**Task 1**

**What are the components of the OS’s kernel**

1) Process Management.

2) Memory management.

3) Device Management.

4) Interrupt Handling.

5) I/O Communication.

**Task 2**

**Password library**

Make Pass

**Task 3**

How to write c++ code in Jupyter notebook

Before you install the modules, you want to set up your own environment to prevent conflicts with your default setup. Open up a terminal and type ‘conda activate’. Enter the following commands

conda create -n cling

Next, you want to install cling to your particular environment.

conda install xeus-cling -c conda-forge

Finally, install Xeus:

conda install xeus -c conda-forge

Open a shell and switch to your cling environment.

conda activate cling

Open a notebook

jupyter notebook

**task 4**

a python code that pass by 2 one time and pass by times 2 another time

**Task 5**

how to make infifnite for loop in python without using extra memory

def infinity():

while True:

yield

for \_ in infinity():

pass

**Task 6**

what is dependency injection

Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. It's a very useful technique for testing, since it allows dependencies to be mocked or stubbed out.

Dependencies can be injected into objects by many means (such as constructor injection or setter injection). One can even use specialized dependency injection frameworks (e.g. Spring) to do that, but they certainly aren't required. You don't need those frameworks to have dependency injection. Instantiating and passing objects (dependencies) explicitly is just as good an injection as injection by framework.

[**https://stackoverflow.com/questions/130794/what-is-dependency-injection**](https://stackoverflow.com/questions/130794/what-is-dependency-injection)

[**https://www.youtube.com/watch?v=Eqi-hYX50MI**](https://www.youtube.com/watch?v=Eqi-hYX50MI)

[**https://www.youtube.com/watch?v=IKD2-MAkXyQ**](https://www.youtube.com/watch?v=IKD2-MAkXyQ)

**Task 7**

what are the clean code rules or principles

* Write code as simply as possible: KISS
* Avoid unnecessary repetition: DRY
* Delete what is not needed: YAGNI
* Readability over conciseness
* Composition over inheritance
* Favor readability
* Practice consistency
* Don't Repeat Yourself

**Task 8**

**why recursive is faster than iterative**

The recursive function runs much faster than the iterative one . The reason is because in the latter, for each item, a CALL to the function st\_push is needed and then another to st\_pop . In the former, you only have the recursive CALL for each node.

The fact is that recursion is rarely the most efficient approach to solving a problem, and iteration is almost always more efficient . This is because there is usually more overhead associated with making recursive calls due to the fact that the call stack is so heavily used during recursion.

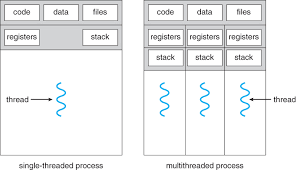
iterative solutions are usually faster than recursive solutions when it comes to speed. ... In a standard programming language, where the compiler doesn't have tail-recursive optimization, Recursive calls are usually slower than iteration.

<https://iq-faq.com/en/Q%26A/page=fea7a01ddceb8c974ce7fcce54dbd150>

**Task 9**

**Threads**

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, ( and a thread ID). Traditional ( heavyweight ) processes (HWP) 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. 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.

