A **hard link** and a **soft link** (also known as a symbolic link) are both ways to reference files in a filesystem, but they function in different ways:

**1. Hard Link:**

* A hard link is essentially another name for an existing file. Both the original file and the hard link share the same **inode** (an index that contains metadata about the file, including its location on disk), meaning they point to the same physical data on the disk.
* Changes made to the data through the original file or the hard link are reflected in both because they both refer to the same data.
* If the original file is deleted, the hard link still allows access to the data because the data itself is not deleted until all hard links (including the original) are removed.
* Hard links cannot span different file systems or partitions and cannot be created for directories.

**Example of a hard link**:

A **hard link** and a **soft link** (also known as a symbolic link) are both ways to reference files in a filesystem, but they function in different ways:

**1. Hard Link:**

* A hard link is essentially another name for an existing file. Both the original file and the hard link share the same **inode** (an index that contains metadata about the file, including its location on disk), meaning they point to the same physical data on the disk.
* Changes made to the data through the original file or the hard link are reflected in both because they both refer to the same data.
* If the original file is deleted, the hard link still allows access to the data because the data itself is not deleted until all hard links (including the original) are removed.
* Hard links cannot span different file systems or partitions and cannot be created for directories.

**Example of a hard link**:

# Create a file

echo "This is a file." > file.txt

# Create a hard link to the file

ln file.txt hard\_link.txt

# Now both file.txt and hard\_link.txt point to the same data

cat hard\_link.txt # Output: This is a file.

# Modifying the original file

echo "Added content" >> file.txt

# Check the hard link

cat hard\_link.txt # Output: This is a file. Added content

# Deleting the original file

rm file.txt

# The hard link still works

cat hard\_link.txt # Output: This is a file. Added content

In this example, even after deleting file.txt, the data can still be accessed through hard\_link.txt because they point to the same data on disk.

**2. Soft Link (Symbolic Link):**

* A soft link (symbolic link) is a shortcut that points to another file or directory by its path. It contains the path to the target file rather than pointing to the actual data on disk.
* If the original file is deleted, the symbolic link becomes a **broken link** and does not provide access to the data anymore.
* Soft links can span across different filesystems or partitions and can be used to link directories.

**Example of a soft link**:

# Create a file

echo "This is a file." > file.txt

# Create a soft link to the file

ln -s file.txt soft\_link.txt

# Now soft\_link.txt points to the path of file.txt

cat soft\_link.txt # Output: This is a file.

# Modifying the original file

echo "Added content" >> file.txt

# Check the soft link

cat soft\_link.txt # Output: This is a file. Added content

# Deleting the original file

rm file.txt

# The soft link is now broken

cat soft\_link.txt # Output: cat: soft\_link.txt: No such file or directory

In this case, after deleting file.txt, the soft link (soft\_link.txt) no longer works because it only points to the path of the file, and the target file no longer exists.

**Key Differences:**

| **Hard Link** | **Soft Link (Symbolic Link)** |
| --- | --- |
| Points to the same inode and physical data on disk. | Points to the file path (symbolic reference). |
| Changes in one link affect all links (since they refer to the same data). | Changes in the original file are reflected in the symbolic link. |
| The data remains accessible even if the original file is deleted (as long as a hard link exists). | The symbolic link breaks if the original file is deleted. |
| Cannot reference directories (in most systems) and cannot span different file systems. | Can link to directories and can span different file systems. |
| More efficient in terms of storage, as no additional metadata is stored. | Less efficient as the symbolic link contains a path reference. |

Both types of links have their own use cases depending on the situation, such as linking files in different directories (soft links) or having multiple filenames pointing to the same data (hard links).

**What are System Calls?**

* **System calls** are the mechanism used by an application to request services from the operating system kernel (e.g., file handling, process control, memory management).

**10. What is the difference between Preemptive and Non-Preemptive Scheduling?**

* **Preemptive Scheduling** allows the OS to interrupt and switch between processes, ensuring that higher-priority tasks can preempt lower-priority ones.
* **Non-Preemptive Scheduling** ensures that once a process starts, it runs to completion unless it voluntarily yields the CPU.

**11. What is a Critical Section?**

* A **critical section** is a part of a program where shared resources are accessed. Synchronization techniques (like semaphores, mutexes) are used to prevent race conditions.

**12. What are Semaphores and Mutexes?**

* Both are synchronization tools to control access to shared resources.
* A **semaphore** is an integer that can be incremented or decremented and allows multiple threads to access a resource (depending on the value).
* A **mutex** (mutual exclusion) is a lock that allows only one thread to access a resource at a time.

