ECE-C353 - Systems Programming

Homework Assignment 5

For this assignment, base your solution on the following skeleton code::

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#define NUM_WORKER_THREADS 20

struct thread_data {
    pthread_t tid;
    unsigned int num;
};

struct workers_state {
    int still_working;
    pthread_mutex_t mutex;
    pthread_cond_t signal;
};

static struct workers_state wstate = {
    .still_working = NUM_WORKER_THREADS,
    .mutex = PTHREAD_MUTEX_INITIALIZER,
    .signal = PTHREAD_COND_INITIALIZER
};

static unsigned int result[NUM_WORKER_THREADS];

void* worker_thread (void* param)
{

}

int main (int argc, char** argv)
{

}
```

(continued on next page)

The main function shall:

1. Dynamically allocate `NUM_WORKER_THREADS` (i.e. 20) `struct thread_data` structures onto the heap.
   - Store the starting memory address of where these structures live in the heap into a variable named threads.
   - Individual `struct thread_data` structures should be accessible using array notation (e.g. `threads[i]`)
2. Assign a number to each worker thread.
   - A worker thread's number is an integer between 0 and `NUM_WORKER_THREADS`-1.
   - Each worker thread's number should be unique.
   - A worker thread's number is used to index into the array `threads[]` to get its `thread_data` structure.
   - A thread's number should be stored into the `num` member of its `thread_data` structure.
3. Create `NUM_WORKER_THREADS` worker threads.
   - Each worker thread should start in the function `worker_thread()`
   - An individual worker thread's `thread_data` structure should be passed as the parameter to `worker_thread()` during thread creation.
   - The Thread ID provided by `pthread_create()` during the thread creation process should be directly stored into the `tid` member of the thread's personal `thread_data` structure.
   - Worker threads should be in a detached state
4. Check if all worker threads have completed by checking the value of the `still_working` member of the `wstate` structure located in the `DATA` segment.
   - The main thread should check for this using the provided condition variable `signal`
5. Free the struct `thread_data` structures that were allocated onto the heap once they are no longer necessary.
6. Sum all of the elements of the global array `result[]` into a local variable named `total`.
7. Print the value of `total` to the screen.

Each worker thread shall:

1. Square their thread number and store the result into the global array `result[]`
   - Access to the global array `result[]` should be indexed by a worker thread's number
2. Decrement the variable `still_working`, which is a member of `wstate`.
3. Notify any threads that may be waiting for the state of `still_working` to change.

Additionally, after `total` has been printed to the screen, your main function should perform the following computation for C using a single thread (this is easily accomplished with a for-loop):

$C = \sum_{i=0}^{499} i^2$

Print the value of C to the screen as a check to see if `total` was computed properly in your multi-threaded approach.

Upload your code to BBLearn as a single C file named `abc123_hw5.c`

(Again, replace `abc123` with your Drexel ID)