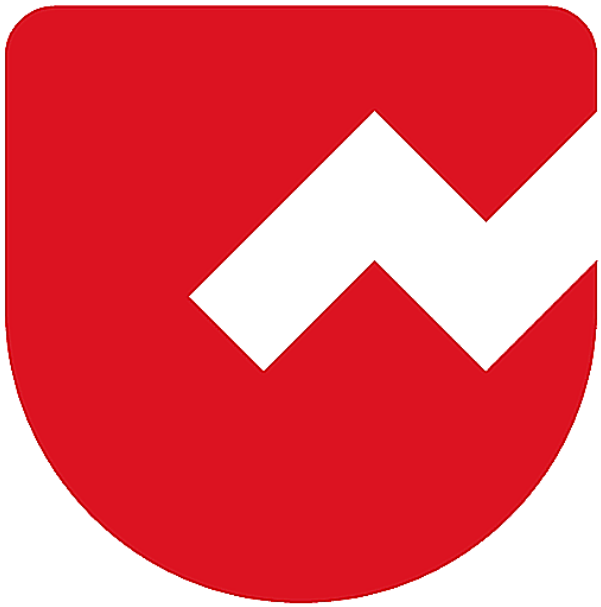
Zoho Schools for Graduate Studies



Notes

Session 1

Multithreading

What is a process?

A process is a program in running state. Process exists in primary memory. Resources shared during process execution CPU time, primary memory, heap, stack I/O classess.

PROCESS

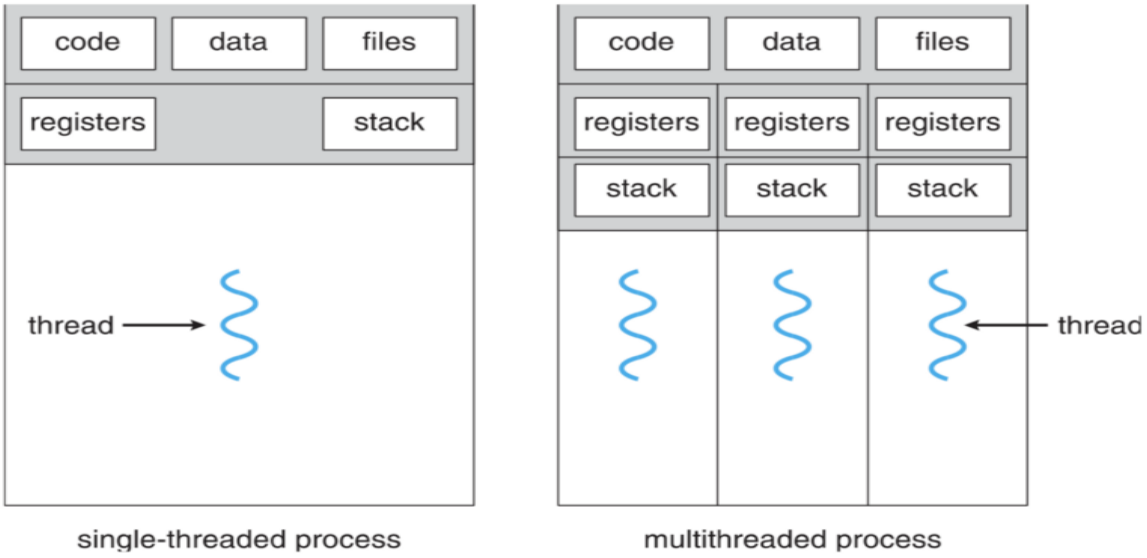
- There are two basic units of execution : process and thread
- Concurrent programming uses threads
- Active processes are processes in running state.
- There exists one or more threads in every process

