

CSCI 5103 - OPERATING SYSTEMS

PROGRAMMING ASSIGNMENT-1

DESIGN DOCUMENT

GROUP MEMBERS:

Sai Pratyusha Attanti (*attan005@umn.edu, Student ID: 5656785*)

Navya Ganta (*ganta016@umn.edu, Student ID: 5673223*)

In this project, we implemented the user thread library that provides API for different thread functions.

1. `uthread_init`: This function creates the main thread and assigns the quantum to it. And also setups the signal handler.
2. `uthread_create`: This function helps us create new threads and assign a task to them. When a thread is created its state is set to READY and is pushed into the ready queue and waits for the scheduler.
3. `uthread_yield`: `uthread_yield` is used for the context switching of the threads. When the quantum expires this function removes the current thread from the running state, moves it back to the ready queue, and assigns the scheduler to the next available thread.
4. `uthread_join`: This function blocks the threads from which it is called until the thread specified in the join call is terminated.
5. `uthread_exit`: When the current thread finishes its execution this function is called. In this function, the state of the thread is set to finished and it is pushed to the finished queue.
6. `uthread_suspend`: This function blocks the specified thread and is pushed to the block queue.
7. `Uthread_resume`: This function resumes the blocked thread and moves it back to the ready queue from the block queue.

Scheduling: For scheduling the threads we implemented Round-robin scheduling algorithm where each thread is assigned a quantum for fair scheduling. Once the quantum expires the currently executing thread is preempted and added at the end of the ready queue and the next available thread in the ready queue is assigned to the processor.

When the thread is preempted its context is stored using the `get_context()` function and the context of the next thread is loaded using the `set_context()` and the execution will start from the previously stored state of this thread.

Design Decisions:

A. What (additional) assumptions did you make?

1. We assumed that the system is a single processor and can only run one thread at a time.
2. And we assumed that the number of threads that will be created will always be below the system's memory and also the number of threads created will be less than 100.
3. We designed our scheduling algorithm assuming that SIGVTALRM is the only signal to interrupt the threads.
4. Also, there will not be any deadline situations
5. Each thread will have the same quantum assigned to it.

B. Descriptions of test cases you provide and how to use them to test your implementation, e.g. what's the purpose of each test case, what specific functionality does each test case focus on, what is the expected output of each test case, etc.?

In addition to the test cases provided by the instructor, we also added a few additional test cases for testing the functionality of our UTL functions. **We added these test cases to the test.cpp file.**

For testing the test cases:

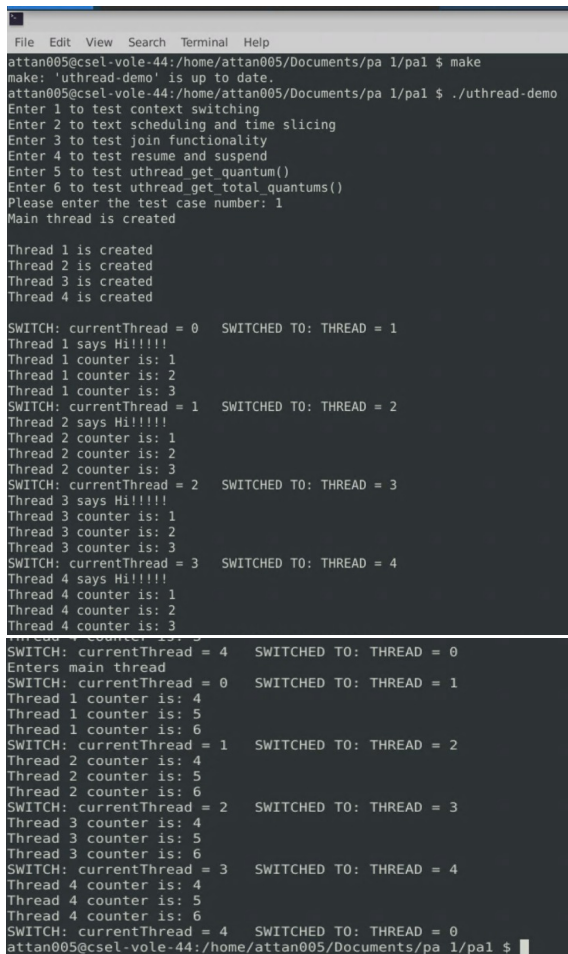
1. If you want to run test.cpp file, update the main.o with test.o in the Makefile.
2. Run ``make`` to compile the program
3. For running test case file test.cpp
``./uthread-demo``

