# 概论

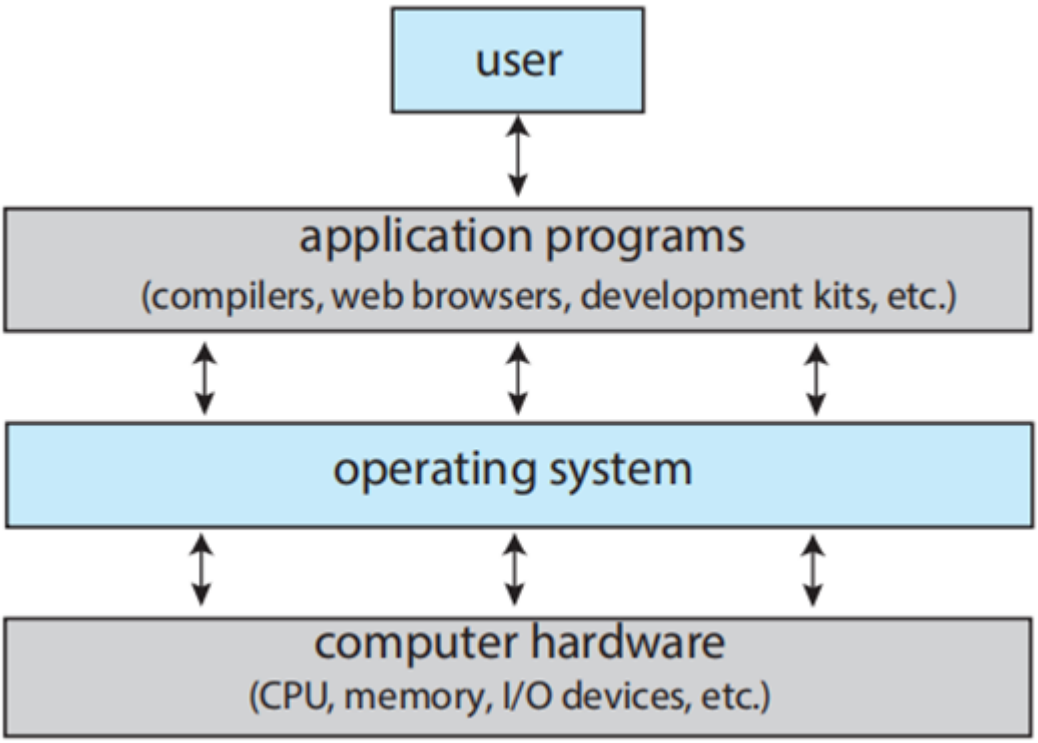
## 操作系统的定义、功能、提供的服务

#### 定义

操作系统是管理计算机硬件的软件。它还为应用程序提供基础，并充当计算机用户和计算机硬件之间的中介。

实际上没有一个universally的定义

“Everything a vendor ships when you order an operating system” is a good approximation




