Operating System

“Nên tránh những cuộc tranh luận
không có tính xây dựng, vô bổ”

Outline

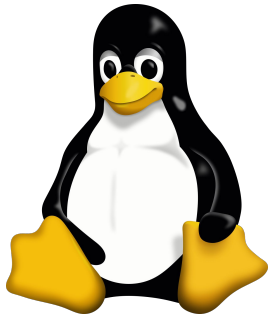
1. Introduction
    - Operating System
    - Kernel
2. Core Functions
    - Process Management
    - Memory Management
    - I/O
    - Bottlenecks
3. Shell Script

1. Introduction

1.1. Operating System

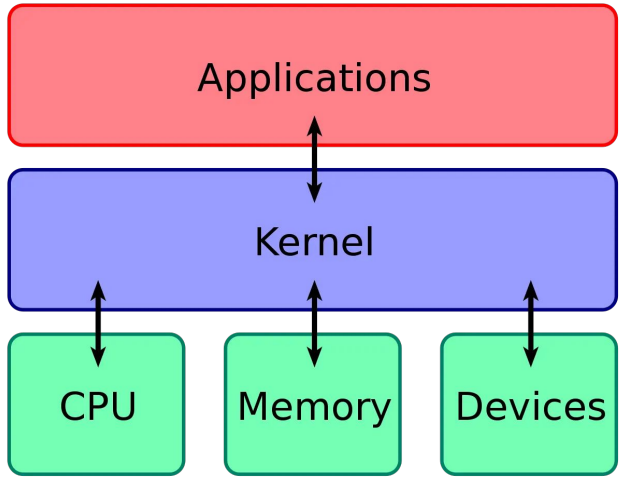
- Operating System (OS) is **system software** that **manages hardware and software resources** and provides **common services for programs**
- OS is **acts as a bridge between hardware and user**. It provides a user interface and controls the hardware so that software can function

UNIX®
An Open Group Standard



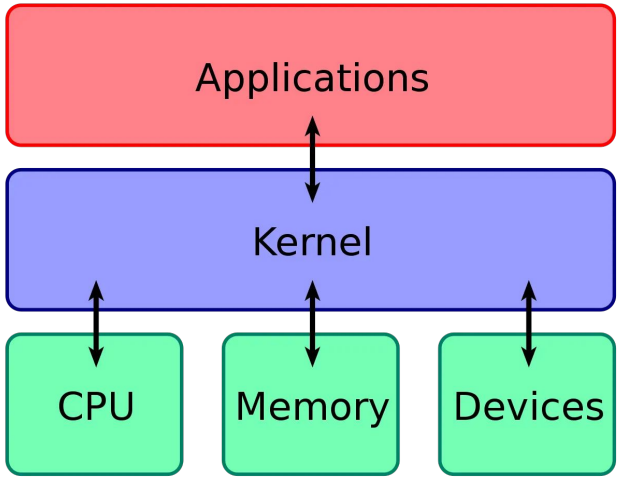
1.2. Kernel*

- The core of the operating system is the kernel.
- **A bridge between applications and hardware devices.**
- Kernel is programs.
- Why Kernel?
    - Computers are composed of various hardware devices: CPU, mem, disk, network, …
    - Every app implement communication protocol with hardware devices



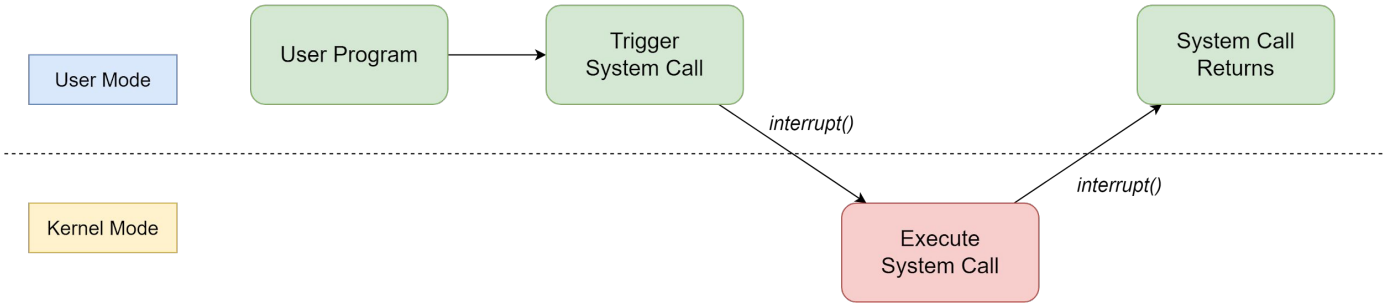
1.2. Kernel

- Applications only need to care about interacting with the kernel.
- **The kernel has very high permissions** and can control hardware such as **CPU, memory, hard disk, etc.** While applications have very small permissions.



1.3. How The Kernel Works?*

- Most operating systems divide the memory into two areas:
  - Kernel space: this memory space can only be accessed by kernel programs
  - User space: this memory space is exclusively used by applications
- When a program uses kernel space, the program is executed in kernel mode.
- When an application uses **a system call, an interrupt is generated**. After an interrupt occurs, CPU will execute the kernel program.

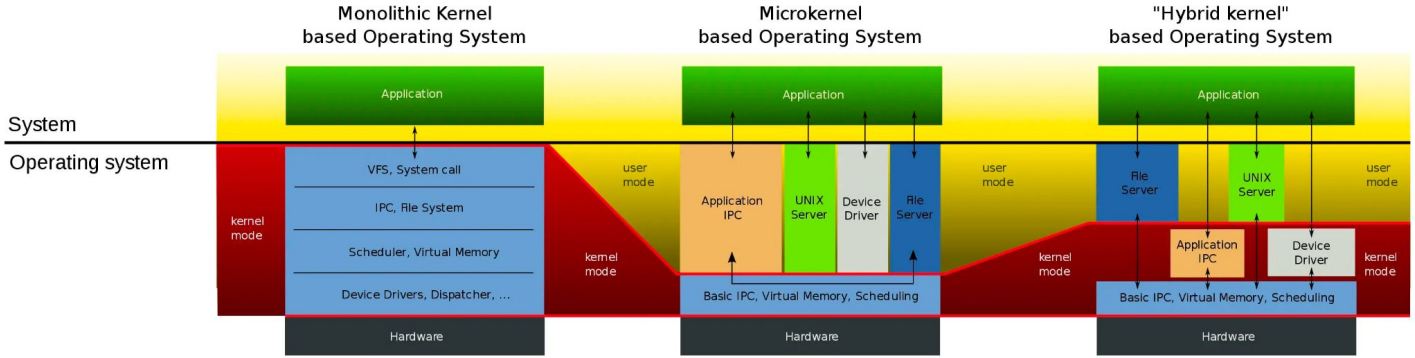


1.4. What Capabilities Does the Kernel Have?

- Manage processes and threads
- Manage memory
- Manage hardware devices
- Provide system calls. If the application wants to run services that run with **higher privileges, then a system call** is required