Running this command will prompt us to enter the test case number we want to run. Please number the desired test case number.

a. Test case 1:

This test case tests the **context-switching** functionality and also helps us in verifying if the context of the thread is saved and loaded correctly.

For this purpose, we created 4 threads and each thread should print numbers from 1 to 6.



```
File Edit View Search Terminal Help
attan005@cseel-vole-44:/home/attan005/Documents/pa 1/pal $ make
make: 'uthread-demo' is up to date.
attan005@cseel-vole-44:/home/attan005/Documents/pa 1/pal $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get quantum()
Enter 6 to test uthread_get total quantum()
Please enter the test case number: 1
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created

SWITCH: currentThread = 0 SWITCHED TO: THREAD = 1
Thread 1 says Hi!!!!!!
Thread 1 counter is: 1
Thread 1 counter is: 2
Thread 1 counter is: 3
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2
Thread 2 says Hi!!!!!!
Thread 2 counter is: 1
Thread 2 counter is: 2
Thread 2 counter is: 3
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3
Thread 3 says Hi!!!!!!
Thread 3 counter is: 1
Thread 3 counter is: 2
Thread 3 counter is: 3
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4
Thread 4 says Hi!!!!!!
Thread 4 counter is: 1
Thread 4 counter is: 2
Thread 4 counter is: 3
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 0
Enters main thread
SWITCH: currentThread = 0 SWITCHED TO: THREAD = 1
Thread 1 counter is: 4
Thread 1 counter is: 5
Thread 1 counter is: 6
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2
Thread 2 counter is: 4
Thread 2 counter is: 5
Thread 2 counter is: 6
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3
Thread 3 counter is: 4
Thread 3 counter is: 5
Thread 3 counter is: 6
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4
Thread 4 counter is: 4
Thread 4 counter is: 5
Thread 4 counter is: 6
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 0
attan005@cseel-vole-44:/home/attan005/Documents/pa 1/pal $
```

In the above screenshots you can see that the threads are scheduled in the FIFO manner, and once the current thread is preempted its state is stored and resumed from the same state when this thread is scheduled to the processor next time.

b. Test case 2:

In this test case, we are testing the **scheduling and the time-slicing** functionality of the scheduling algorithm.

We are creating 4 threads and each thread should print numbers from 1 to 6. For round robin scheduling algorithm we are assigning a quantum to the threads.

