***Afzal Ahmed Assignment 2***

***IOT046458 Batch 2, Quarter 3***

***Class Time: 1:30 – 3:30***

**Asynchronous Rust**

*Summary of the article written by Thomas Heartman­­­­­­­­­*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Default environment of the Rust programming language is synchronous. In synchronous programming operation a task blocks instructions until it is completed. Thomas Heartman and the Rust community is excited about the new functionality being stabilized which is called Asynchronous Rust or simply Async Rust. It differs from synchronous Rust, because in Async rust operations may continue without blocking other operations.

Another difference between them is that the Async operations are completed by kick starting or firing an event. Async Rust may be defined as concurrent programming in which multiple computations run at the same time in single thread. Remember concurrent programming in Async rust is different from multithreading. These two have different use cases. For example, multithreaded programming is better suited for CPU bound tasks which can be allocated to multiple processor cores. On the other hand, the use case of Async or concurrent programming is for IO bound tasks, in which a task waits a response from a server and in the mean while does not block other tasks while performing multiple IO bound tasks.

However, if we want the result of some Async operation we must .await it. The value returned by such operations are called futures (futures are traits).

Standard library in Rust does not have an executor, which is required for Async programming. The solution to this problem is the external crate shown below which provides executor

**async\_std::task**

another such crate is “**tokio**”

The syntax to declare Async functions is shown below

Async fn f() -> i32 {

// Body of function

}

main function is not Async in Rust. See the code below

fn main() {

task :: block\_on(execute());

}

In the code above main fires executes function and then waits for it to finish.

task :: block\_on is the synchronous function that starts an Async operation and blocks until it is finished. Why it has been used because main() itself is not Async and can not use .await which is a feature of Async programming. In short, we cant use .await in main but can block Async operations using block\_on() function.

Async Rust looks an exciting addition to the Rust’s powers, but you need to think of its use case before implanting. It is sitable for single threaded IO based application where a task waits for something (a response from a remote serve) and in the mean while continues other operations.