1.5. Linux Design

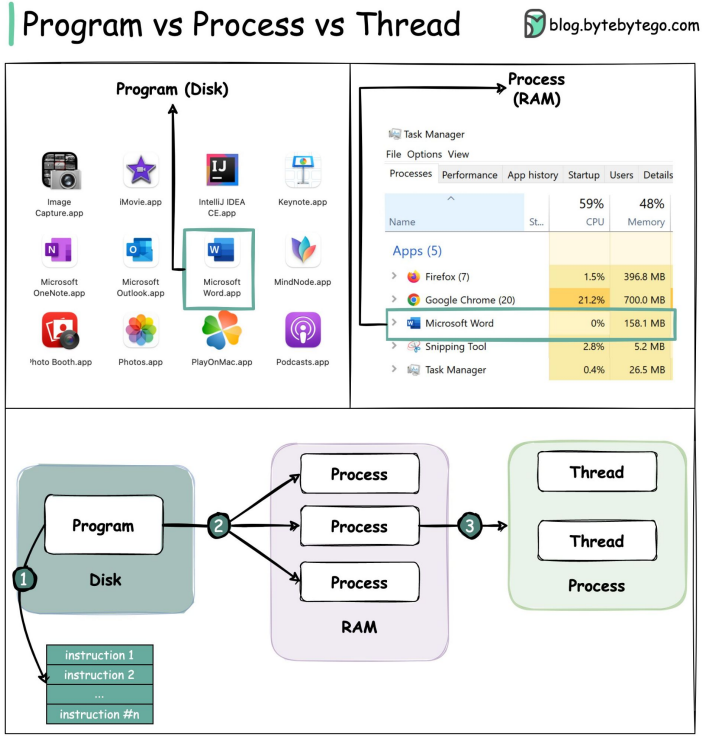
- MultiTask: multiple tasks can be executed at the same time
- SMP (Symmetric Multiprocessing): each CPU has an equal status and has the same rights to use resources.
- ELF: Executable File Link Format. is the storage format of executable files in the Linux operating system
- Monolithic Kernel (Macro kernel): all modules of the system kernel run in the kernel state.
- **Linux is a macro kernel, Window is a hybrid kernel.**



2. Process Management

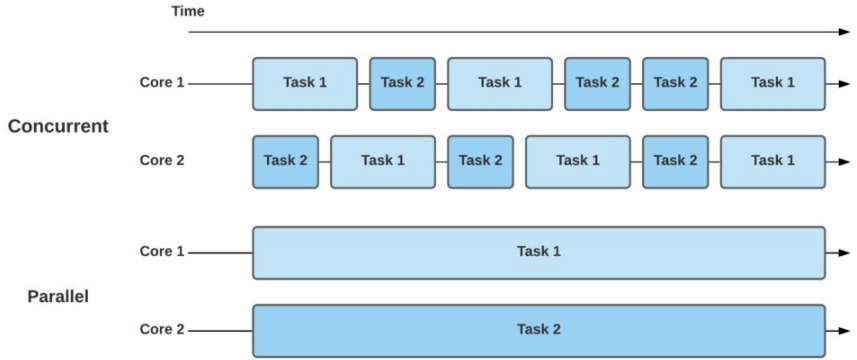
2.1. Concepts

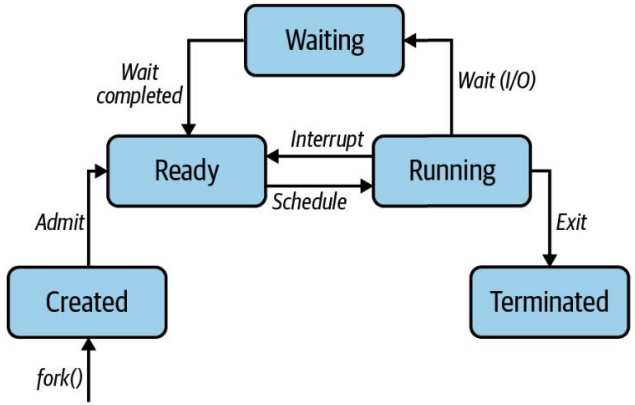
- **Program**: An **executable file** that contains code and is stored as **a file on disk**
- **Process**: When we **run the program file**, it will be loaded into memory and CPU will execute each instructions in the program
- **Thread**: A thread is an execution **flow within a process**.



2.2. Process

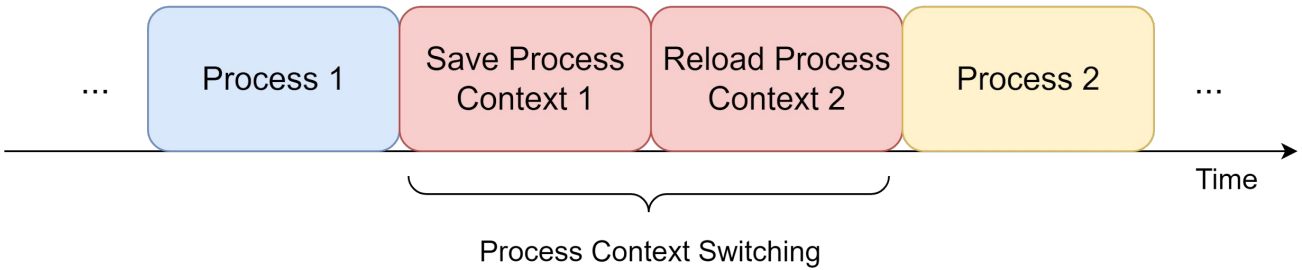
- Although a single-core CPU can only run one process at a certain moment. But during 1 second, it may be running multiple processes. This is called **concurrency**.
- The process control block (PCB) data structure is used to describe the process:
  - Process ID
  - User Id
  - Current Status
  - Priority
  - Memory address space, opened files, …
  - The values of each register in CPU





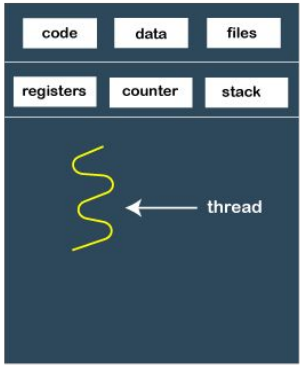
2.3. Process Context Switching

- Before each task is run, the CPU needs to know where the task is loaded and where it starts to run.
  - The CPU register
  - The program counter (PC) is used to store the location of the next instruction to be executed.
- Context Switching is **very critical.**

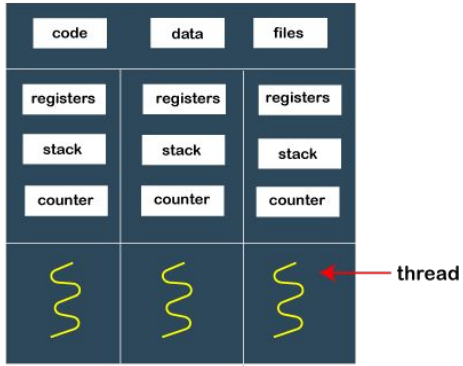


2.4. Thread

- A thread is an execution flow within a process.
- A process can have multiple threads. Each thread can execute concurrently
- **Multiple threads in the same process can share resources** such as code segments, data process (heap), open files, etc.
- Each thread has an independent set of registers and stacks
- The **context switching between threads is much cheaper** than between processes.