Case 1: Quantum = 100microseconds.

```
-----
Switch: currentThread = 4    SWITCHED TO: THREAD = 5
-----
Testing for Quantum=100 microseconds
Main thread is created

Thread 5 is created
Thread 6 is created
Thread 7 is created
Thread 8 is created

SWITCH: currentThread = 0    SWITCHED TO: THREAD = 5

Thread 5 says Hi!!!!!!
Thread 5 counter is: 1
Thread 5 counter is: 2
SWITCH: currentThread = 5    SWITCHED TO: THREAD = 6

Thread 6 says Hi!!!!!!
Thread 6 counter is: 1
Thread 6 counter is: 2
SWITCH: currentThread = 6    SWITCHED TO: THREAD = 7

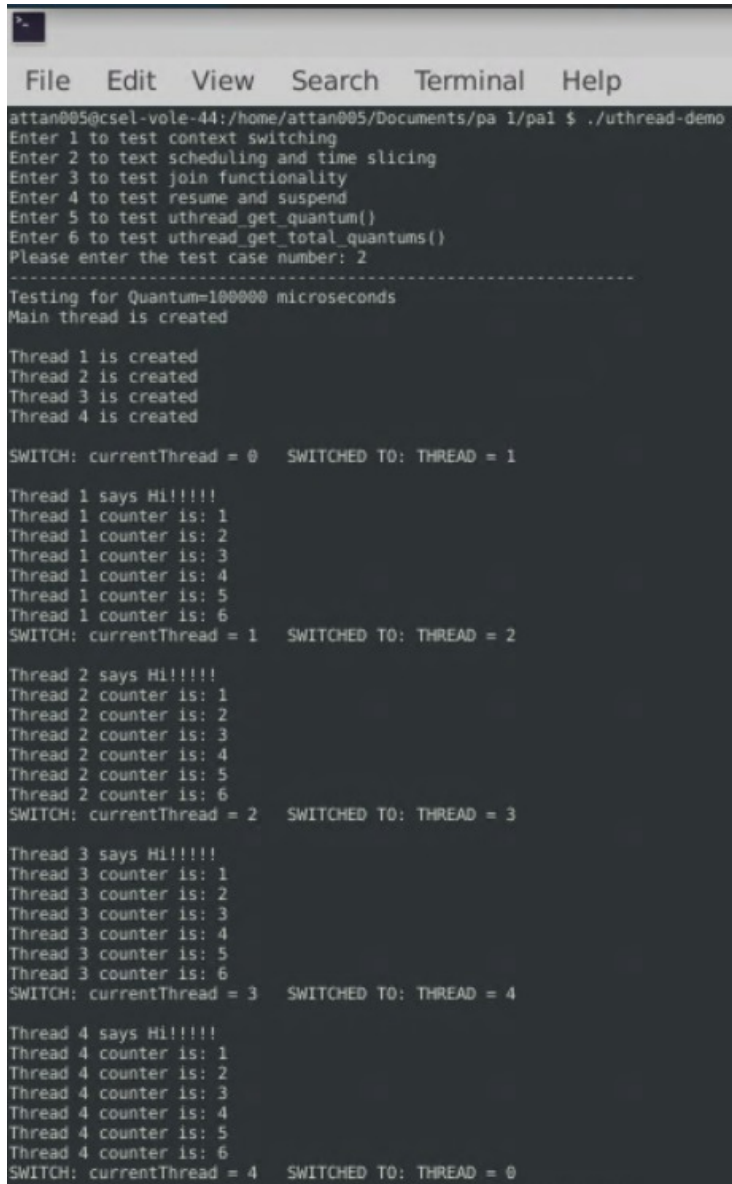
Thread 7 says Hi!!!!!!
Thread 7 counter is: 1
Thread 7 counter is: 2
SWITCH: currentThread = 7    SWITCHED TO: THREAD = 8

Thread 8 says Hi!!!!!!
Thread 8 counter is: 1
Thread 8 counter is: 2
SWITCH: currentThread = 8    SWITCHED TO: THREAD = 5
Thread 5 counter is: 3
SWITCH: currentThread = 5    SWITCHED TO: THREAD = 6
Thread 6 counter is: 3
