Topic 3 Asynchronous Programming

Today's lesson will use callbacks for asynchronous programming. This topic is important because it relates to other topics or network theory about transport and applications. Because of network latency, some information would make our network disconnect and never get the job done, so asynchronous programming is what we need.

- Libuv: It is the multi-platform C library that provides the third construct event queue when we use JavaScript.
  - An asynchronous invocation gives a task the right to continue executing the task without waiting for a response from the previous task, either read or invoked.
  - When Libuv accepts an asynchronous task, if the task needs to use CPU or IO related programs, it places the task in the task table.
    - When a CPU task is needed, the worker thread accepts the task when it is idle.
    - When I/O receives a task, the event cycle polls at a specified time.
  - The asynchronous API methods use the function callback, which tells the result of the asynchronous call.
- DNS resolution: it is a network service that can change human-readable areas into machine-readable IP addresses. The C-ARES library provides DNS support using the C language. （Will be learned later in the semester）
- Multiple synchronization tasks: It means that each subsequent task must wait for a response from the previous task
  - It moves after_function to the event queue, and the event loop periodically checks V8 to see if its stack is empty, but next_task has not yet completed, so V8 is not empty.(Result is Undefined)
- Continuation Passing Style: Functions are executed in coded order, Mars functions are not executed until Venus is complete and records is added. The resulting Venus IP address is always printed before Mars.
- N Domains Concurrently(Loop): In most Java or C languages, there is no difference between setting i in a for loop and setting i in a for loop peripheral.
  - This step does not apply to asynchronous code, which does not have to set the variable i outside the loop. The result prints FINISHED and prints 5 five times.
  - When we define i in the loop with let, it not only prints FINISHED but also 0-4.
- N Domains Synchronously(Recursion): We can't send too many requests to DNS on the first row, 6 or 7 times is allowed, but for example 100 or 1000 times is not allowed, because DNS has a rate limit to prevent server attacks.
  - Going from Synchronous to Concurrent: Create a file with numbers of domains, read the data, convert each line of data to an IP address and print it to the console.
  - Multiple fields will not guarantee order.
  - The functions in code will read into the system, but the instructions will not wait for

    completion.

- Going from Concurrent to Synchronous: A set of valid fields must be successfully converted before moving on to the next function.
- Compressing Data with deflate: Compression makes the total number of bytes in the file smaller, but the resulting information is the same because it's a way of re-encoding the information to make it smaller in bytes. This process takes a lot of time because it is CPU intensive. The API is designed to be asynchronous.
    - It is written by putting functions inside functions and accessing one function from another.
    - The downside of this approach is that it makes the code extremely complex to use function after function.
- Decompress data using Inflate: By reading a binary file, encoding NULL will be used, and when the data is binary, opening it will only get some garbage characters. Therefore, with NULL encoding, the system uncompresses the file with zlib.inflate after moving it into memory. When you unzip it, you get a buffer, which you can convert to a string.