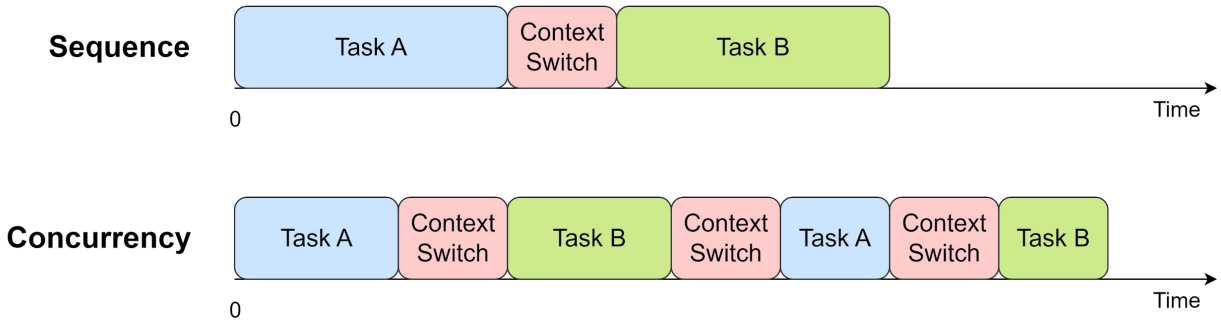
Single-threaded process



Multi-threaded process

2.5. Principle

- **For one core**, executing A and B **sequentially will always be faster than** executing A and B **concurrently** through time-slicing.

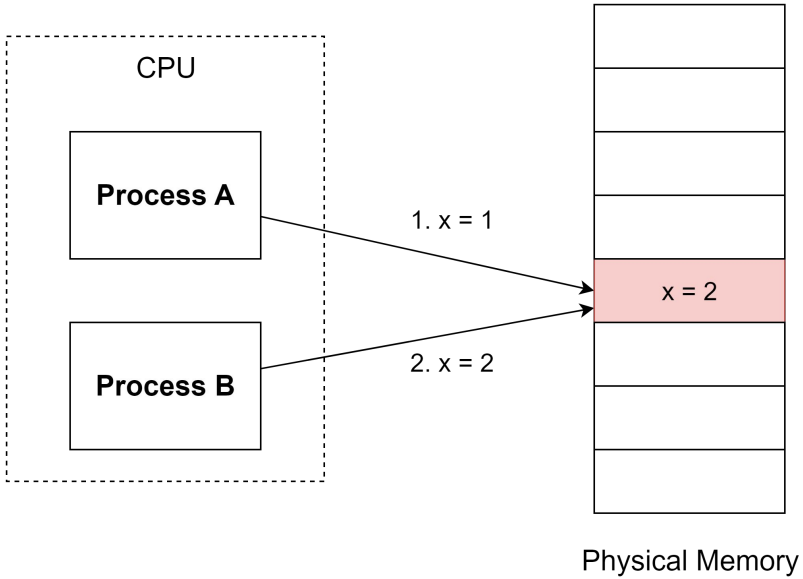


2. Memory Management

2.1. Problem of Memory Management

Problem:

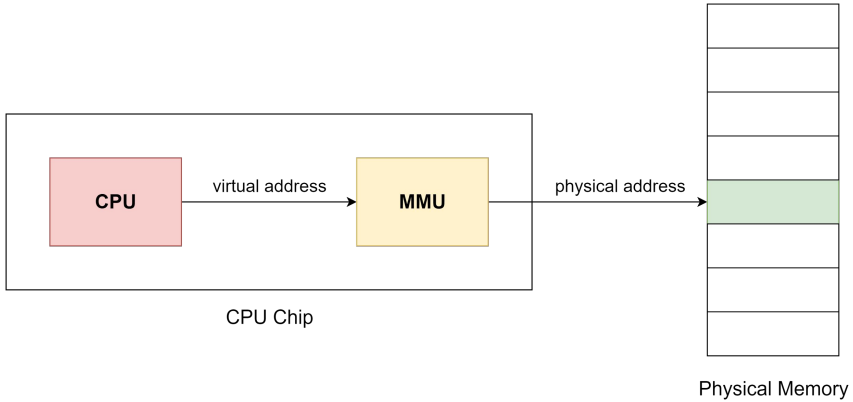
- **Two programs use the same memory space.**
- The first program writes a new value at location 0x00002000. The second program overwrite value at that location → Conflict



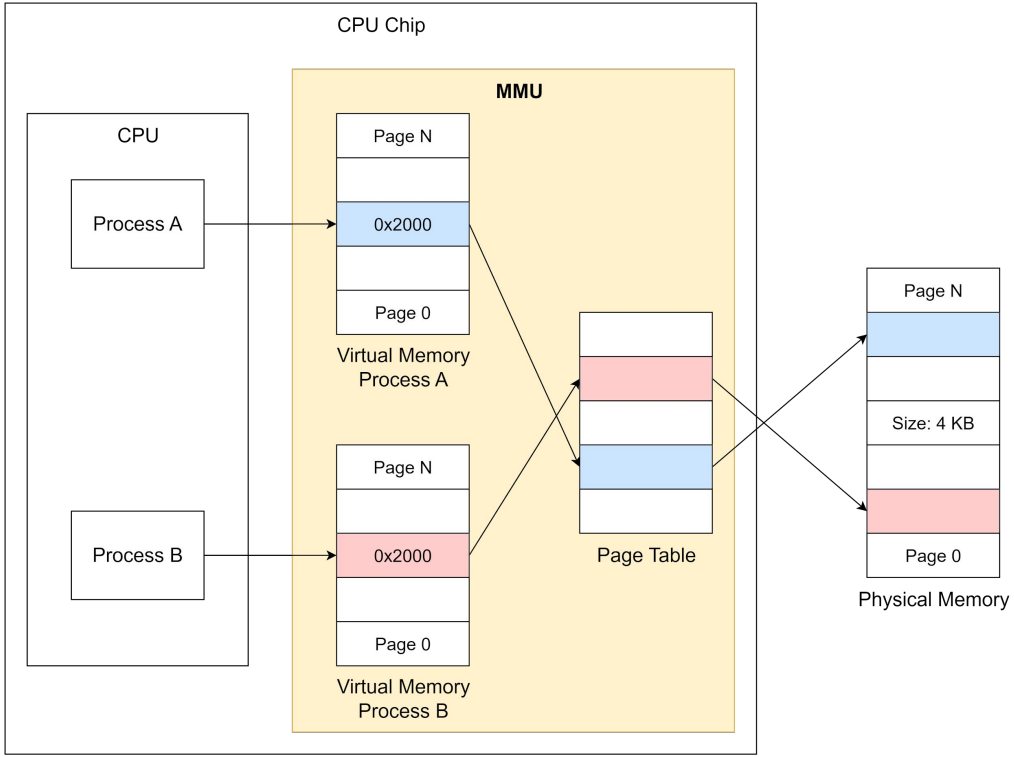
2.2. Virtual Memory

Solution: **Virtual Memory**

- **Virtual Memory Address: each program has its own virtual address space**
- Physical Memory Address: the address space exists in the hardware actually
- **The virtual address is converted into a physical address using the Memory Management Unit (MMU) in the CPU chip.**
- Entries in the page table also have some bits that mark attributes, such as controlling the read and write permissions of a page → better security