**功能和服务**

**用户视角**：

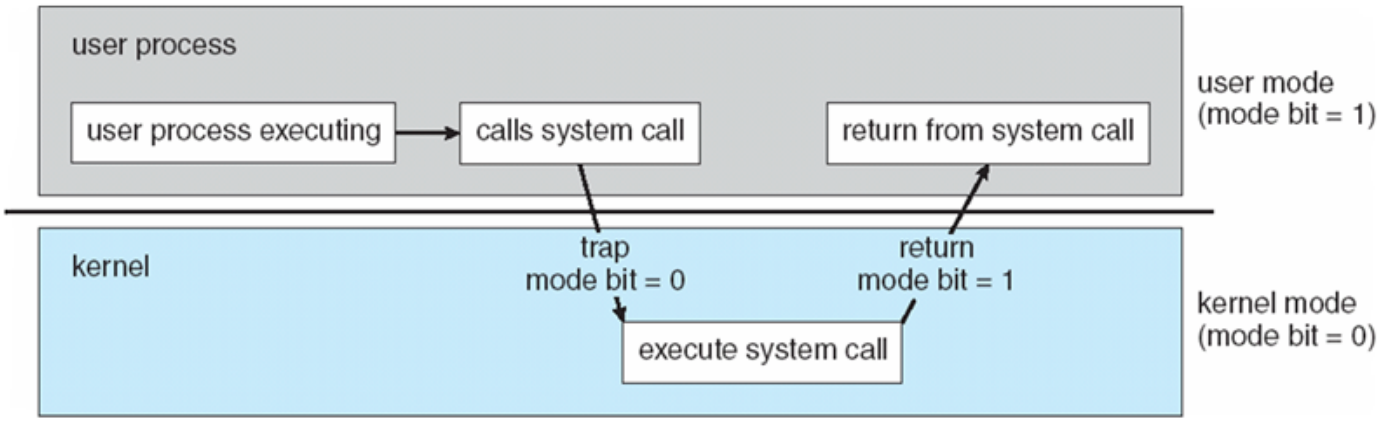
- 提供**便利性、易用性和良好的性能。**
- 对于资源丰富的共享计算机（如**主机**或**小型机**），需要满足所有用户的需求。
- 手持设备资源有限，需优化可用性和电池寿命。
- 嵌入式计算机通常没有用户界面，主要用于设备和汽车中。

**系统视角**：

- **资源分配器**： 管理系统资源，在冲突请求间进行决策，以确保资源使用的高效性和公平性。
- **控制程序**： 控制程序的执行，防止错误和不当使用计算机。

# 操作系统的运行

🚀 **内核态与用户态**




- **目的**：双模式操作用于保护操作系统自身及其他系统组件，防止用户程序的错误或恶意行为破坏系统。
- **两种模式**：
  - **用户模式**（User mode）：运行用户代码，限制权限。
  - **内核模式**（Kernel mode）：运行操作系统代码，具有完全的硬件访问权限。
- **模式位**：硬件提供模式位（Mode bit），用于区分当前是用户模式还是内核模式：
  - 用户模式：模式位设置为用户状态。
  - 内核模式：模式位设置为内核状态。
- **特权指令**：某些指令被指定为特权指令，仅能在内核模式下执行，以保护系统资源。
- **模式切换**：
  - 系统调用（System call）会将模式从用户模式切换为内核模式。
  - 系统调用返回时，模式会切换回用户模式。

