Lab 5

Objective:

This lab focuses on writing C programs to demonstrate single-threaded and multi-threaded concepts using the `pthread` POSIX calls.

Thread Function	Thread call
Initialises a thread's attribute structure	`int pthread_attr_init(pthread_attr_t *attr)`
Creates a new thread	`int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg)`
Suspends the execution of the calling thread until the target thread ends	`int pthread_join(pthread_t thread, void **value_ptr)`
Releases the CPU to let another thread run	`int pthread_yield(void)`
Terminates the calling thread	`int pthread_exit(void *value_ptr)`

Table 1. Pthread Function Calls

Activity 1

Download the `samplethread.c` file from myElearning.

- Compile the program to ensure that there are no errors. (`gcc samplethread.c`)
- Run the program (`./a.out`)
- Ensure that you understand fully what is happening in this program

Activity 2

Download the `avg-no-threads.c` file from myElearning.

- Compile the program to ensure that there are no errors.
- Run the program and ensure that you understand fully what is happening in this program

Activity 3

Download the `avg-1-thread.c` file from myElearning.

- Compile the program to ensure that there are no errors.
- Run the program and ensure that you understand fully what is happening in this program
- One thread is created which calculates the sum of the 10 values in the data array of random numbers. This thread returns the sum to the main thread which then calculates the average.

Activity 4

Duplicate the `avg-1-thread.c` file and save it as `avg-many-threads.c`

- Modify the program to accept two parameters from the command line:
  - The number of values (`n`) to store in the data array.
  - The number of threads (`k`) to use to calculate the average
- The program should use `k` threads to calculate the sum of the values in the data array of `n` random numbers.
- In this activity, each thread is given a starting index into the array and a range it needs to cover. It sums the values in that range and returns the partial sum in the result result structure.
- The main thread spawns each thread one at a time in a loop, giving each its own argument structure. The argument structure should be filled in with the starting index where that thread is supposed to start and the range of values it needs to cover.
- After the main thread spawns all of the sum threads, it should go into another loop and join with the sum threads, one-at-a-time, in the order that they were spawned.

Bonus Activity

Write a multithreaded C program to calculate and display the sum of two `n x n` matrices.

samplethread.c

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

int sum; /* this data is shared by the thread(s) */

void *calculate_sum(void *param); /* the thread */

int main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of attributes for the thread */
    char arg[10]; sprintf(arg,"%d",10); /* in the book, this value is taken from command line */

    /* get the default attributes */
    pthread_attr_init(&attr);

    /* create the thread */
    pthread_create(&tid,&attr,calculate_sum,arg);

    /* now wait for the thread to exit */
    pthread_join(tid,NULL);

    printf("sum = %d\n",sum);
}

/**
 * The thread will begin control in this function
 */
void *calculate_sum(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    if (upper > 0) {
        for (i = 1; i <= upper; i++)
            sum += i;
    }

    pthread_exit(0);
}
```

avg-no-threads.c

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

/*
 * program to find the average value of a set of random numbers
 *
 * usage: avg-nothread count
 *
 * where count is the number of random values to generate
 */

char *Usage = "usage: avg-nothread count";

#define RAND() (drand48()) /* basic Linux random number generator */

int main(int argc, char **argv)
{
    int i;
    int n;
    double *data;
    double sum;
    int count;

    count = atoi(argv[1]);  /* count is first argument */

    /*
     * make an array large enough to hold #count# doubles
     */
    data = (double *)malloc(count * sizeof(double));

    /*
     * pick a bunch of random numbers
     */
    for(i=0; i < count; i++) {
        data[i] = RAND();
    }

    sum = 0;
    for(i=0; i < count; i++) {
        sum += data[i];
    }

    printf("the average over %d random numbers on (0,1) is %f\n",
           count, sum/(double)count);

    return(0);
}
```

avg-1-thread.c

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

#include <pthread.h>


char *Usage = "usage: avg-1-thread count";

#define RAND() (drand48()) /* basic Linux random number generator */

struct arg_struct     /* data type for arguments passed to a thread: 2 args */
{
    int size;           //size of array
    double *data;       // pointer to the array
};

struct result_struct /* data type for results returned by a thread */
{
    double sum;
};

/*
 Each thread begins with a function call that defines the 'body' of the thread.
 The function: void *SumThread(void *arg) is the 'entry point' for our thread.
 Linux specifies that these entry point functions should have the format:
 - take one argument of type void *
 - return a single argument of type void *

 (void *) is a legal pointer to any data type (char, int, double etc).
 This tells the compiler that our program will determine the data type
 at run time under its own control.


 */
void *SumThread(void *arg)
{
    int i;
    double my_sum;
    struct arg_struct *my_args;
    int my_size;
    double *my_data;
    struct result_struct *result;

    printf("sum thread running\n");
    fflush(stdout);

    /*Unmarshaling: the thread translates the generic input argument void*
      into the structure that it understands: arg_struct. This is done
     by casting the arg pointer to type struct and assigning it to my_args
     */
    my_args = (struct arg_struct *)arg;

    /* Allocate memory to transmit results when the thread has completed.
     */
    result = (struct result_struct *)malloc(sizeof(struct result_struct));

    my_size = my_args->size;
    my_data = my_args->data;  // assign pointer to array of numbers - thread's local copy

    free(my_args); //release the memory for the arguments structure as it is no longer needed

    my_sum = 0.0;
    for(i=0; i < my_size; i++) {
        my_sum += my_data[i];
    }

    result->sum = my_sum;  // load the sum in the results structure
    printf("sum thread done, returning\n");
    fflush(stdout);

    return((void *)result);   // marshaling: the
}
```

avg-1-thread.c continued

```
int main(int argc, char **argv)
{
    int i;
    int n;
    double *data;
    int count;
    struct arg_struct *args;
    pthread_t thread_id;
    struct result_struct *result;
    int err;

    count = atoi(argv[1]);  /* count is first argument */


    data = (double *)malloc(count * sizeof(double));

    for(i=0; i < count; i++) {
        data[i] = RAND();
    }

    args = (struct arg_struct *)malloc(sizeof(struct arg_struct));

    args->size = count;
    args->data = data;

    printf("main thread forking sum thread\n");
    fflush(stdout);

    err = pthread_create(&thread_id, NULL, SumThread, (void *)args);

    printf("main thread running after sum thread created, about to call join\n");
    fflush(stdout);

    err = pthread_join(thread_id,(void **)&result);

    printf("main thread joined with sum thread\n");
    fflush(stdout);

    printf("the average over %d random numbers on (0,1) is %f\n",
           count, result->sum / (double)count);

    free(result);
    free(data);

    return(0);
}
```