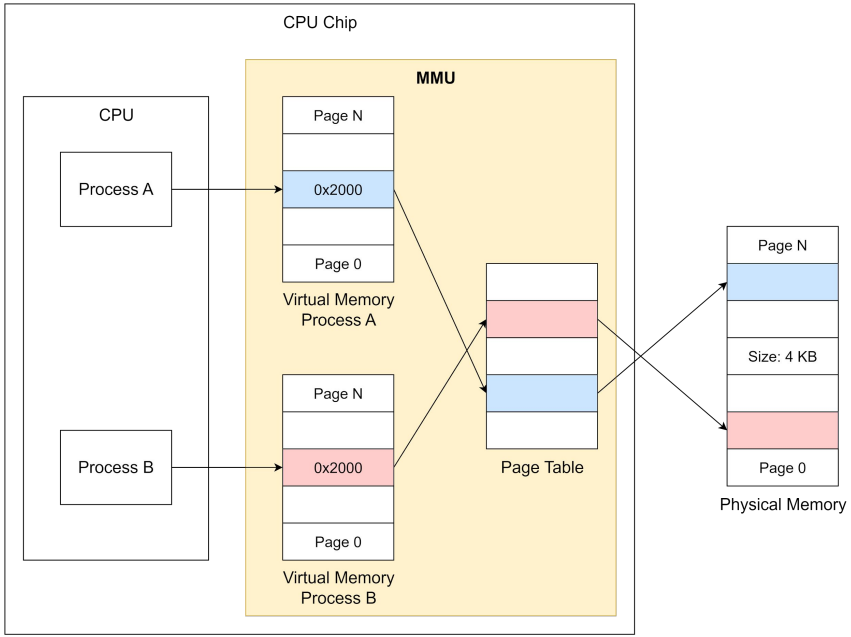


2.2. Virtual Memory



2.3. Problem of Virtual Memory

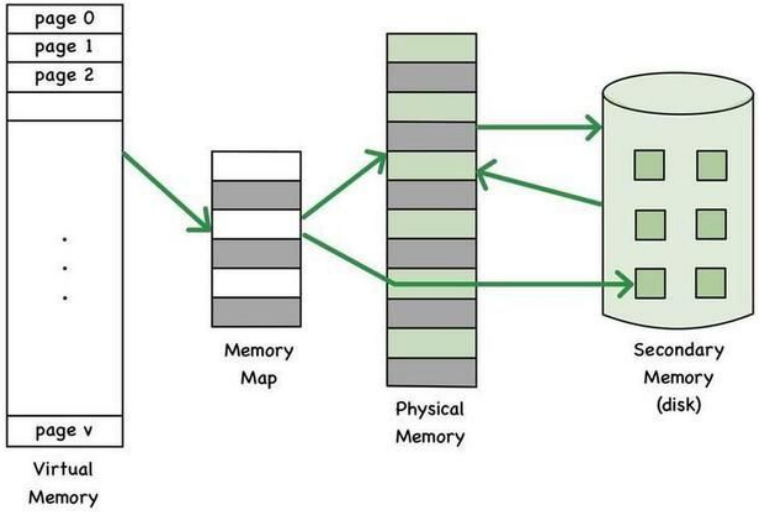
- Problem: processes can use more running memory than the physical memory size



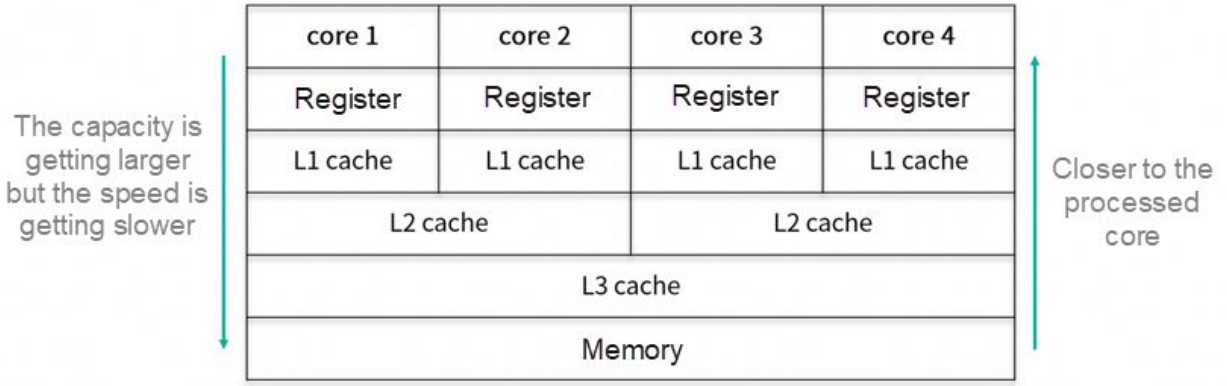
2.3. Swap Partition

Solution: Reclaiming memory

- Virtual Memory = Physical Memory + **Swap memory** (hard disk memory)
- For those memories that are not frequently used, we can Swap it out of physical memory, such as the swap area on the hard disk.
- Drawback?
- Performance impacts



2.4. Memory Model*

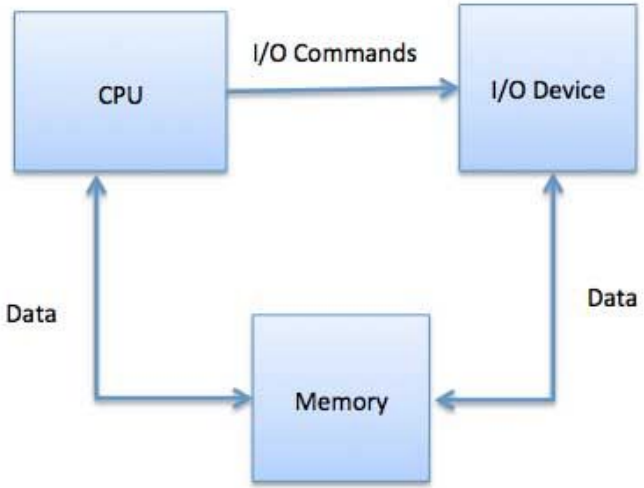


- **Hot data or share data in memory could have a copy in L1, L2, L3 cache.**
- **Data in L1 - Core 1 and L1 - Core 3 and Memory might be different.**
  → need a sync mechanism.

