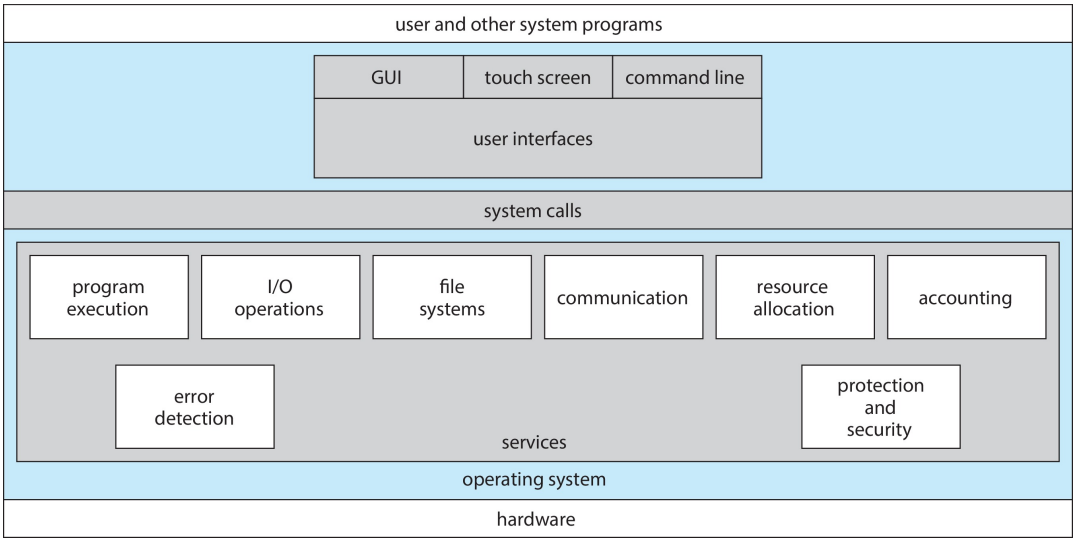
System Structure

The User Interface

- **Operating systems** provide an **environment** for the **execution of programs**, and **services to those programs**.
- One set of **operating system services provides functions** that are helpful to the **user**.
- The **user interface** is the way **users interact with the operating system**.
- There are **two types** of **user interfaces**:
  1. The **command-line interface (CLI)** is a way the **user** can **interact** with the **operating system** through a set of **commands**.
  2. The **graphical user interface (GUI)** is a way the **user** can **interact** with the **operating system** in a **graphical environment**.
- Any operating system **user interface** should allow the user to **perform file-system manipulation**, **execute programs**, and **interact with IO devices**.
- Additionally, the **user interface** should handle **error detection, resource allocation, and device security** behind the scenes.



System Calls

- A **system call** is a **programming interface** to the **services** provided by the **operating system**.
- **System calls** are invoked with the system's **assembly language**.
- To make **software development easier** operating systems typically provide a **high-level programming interface** that **abstracts** the **system calls**.
  - **Windows systems** use the **Win32 API**.
  - **POSIX-based systems** (UNIX, Linux, MacOS, etc) use the **POSIX API**.
- **System calls** typically require **parameters**. There are **three methods** that can be used to **pass parameter to the operating system**:
  1. Pass the parameters in registers. (Max amount of parameters is restricted to register count)

2. Store the parameters in a block / table that is in the system's primary memory, and pass the memory location of the table in a register. (Used by Linux)
3. Push the parameters onto the stack, and the operating system will pop them off.

- Method 2 and 3 **do not have a maximum amount of parameters**.

- **System calls** are needed for: **process control, file management, device management, information maintenance, communication, and permission management**.

System Programs

- **System programs** can be thought of a **bundles** of **system calls** that provide basic functionality to **users** so they do not need to write their own programs to solve common problems.
- A **daemon** is a **system program** that runs as a **background process**, and usually provides some type of service.

Separation of Mechanism and Policy

- An operating system **policy** is a way of choosing an **action to perform**.
- An operating system **mechanism** is the **implementation** that **enforces a policy**.
- **Policies and mechanism should be separated**. The policy remains the same, but the **mechanism may vary** depended on the type of system the operating system is running on.

Monolithic and Modular Operating Systems

- A **monolithic operating system** (tightly-coupled system) is an operating system that uses **one static-compiled image** and runs in an **all or nothing mode**.
- A **layered operating system** is an operating system that is divided into several layers with each layer interfacing with the one below it. The first layer (Layer 0) is the hardware, and the final layer (Layer N) is the user interface.
- The **layered approach** has a lot of **overhead** because each user functionality has to **traverse all of the layers**.
- A **modular operating system** (loosely-coupled system) is an operating system that is built with various tasks **divided** into **distinct processes** each of which contain their own **interface**.
  - One way to implement a **modular operating system** is with a **microkernel**. This approach contains a kernel that is the **near-minimum amount of software that can provide methods to implement an operating system**. **Modules** are built on top of **microkernel**, and communicate with each-other with **message passing** through the kernel.
  - **Microkernels** are generally more **secure, reliable, and maintainable**. The only downside is that **modular communication has more overhead**.
- **Modular operating systems** use **Loadable Kernel Modules (LKM)** which gives the operating system the ability to **add or remove modules at runtime** without the need to recompile the kernel or restart the system.
- **Monolithic operating systems** are **difficult to implement and maintain**, but have **less overhead**.
- You can also have a **hybrid operating system** which is partially **monolithic** and **modular**.