Task 7: what are the clean code rules or principles?

Meaningful Names: Name your symbols clearly, in a clear and searchable manner that avoids mind mapping, puns, and wit. Choose one word for each concept and keep the names short.

Write neat jobs: small, well defined, and free from side effects. Get them to do one thing well and one thing only. Keep your code dry, organized, and focused.

Use the right comments: Avoid redundant, misleading, prescriptive, or everyday comments, but keep and use legal, educational, and explanatory comments.

Format your code correctly, according to the standard agreed in your product and team

Avoid unnecessary classes, and keep them small and cohesive. Define your stripping well, focus on responsibilities and solitude.

Handle all errors: define normal flow, use unchecked exceptions, use unchecked exceptions, provide context when throwing

Test your code. At least with unit tests. Integration, component, and comprehensive tests are also preferred. Use all possible guarantees.

Task 8: why recursive is faster than iterative?

recursive	iterative
recursion is always called on a function	iterative code is applied on variables and is a set of instructions
Recursive code will terminate on a base case condition	iterative code will either run for a particular number of loops or until a specified condition is met.
An infinite recursion can lead to the system crashing	iterative code, it will just consume more CPU cycles
Recursive code has an overhead time for each recursive call	iterative code has no an overhead time for each recursive call.

- Recursive code is smaller and neater in length than iterative code.
- Recursive code is slower than iterative code as it not only runs the program but also has to invoke stack memory.
- Recursion uses the stack to store the variable changes for each recursive call, whereas iterative code does not.

Orange Digital Center

Task 9 : Threads research

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. In many respect, threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. Like a traditional process i.e., process with one thread, a thread can be in any of several states . Each thread has its own stack. Since thread will generally call different procedures and thus a different execution history. This is why thread needs its own stack. An operating system that has thread facility, the basic unit of CPU utilization is a thread

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers.

• Traditional processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time.

• As shown in Figure 4.1, multi-threaded applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.

We use threads due to many reasons. Threads plays a key role in the designing of an operating system. A process with multiple threads make a great server for example printer server. Because threads can share common data, they do not need to use inter process communication .Because of the very nature, threads can take advantage of multiprocessors.

Threads are cheap as They only need a stack and storage for registers therefore, threads are cheap to create. Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources. Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

MULTI THREAD PROCESS

Multithreading is mainly found in multitasking operating systems. Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process's resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. Multithreading can also be applied to a single process to enable parallel execution on a multiprocessing system

• There are two types of threads to be managed in a modern system: User threads and kernel threads.

• User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.

• Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

• In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

A. Many to one
Where many user level threads are mapped to as single kernel thread. In such model the thread is managed by thread library in the user space which makes such model efficient. However the entire threads can be blocked if a thread makes a blocking system call. Also; with multiprocessors system; threads can access the kernel with only a single thread at a time and as such; multiple threads are unable to take advantage on multiprocessor and run in parallel on multiprocessors. With such model, developer will be able to create as many threads as required; however the true concurrency is not implemented since the kernel can schedule only one thread at a time.

B. One to one

Where each user thread is mapped to the kernel thread. Such mechanism provides concurrency operation than the many-to –many model where other threads are allowed to even when one of these threads makes a blocking systemcall. Also such system allows multiple threads to run in parallel on multiprocessors. Since such model creates a corresponding kernel thread with every creating of user thread, on overhead of creating kernel threads can affect the performance of an application and as such this implementation can be restricted with the number of threads that can be supported be the system.

C. Many to many

The many to many model multiplexes any number of user threads onto an equal or smaller, number of kernel threads , combining the best of the one-to –one and many to one models. Users have no restrictions on the number of threads created .Blocking kernel system calls do not block the entire process. Processed can be split across multiple process. Individual processes may be allocated variable number of kernel threads.

ADVANTAGES:

Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others This is particularly true when one of its task may block, and it is desired to allow the other tasks to proceed without blocking