SWITCH: currentThread = 6    SWITCHED TO: THREAD = 7
Thread 7 counter is: 3
SWITCH: currentThread = 7    SWITCHED TO: THREAD = 8
Thread 8 counter is: 3
SWITCH: currentThread = 8    SWITCHED TO: THREAD = 5
Thread 5 counter is: 4
SWITCH: currentThread = 5    SWITCHED TO: THREAD = 6
Thread 6 counter is: 4
Thread 6 counter is: 5
SWITCH: currentThread = 6    SWITCHED TO: THREAD = 7
Thread 7 counter is: 4
Thread 7 counter is: 5
SWITCH: currentThread = 7    SWITCHED TO: THREAD = 8
Thread 8 counter is: 4
Thread 8 counter is: 5
SWITCH: currentThread = 8    SWITCHED TO: THREAD = 5
Thread 5 counter is: 5
Thread 5 counter is: 6
SWITCH: currentThread = 5    SWITCHED TO: THREAD = 6
Thread 6 counter is: 6
SWITCH: currentThread = 6    SWITCHED TO: THREAD = 7
Thread 7 counter is: 6
SWITCH: currentThread = 7    SWITCHED TO: THREAD = 8
Thread 8 counter is: 6
SWITCH: currentThread = 8    SWITCHED TO: THREAD = 0
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pa1 $
```