3. I/O

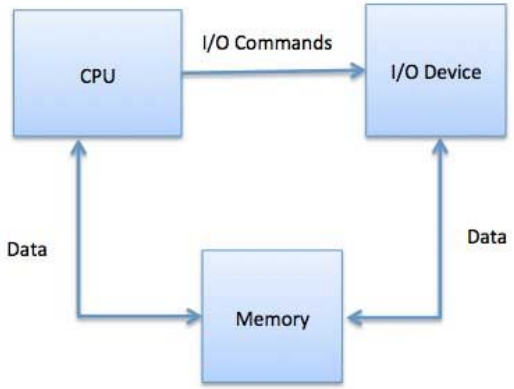
3.1. What is I/O operation?*

- I/O operation: transferring data between a computer and the external environment.
- Types:
  - Disk I/O
  - Network I/O
  - Peripheral I/O: keyboards, mice, printers, …
- Accessing RAM is not a I/O operation



3.2. Bottlenecks*

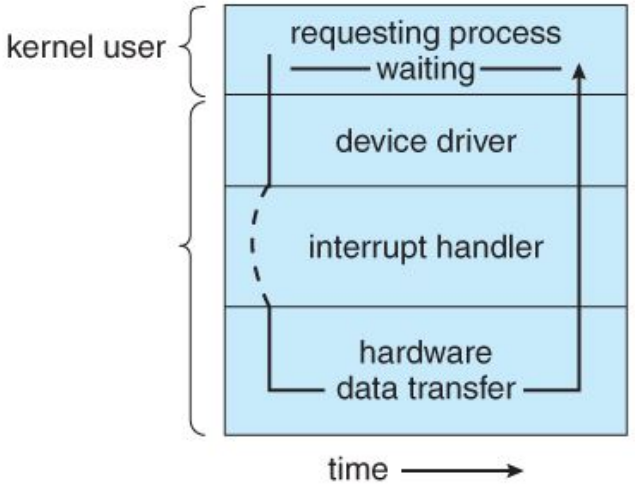
- What are bottleneck groups that a backend app can have?
- Bottleneck groups can be:
  - **CPU**
    - *Context switches*
    - *IO waits*
  - **Memory**
    - *Size*
  - **Disk**
    - I/O:
      - IOPS depends on hardware
      - *Sequential I/O >> Random I/O*
    - Fragmentation
  - **Network**
    - Bandwidth
    - *I/O models*



3.3. How I/O Operations Work?

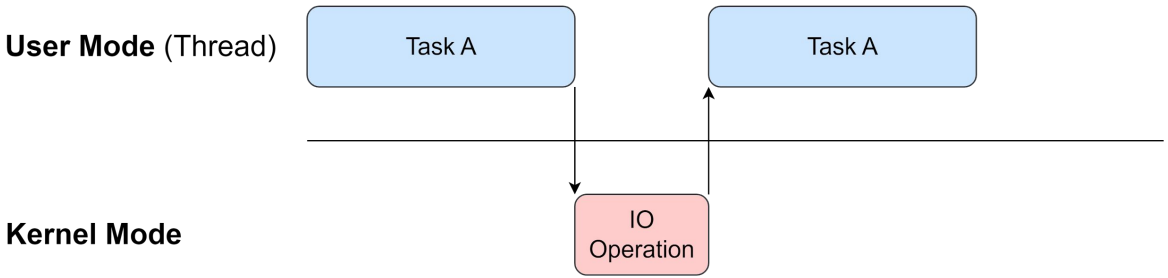
1. An application makes a request for an I/O operation.
2. The application issues a system call to the OS, asking it to perform the I/O operation.
3. The OS talk with the hardware through drivers/kernel
4. Data Transfer
5. Once the I/O operation is completed, the device informs the CPU, through an interrupt.
6. Return Control to Application

- Note: Direct Memory Access (DMA) is a Hardware feature to transfer data between memory and a peripheral device without involving CPU

