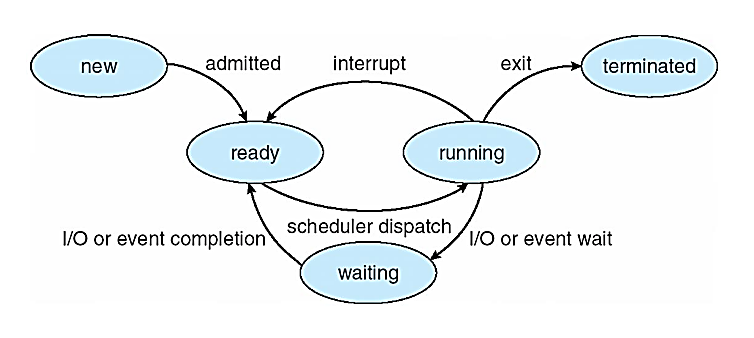
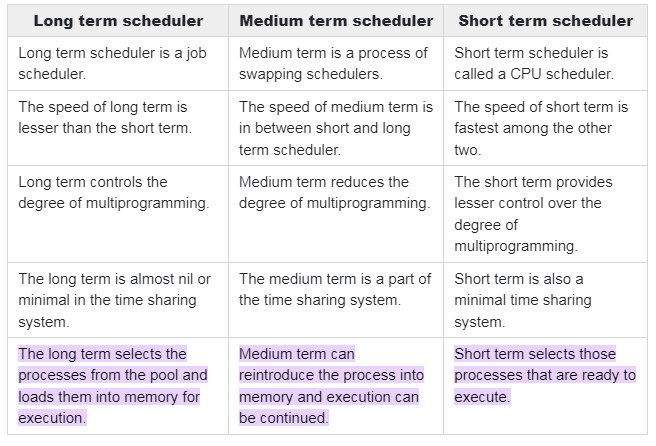
1. **What is a process? Describe the different states of a process using state diagram.**

Answer: A process is a running program that serves as the foundation for all computation. The procedure is not the same as computer code, although it is very similar. In contrast to the program, which is often regarded as some 'passive' entity, a process is an 'active' entity.

Process State: As a process executes, it changes state

**1.new:** The process is being created **2. running:** Instructions are being executed **3. waiting:** The process is waiting for some event to occur **4. ready:** The process is waiting to be assigned to a processor **5. terminated:** The process has finished execution

1. **Describe the differences among short-term, medium-term, and long-term scheduling.**



1. **The Sun UltraSPARC processor has multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?**

Answer: The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

1. **Consider a multiprocessor system and a multithreaded program written using the many to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system. Discuss the performance implications of the following scenarios.**

**a. The number of kernel threads allocated to the program is less than the number of processors**.

Answer: The process won't use all the processors in the system. All the work of the process will be done by a few processors, taking more time for the process to finish.

**b. The number of kernel threads allocated to the program is equal to the number of processors.** Answer: The process will utilize all the processors in the system; therefore, the work will finish as soon as possible. But if a thread is blocked by a system call, then a processor is not utilized while the thread waits, lowering the system usage.

**c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.**

Answer: All the processors in the system will be used by the process. The OS will have to schedule the K threads over the P processors, with a little overhead for the context switching. However, if a thread running in some processor gets blocked by a system call, the scheduler can schedule another ready thread in that processor, maximizing system utilization.

1. What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?
2. **Management and Scheduling**: User-Level Threads: Managed and scheduled by the user-level thread library, not by the OS kernel. The kernel is unaware of these threads and manages them as a single process**. Kernel-Level Threads:** Directly managed and scheduled by the OS kernel. Each thread is known to the kernel, allowing finer control over their management.
3. **Performance and Overhead:** User-Level Threads: Typically faster in terms of creation, context switching, and management since these operations are handled within the user space without kernel intervention. Kernel-Level Threads: Incur more overhead due to system calls required for management, but can leverage multi-core processors more effectively as the kernel can schedule threads on different processors.

