REPORT OF ASSIGNMENT 3

What are Barriers?: A barrier is a tool for synchronizing the activity of a number of threads. When a thread reaches a barrier point, it cannot proceed until all other threads have reached this point as well. When the last thread reaches the barrier point, all threads are released and can resume concurrent execution.

Design and Analysis of Barrier(New, Without using library functions): C++11 Semaphore.h and Thread.h libraries are used for maintaing threads and semaphores. Following is the pseudocode for Readers and Writers.

```
Initaialisations:       sem_t mutex_ = 1;
                        sem_t mutex2_ = 1;
                        int barcount, currentBarCount;
                        double *avg_time;


barrier_init(THREAD_COUNT):     barCount=THREAD_COUNT;
                                 currentBarCount=0;
                        }


barrier_point():                        sem_wait(&mutex_);
                                        currentBarCount++;
                                        sem_post(&mutex_);

                                        if(currentBarCount<barCount)
                                                sem_wait(&mutex2_);
                                        else{
                                                for(int i=1; i<barCount; i++)
                                                sem_post(&mutex2_);
                                                currentBarCount=0;
                                        }
                                        while(currentBarCount);

newBarr(int thread_index):      for i=0; i<k; i++
                                        //sleep(spend time) for some time before
                                                reaching barrier
                                        //start the timer to record time
                                        barrier_point();
                                        //after barrier
                                        //end the timer and calculate time
                                        //sleep(spend time) for some time after reaching
                                                barrier
```

Above mentioned code implements barriers without using built in library functions.
currentBarCount: keeps track of how many have threads have reached barrier
barCount: keeps thread of number of threads
avg_time: keeps track of average of threads
mutex_: semaphore for incrementing currentBarCount across threads
mutex2_: semophore for blocking threads before all of them reaches barrier_point

barrier_int(): initialise the barCount and currentBarCount
barrier_point(): function to manage threads after they reach barrier point

Once the thread reaches the barrier point the thread gets blocked till all the thread reaches barrier point. mutex2_ semaphore is used for this purpose.

Design and Analysis of Barrier(Pthread, Using library functions): C++11 Semaphore.h and Thread.h libraries are used for maintaing threads and semaphores. Following is the pseudocode for Readers and Writers.

```
Initaialisations:     pthread_barrier_t mybarrier
                      double *avg_time;

newBarr(int thread_index):      for i=0; i<k; i++
                                      //sleep(spend time) for some time before
                                            reaching barrier
                                      //start the timer to record time
                                      pthread_barrier_wait(&mybarrier);
                                      //after barrier
                                      //end the timer and calculate time
                                      //sleep(spend time) for some time after reaching
                                            barrier

main():               //create and initialize the threads
                      pthread_barrier_init(&mybarrier, NULL, n);
                      pthread_barrier_destroy(&mybarrier);
```

Above mentioned code implements barriers using built in library functions.
avg_time: keeps track of average of threads
mybarrier: barrier variable to implement barriers

The `pthread_barrier_wait()` function synchronizes participating threads at the barrier pointed to by the *barrier* argument. The calling thread blocks until the

required number of threads have called `pthread_barrier_wait()` specifying *barrier*.

When the required number of threads have called `pthread_barrier_wait()` specifying the barrier, the constant `PTHREAD_BARRIER_SERIAL_THREAD` is returned to one thread and zero is returned to each of the remaining threads. The barrier is then reset to the state it had after the most recent `pthread_barrier_init()` call that referenced it.

The `pthread_barrier_wait()` function should not be used with an uninitialized barrier.

When a thread blocked on a barrier receives a signal, that thread resumes waiting at the barrier upon return from the signal handler if the required number of threads have not arrived at the barrier while the signal handler was executing. Otherwise, the thread continues as normal from the completed barrier wait.

A thread that has blocked on a barrier does not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

The `pthread_barrier_destroy()` function destroys the barrier pointed to by the *barrier* argument. It also releases the resources used by that barrier. Once a barrier has been destroyed, it should be reinitialized by a `pthread_barrier_init()` call before it is used again.

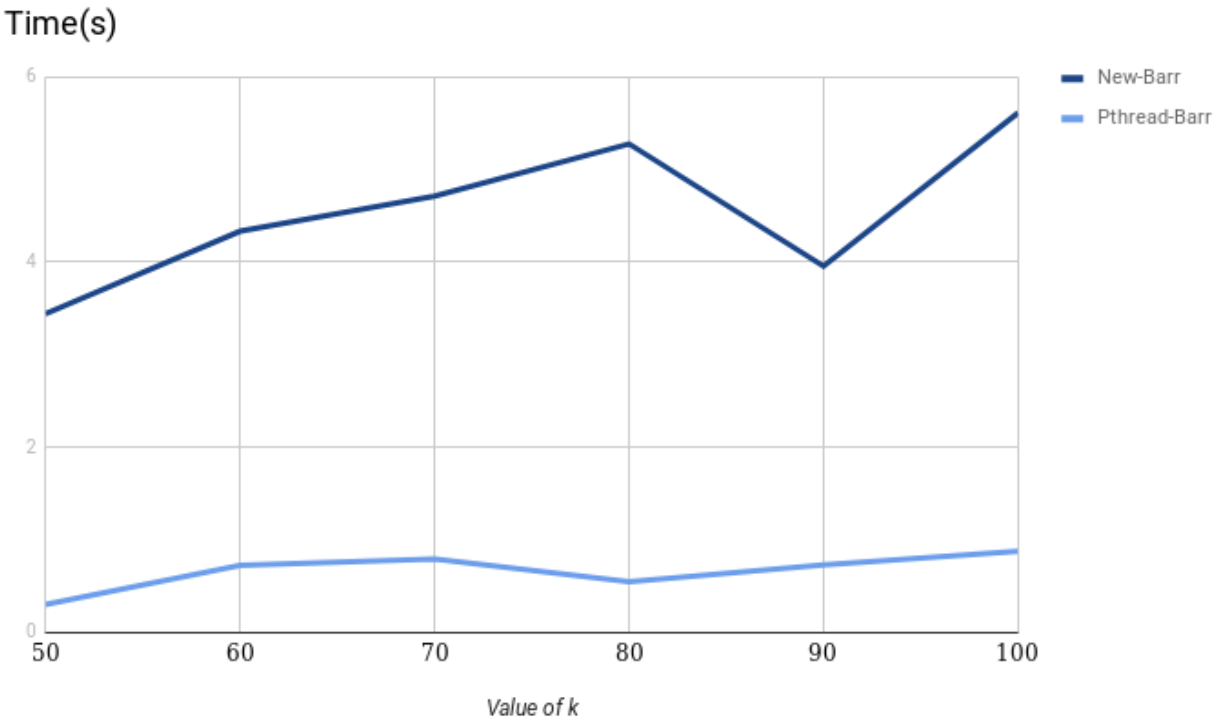
The `pthread_barrier_init()` function allocates the resources needed for the specified *barrier* and initializes that barrier with the attributes pointed to by the *attr* argument. When *attr* is NULL, the specified barrier is initialized with the default attributes. You should always initialize a barrier with this function before making any other reference to that barrier.