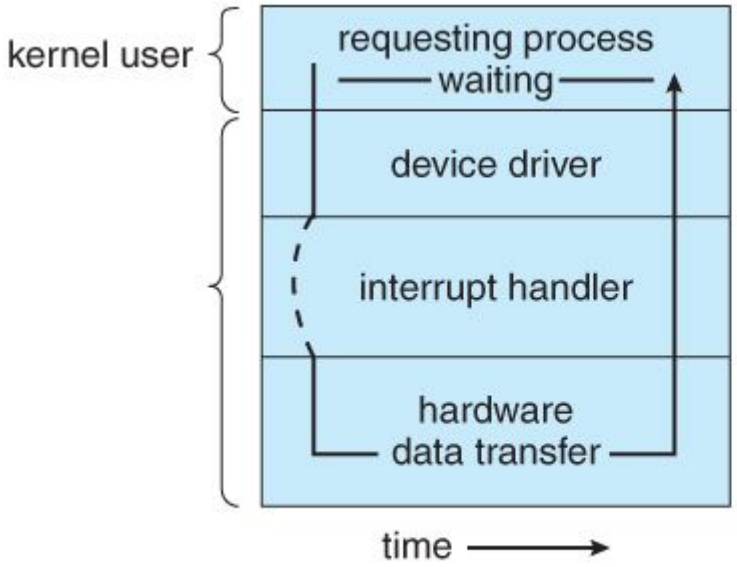


3.4. Why can I/O operations be bottlenecks*

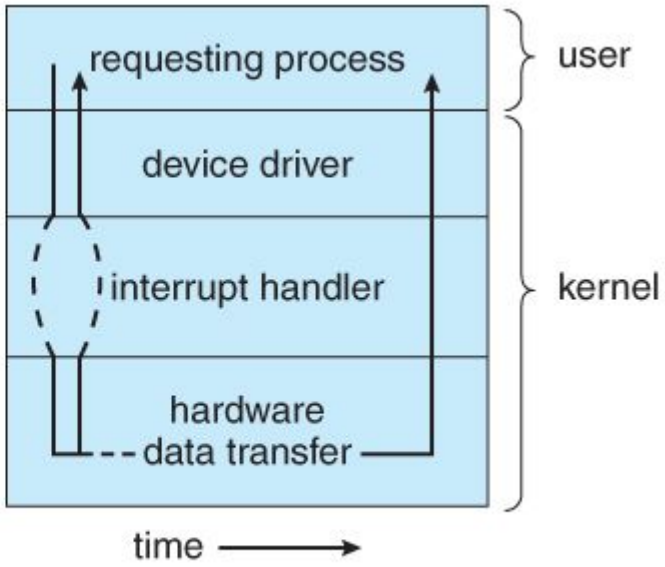
- Blocking I/O



3.5. Types of I/O Operations

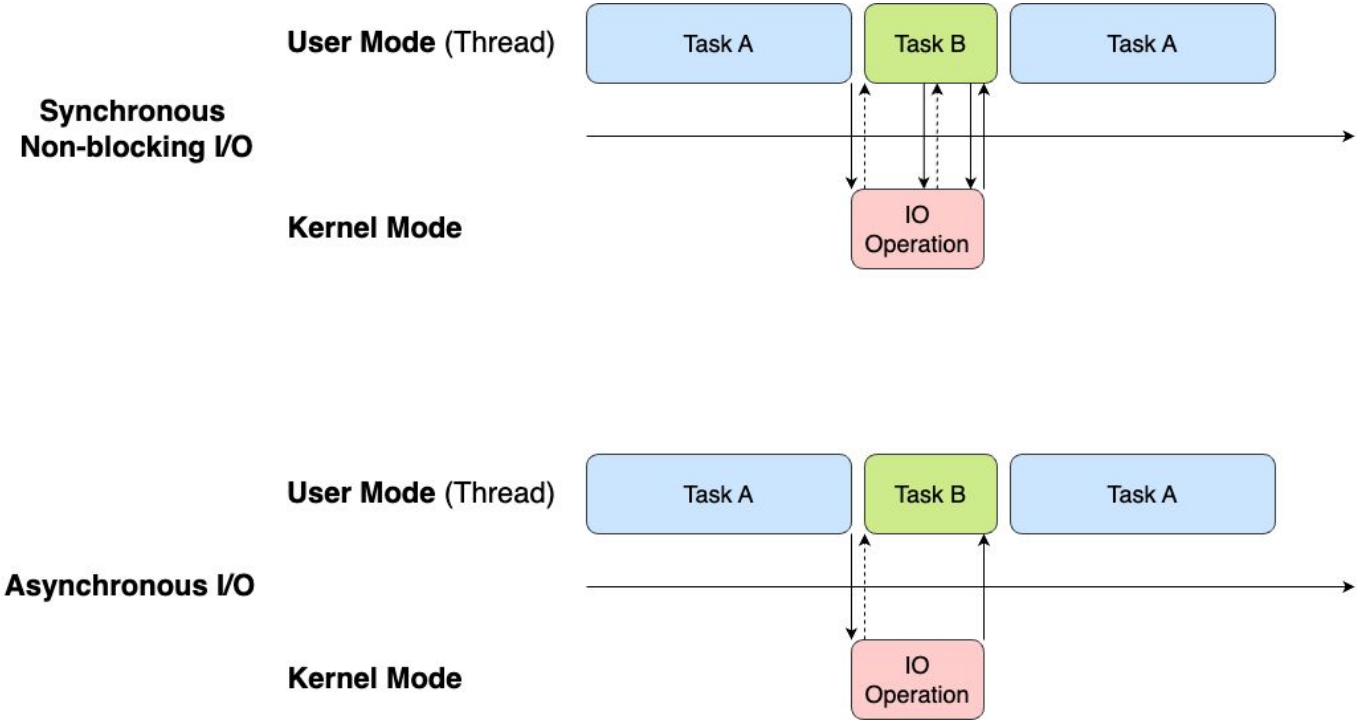


Blocking



Non-blocking

3.6. Non-Blocking I/O (Synchronous)



3.7. Blocking IO Implementation*

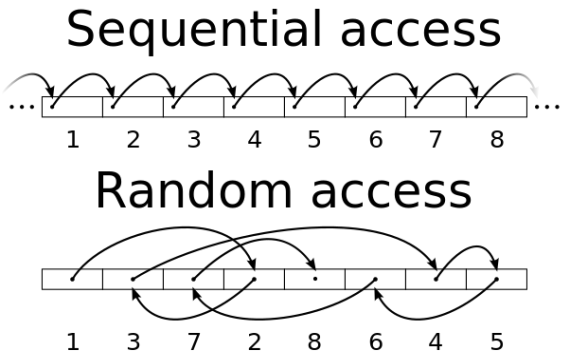
```java
import java.io.FileInputStream;
import java.io.IOException;

public class SynchronousFileRead {
    public static void main(String[] args) {
        FileInputStream fis = null;
        try {
            fis = new FileInputStream("example.txt");
            int content;
            while ((content = fis.read()) != -1) {
                // process the content
                System.out.print((char) content);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fis != null) {
                    fis.close();
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

3.8. Nonblocking IO Implementation*

```java
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousFileChannel;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.concurrent.Future;

public class AsynchronousFileRead {
    public static void main(String[] args) {
        try (AsynchronousFileChannel fileChannel =
AsynchronousFileChannel.open(Paths.get("example.txt"), StandardOpenOption.READ)) {
            ByteBuffer buffer = ByteBuffer.allocate(1024);
            Future<Integer> result = fileChannel.read(buffer, 0); // position 0

            // Do something else while the read operation completes
            while (!result.isDone()) {
                System.out.println("Doing something else while reading...");
            }

            // Check how many bytes were read
            int bytesRead = result.get();
            System.out.println("Bytes read: " + bytesRead);

            // Optionally, process the data
            buffer.flip();
            while (buffer.hasRemaining()) {
                System.out.print((char) buffer.get());
            }
            buffer.clear();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.9. File Access*

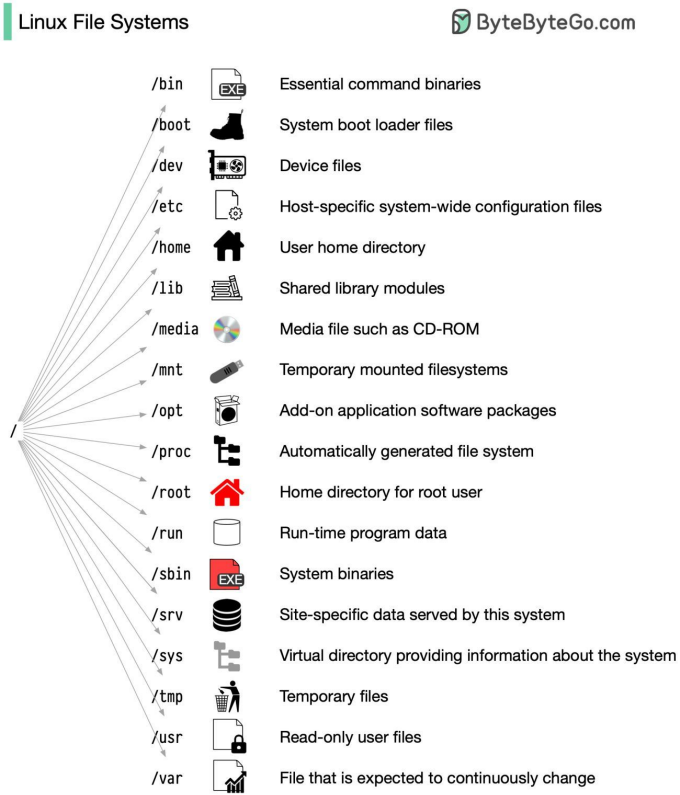
- Are Disk Accesses slow or fast?
- **A wide misconception: disks are slow**
- Disk rotates in one direction
- **Sequential access is much faster than random access** (x3 - 4), which jumps to many different locations on disk
- [Understanding I/O: Random vs Sequential | flashdba]



