
| **System calls** | interface to services; used through an API (Windows, POSIX, Java) |
| **Six types** | process, file, device, information, communications, protection |
| **IPC** | message passing or shared memory |
| **Build & run** | compiler → linker → loader → process in memory |