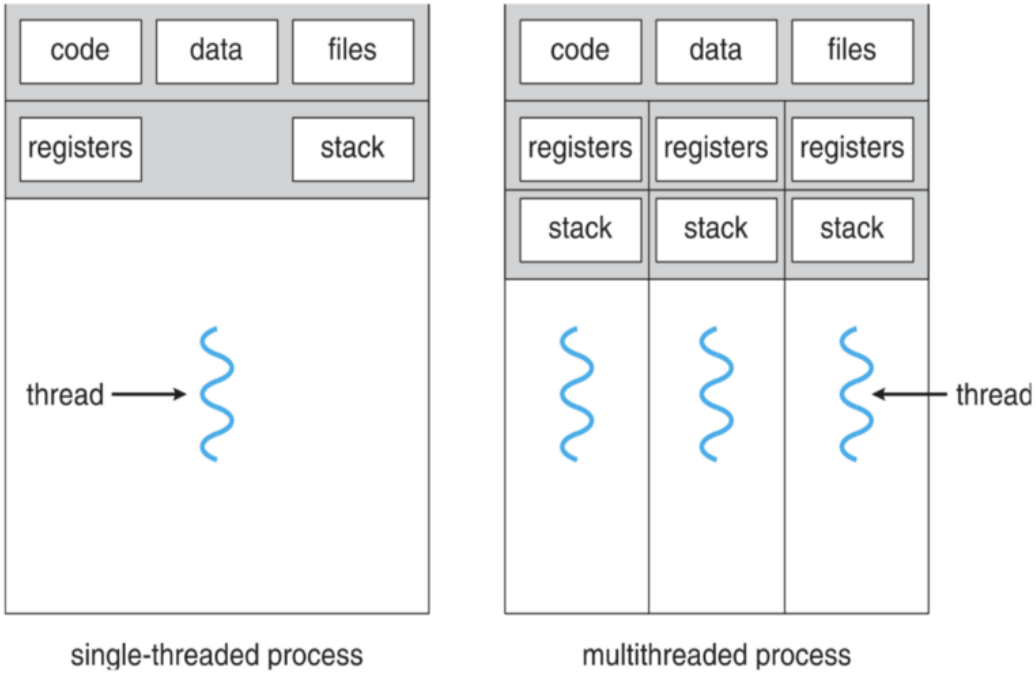
Process has a self contained execution environment. To facilitate communication between processes, OS supports Inter Process Communication (IPC) using pipes and sockets. Most implementation of JVM run as a single process.

THREADS:

Threads are light weight processes. Threads exist within a process. Threads share process resources such as memory and open files.

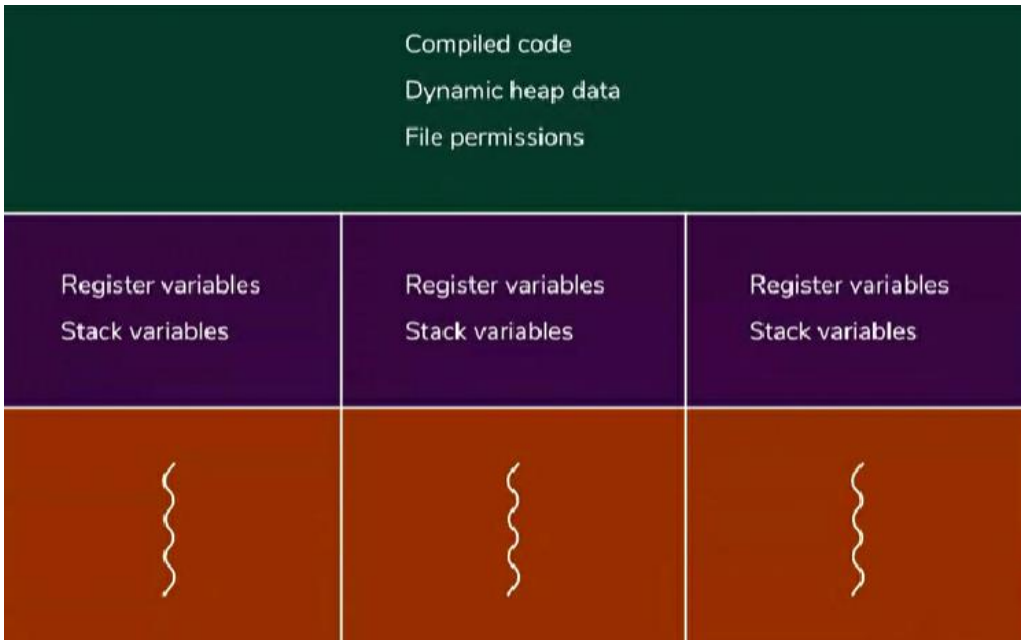
We start with one thread(main thread) which has the ability to create multiple threads.





Each thread has its own call stack and PC register.

Creating a new thread requires fewer resources than creating a new process.



MULTITHREADING

Multithreading is the capability for a single CPU core to execute multiple threads at once. This improves system utilization and responsiveness by more efficiently splitting up tasks

Types of Threads :

- **Kernel Thread**: A thread managed directly by the operating system kernel.
- **User Thread**: A thread managed by the user-level thread library, without kernel involvement.
- **Daemon Thread**: A background thread in Java that runs continuously and does not prevent the JVM from exiting.

Context Switching

Context switching allows CPU cores to alternate between ready and blocked processes to best take advantage of limited computing resources.

Preemption

Preemption occurs when a process is temporarily interrupted by an external scheduler to prioritize a more important task.

Blocked Process

A process is *blocked* when it has to wait for a contested, limited, or slow resource, such as accessing a specific file or waiting for a network request or waiting for I/O.