Single-threaded and multithreaded processes

• Most modern applications are multithreaded

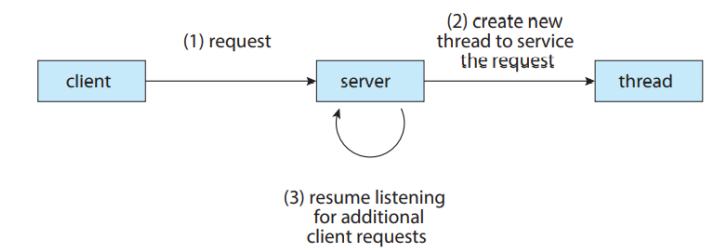
• Threads run within application

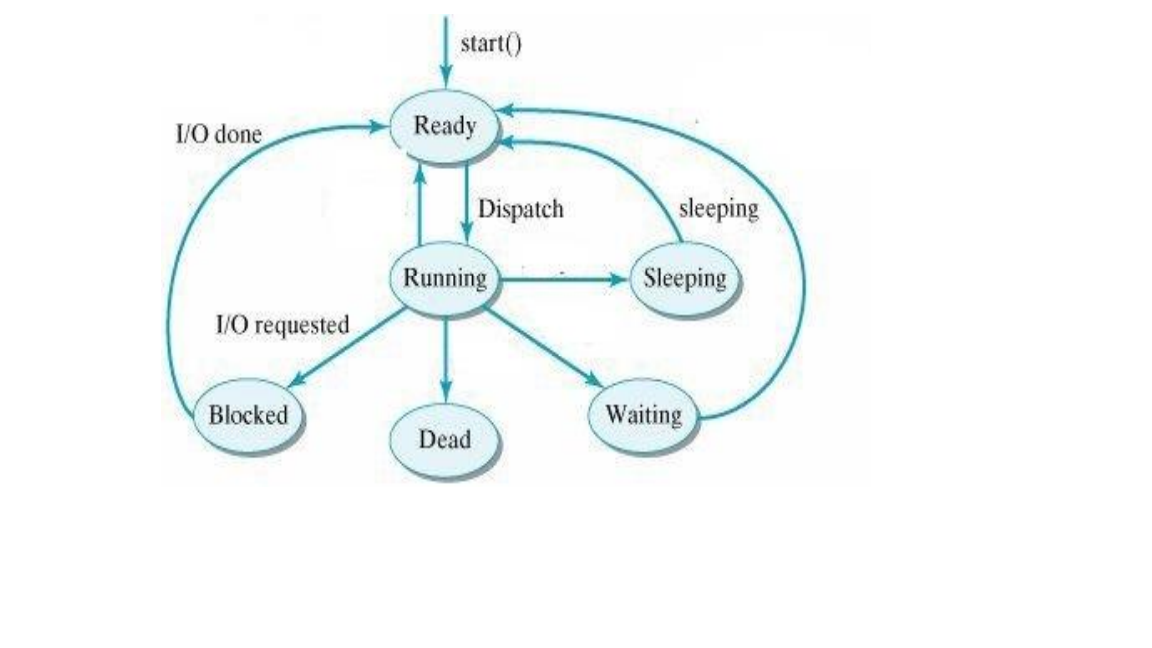
• Multiple tasks with the application can be implemented by separate threads o Update display o Fetch data o Spell checking o Answer a network request

• Process creation is heavy Process creation is heavy-weight while thread creation is weight while thread creation is light-weight

• Can simplify code, increase efficiency

• Kernels are generally multithreaded

In certain situations, a single application may be required to perform several similar tasks. For example, a web server accepts client requests for web pages, images, sound, and so forth. A busy web server may have several (perhaps thousands of) clients concurrently accessing it. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time, and a client might have to wait a very long time for its request to be serviced. One solution is to have the server run as a single process that accepts requests. When the server receives a request, it creates a separate process to service that request. In fact, this process-creation method was in common use before threads became popular. Process creation is time consuming and resource intensive, however. If the new process will perform the same tasks as the existing process, why incur all that overhead? It is generally more efficient to use one process that contains multiple threads. If the web-server process is multithreaded, the server will create a separate thread that listens for client requests. When a request is made, rather than creating another process, the server creates a new thread to service the request and resumes listening for additional requests.

**Thread Life cycle:**

1. Born State: A thread that has just created.

2. Ready State: The thread is waiting for the processor (CPU).

3. Running: The System assigns the processor to the thread means that the thread is being executed.

4. Blocked State: The thread is waiting for an event to occur or waiting for an I/O device.

5. Sleep: A sleeping thread becomes ready after the designated sleep time expires.

6. Dead: The execution of the thread is finished.