Here are some important **Operating System (OS)** interview questions along with brief explanations that cover a wide range of topics:

**1. What is a Process?**

* A **process** is an instance of a program in execution. It contains the program code, its current activity (including program counter, registers, etc.), and the memory allocated to it.

**Follow-up**: What are the different states of a process?

* States include **New**, **Ready**, **Running**, **Waiting (Blocked)**, and **Terminated**.

**2. What is the difference between Process and Thread?**

* A **process** is an independent program in execution, while a **thread** is the smallest unit of execution within a process.
* Threads within the same process share the same memory space, whereas processes do not share memory by default.

**3. What is a Context Switch?**

* A **context switch** occurs when the CPU switches from one process (or thread) to another. The OS saves the state (context) of the current process and loads the context of the new process to be executed.

**Follow-up**: What is the overhead of a context switch?

* It requires saving and restoring process states, which takes time and resources.

**4. What is Deadlock?**

* A **deadlock** is a situation in a multitasking environment where a set of processes are blocked because each process is waiting for a resource that another blocked process holds.

**Follow-up**: What are the four conditions for deadlock?

* **Mutual Exclusion**: Only one process can use a resource at a time.
* **Hold and Wait**: A process is holding at least one resource and waiting for additional resources.
* **No Preemption**: A resource cannot be forcibly taken from a process.
* **Circular Wait**: A set of processes are waiting in a circular chain.

**5. What is Virtual Memory?**

* **Virtual memory** is a memory management technique where secondary memory (like a hard disk) is used as an extension of main memory (RAM). It allows processes to run even if they require more memory than is physically available.

**Follow-up**: What is paging and segmentation in virtual memory?

* **Paging**: Divides memory into fixed-size blocks called **pages**.
* **Segmentation**: Divides memory into variable-size segments based on logical divisions in a program (like functions or data structures).

**6. What is the difference between Paging and Segmentation?**

* **Paging** breaks memory into fixed-size blocks called pages and is more focused on managing memory uniformly.
* **Segmentation** breaks memory into variable-size segments based on logical divisions within a program (e.g., code segment, data segment).

**7. What is a CPU Scheduler?**

* The **CPU scheduler** is part of the operating system responsible for deciding which process or thread gets to use the CPU at any given time.

**Follow-up**: What are the different CPU scheduling algorithms?

* **FCFS** (First Come First Serve)
* **SJF** (Shortest Job First)
* **Round Robin**
* **Priority Scheduling**

**8. What is Thrashing?**

* **Thrashing** occurs when a system spends more time swapping pages in and out of memory (due to high paging activity) than executing processes, leading to poor system performance.

**9. What are System Calls?**

* **System calls** are the mechanism used by an application to request services from the operating system kernel (e.g., file handling, process control, memory management).

**10. What is the difference between Preemptive and Non-Preemptive Scheduling?**

* **Preemptive Scheduling** allows the OS to interrupt and switch between processes, ensuring that higher-priority tasks can preempt lower-priority ones.
* **Non-Preemptive Scheduling** ensures that once a process starts, it runs to completion unless it voluntarily yields the CPU.

**11. What is a Critical Section?**

* A **critical section** is a part of a program where shared resources are accessed. Synchronization techniques (like semaphores, mutexes) are used to prevent race conditions.

**12. What are Semaphores and Mutexes?**

* Both are synchronization tools to control access to shared resources.
* A **semaphore** is an integer that can be incremented or decremented and allows multiple threads to access a resource (depending on the value).
* A **mutex** (mutual exclusion) is a lock that allows only one thread to access a resource at a time.

**13. What is Demand Paging?**

* **Demand paging** loads pages into memory only when they are needed, as opposed to loading the entire process into memory at once.

**15. What is the Banker's Algorithm?**

* The **Banker's Algorithm** is a deadlock avoidance algorithm. It simulates the allocation of resources and checks if the system is in a **safe state** by ensuring that no deadlock occurs when allocating resources.

**19. What is a Zombie Process?**

* A **zombie process** is a process that has completed execution but still has an entry in the process table (because its parent hasn't read its exit status). The process is "dead" but still consuming system resources.

A **zombie process** is created when a process has completed its execution but still has an entry in the process table. This happens because the process's parent has not yet read its exit status.

**How a Process Becomes a Zombie:**

* **Process Execution**:
  + A process starts executing and runs to completion, either by calling exit() or finishing its main function.
* **Exit Status**:
  + Upon termination, the process sends an exit status (a small amount of information about how the process terminated) to its parent process.
* **Parent Process Reads Exit Status**:
  + Normally, the parent process calls the wait() or waitpid() system call to read the exit status of its child process. This is important for resource cleanup.
* **If the Parent Does Not Read the Exit Status**:
  + If the parent process does not call wait() or waitpid(), the child process remains in the **"zombie" state**. The process is terminated, but its entry in the process table remains until the parent reads its exit status.
  + The zombie process is "dead" because it no longer consumes any CPU time or performs any operations, but it is not completely removed from the system as its entry in the process table persists.