Advantages of Process

- Processes work independently in their own memory, ensuring no interference and better security.
- Resources like CPU and memory are allocated effectively to optimize performance.
- Processes can be prioritized to ensure important tasks get the resources they need.
-

Disadvantages of Process

- Frequent switching between processes can slow down the system and reduce speed.
- Improper resource management can cause deadlocks where processes stop working and block progress.
- Having too many processes can make the process table take up a lot of memory. This can also make searching or updating the table slower, which can reduce system performance.

Advantages of Thread

- When there is a lot of computing and input/output (I/O) work, threads help tasks run at the same time, making the app faster.
- Another advantage for having threads is that since they are lighter weight than processes, they are easier (i.e., faster) to create and destroy than processes.
- Many apps need to handle different tasks at the same time. For example, a web browser can load a webpage, play a video, and let you scroll all at once. Threads make this possible by dividing these tasks into smaller parts that can run together.

Disadvantages of Thread

- Threads in the same process are not completely independent like separate processes. They share the same memory space including global variables. This means one thread can accidentally change or even erase another thread's data as there is no protection between them.
- Threads also share resources like files. For example - if one thread closes a file while another is still using it, it can cause errors or unexpected behavior.
- If too many threads are created they can slow down the system or cause it to run out of memory.

States of thread

The states of a thread in Java are:

1. New - The thread is created but not yet started.
2. Runnable - The thread is ready to run and waiting for CPU time.
3. Running - The thread is actively executing.
4. Blocked - The thread is waiting to acquire a lock to enter a synchronized block/method.
5. Waiting - The thread is waiting indefinitely for another thread's action.
6. Timed Waiting - The thread is waiting for a specified time.
7. Terminated (Dead) - The thread has finished execution or exited due to an exception.

Creating Thread

1. Extending Thread
2. Implementing Runnable

Thread should override the method run().

Exception handling should be used to handle InterruptedException. The join() method is used to pause the execution of the current thread until the specified thread has finished executing.

SESSION 2

Synchronization

Synchronization is a mechanism to control access to shared resources in multithreaded programs to avoid inconsistency and ensure thread safety.

Race Condition

- Occurs when two or more threads access shared data simultaneously, and the final result depends on the thread execution order.
- Can lead to unpredictable results and bugs.

Critical Section

- A part of the code where the shared resource is accessed.
- Only one thread should execute in the critical section at a time to avoid race conditions.

Mutual Exclusion

- Ensures that only **one thread** can access the **critical section** at a time.
- Achieved in Java using synchronized keyword or other synchronization constructs.

Static Synchronization

- Synchronizing a static method so the lock is applied on the class object (ClassName.class) rather than on instance objects.
- Useful when shared resources are static.

Interthread Communication

- Mechanism where threads **communicate using wait(), notify(), and notifyAll() methods.**

- Allows efficient coordination by letting threads wait and notify each other about resource availability.

Starvation

- Happens when a thread is **perpetually denied access** to resources because other higher-priority threads are always favored.
- A technique to **gradually increase the priority** of waiting threads to prevent starvation.

Deadlock

- A situation where two or more threads are waiting on each other to release resources, and none can proceed.
- Typically occurs when multiple threads hold locks and wait for each other's locks.

Semaphores

- A synchronization tool that controls access to a shared resource using counters.
- Java provides Semaphore class in java.util.concurrent package.

Lifecycle Management of Thread:

- start():
  Initiates the execution of a thread by invoking its run() method in a new, separate thread of control.
- run():
  This method contains the code that the thread will execute. It is automatically called by the Java Virtual Machine (JVM) when start() is invoked.
- sleep(long millis):
  Causes the currently executing thread to temporarily pause its execution for the specified number of milliseconds.
- join():
  Allows one thread to wait for the completion of another thread. The calling thread will block until the target thread finishes its execution.

- isAlive():
  Returns true if the thread is currently alive (has been started and has not yet terminated), and false otherwise

Thread Information and Control:

- currentThread(): A static method that returns a reference to the currently executing thread object.
- getName(): Returns the name of the thread.
- setName(String name): Sets the name of the thread.
- getPriority(): Returns the priority of the thread.
- setPriority(int newPriority): Sets the priority of the thread. Thread priorities range from Thread.MIN_PRIORITY to Thread.MAX_PRIORITY.
- yield(): A hint to the scheduler that the current thread is willing to yield its current use of a processor, allowing other threads to run.
- interrupt(): Interrupts a thread. This sets the thread's interrupted status to true.
- isInterrupted(): Tests whether the current thread has been interrupted.
- interrupted(): Tests whether the current thread has been interrupted. The interrupted status of the thread is cleared by this method.

Synchronization and Inter-Thread Communication (Used with Object class methods):

- wait(): Causes the current thread to wait until another thread invokes the notify() or notifyAll() method for this object, or a specified amount of time has elapsed.
- notify(): Wakes up a single thread that is waiting on this object's monitor.
- notifyAll(): Wakes up all threads that are waiting on this object's monitor.