#### 中断、定时器

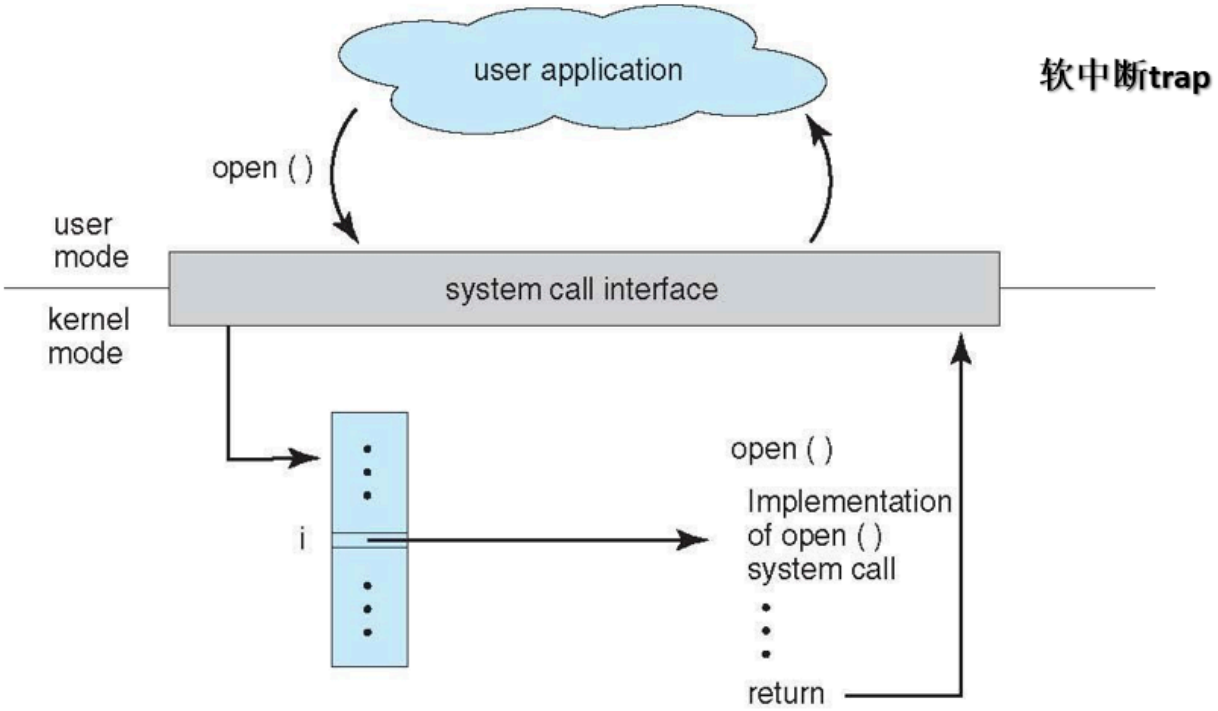
##### 中断

- 中断的作用：中断将控制权转移到中断服务例程（ISR），通过中断向量实现，该向量存储所有服务例程的地址。
- 中断的保存机制：中断架构需要保存被中断指令的地址，以便在处理完成后恢复程序执行。
- **陷阱和异常**：陷阱或异常是软件生成的中断，通常由错误或用户请求触发。
- 操作系统的特点：**操作系统以中断驱动方式运行**，用于处理各种硬件和软件事件。

##### 定时器

- **防止无限循环和资源独占**：计时器用于防止进程陷入无限循环或长时间占用系统资源，确保操作系统可以定期收回控制权。
- **计时器的工作机制**：
  - 计时器被设置为在特定时间段后中断计算机。
  - 硬件中维护一个计数器，该计数器由物理时钟递减。
  - 当计数器减至零时，会触发一个中断。
- **操作系统的控制**：
  - 操作系统通过特权指令设置计时器的计数器值。
  - 计时器在调度每个进程之前被设置，以确保操作系统能够在指定时间后重新获得控制权。
- **中断处理**：如果进程超出了分配的时间，操作系统可以通过计时器中断终止该进程或重新调度其他进程。

### 🚀 系统调用




用户程序与操作系统内核之间进行交互的接口

#### Implementation

- 通常，每个系统调用都关联一个编号。
  - **系统调用接口**通过这些编号维护一个索引表。
- 调用者无需了解系统调用的具体实现方式。
  - 只需遵循API，并理解操作系统在系统调用后的行为。
  - API隐藏了操作系统接口的大部分细节，程序员无需直接接触底层实现。

