Now since we are able to run JS in our terminal the options are lot we can make

↳ Server side Applications → file system
↳ Desktop " → timer
↳ IOT " → process & runtime variables

I/O operations -

I/O operations in computer system are one of the expensive operation that exist inside our computer system

↳ The implementation of I/O can be of two types

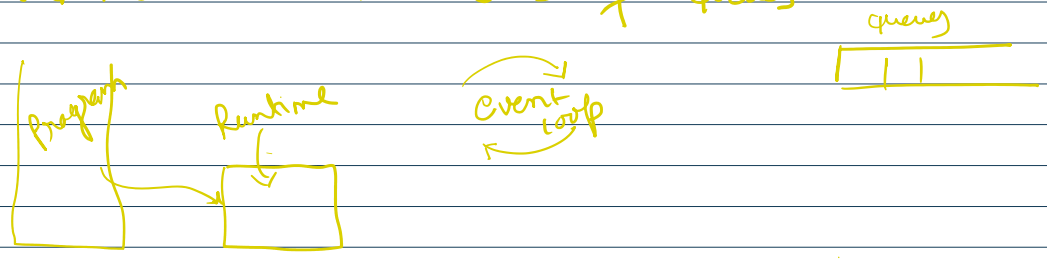
(i) Blocking

(ii) Non Blocking

Blocking I/O means let's say we are executing a process in a particular thread and that process is taking some input from user. Now the entire thread is actually blocked until and unless we are coming back from the user and started executing the remaining steps. The thread will actually wait. So, if we want to do multiple things parallelly we might have to trigger multiple thread together at the same time

whereas it's not the case in NON BLOCKING I/O

Non Blocking I/O handles things in a very different way, we know this from the asynchronous nature of JS. We learned the architecture that there are set of queues

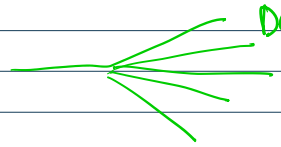


there is something called as event loop, runtime which actually executes our features & from our actual piece of code we trigger the runtime meanwhile behind the scenes the event loop is running if the runtime completes it's features and there is still something going on in the program it's going to passes its callback inside the queue.

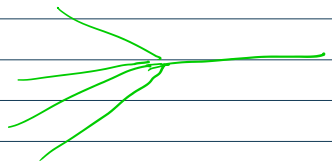
In the same mechanism Nodejs non-blocking I/O works
In Nodejs also there's something called as event loop, there are a bunch of queues that we can maintain. Now actually when we

In Nodejs also there's something called as event loop, there are a bunch of queues that we can maintain. Now actually when we say there is a runtime and inside the runtime we send a signal to execute a feature. Runtime is comprised of a lot of things, it provides a lot of resources to JS. It's not going to execute whatever instruction you have given. So there is something called as **Event Demultiplexer**.

In order to understand event demultiplexer we need to first understand multiplexing demultiplexing.



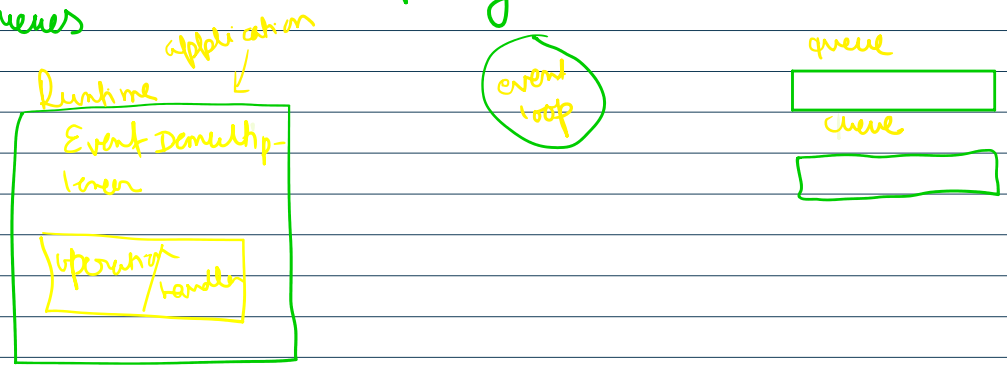
DeMultiplexing : Distributing one signal into multiple signal



Multiplexing : Combining multiple signal into one

All the I/O operation and heavy operations are considered as events. So whenever the end application here in this case JS triggers that it's a runtime feature. What it does is, it actually registers a new entry in the event demultiplexer and if it's a callback based application it also registers the handler inside it and it starts executing the operation.

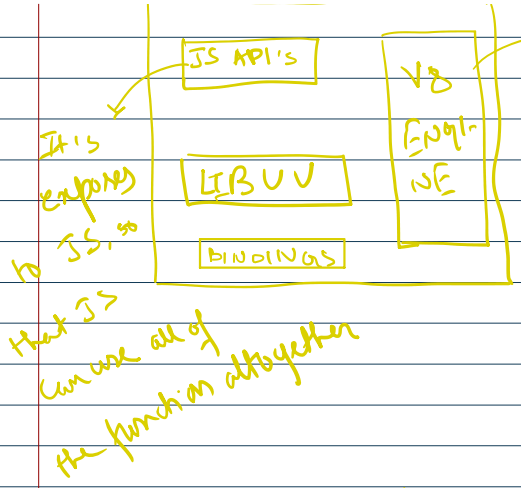
Meanwhile when the operation is done and if the application is still getting executed then it doesn't halt the execution, it just parks whatever was the corresponding handler inside the corresponding queues



Similar thing exist in Nodejs as well. That's what makes Nodejs non-blocking. So, if we are writing our services inside nodejs we will be having nonblocking architecture altogether.

What NODE.JS IS MADE UP OF ?

It is a JS engine



developed by google for responsible
This V8 engine is actually responsible for understanding what JS syntax we are running, and running JS in the corresponding runtime

LIBUV – library prepared by the NodeJS core team,
- it makes node compatible to all operating systems
- It's written in C++, it's very fast
- So, technically LIBUV represents kind of IO binding. It provi low level IO based implementations in more compatible form for node with respect to any OS.

BINDINGS – Special programs that helps Nodejs to interact with LIBUV library

So, a lot of things together prepare the NodeJS architecture and that's the fun part.

↳ It gets V8 engine from chrome which is one of the fastest running JS engines it works on cutting edge architecture.

↳ Provides a lot of API's to JS so that JS can leverage OS based feature.

↳ We have LIBUV, that actually works as an interface between the Nodejs and different OS and giving a consistent setup of APIs to Nodejs to use despite what operating system the user is working on.

↳ And there are a set of bindings that actually are set of program that can help Nodejs runtime to actually interact with LIBUV.

So all of these things together actually help us to prepare a runtime like NODE JS

TO Get Started in NODE.JS

→ Globals

→ Create our own module

→ Module Pattern → use 3rd party

→

→ Streams

☆ Globals in Node.JS –

Global variable are available in every part of code in Node.js runtime. These are important for a lot of features. Some of them are

→ process global : if you want to access the processes running, the environment variable for that process.

→ __dirname : Using this we can actually access the current directory we are trying to execute the process.

→ require : helps us to require other modules

→ global

→ module

Note : → There will be cases when some of the globals will not be available

→ Never try to update your global object,