Assignment 6 – Multithreaded Web Server Using Rust

By – Viveka Kumar (J01258735)

Objective:

To implement a basic yet functional **multithreaded web server** in Rust, which demonstrates:

- TCP Socket Programming
- Basic HTTP parsing
- Serving static HTML pages
- Managing concurrency using a **custom thread pool**
- Graceful shutdown and cleanup of threads

Steps followed and features implemented

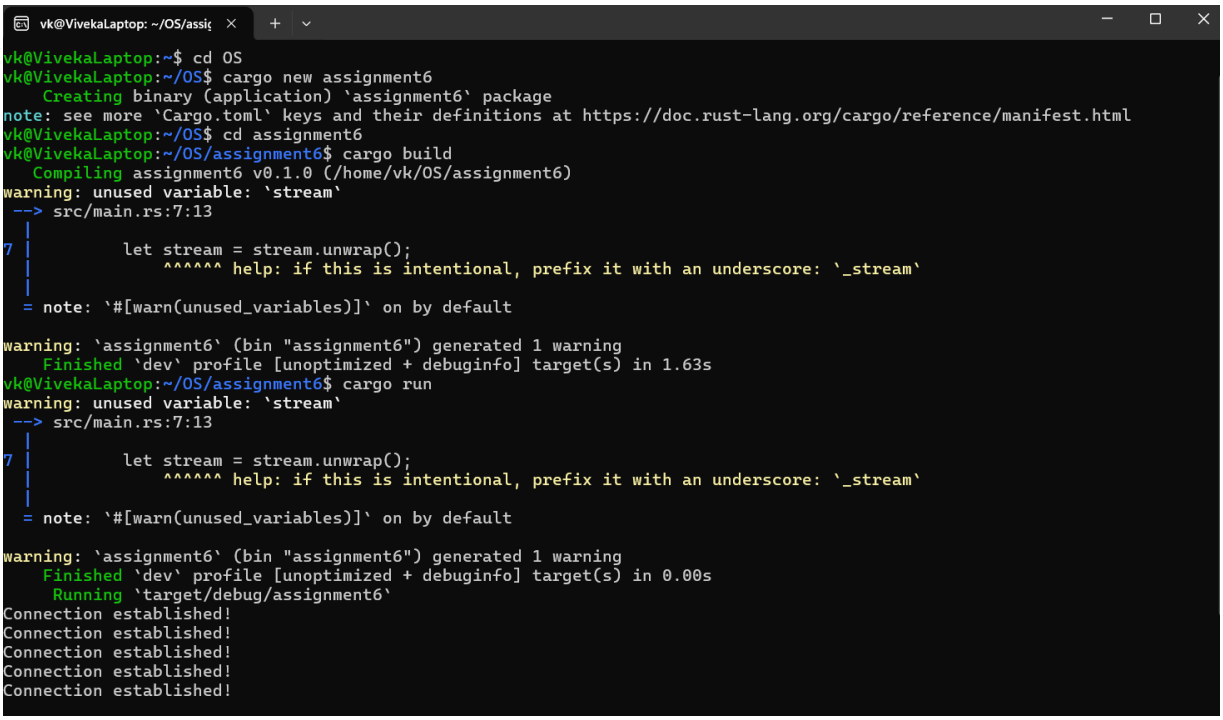
Creating a new project as in rust as – "assignment6 "

- Listening to TCP Port - Adding the **TCPListener** that will listen to at the local address 127.0.0.1:7878 for incoming TCP streams in main.rs file

Code line:

```
let listener = TcpListener::bind("127.0.0.1:7878").unwrap();
```

Output:



- Accepting incoming connections and handling them in a thread pool:
  Code line:

```
for stream in listener.incoming().take(2) {
    let stream = stream.unwrap();

    pool.execute(|| {
        handle_connection(stream);
    });
}
```

- Reading the HTTPS Request
  Code Lines:

```
let buf_reader = BufReader::new(&stream);
let request_line = buf_reader.lines().next().unwrap().unwrap();
```

- Now we are Returning HTTP Responses, using the routing logic on URL & writing the response to the stream (stream.write_all)

  Code Lines:

```
let (status_line, filename) = match &request_line[..] {
    "GET / HTTP/1.1" => ("HTTP/1.1 200 OK", "hello.html"),
    "GET /sleep HTTP/1.1" => {
        thread::sleep(Duration::from_secs(5));
        ("HTTP/1.1 200 OK", "hello.html")
    }
    _ => ("HTTP/1.1 404 NOT FOUND", "404.html"),
};
```

```
let contents = fs::read_to_string(filename).unwrap();
let length = contents.len();

let response =
    format!("{status_line}\r\nContent-Length: {length}\r\n\r\n{contents}");

stream.write_all(response.as_bytes()).unwrap();
```

- Thread Pool Implementation in “ lib.rs ” – using “impl Treadpool”

```rust
impl ThreadPool {
    /// Create a new ThreadPool.
    ///
    /// The size is the number of threads in the pool.
    ///
    /// # Panics
    ///
    /// The `new` function will panic if the size is zero.
    pub fn new(size: usize) -> ThreadPool {
        assert!(size > 0);

        let (sender, receiver) = mpsc::channel();
        let receiver = Arc::new(Mutex::new(receiver));

        let mut workers = Vec::with_capacity(size);

        for id in 0..size {
            workers.push(Worker::new(id, Arc::clone(&receiver
        }

        ThreadPool {
            workers,
            sender: Some(sender),
        }
    }

    pub fn execute<F>(&self, f: F)
    where
        F: FnOnce() + Send + 'static,
    {
        let job = Box::new(f);
        self.sender.as_ref().unwrap().send(job).unwrap();
    }
}
```

- Implementing **Graceful Shutdown of Threads** in lib.rs :

```rust
impl Drop for ThreadPool {
    fn drop(&mut self) {
        drop(self.sender.take());

        for worker in &mut self.workers {
            println!("Shutting down worker {}", worker.id);

            if let Some(thread) = worker.thread.take() {
                thread.join().unwrap();
            }
        }
    }
}
```

- Worker thread managing sender and exiting loop when needed:

```rust
impl Worker {
    fn new(id: usize, receiver: Arc<Mutex<mpsc::Receiver<Job>>>) -> Worker {
        let thread = thread::spawn(move || loop {
            let message = receiver.lock().unwrap().recv();

            match message {
                Ok(job) => {
                    println!("Worker {id} got a job; executing.");
                    job();
                }
                Err(_) => {
                    println!("Worker {id} disconnected; shutting down.");
                    break;
                }
            }
        });
```

Terminal Output:

```
vk@VivekaLaptop:~/OS/assignment6$ cargo build
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.03s
vk@VivekaLaptop:~/OS/assignment6$ cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/assignment6`
Worker 0 got a job; executing.
Shutting down.
Shutting down worker 0
Worker 1 got a job; executing.
Worker 2 disconnected; shutting down.
Worker 3 disconnected; shutting down.
Worker 0 disconnected; shutting down.
Shutting down worker 1
thread '<unnamed>' panicked at src/main.rs:28:50:
called `Option::unwrap()` on a `None` value
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
thread 'main' panicked at src/lib.rs:57:31:
called `Result::unwrap()` on an `Err` value: Any { .. }
```

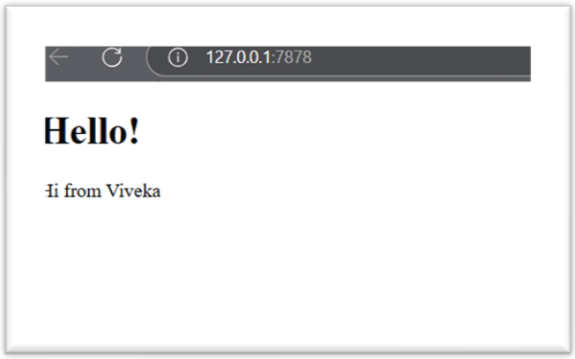
Web Results

- **Display page Code – hello.html**

hello.html > ...

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Hello!</title>
  </head>
  <body>
    <h1>Hello!</h1>
    <p>Hi from Viveka </p>
  </body>
</html>
```

Output:

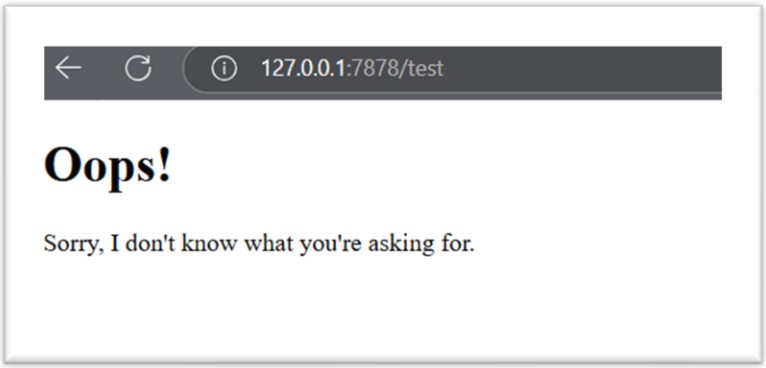


- **Error message Display – 404.html**

Code:

```
404.html > ...
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8">
5       <title>Not Found</title>
6     </head>
7     <body>
8       <h1>Oops!</h1>
9       <p>Sorry, I don't know what you're asking for.</p>
10    </body>
11  </html>
```

Output:



Appendix:


main.rs


lib.rs


hello.html


404.html