600086 Lab Book

Week 2 – Lab B

Date: 12th Feb 2022

Q1. First threads

Question:

Replace the synchronous call to your function with an asynchronous call.

Solution:

```
fn main()
{
    std::thread::spawn(move || my_function() );
    std::thread::spawn(move || second_function() );
    std::thread::sleep(std::time::Duration::new(1,0));
}

fn my_function()
{
    println!("Hello, world!");
}

fn second_function()
{
    println!("Sneaky sneaky")
}
```

Test data:

n/a

Sample output:

```
PS C:\Temp\600086-wjv-lab-b-zaryc0\first_thread> cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
     Running `target\debug\first_thread.exe`
Hello, world!
Sneaky sneaky
```

Reflection:

This is threading 101. Adding the Time delay thread to the function is necessary to alleviate the race condition created by spawning asynchronous threads with no other content to the program causing it to terminate before all of the thread shave a chance to complete.

Metadata:

Threads

Further information:

Unsure of the use of mut?

Q2. Joining threads

Question:

Replace the synchronous call to your function with an asynchronous call.

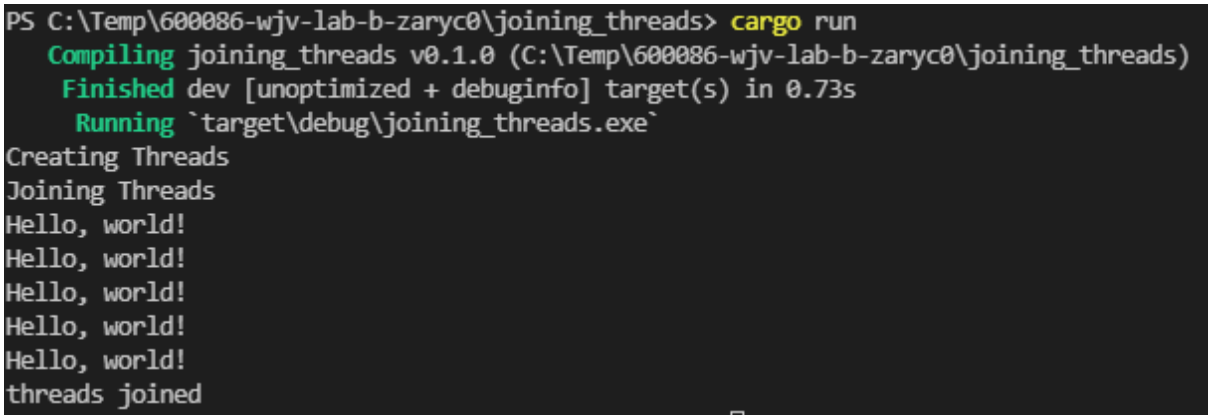
Solution:

```
fn main()
{
    let num_of_threads : i32 = 5;
    let mut list_of_threads = vec!();

    println!("Creating Threads");

    for _id in 0..num_of_threads
    {
        list_of_threads.push(std::thread::spawn(move || my_function() ));
    }

    println!("Joining Threads");
    for thread in list_of_threads
    {
        thread.join().expect("join failed");
    }
    println!("threads joined");
}

fn my_function()
{
    println!("Hello, world!");
}
```

Test data:

n/a

Sample output:



Reflection:

This is joining the threads back together again it resynchronizing them ensuring that all of the threads have completed their tasks before ending the program this helps alleviate the race condition from the Q1 solution where the threads have to race against the main program to execute their functions.

Metadata:

Threads

Further information:

Q3 Experimentation

Question:

Now that you have the basic framework for creating and joining threads, experiment with giving the threads items of work, as well as altering the number of threads used.

Solution:

```rust
fn main()
{
    let num_of_threads : i32 = 12;
    let mut list_of_threads = vec!();

    println!("Creating Threads");

    for _id in 0..num_of_threads
    {
        list_of_threads.push(std::thread::spawn(move || perform_task(_id) ));
    }

    println!("Joining Threads");
    for thread in list_of_threads
    {
        thread.join().expect("join failed");
    }
    println!("threads joined");
}

fn perform_task(id:i32)
{
    let result:i32 = id * id;
    println!("Thread: {} Result is {}", id,result);
}
```

Test data:

n/a

Sample output:

```
PS C:\Temp\600086-wjv-lab-b-zaryc0\joining_threads> cargo run
   Compiling joining_threads v0.1.0 (C:\Temp\600086-wjv-lab-b-zaryc0\joining_threads)
    Finished dev [unoptimized + debuginfo] target(s) in 0.68s
     Running `target\debug\joining_threads.exe`
Creating Threads
Thread: 0 Result is 0
Thread: 1 Result is 1
Thread: 2 Result is 4
Thread: 3 Result is 9
Joining Threads
Thread: 5 Result is 25
Thread: 6 Result is 36
Thread: 7 Result is 49
Thread: 8 Result is 64
Thread: 9 Result is 81
Thread: 4 Result is 16
Thread: 11 Result is 121
Thread: 10 Result is 100
threads joined
```

Reflection:

I added a task to square the ID value as part of the thread function it then printed the result to the terminal along with the thread ID.

Metadata:

Threads

Further information: