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:

### 

### 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:

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:

### 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: