1 Hands-on; Shared Memory I; Threads

1. **Creation and Termination Threads**, This example program creates 5 threads with the **pthread_create()** routine. Each thread prints a "Hello World!" message, and then terminates with a call to **pthread_exit()**. Compile as

   ```
   gcc -o code41 code41.c -lpthread
   ```

2. **Passing Arguments to Threads 1**, This example program demonstrates how to pass a simple integer to each thread.

3. **Passing Arguments to Threads 2**, This example program shows how to setup/pass multiple arguments via a structure. Each thread receives a unique instance of the structure.

4. **Passing Arguments to Threads 3 - Incorrectly**, This example program performs argument passing incorrectly.

   - It passes the address of variable t, *which is shared memory space and visible to all threads.*
   - The loop which creates threads modifies the contents of the address passed as an argument, *possibly before the created threads can access it.*

5. **Joining Threads**, This example program demonstrates how to "wait" for thread completions by using the Pthread join routine. Since some implementations of Pthreads may not create threads in a joinable state, the threads in this example are explicitly created in a joinable state so that they can be joined later. Compile as

   ```
   gcc -o code45 code45.c -lpthread -lm
   ```

6. **Exercise:** Complete this template; which is a thread program for finding the minimum of a list of integers;

   - The list is partitioned equally among the threads.
   - The size of each thread's partition is stored in the variable (*partial_list_size*).
   - The pointer to the start of each thread's partial list is passed to it as the pointer (*list_ptr*).
   - The test-update operation for *minimum_value* is protected by the mutex-lock *minimum_value_lock*.
   - Threads execute *pthread_mutex_lock* to gain exclusive access to the variable *minimum_value*.
   - Once this access is gained, the value is updated as required, and the lock subsequently released.
   - Since at any point of time, only one thread can hold a lock, only one thread can test-update the variable.