600086 Lab Book

# Week 6 – Lab H

Date: 24th Mar 2022

## Exercise 1. Condition variables

### Questions:

1. create a rust program using a producer consumer architecture with condition variables to prevent race conditions
2. modify the code to allow the code to work with multiple consumers

### Solution:

use std::sync::{Arc, Mutex, Condvar};

pub fn main()

{

    println!("Begin");

    let D = Arc::new(Data::new());

    let D\_clone = D.clone();

    let loops = 5;

    let producers = std::thread::spawn(move || producer\_main( &D,loops));

    let consumers = std::thread::spawn(move || consumer\_main(&D\_clone, loops));

    producers.join();

    consumers.join();

    println!("Cease");

}

pub fn producer\_main(data : & Data, loop\_limit : u32)

{

    let data = data;

    for \_i in 0..loop\_limit

    {

        let mut full = data.value.lock().unwrap();

        while \*full

        {

            full = data.condition.wait(full).unwrap();

        }

        //"produce" by setting the mutex value to true

        \*full = true;

        println!("Producer\_ID: {} has produced",\_i);

        data.condition.notify\_all(); // notify any waiting threads

    }

}

pub fn consumer\_main(data : & Data,loop\_limit : u32)

{

    for \_i in 0..loop\_limit

    {

        let mut full = data.value.lock().unwrap();

        while !\*full // de reference the guard variable

        {

            full = data.condition.wait(full).unwrap();

        }

        \*full = false;

        println!("Consumer\_ID: {} has consumed",\_i);

        data.condition.notify\_all();

    }

}

pub struct Data

{

    condition : Condvar,

    value : Mutex<bool>

}

impl Data

{

    pub fn new() -> Data

    {

        Data

        {

            condition : Condvar::new(),

            value : Mutex::new(false)

        }

    }

}

The above code implements a struct called Data which has a CondVal and a mutex Boolean value the Mutex bool is the value of the data and the CondVal is there to organise the access to the variable

When run it produces the output shown in the sample output section ref 1

1. To run the code with multiple consumers and producers I created a thread\_pool variable and pushed multiple threads of each kind to create multiple consumer and producer threads.

pub fn main()

{

    println!("Begin");

    let D = Arc::new(Data::new());

    let mut thread\_pool = Vec::new();

    let loops = 5;

    let num\_producers = 2;

    let num\_consumers = 2;

    for \_i in 0..num\_producers

    {

        let d\_clone = D.clone();

        thread\_pool.push(std::thread::spawn(move || producer\_main(&d\_clone, loops)));

    }

    for \_i in 0..num\_consumers

    {

        let d\_clone = D.clone();

        thread\_pool.push(std::thread::spawn(move || consumer\_main(&d\_clone, loops)));

    }

    for t in thread\_pool

    {

        t.join();

    }

    println!("Cease");

}

### Test data:

N/A

### Sample output:

|  |  |
| --- | --- |
| ref | output |
| 1 | Text  Description automatically generated |
| 2 | Text  Description automatically generated |

### Reflection:

Part one I produced a producer consumer structured program, part 2 I scaled this program to work with multiple consumers and producers however this is not a massively scalable solution as there is still only one piece of data being accessed meaning it bottlenecks the parallel program and makes it very concurrent slightly defeating the purpose.

### Metadata:

Produce, consumer, architecture, Condvar

### Further information:

N/A

## Q2. Striped Arrays

### Question:

1. Add timing code to the provided program to measure the data access time, vary the number of threads to check how this changes the value
2. Modify the code to simulate random access

### Solution:

1. Added a start\_time and end\_time variable to the code that wraps the thread creation and joining to time the data access portion of the programme I also modified the strip sizes so that it is the total divided by the number of threads so that the same number of accesses are made.

A screenshot of a computer

Description automatically generated

1. To simulate random access I implemented the following code as per the lab notes in the thread\_main

Text

Description automatically generated

### Test data:

N/A

### Sample output:

|  |  |  |
| --- | --- | --- |
| No. of threads | Output - seq | Output - rand |
| 1 |  |  |
| 2 |  |  |
| 4 |  |  |
| 8 |  |  |
| 16 |  |  |
| 32 |  |  |

### Reflection:

In the sequential implementation the run time gets greater every time as the amount of work being done is very small compared to the overhead of spinning up and joining threads.

When implementing the random-access code. Expected that the code would have roughly the same access time however on average the access time was roughly double that of the sequential implementation

### Metadata:

Stripped Array

### Further information:

N/A