**Orphan Process**

* **Definition**: An **orphan process** is a process that is still executing but whose parent process has terminated. This can happen if the parent process ends before the child process.
* **State**: The process is still active and running, but it no longer has a parent process.
* **Resource Usage**: Orphan processes continue to consume CPU and memory resources as they are still in execution.
* **Cleanup**: Orphan processes are adopted by the init process (PID 1). The init process becomes the new parent and is responsible for cleaning up the orphan process when it terminates.
* **Summary of Differences**

| **Feature** | **Zombie Process** | **Orphan Process** |
| --- | --- | --- |
| **Definition** | Completed execution, waiting for parent to read status | Running process whose parent has terminated |
| **State** | Dead (not running) | Alive (still running) |
| **Resource Usage** | Occupies process table entry, no CPU/memory usage | Consumes CPU and memory resources |
| **Cleanup** | Cleared when parent reads exit status or if init adopts it | Adopted and managed by init process |

**3. What is a Context Switch?**

* A **context switch** occurs when the CPU switches from one process (or thread) to another. The OS saves the state (context) of the current process and loads the context of the new process to be executed.

**4. What is the difference between internal and external fragmentation?**

* Answer: Internal fragmentation occurs when allocated memory is slightly larger than the requested memory, leading to wasted space. External fragmentation occurs when there is enough total memory but it is scattered in non-contiguous blocks.

**5. What is Virtual Memory?**

* **Virtual memory** is a memory management technique where secondary memory (like a hard disk) is used as an extension of main memory (RAM). It allows processes to run even if they require more memory than is physically available.

What is paging and segmentation in virtual memory?

* **Paging**: Divides memory into fixed-size blocks called **pages**.
* **Segmentation**: Divides memory into variable-size segments based on logical divisions in a program (like functions or data structures).

Here are some important **Operating System (OS)** interview questions along with brief explanations that cover a wide range of topics:

**1. What is a Process?**

* A **process** is an instance of a program in execution. It contains the program code, its current activity (including program counter, registers, etc.), and the memory allocated to it.

**Follow-up**: What are the different states of a process?

* States include **New**, **Ready**, **Running**, **Waiting (Blocked)**, and **Terminated**.

**2. What is the difference between Process and Thread?**

* A **process** is an independent program in execution, while a **thread** is the smallest unit of execution within a process.
* Threads within the same process share the same memory space, whereas processes do not share memory by default.

**3. What is a Context Switch?**

* A **context switch** occurs when the CPU switches from one process (or thread) to another. The OS saves the state (context) of the current process and loads the context of the new process to be executed.

**Follow-up**: What is the overhead of a context switch?

* It requires saving and restoring process states, which takes time and resources.

**4. What is Deadlock?**

* A **deadlock** is a situation in a multitasking environment where a set of processes are blocked because each process is waiting for a resource that another blocked process holds.

**Follow-up**: What are the four conditions for deadlock?

* **Mutual Exclusion**: Only one process can use a resource at a time.
* **Hold and Wait**: A process is holding at least one resource and waiting for additional resources.
* **No Preemption**: A resource cannot be forcibly taken from a process.
* **Circular Wait**: A set of processes are waiting in a circular chain.

**5. What is Virtual Memory?**

* **Virtual memory** is a memory management technique where secondary memory (like a hard disk) is used as an extension of main memory (RAM). It allows processes to run even if they require more memory than is physically available.

**Follow-up**: What is paging and segmentation in virtual memory?

* **Paging**: Divides memory into fixed-size blocks called **pages**.
* **Segmentation**: Divides memory into variable-size segments based on logical divisions in a program (like functions or data structures).

**6. What is the difference between Paging and Segmentation?**

* **Paging** breaks memory into fixed-size blocks called pages and is more focused on managing memory uniformly.
* **Segmentation** breaks memory into variable-size segments based on logical divisions within a program (e.g., code segment, data segment).

**7. What is a CPU Scheduler?**

* The **CPU scheduler** is part of the operating system responsible for deciding which process or thread gets to use the CPU at any given time.

**Follow-up**: What are the different CPU scheduling algorithms?

* **FCFS** (First Come First Serve)
* **SJF** (Shortest Job First)
* **Round Robin**
* **Priority Scheduling**