4. File System

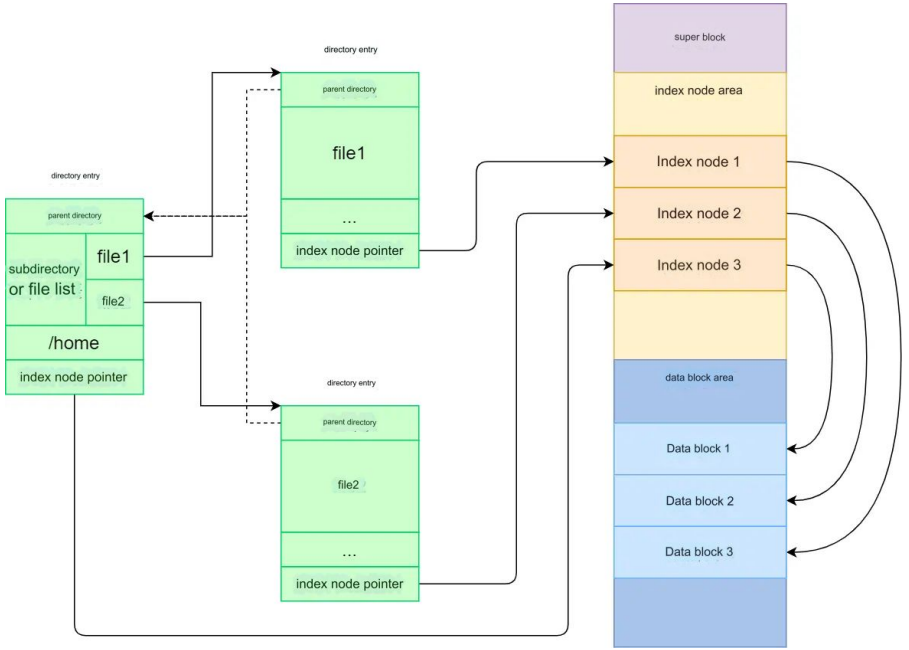
4.1. Introduction*

- The file system is the subsystem of the OS responsible for managing persistent data.
- **In Linux, "Everything is a file."**
- File descriptor is the identifier of a opening file, a socket connection, device, pipe (IPC).
- There is **a limited number of File Descriptors** per process, usually 1024 per process. But we can configure it.



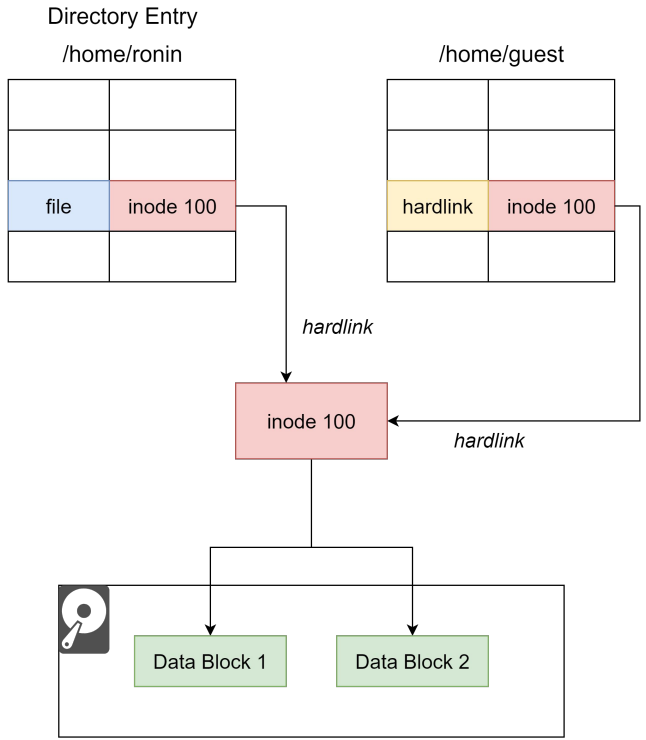
4.2. Inode

- **Index nodes (Inodes)** are used to record meta-information of files, such as inode number, file size, **data location on disk**, etc.
- The index node is the unique identifier of the file.
- **Directory entries (aka filenames)** are used to record file names, index node pointers, and hierarchical relationships with other directory entries.



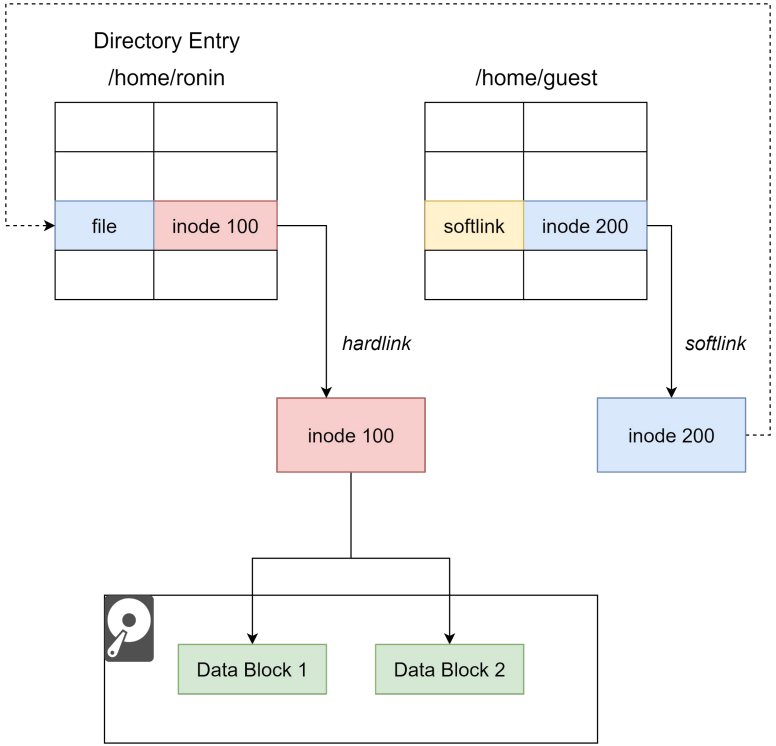
4.3. Hard Link

- A hard link: multiple filenames (directory entries) that points to the same file (to the same inode) on disk.
- **Inode Sharing**: Hard links share the same inode number as the original file
- Deleting a hard link does not delete the actual file data; the data remains accessible as long as there is at least one hard link or the original file name pointing to it.



