Milestone 1 Report

Skeleton code

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void *func1(void *arg)
{

    printf("Inside the first thread\n");
    pthread_t id = pthread_self();
    printf("first thread id %lu\n", (unsigned long)id);
    printf("exiting first thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}

void *func2(void *arg)
{

    printf("Inside the second thread\n");
    pthread_t id = pthread_self();
    printf("second thread id %lu\n", (unsigned long)id);
    printf("exiting second thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}
void *func3(void *arg)
{

    printf("Inside the third thread\n");
    pthread_t id = pthread_self();
    printf("third thread id %lu\n", (unsigned long)id);
    printf("exiting third thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}
void *func4(void *arg)
{
```

```
    printf("Inside the forth thread\n");
    pthread_t id = pthread_self();
    printf("forth thread id %lu\n", (unsigned long)id);
    printf("exiting forth thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}

// Driver code
int main()
{
    pthread_t ptid1;
    pthread_t ptid2;
    pthread_t ptid3;
    pthread_t ptid4;
    pthread_attr_t attr;
    pthread_attr_setschedpolicy(&attr, SCHED_OTHER);
    clock_t start, end;
    double cpu_time_used;
    start = clock();
    // Creating a new thread
    pthread_create(&ptid1, NULL, &func1, NULL);
    pthread_create(&ptid2, NULL, &func2, NULL);
    pthread_create(&ptid3, NULL, &func3, NULL);
    pthread_create(&ptid4, NULL, &func4, NULL);
    printf("This line may be printed before thread terminates\n");

    // Compare the two threads created
    if (pthread_equal(ptid1, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid2, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid3, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid4, pthread_self()))
```

```
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid1, ptid2))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid1, ptid3))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid2, ptid3))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");

    // Waiting for the created thread to terminate
    pthread_join(ptid1, NULL);
    pthread_join(ptid2, NULL);
    pthread_join(ptid3, NULL);
    pthread_join(ptid4, NULL);

    printf("This line will be printed after thread ends\n");
    end = clock() - start;
    cpu_time_used = ((double)end) / CLOCKS_PER_SEC;
    printf("%i\n", end);
    printf("%.4f\n", cpu_time_used);
    pthread_exit(NULL);
    return 0;
}
```

Line by Line Code Explanation

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

These are preprocessor directives to include necessary header files for pthread (POSIX threads), standard input/output operations, standard library functions, and time-related functions.

```
void *func1(void *arg)
{

    printf("Inside the first thread\n");
    pthread_t id = pthread_self();
    printf("first thread id %lu\n", (unsigned long)id);
    printf("exiting first thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}

void *func2(void *arg)
{

    printf("Inside the second thread\n");
    pthread_t id = pthread_self();
    printf("second thread id %lu\n", (unsigned long)id);
    printf("exiting second thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}
void *func3(void *arg)
{

    printf("Inside the third thread\n");
    pthread_t id = pthread_self();
    printf("third thread id %lu\n", (unsigned long)id);
    printf("exiting third thread id %lu\n", (unsigned long)id);
    pthread_exit(NULL);
}
void *func4(void *arg)
{

    printf("Inside the forth thread\n");
    pthread_t id = pthread_self();
    printf("forth thread id %lu\n", (unsigned long)id);
    printf("exiting forth thread id %lu\n", (unsigned long)id);
```

```
    pthread_exit(NULL);
}
```

This defines a function func1 which takes a void pointer as an argument and returns a void pointer. Inside the function, it prints a message indicating it's inside the first thread, retrieves the ID of the thread, prints it, and then exits the thread.

Similarly, func2, func3, and func4 are defined with the same functionality for second, third, and fourth threads respectively.

```
int main()
{
    pthread_t ptid1;
    pthread_t ptid2;
    pthread_t ptid3;
    pthread_t ptid4;
    pthread_attr_t attr;
    clock_t start, end;
    double cpu_time_used;
    start = clock();
```

In the main function, it declares variables to hold thread IDs (ptid1, ptid2, ptid3, ptid4), a pthread attribute object attr, and variables for timing the execution. Then it initializes a clock to measure the execution time.

```
    pthread_attr_setschedpolicy(&attr, SCHED_OTHER);
```

It sets the scheduling policy for the threads to various scheduling algorithms, SCHED_OTHER, SCHED_RR, SCHED_FIFO.

```
    // Creating a new thread
    pthread_create(&ptid1, NULL, &func1, NULL);
    pthread_create(&ptid2, NULL, &func2, NULL);

    pthread_create(&ptid3, NULL, &func3, NULL);
    pthread_create(&ptid4, NULL, &func4, NULL);
```

Here, it creates four threads using the pthread_create() function, each calling a different function (func1, func2, func3, func4) which were defined earlier.

```
    printf("This line may be printed before thread terminates\n");
```

It prints a message indicating that this line may be printed before the threads terminate.

```
    // Compare the two threads created
    if (pthread_equal(ptid1, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid2, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid3, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid4, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid1, ptid2))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid1, ptid3))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
    if (pthread_equal(ptid2, ptid3))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");
```

It checks if the thread ID of ptid1 is equal to the thread ID of the main thread using the pthread_equal() function. If they are equal, it prints that threads are equal, otherwise not equal. Similarly, it does the same for ptid2, ptid3, and ptid4.

```
    // Waiting for the created thread to terminate
    pthread_join(ptid1, NULL);
    pthread_join(ptid2, NULL);
    pthread_join(ptid3, NULL);
    pthread_join(ptid4, NULL);
```

It waits for the four threads (ptid1, ptid2, ptid3, ptid4) to terminate using the pthread_join() function.

```
    printf("This line will be printed after thread ends\n");
    end = clock() - start;
    cpu_time_used = ((double)end) / CLOCKS_PER_SEC;
    printf("%i\n", end);
    printf("%.4f\n", cpu_time_used);
    pthread_exit(NULL);
    return 0;
}
```

After all threads have terminated, it prints a message indicating that this line will be printed after the threads end, calculates the CPU time used, and prints it. Finally, it exits the main thread.