#### 参数传递

- 有三种常见的方法用于将参数传递给操作系统：

  **最简单的方法** - 将参数存储在寄存器中：

  - 在某些情况下，参数可能多于寄存器的数量。

  **存储块或表的方法**：

  - 参数存储在内存中的一个块或表中，并将该块的地址作为参数存储在寄存器中。
  - Linux和Solaris采用这种方法。

  **使用堆栈**：

  - 参数由程序压入堆栈，并由操作系统从堆栈中弹出。

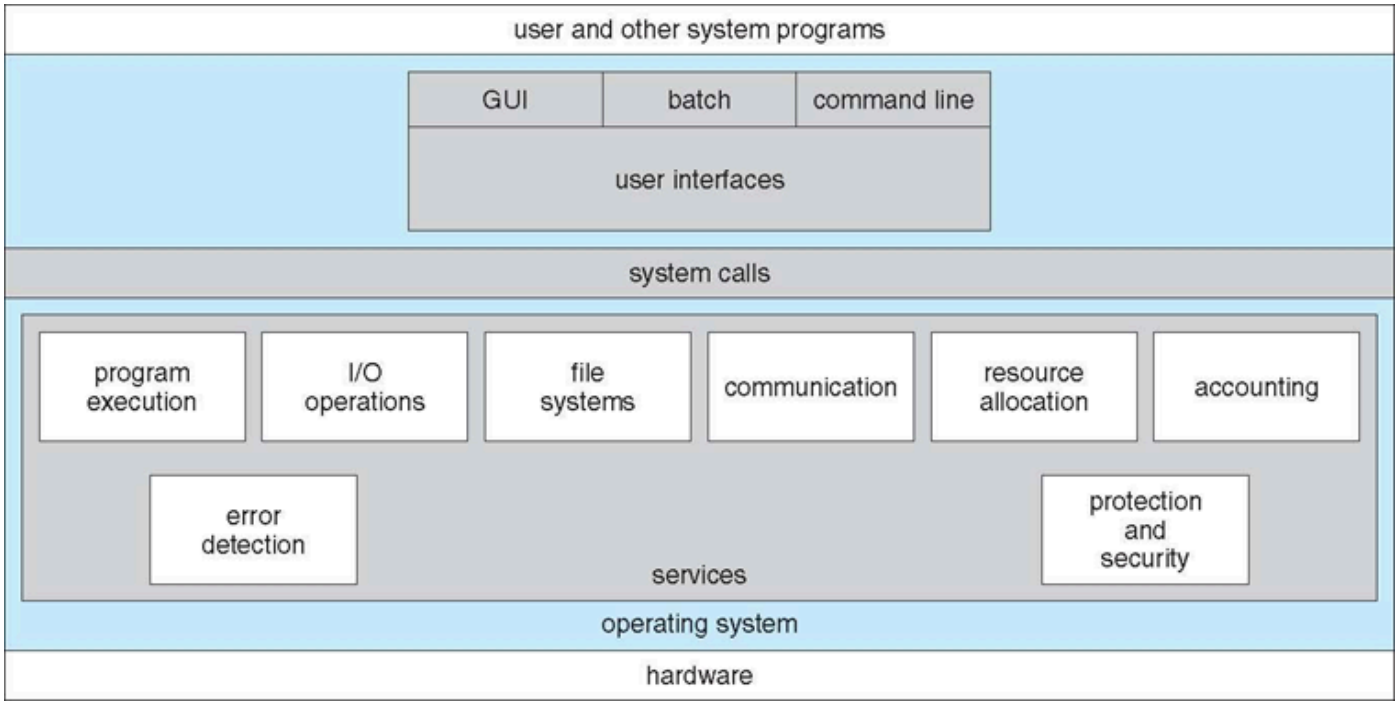
- 使用块和堆栈的方法不限制传递参数的数量或长度。

#### Types of System Calls

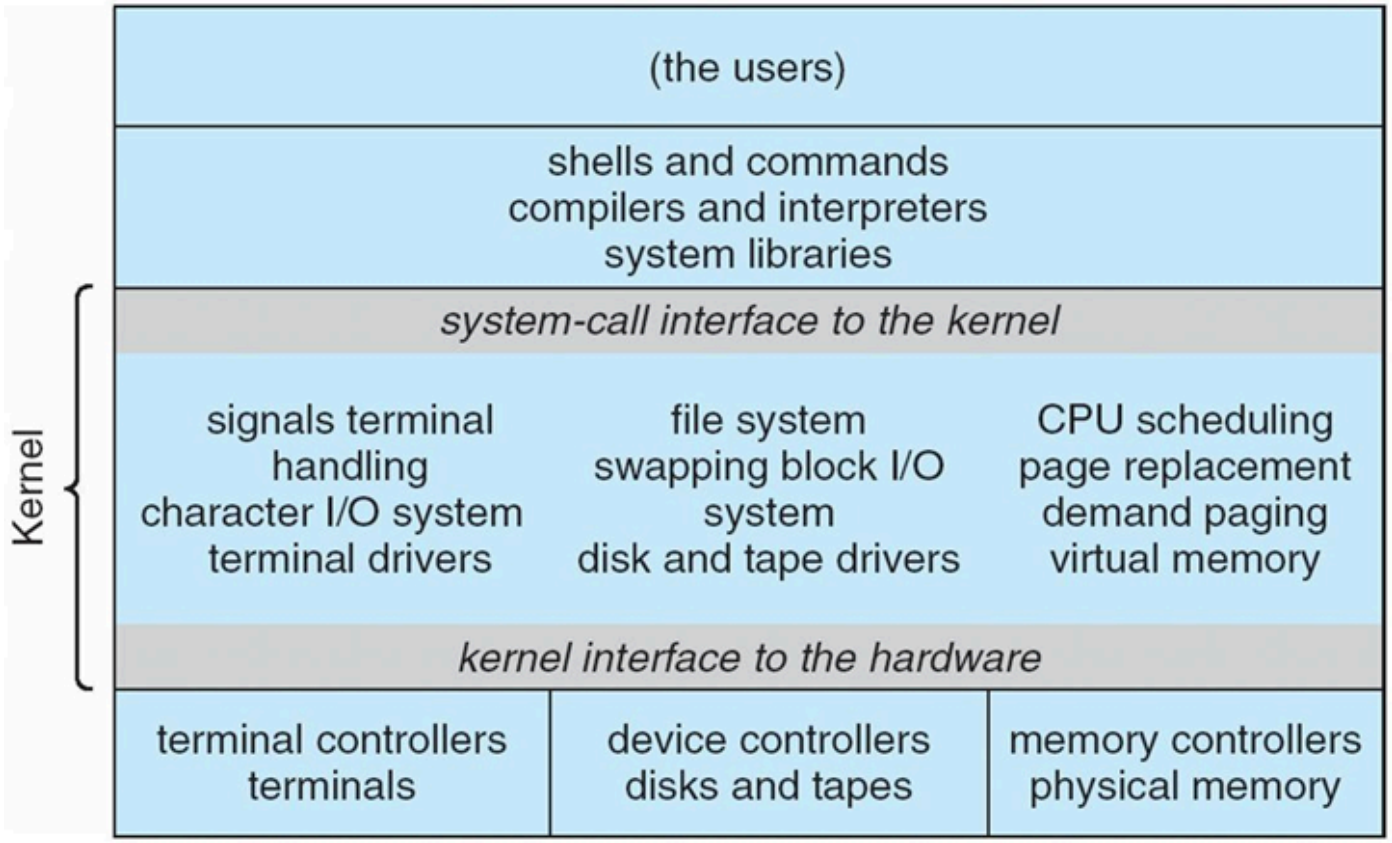
- Process control
- File management
- Device management
- Information maintenance

- Communications

# 操作系统的结构




**单片结构（UNIX）**




由于硬件功能的限制，最初的UNIX操作系统结构有限。UNIX操作系统由两部分组成：

1. **系统程序（Systems programs）**
2. **内核（The kernel）**
    - 包含系统调用接口以下和物理硬件以上的所有内容。
    - 提供文件系统、CPU调度、内存管理以及其他操作系统功能；这些功能都集中在一个层次中完成。

#### 宏内核

宏内核将操作系统的大部分功能集成在一个大的内核中，所有关键服务（如文件系统、内存管理、设备驱动、网络协议等）都在内核模式下运行。

**优点**：

1. **高性能**： 内核中的所有组件共享相同的地址空间，通信效率较高，避免了用户空间与内核空间之间的开销。
2. **简单设计**： 功能集中在内核中，系统设计和实现较为直观。
3. **成熟性**： 宏内核已经被广泛应用（如Linux和Windows），具有丰富的功能和良好的兼容性。