4.4. Soft Link

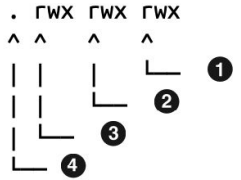
- A soft link is another file, a reference to another file
- **Soft links have their own inode** number and **store the path** of the original file they link to.
- **If the original file is deleted, moved, or renamed, the soft link will break**



4.6. Access Control

- Scopes of permissions:
  - User: The owner of the file
  - Group: Has one or more members
  - Other: The category for everyone else
- Types of access:

Permission	Pattern	Representation
None	- - -	0
Execute	- - x	1
Write	- w -	2
Read	r - -	4



❶ Permissions for others

❷ Permissions for the group

❸ Permissions for the file owner

❹ The file type (Table 4-2)

4.6. Access Control

- Why 0, 1, 2, 4?
- Deny all permissions: $ sudo chmod 000 <file_name>
- Ex 1:
  - User: full permissions
  - Group: read + execute
  - Other: deny all permissions

Permission	Pattern	Representation
None	- - -	0
Execute	- - x	1
Write	- w -	2
Read	r - -	4

5. Linux Commands

5.1. Common Commands

- head: shows the first lines of a file
- tail: shows the last lines of a file
- sed: stream editor
- awk: it's a programming language designed for text processing.
- curl: a tool to make HTTP requests.
- wget: a tool for retrieving files using HTTP, HTTPS , or FTP.
- …

5.2. Shell

- Shell: command-line
- Common types of shells on Linux:
  - sh
  - Bash (default)
  - Csh
  - **Zsh (recommended)**
- Exercise 1: Rename image files with date prefix

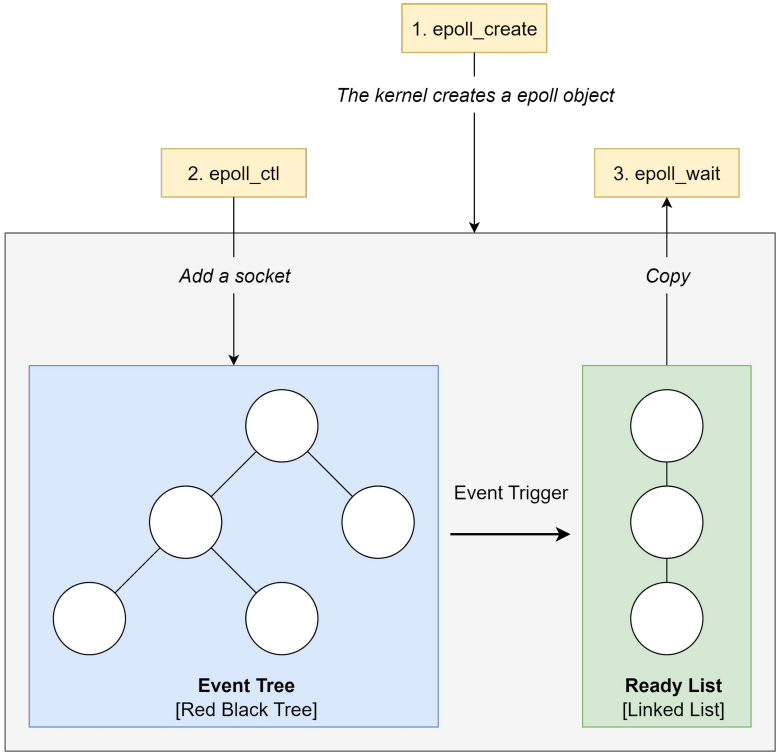
6. Network

6.1. How Many Connections Can a Server Serve?*

- It will be mainly limited by multiple factors:
  - File descriptor: The default value is 1024, but we can increase the number of file descriptors
  - **System memory**: each connection will occupy a certain amount of memory
  - **Threading Design Pattern**
- Traditional threading pattern: a connection → a thread.
  - A thread handles I/O operations, serialization, processing → it's inefficient
  - If threads are frequently created and destroyed, the system overhead will be considerable.
    → Solution: thread pool, but it's inefficient (the reason will be discussed later)
- Context switching between threads is still heavy.

6.2. Multiplexing I/O

- I/O multiplexing can **handle the I/O of multiple sockets/files in only one process**.
- The process can obtain multiple events from the kernel through a system call function epoll_wait()
- select/poll: When a network event occurs, the kernel needs to traverse the process concerned Socket collection
- epoll uses "red-black tree" + event-driven mechanism

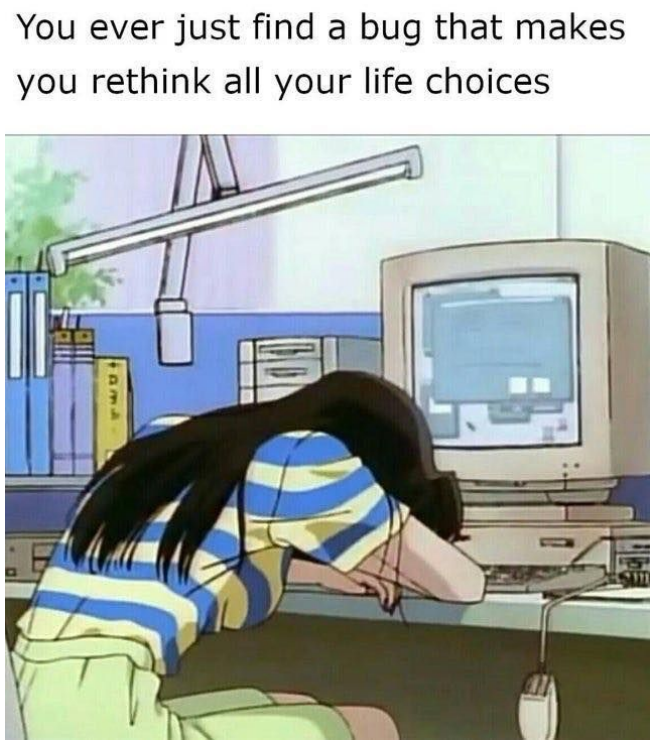


Recap

- Context switch is costly
  - System call (Kernel Mode)
  - For one core, executing A and B sequentially will always be faster than executing A and B concurrently.
- Optimize I/O operations: Find how I/O operations work under lib?
  - Non-blocking >> Blocking
  - Disk: Sequential I/O >> Random I/O
  - Network: Multiplexing I/O
- In Linux, everything is file

Homework

- Exercise 1: Replace env var in a manifest file
  - “Image: $IMAGE_TAG” in file deployment.yaml
  - Replace $IMAGE_TAG by using sed, awk to edit (value: ronin:v0.0.1)
- Exercise 2: Write shell script to open your workspace
  - Editor
  - Browser
  - Postman
  - Containers: mysql, redis, …
  - Ex: ./container.sh mysql up



References

- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html
- https://highscalability.com/big-list-of-20-common-bottlenecks/

Thank you 🙏