Lab

Pre-Lab Task

- Ran prelab task

```
➜ PRAC04 git:(master) X gcc hello.c -o hello -lpthread
➜ PRAC04 git:(master) X ./hello

 Created new thread (140429824038656) ...
Hello from new thread - got 11
```

Practical Task

Task 1

```
➜ PRAC04 git:(master) X gcc hello1.c -o hello1 -lpthread
➜ PRAC04 git:(master) X ./hello1
I am thread 140325348751168 Created new thread (140325348747008) ...
Hello from new thread (140325348747008)- got 140325348751168
```

Task 2

```
➜ PRAC04 git:(master) X gcc hello2.c -o hello2 -lpthread
➜ PRAC04 git:(master) X ./hello2
I am thread 140708952246080 Created new thread (140708952241920) ...
➜ PRAC04 git:(master) X
```

- The main thread did not have pthread_exit() and instead used the implicit exit

```
To allow other threads to continue execution, the main thread should
terminate by calling pthread_exit() rather than exit(3).
```

- Considering the delay too, this meant that the second thread did not have enough time to execute before being terminated

```
➜ PRAC04 git:(master) X gcc hello2.c -o hello2 -lpthread
➜ PRAC04 git:(master) X ./hello2
Hello from new thread (139826370205440)- got 139826370209600
I am thread 139826370209600 Created new thread (139826370205440) ...
➜ PRAC04 git:(master) X
```

- In this case, the sleep causes the second thread to execute the print statement and terminate before the main thread gets to print and terminate.

Task 3

```
➜ PRAC04 git:(master) X gcc hello3.c -o hello3 -lpthread
➜ PRAC04 git:(master) X ./hello3

 Created new thread (140619010905856) ...
Hello from new thread 140619010905856 - got 140619010910016
➜ PRAC04 git:(master) X
```

- Yes, the output is what I expected. The second thread calls pthread_join, passing in the master thread's ID. This means that the second thread waits for the main thread to run its print statement and terminate, before running its own print statement and terminating.

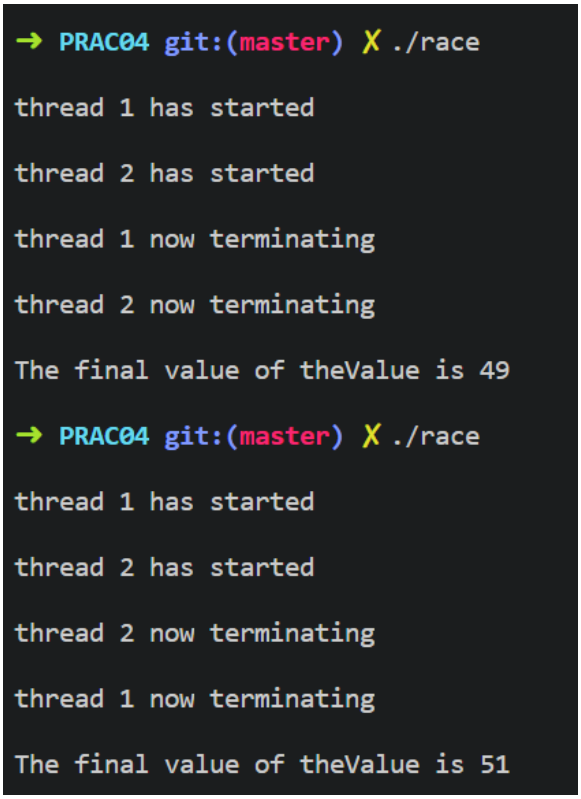
Task 4

```
➜ PRAC04 git:(master) X ./hellomany 12
I am thread 140679021758272. Created new thread 140679021754112 in iteration 0
Hello from thread 140679021754112. I was created in iteration 0
I am thread 140679021758272. Created new thread 140679013361408 in iteration 1
Hello from thread 140679013361408. I was created in iteration 1
I am thread 140679021758272. Created new thread 140679004968704 in iteration 2
Hello from thread 140679004968704. I was created in iteration 2
I am thread 140679021758272. Created new thread 140678996465408 in iteration 3
Hello from thread 140678996465408. I was created in iteration 3
I am thread 140679021758272. Created new thread 140678996465408 in iteration 4
I am thread 140679021758272. Created new thread 140679004968704 in iteration 5
Hello from thread 140679004968704. I was created in iteration 5
Hello from thread 140678996465408. I was created in iteration 4
Hello from thread 140679013361408. I was created in iteration 6
I am thread 140679021758272. Created new thread 140679013361408 in iteration 6
I am thread 140679021758272. Created new thread 140678996465408 in iteration 7
Hello from thread 140678996465408. I was created in iteration 7
I am thread 140679021758272. Created new thread 140678996465408 in iteration 8
I am thread 140679021758272. Created new thread 140679013361408 in iteration 9
Hello from thread 140678996465408. I was created in iteration 8
Hello from thread 140679013361408. I was created in iteration 9
Hello from thread 140679004968704. I was created in iteration 10
I am thread 140679021758272. Created new thread 140679004968704 in iteration 10
I am thread 140679021758272. Created new thread 140679004968704 in iteration 11
Hello from thread 140679004968704. I was created in iteration 11
```

Pre-class exercise

```
➜ PRAC04 git:(master) X ./thread

thread 1 is executing

thread 2 has started

thread 1 now terminating

thread 2 now terminating

➜ PRAC04 git:(master) X ./thread

thread 1 is executing

thread 2 has started

thread 2 now terminating

thread 1 now terminating
```

Task 5



Task 6

- Mutex is a locking mechanism that allows only one thread to access a resource, given that they have the mutex. The thread then releases the mutex when it is done.
- Semaphore is a signalling mechanism that has wait and signal operations that allows multiple threads to access the same critical section.

```
pthread_mutex_init: Initialises the mutex, being able to pass in custom attributes
```

```
pthread_mutex_lock: Locks a given mutex object
```

```
pthread_mutex_unlock: Unlocks a given mutex object
```

```
sem_init: Initialises an unnamed semaphore
```

```
sem_wait: Locks a given semaphore value (by decreasing the value)
```

```
sem_post: Unlocks a given semaphore value (by incrementing it)
```

Mutex Solution

```
➜ PRAC04 git:(master) X ./race_mutex

thread 1 has started

thread 1 now terminating

thread 2 has started

thread 2 now terminating

The final value of theValue is 50
```

- Both threads are locked and unlocked using Pthread MUTEX methods

Semaphore Solution

```
➜ PRAC04 git:(master) X ./race_semaphore

thread 1 has started

thread 1 now terminating

thread 2 has started

thread 2 now terminating

The final value of theValue is 50
```

There are not many differences between the actual implementation, besides the fact that semaphores use a value to increment and decrement and mutex is a single object.

Semaphores can have variable values where the count is the number of available resources, this allows more granularity in choosing which resources to restrict.