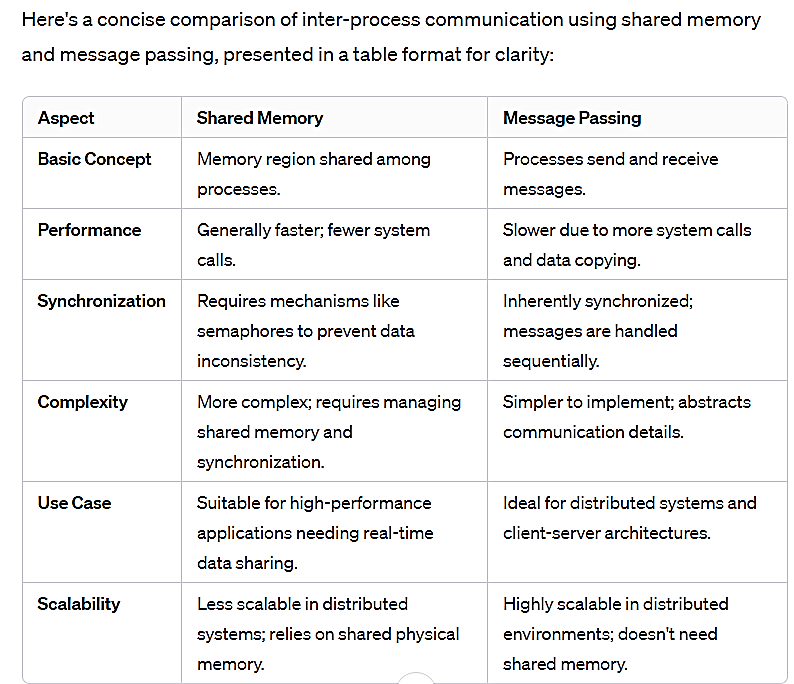
**Circumstances Favoring Each Type**

User-Level Threads are better when: High-frequency thread creation and destruction are required, as they have lower overhead. Applications need fine-grained control over thread scheduling and management.

Kernel-Level Threads are better when: The application can benefit from multi-core processors, as the kernel can schedule threads across multiple cores. Need for stronger isolation between threads, as each kernel-level thread operates in a separate memory space. Integration with certain kernel-level services and functionalities is needed.

1. **Describe and contrast inter-process communication using shared memory and message passing.**

**Answer:** Shared memory system is the fundamental model of inter process communication. In a shared memory system, in the address space region the cooperating communicates with each other by establishing the shared memory region. Shared memory concept works on fastest inter process communication. If the process wants to initiate the communication and it has some data to share, then establish the shared memory region in its address space. After that, another process wants to communicate and tries to read the shared data, and must attach itself to the initiating process’s shared address space. **Message Passing** provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. For example − Chat program on the World Wide Web. Message passing provides two operations which are as follows − • Send message • Receive message Messages sent by a process can be either fixed or variable size.



1. **Describe how processes are created and terminated in an operating system, including developing programs using the appropriate system calls that perform these operations.**

Processes in an operating system (OS) are fundamental units of execution, and their creation and termination are essential aspects of OS functionality. Here’s how these operations typically work:

**Process Creation**

**System Calls**: Processes are usually created through system calls. In Unix-like systems, this is often done using fork() and exec() calls. fork(): Creates a new process by duplicating the calling process. The new process, known as the child, is an exact copy of the parent process except for returned values. exec(): Used after a fork(). It replaces the process's memory space with a new program. The exec() family of functions loads and runs a new program, so the process's code, data, heap, and stack are replaced.

**Steps in Creation**: The OS allocates space for the process (memory, process control block, etc.). The new process is initialized with a copy or reference to the resources of the parent process. The OS schedules the new process for execution.

**Process Termination**

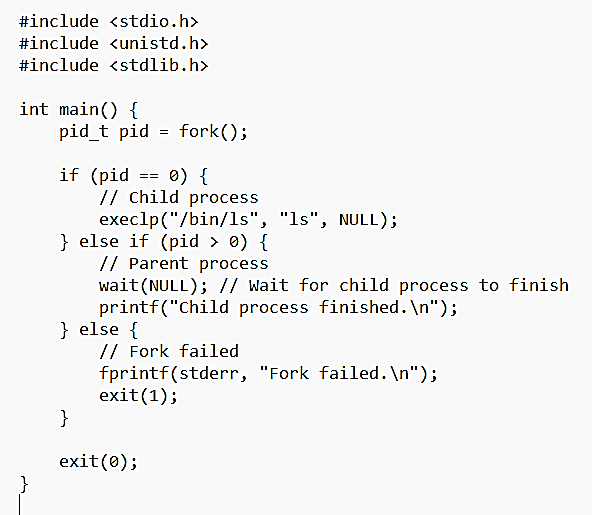
**System Call:** Usually exit() in Unix-like systems. exit(): Ends the calling process and releases its resources.

**Steps in Termination:** The process releases the resources it's using (like memory and file handles). The OS updates its process and resource tables to reflect that the resources are available. Parent processes may be notified of the termination (for example, via a wait(

system call).

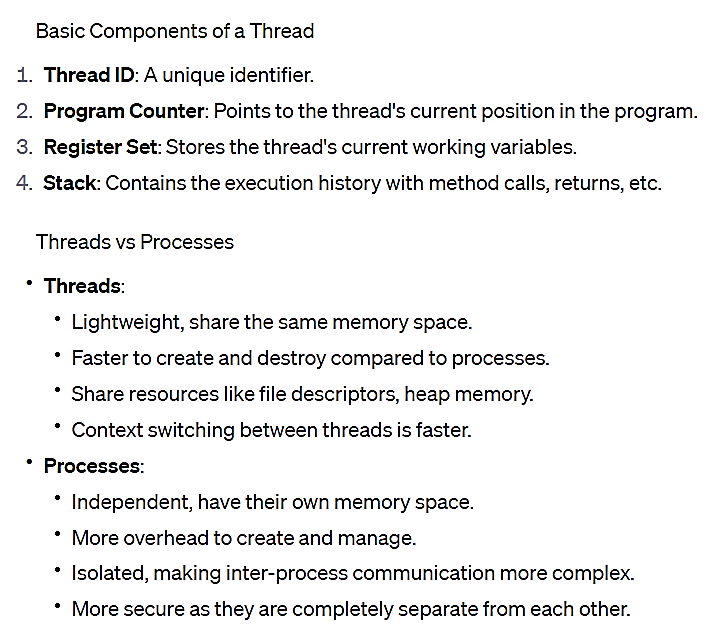
Programming Example

In programming, use fork() to create a process and exec() to run a new program in the child process. Terminate processes with exit() to ensure proper resource cleanup.



1. **Describe client-server communication using sockets and remote procedure calls.**

**Using Sockets**  
 Description: Sockets are an endpoint for sending and receiving data across a network. They are used to establish a connection between a client and a server. **How it Works:** Server: Initializes a socket, binds it to an IP address and port, and listens for connections. Client: Creates a socket and connects it to the server's IP address and port. Once connected, data can be sent back and forth over this socket connection.   
**Using Remote Procedure Calls (RPC) Description:** RPC allows a program to cause a procedure to execute on another address space (commonly on another physical machine). How it Works: The client makes a procedure call that sends a request to the server and waits for a response. The server processes the request, executes the procedure, and sends back results. This process abstracts the network communication, making the remote procedure appear like a local one.

1. **Identify the basic components of a thread, and contrast threads and processes**
2. **Describe the benefits and challenges of designng multithreaded applications**

**Benefits:** **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces

**Resource Sharing** – threads share resources of process, easier than shared memory or message **passing Economy** – cheaper than process creation, thread switching lower overhead than context switching

**Scalability** – process can take advantage of multicore architectures

**Disadvantages of a Multithreaded Application** •Difficulty of writing code •Difficulty of debugging •Difficulty of managing concurrency •Difficulty of testing •Difficulty of porting existing code •Remove static variables • Replace any function calls that are not thread-safe • Replace any other code that is not thread-safe

1. **Illustrate different approaches to implicit threading including thread pools, fork-join, and Grand Central Dispatch**

Implicit Threading Approaches

**1. Thread Pools Description:** A thread pool maintains a pool of worker threads. Instead of creating new threads for each task, tasks are assigned to existing threads. A fixed number of threads are created at startup and maintained in a pool. When a task arrives, it's assigned to a free thread. After the thread completes the task, it returns to the pool for the next task. Benefits: Reduces the overhead of thread creation and destruction, improves performance, and limits the number of simultaneously running threads.

**2. Fork-Join Description:** A parallel programming model that divides a task into sub-tasks, processes them in parallel, and then joins the results. How it Works: The main task (the "fork") splits into multiple parallel sub-tasks. Once these sub-tasks are completed, their results are combined back together (the "join"). Benefits: Simplifies parallel data processing, particularly in recursive algorithms and divide-and-conquer strategies.

**3. Grand Central Dispatch (GCD) Description:** A technology developed by Apple to optimize application support for concurrent code execution on multicore systems. How it Works: Uses dispatch queues to execute blocks of code either synchronously or asynchronously. GCD manages the thread pool automatically, optimizing based on system conditions. Benefits: Simplifies code for concurrent operations, efficiently uses system resources, and automatically manages threads.

1. **Describe how the Windows and Linux operating systems represent threads**

**Windows**

In Windows, threads are represented as objects within the kernel. Each thread has a thread kernel object, a stack, a thread environment block, and a user-mode stack. Threads share resources like memory space and file handles with other threads in the same process. Windows uses a preemptive scheduling model to manage threads.

**Linux**

In Linux, threads are treated similarly to processes but share resources like memory and file descriptors. Linux uses a 1:1 threading model where each user-level thread maps to a kernel thread. Threads are represented by task struct in the kernel, which contains thread state, scheduling information, and more. Linux's scheduler sees and schedules each thread independently.

**13.What resources are used when a thread is created? How do they differ from those used when a process is created?**

Answer: Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

1. **Identify the separate components of a process and illustrate how they are represented in an operating system.**

Answer: Components of a Process in Operating Systems Process Control Block (PCB): Central data structure that contains process information like process state, program counter, CPU registers, memory management information, accounting information, and I/O status information. Stack: Stores the temporary data such as method/function parameters, return address, and local variables. Heap: Area of memory used for dynamic memory allocation where variables are allocated and de-allocated in an arbitrary order. Text Section: Contains the executable code of the process. Data Section: Stores global and static variables. In an operating system, a process is typically represented by a data structure (like PCB in Linux or PROCESS in Windows) that contains all the information needed to manage and execute the process.

1. **Difference between parallelism and concurrency. Difference between data and task parallelism**

