Design Choices for Programming Assignment 1

What (additional) assumptions did you make?

1. Uthread Library does not provide deadlock-prevention facilities, and it remains the responsibility of users to ensure that they design deadlock free programs.

   More specifcally, the program should be designed in a way that threads in READY queue will eventually get the resources it needs, instead of never again changing state.

   We implmented a round-robin scheduler, so that every thread is allotted the same time slice to run. The length of the time slice is set in `uthread_init(time_slice)`.

   The user should choose the length of time slice wisely. For example, in our testcases, there are no system call that blocks, so longer time slice is preferred for less context switching time.

2. `disableInterrupts()` only block SIGVTALRM signal.

   ```
   // Block signals from firing timer interrupt
   static void disableInterrupts()
   {
       /* define a new mask set */
       sigset_t mask_set;

       /* first clear the set (i.e. make it contain no signal numbers) */
       sigemptyset(&mask_set);

       /* add the SIGVTALRM signals to our mask set */
       sigaddset(&mask_set, SIGVTALRM);

       sigprocmask(SIG_BLOCK, &mask_set, NULL);
   }
   ```

   So, when we call `disableInterrupts()` after entering uthread library functions that modify critical data, we are able to disable timer interrupts (SIGVTALRM), but not other OS interrupts.

3. This user program should follow the thread create/join strategy, which means that the parent thread will wait for child threads to terminate by calling the pthread_join() function.

   For example, in our testcases, we use a **for** loop to wait on several threads using `uthread_join()` function.

4. Maximum number of threads (MAX_THREAD_NUM) and stack size (STACK_SIZE) per thread are predefine in uthead library. The user program should not attempt to exceed the limits.

Descriptions of testcases your provide and how to use them to test your implementation,

e.g. what's the purpose of each testcase, what specific functionality does each testcase focus on, what is the expected output of each testcase, etc.

Tests.cpp contains all test cases, to run test case, please run the following command in order: make clean make ./uthread-demo

Test Case 1

This test focuses on testing uthread_yeild() function. In this test case, we created two threads, both of the threads start from 0 and end at 9. Each thread has its own quantum, after each quantum the thread will yeild. The quantum for the first thread 100 is 2 and the quantum for the second thread 101 is 3. The output of the test case is in the folllowing section. As we can see, thread 100 yeilds at every two quantums and thread 101 yeilds at every three quantums.

```
 TEST YEILD FUNCTION:
Thread 100 STARTS
Thread 100 starts at 0
Thread 100 starts at 1
Thread 101 STARTS
Thread 101 starts at 0
Thread 101 starts at 1
Thread 101 starts at 2
NOW Thread 100 is running
Thread 100 starts at 2
Thread 100 starts at 3
NOW Thread 101 is running
Thread 101 starts at 3
Thread 101 starts at 4
Thread 101 starts at 5
NOW Thread 100 is running
Thread 100 starts at 4
Thread 100 starts at 5
NOW Thread 101 is running
Thread 101 starts at 6
Thread 101 starts at 7
Thread 101 starts at 8
NOW Thread 100 is running
Thread 100 starts at 6
Thread 100 starts at 7
NOW Thread 101 is running
Thread 101 starts at 9
Finished running Thread  101
NOW Thread 100 is running
Thread 100 starts at 8
Thread 100 starts at 9
NOW Thread 100 is running
Finished running Thread  100
EXIT
```

Test Case 2

This test focuses on testing uthread_self() function. If we pass the correct id we will get a meassage saying that "test for uthread_self() function succeeds", otherwise we will get a test fails message. In this test case, we created two threads. We passed the correct thread_id to the first thread 102 and passed a wrong thread_id to the second thread 103. Therefore, we will get a succeed message for thread 102 and a fail message for thread 103. The output showa as follows.

```
 TEST SELF FUNCTION:
Test uthread_self() for thread 102 succeeds
Test uthread_self() for thread 103 fails
EXIT
```

Test Case 3

This test case is designed to test uthread_join() function. We created three threads. Unlike test case 1, for these three threads we assigned them start and end number, so the thread will start from a specific number and ends at a sepcific number. The first thread starts at 1, ends at 1, and yeild in 2 quantums. The second thread starts at 13, ends at 19, and yeild in 4 quantums. The last thread starts at 25, ends at 30 and yeild in 3 quantums. As the following output shows, thread 104 finished before yeilding so

it will be added to finished queue. Then the processor will wait for 105 and 106 and exit the main threads when done waiting.

```
 TEST JOIN FUNCTION:
Thread 104 is joininig
Thread 104 STARTS
Thread 104 starts at 1
Thread 105 STARTS
Thread 105 starts at 13
Thread 105 starts at 14
Thread 105 starts at 15
Thread 105 starts at 16
Thread 106 STARTS
Thread 106 starts at 25
Thread 106 starts at 26
Thread 106 starts at 27
Thread 105 is joininig
Thread 105 starts at 17
Thread 105 starts at 18
Thread 105 starts at 19
Thread 106 starts at 28
Thread 106 starts at 29
Thread 106 starts at 30
Thread 106 is joininig
EXIT
```

Test Case 4

This test case is supposed to test uthread_suspend() and uthread_resume(). In this test case, we created three threads and suspended the second one when first thread executes and resumed it after two thread finished executing.

```
 TEST SUSPEND AND RESUME FUNCTION:
Thread 108 is suspended by thread 107
Thread 107 STARTS
Thread 107 starts at 0
Thread 109 STARTS
Thread 109 starts at 0
Thread 107 starts at 1
Thread 109 starts at 1
Thread 107 starts at 2
Thread 109 starts at 2
Thread 107 starts at 3
Thread 109 starts at 3
Thread 107 starts at 4
Thread 109 starts at 4
Thread 107 starts at 5
Thread 109 starts at 5
Thread 107 starts at 6
Thread 109 starts at 6
Thread 107 starts at 7
Thread 109 starts at 7
Thread 107 starts at 8
Thread 109 starts at 8
Thread 107 starts at 9
Thread 109 starts at 9
Resume thread 108
Thread 108 STARTS
Thread 108 starts at 0
Thread 108 starts at 1
Thread 108 starts at 2
Thread 108 starts at 3
Thread 108 starts at 4
Thread 108 starts at 5
Thread 108 starts at 6
Thread 108 starts at 7
Thread 108 starts at 8
Thread 108 starts at 9
EXIT
```

Note

uthread_create() and uthread_init() are used in every test cases so we did not implement separate test cases for these two functions.

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

The library created stub function to call thread entry fuction. Our library can customize input type and pass it to the void* argument of the thread entry level function, which is the second argument of the `uthread_create()`. The output is passed and updated during calling `uthread_join`.

To invoke execution of a new thread, the program counter has to be switched to the thread entry function and change the sp to the specified memory location.

In order to do so, `makecontext` points the program counter of the context, which is passed to the function as a parameter, to the thread entry function and puts the function arguments on the stack of the context.