In this case, we used a quantum of 100microseconds. So since the quantum is small the threads will be preempted very frequently so more number context switchings. And resumed back from the previously stored state. We can also notice that threads are scheduled in a FIFO manner, i.e once the thread is

preempted it is added to the end of the ready queue, and the thread in the front of the ready queue is assigned to the processor.

Case 2: Quantum = 100000 microseconds.



```
File Edit View Search Terminal Help
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pa1 $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get_quantum()
Enter 6 to test uthread_get_total_quantums()
Please enter the test case number: 2
-----
Testing for Quantum=100000 microseconds
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created

SWITCH: currentThread = 0   SWITCHED TO: THREAD = 1

Thread 1 says Hi!!!!!!
Thread 1 counter is: 1
Thread 1 counter is: 2
Thread 1 counter is: 3
Thread 1 counter is: 4
Thread 1 counter is: 5
Thread 1 counter is: 6
SWITCH: currentThread = 1   SWITCHED TO: THREAD = 2

Thread 2 says Hi!!!!!!
Thread 2 counter is: 1
Thread 2 counter is: 2
Thread 2 counter is: 3
Thread 2 counter is: 4
Thread 2 counter is: 5
Thread 2 counter is: 6
SWITCH: currentThread = 2   SWITCHED TO: THREAD = 3

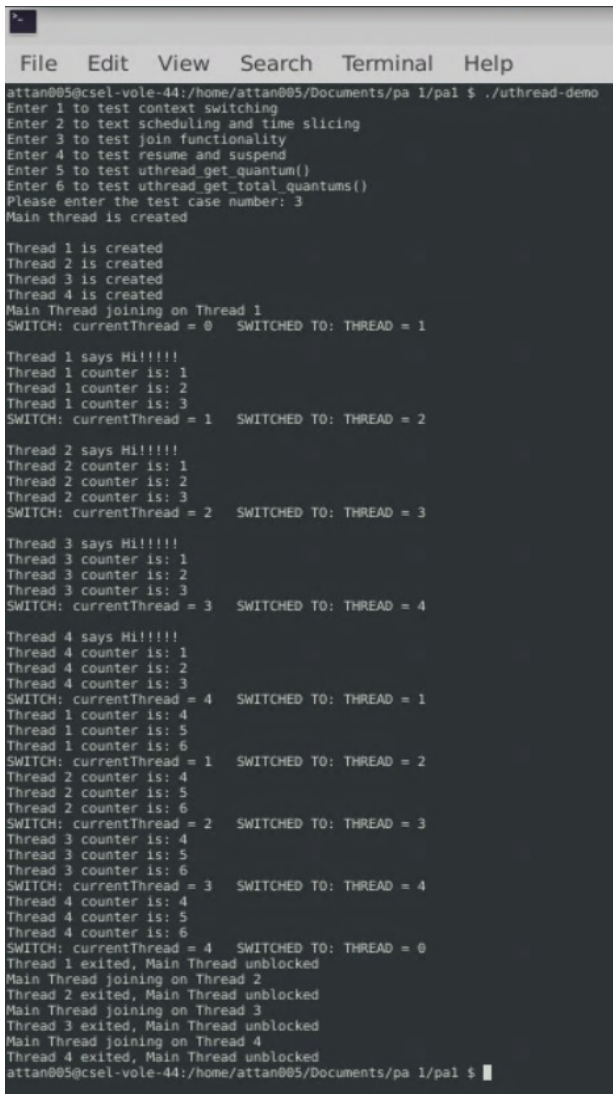
Thread 3 says Hi!!!!!!
Thread 3 counter is: 1
Thread 3 counter is: 2
Thread 3 counter is: 3
Thread 3 counter is: 4
Thread 3 counter is: 5
Thread 3 counter is: 6
SWITCH: currentThread = 3   SWITCHED TO: THREAD = 4

Thread 4 says Hi!!!!!!
Thread 4 counter is: 1
Thread 4 counter is: 2
Thread 4 counter is: 3
Thread 4 counter is: 4
Thread 4 counter is: 5
Thread 4 counter is: 6
SWITCH: currentThread = 4   SWITCHED TO: THREAD = 0
```