The *count* argument to `pthread_barrier_init()` indicates how many threads must call the `pthread_barrier_wait()` function before any of those threads successfully return from the call. *count* must be greater than zero.

When the `pthread_barrier_init()` function fails, the specified barrier is not initialized.

You should only use the object pointed to by *barrier* to perform synchronization. Using copies of that object with `pthread_barrier_destroy()` or `pthread_barrier_wait()` may not produce the desired results.

GRAPH 1: Value of k vs avg_time

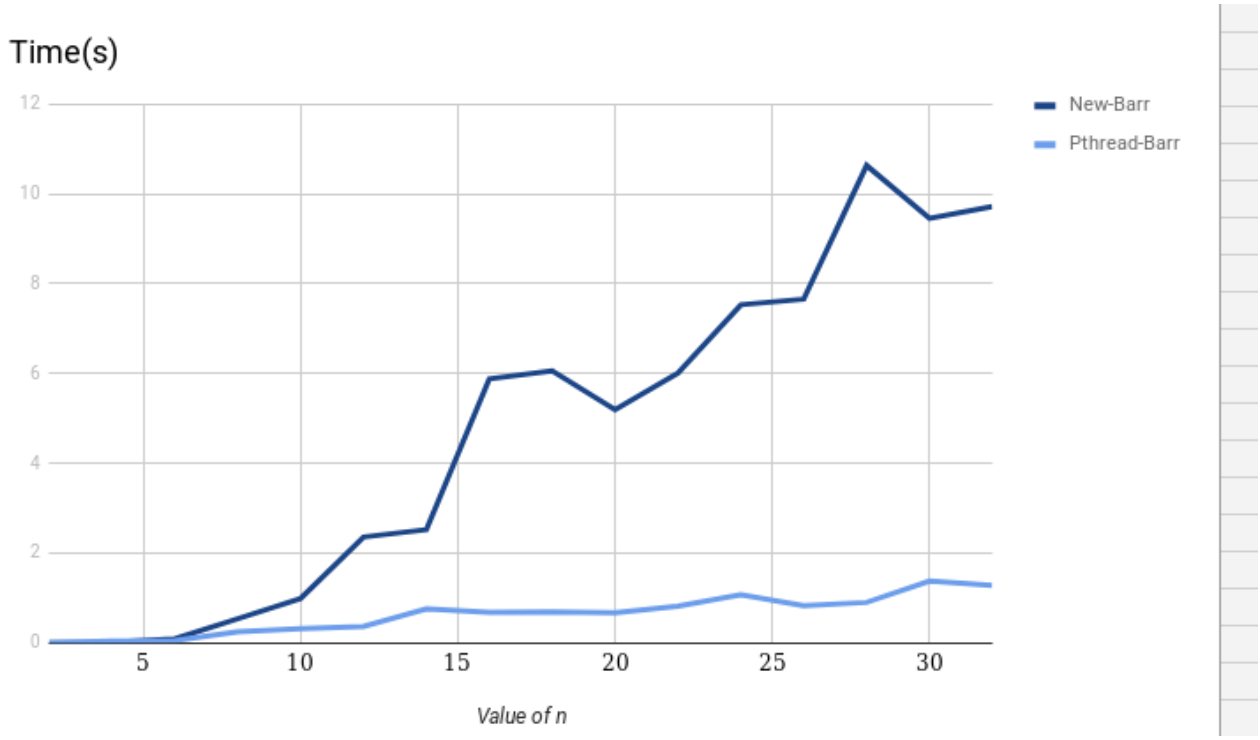


GRAPHICAL INTERPRETATION: Avg_Time of New Barr > Avg_Time of Pthread-Barr

Since the library functions are optimised for performance the behaviour is justified

Value of avg_time generally increases with increasing value of k. However, there is some deviation at some values. This may due to change in environment variables.

GRAPH 2: Value of n vs avg_time



GRAPHICAL INTERPRETATION: Avg_Time of New Barr > Avg_Time of Pthread-Barr

Since the library functions are optimised for performance the behaviour is justified

Value of avg_time generally increases with increasing value of n. However, there is some deviation at some values. This may due to change in environment variables.