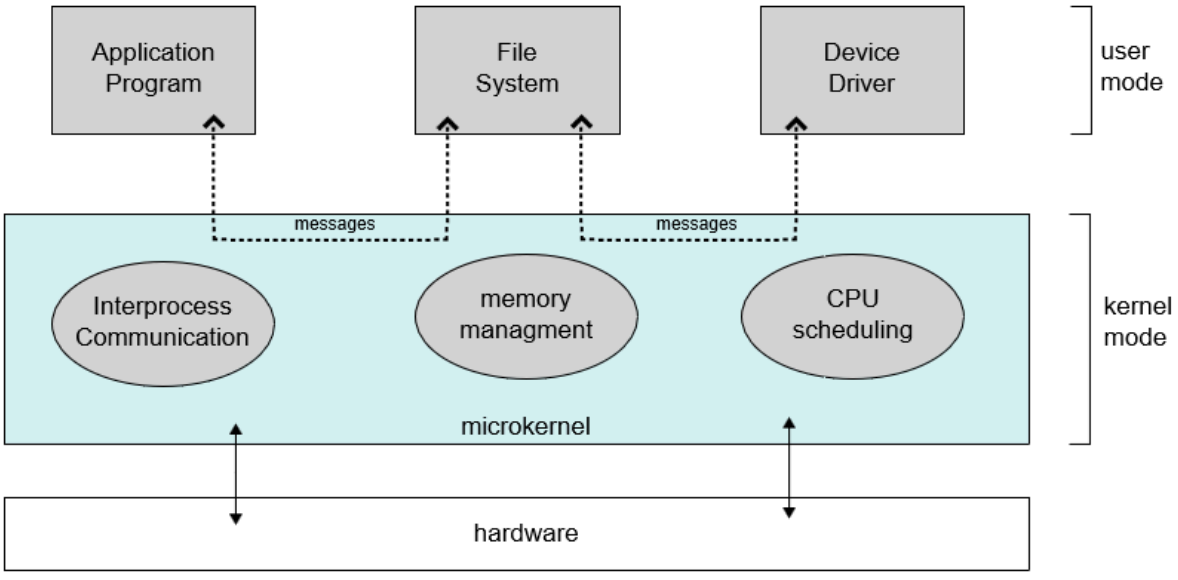
**缺点**：

1. **可靠性较低**： 内核中运行的代码量较大，任何一个组件出错可能导致整个系统崩溃。
2. **扩展性差**： 增加或修改内核功能较为困难，且容易引入新的错误。
3. **安全性不足**： 由于所有组件共享内核空间，一个恶意或有漏洞的模块可能影响整个系统。

宏内核适合需要高性能和功能丰富的系统，但在扩展性和模块化上不如微内核。

**单片结构可以看作宏内核的早期形式**

#### 微内核




**微内核系统结构**

- 将尽可能多的功能从内核移至用户空间。

- 用户模块之间通过**消息传递**进行通信。

**优点**：

- 更容易扩展微内核。
- 更容易将操作系统移植到新架构上。
- 更可靠（在内核模式下运行的代码更少）。
- 更安全。

**缺点**：

- 用户空间与内核空间通信的性能开销较高。

# 操作系统引导过程

- 当系统加电初始化时，执行从一个固定的内存位置开始。
  - 固件ROM（Firmware ROM）用于存储初始引导代码。
- 操作系统必须被加载到硬件中以供启动：
  - 一段小代码—— **引导加载程序（bootstrap loader）** ，存储在ROM或EEPROM中，用于定位内核，将其加载到内存并启动。
  - 有时这是一个两步流程：ROM代码加载固定位置的引导块（boot block），然后引导块从磁盘加载引导加载程序。
- 常见的引导加载程序如 **GRUB** ，允许从多个磁盘、版本或内核选项中选择内核。
- 内核加载后，系统进入运行状态。