In this case, we used a quantum of 100000microseconds, Since the quantum is very large each thread will each more execution time so less number of context switchings. As we can see in the above screenshot, in this case each thread is completing the whole task in one go because of more quantum time.

c. Test case 3:

This test case is testing our **uthread_join** functionality. The main thread should wait till the execution of other threads. Even in this case, 4 threads are created and each thread should print numbers 1 to 6. Here main thread call join and waits for the execution of each of the other threads.



```
File Edit View Search Terminal Help
attan005@csele-vole-44:/home/attan005/Documents/pa 1/pa1 $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get_quantum()
Enter 6 to test uthread_get_total_quantums()
Please enter the test case number: 3
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created
Main Thread joining on Thread 1
SWITCH: currentThread = 0 SWITCHED TO: THREAD = 1

Thread 1 says HI!!!!
Thread 1 counter is: 1
Thread 1 counter is: 2
Thread 1 counter is: 3
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2

Thread 2 says HI!!!!
Thread 2 counter is: 1
Thread 2 counter is: 2
Thread 2 counter is: 3
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3

Thread 3 says HI!!!!
Thread 3 counter is: 1
Thread 3 counter is: 2
Thread 3 counter is: 3
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4

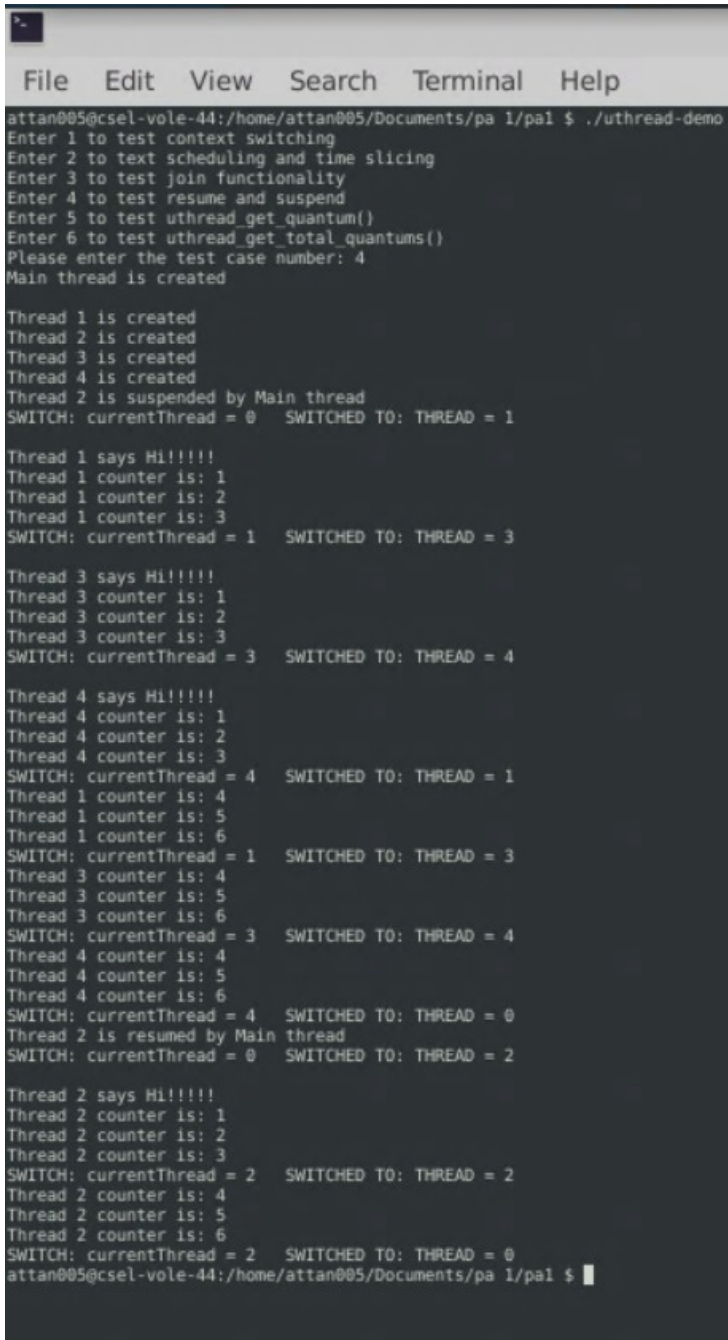
Thread 4 says HI!!!!
Thread 4 counter is: 1
Thread 4 counter is: 2
Thread 4 counter is: 3
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 1
Thread 1 counter is: 4
Thread 1 counter is: 5
Thread 1 counter is: 6
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2
Thread 2 counter is: 4
Thread 2 counter is: 5
Thread 2 counter is: 6
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3
Thread 3 counter is: 4
Thread 3 counter is: 5
Thread 3 counter is: 6
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4
Thread 4 counter is: 4
Thread 4 counter is: 5
Thread 4 counter is: 6
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 0
Thread 1 exited, Main Thread unblocked
Main Thread joining on Thread 2
Thread 2 exited, Main Thread unblocked
Main Thread joining on Thread 3
Thread 3 exited, Main Thread unblocked
Main Thread joining on Thread 4
Thread 4 exited, Main Thread unblocked
attan005@csele-vole-44:/home/attan005/Documents/pa 1/pa1 $
```

In the above screenshot, you can see that thread-0 (Main thread) is been blocked until thread-1 finished its execution. And then join back the ready queue after thread-1 finished its execution. In a similar way, thread-0 also blocks until other threads are completed.

d. Test case 4:

This test case tests the **suspend resume** functionality. Even in this case, 4 threads are created and each thread should print numbers 1 to 6.

When suspend is called on a specific thread it is blocked and sent to the block queue and will be in the same state until it is resumed.



```
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pa1 $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get_quantum()
Enter 6 to test uthread_get_total_quantums()
Please enter the test case number: 4
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created
Thread 2 is suspended by Main thread
SWITCH: currentThread = 0   SWITCHED TO: THREAD = 1

Thread 1 says Hi!!!!!!
Thread 1 counter is: 1
Thread 1 counter is: 2
Thread 1 counter is: 3
SWITCH: currentThread = 1   SWITCHED TO: THREAD = 3

Thread 3 says Hi!!!!!!
Thread 3 counter is: 1
Thread 3 counter is: 2
Thread 3 counter is: 3
SWITCH: currentThread = 3   SWITCHED TO: THREAD = 4

Thread 4 says Hi!!!!!!
Thread 4 counter is: 1
Thread 4 counter is: 2
Thread 4 counter is: 3
SWITCH: currentThread = 4   SWITCHED TO: THREAD = 1
Thread 1 counter is: 4
Thread 1 counter is: 5
Thread 1 counter is: 6
SWITCH: currentThread = 1   SWITCHED TO: THREAD = 3
Thread 3 counter is: 4
Thread 3 counter is: 5
Thread 3 counter is: 6
SWITCH: currentThread = 3   SWITCHED TO: THREAD = 4
Thread 4 counter is: 4
Thread 4 counter is: 5
Thread 4 counter is: 6
SWITCH: currentThread = 4   SWITCHED TO: THREAD = 0
Thread 2 is resumed by Main thread
SWITCH: currentThread = 0   SWITCHED TO: THREAD = 2

Thread 2 says Hi!!!!!!
Thread 2 counter is: 1
Thread 2 counter is: 2
Thread 2 counter is: 3
SWITCH: currentThread = 2   SWITCHED TO: THREAD = 2
Thread 2 counter is: 4
Thread 2 counter is: 5
Thread 2 counter is: 6
SWITCH: currentThread = 2   SWITCHED TO: THREAD = 0
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pa1 $
```

As we can see in the above screenshot, thread-2 is being blocked by the main thread by calling the `uthread_suspend()` function. And then the main thread is resuming back thread-2 after the completion of thread-1 by calling `uthread_resume()`.

e. Test case:5

This function tests the **uthread_get_quantum()** function. And prints the quantum time of each thread in the terminal. We created 4 threads and each thread prints its quantum time.

```
File Edit View Search Terminal Help
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pal $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get_quantum()
Enter 6 to test uthread_get_total_quantums()
Please enter the test case number: 5
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created
Thread 1 Quantum: 1000000
Thread 2 Quantum: 1000000
Thread 3 Quantum: 1000000
Thread 4 Quantum: 1000000
SWITCH: currentThread = 0 SWITCHED TO: THREAD = 1

Thread 1 says Hi!!!!
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2

Thread 2 says Hi!!!!
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3

Thread 3 says Hi!!!!
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4

Thread 4 says Hi!!!!
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 0
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pal $
```

f. Test case 6:

This test case tests the **uthread_get_total_quantums()** function. Here we are assuming that all threads will have the same quantum time and this function will return the summation of the quantum times of all the threads. We created 4 threads and the main thread will print the total quantum time.

```
File Edit View Search Terminal Help
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pal $ ./uthread-demo
Enter 1 to test context switching
Enter 2 to test scheduling and time slicing
Enter 3 to test join functionality
Enter 4 to test resume and suspend
Enter 5 to test uthread_get_quantum()
Enter 6 to test uthread_get_total_quantums()
Please enter the test case number: 6
Main thread is created

Thread 1 is created
Thread 2 is created
Thread 3 is created
Thread 4 is created
Total Quantum time: 5000000
SWITCH: currentThread = 0 SWITCHED TO: THREAD = 1

Thread 1 says Hi!!!!
SWITCH: currentThread = 1 SWITCHED TO: THREAD = 2

Thread 2 says Hi!!!!
SWITCH: currentThread = 2 SWITCHED TO: THREAD = 3

Thread 3 says Hi!!!!
SWITCH: currentThread = 3 SWITCHED TO: THREAD = 4

Thread 4 says Hi!!!!
SWITCH: currentThread = 4 SWITCHED TO: THREAD = 0
attan005@cse1-vole-44:/home/attan005/Documents/pa 1/pal $
```

C. How did your library pass input/output parameters to a thread entry function? What must make context do “under the hood” to invoke the new thread execution?

While creating the thread we will be passing in the thread entry function and its arguments as the arguments to the `uthread_create()` function. And the arguments that are being passed to this thread entry function are of type `void*`. As it is of type `void*` the input can be converted to the desired type.

When a new thread is created make context will point the program counter to the thread entry function of the new thread and then it will change to a stack pointer to the specified memory location which contains the new thread context.