OS-Lab

lab#8

Name: Anas-Altaf

Roll.No : Anas Altaf

C Codes:

T-1:

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

#define NUM_THREADS 5

static int global_variable = 0;

void *perform_work()
{

   global_variable += 2;
   printf("global_variable = %d\n", global_variable);
   return NULL;
}

int main()
{
   pthread_t threads[NUM_THREADS];

   for (int i = 0; i < NUM_THREADS; i++)
   {
       if (pthread_create(&threads[i], NULL, (void *(*)(void
*))perform_work, NULL) != 0)
       {
           perror("Failed to create thread");
           return 1;
```

```
        }
    }

    for (int i = 0; i < NUM_THREADS; i++)
    {
        pthread_join(threads[i], NULL);
    }

    printf("Final value of global_variable = %d\n", global_variable);

    return 0;
}
```

Output:

```
root@xit:/media/xit/2nd/Test# gcc t1.c -lpthread
root@xit:/media/xit/2nd/Test# ./a.out
global_variable = 2
global_variable = 8
global_variable = 4
global_variable = 6
global_variable = 10
Final value of global_variable = 10
root@xit:/media/xit/2nd/Test#
```

T-2:

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

#define MAX_NUMBERS 20

typedef struct
{
    int *numbers;
    int count;
    double average;
    int min;
    int max;
} Stats;
```

```c
void *Calc_average(void *arg)
{
   Stats *stats = (Stats *)arg;
   double sum = 0;

   for (int i = 0; i < stats->count; i++)
   {
       sum += stats->numbers[i];
   }
   stats->average = sum / stats->count;
   return NULL;
}

void *Calc_minimum(void *arg)
{
   Stats *stats = (Stats *)arg;
   stats->min = stats->numbers[0];

   for (int i = 1; i < stats->count; i++)
   {
       if (stats->numbers[i] < stats->min)
       {
           stats->min = stats->numbers[i];
       }
   }
   return NULL;
}

void *Calc_maximum(void *arg)
{
   Stats *stats = (Stats *)arg;
   stats->max = stats->numbers[0];

   for (int i = 1; i < stats->count; i++)
   {
       if (stats->numbers[i] > stats->max)
       {
           stats->max = stats->numbers[i];
       }
   }
```

```c
    return NULL;
}

int main()
{
    Stats stats;
    stats.count = 0;
    stats.numbers = malloc(MAX_NUMBERS * sizeof(int));

    printf("Enter up to %d integers (end with -1):\n", MAX_NUMBERS);

    while (stats.count < MAX_NUMBERS)
    {
        int num;
        scanf("%d", &num);

        if (num == -1)
        {
            break;
        }

        stats.numbers[stats.count++] = num;
    }

    pthread_t avg_thread, min_thread, max_thread;

    pthread_create(&avg_thread, NULL, Calc_average, (void *)&stats);
    pthread_create(&min_thread, NULL, Calc_minimum, (void *)&stats);
    pthread_create(&max_thread, NULL, Calc_maximum, (void *)&stats);

    pthread_join(avg_thread, NULL);
    pthread_join(min_thread, NULL);
    pthread_join(max_thread, NULL);

    printf("The average value is %.2f\n", stats.average);
    printf("The minimum value is %d\n", stats.min);
    printf("The maximum value is %d\n", stats.max);

    free(stats.numbers);
```

```
    return 0;
}
```

Output:

```
root@xit:/media/xit/2nd/Test# gcc t2.c -lpthread
root@xit:/media/xit/2nd/Test# ./a.out
Enter up to 20 integers (end with -1):
12
23 45 22 55 -1
The average value is 31.40
The minimum value is 12
The maximum value is 55
root@xit:/media/xit/2nd/Test#
```

T-3:

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

typedef struct
{
    int *array;
    int left;
    int right;
    int key;
    int result;
} SearchParams;

void *binary_search(void *params)
{
    SearchParams *searchParams = (SearchParams *)params;
    int *array = searchParams->array;
    int left = searchParams->left;
    int right = searchParams->right;
    int key = searchParams->key;
    int foundIndex = -1;

    while (left <= right)
    {
```

```
        int mid = left + (right - left) / 2;

        if (array[mid] == key)
        {
            foundIndex = mid;

            right = mid - 1;
        }
        else if (array[mid] < key)
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }

    searchParams->result = foundIndex;
    return NULL;
}

int main()
{
    int n, key;

    printf("Enter the number of elements in the sorted array: ");
    scanf("%d", &n);

    int *array = malloc(n * sizeof(int));

    printf("Enter the sorted array elements:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("Enter the key to search: ");
    scanf("%d", &key);
```

```
    pthread_t thread1, thread2;
    SearchParams params1 = {array, 0, n / 2 - 1, key, -1};
    SearchParams params2 = {array, n / 2, n - 1, key, -1};

    pthread_create(&thread1, NULL, binary_search, &params1);
    pthread_create(&thread2, NULL, binary_search, &params2);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    int index = params1.result;
    if (index == -1)
    {
        index = params2.result;
    }

    if (index != -1)
    {
        printf("First occurrence of key %d found at index: %d\n", key,
index);
    }
    else
    {
        printf("Key %d not found in the array.\n", key);
    }

    free(array);
    return 0;
}
```

Output:

```
root@xit:/media/xit/2nd/Test# gcc t3.c -lpthread
root@xit:/media/xit/2nd/Test# ./a.out
Enter the number of elements in the sorted array: 8
Enter the sorted array elements:
12 34 54 65 72 83 93 93
Enter the key to search: 93
First occurrence of key 93 found at index: 6
root@xit:/media/xit/2nd/Test#
```

T-4:

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

#define SIZE 3

typedef struct
{
   int (*matrixA)[SIZE];
   int (*matrixB)[SIZE];
   int (*result)[SIZE];
   int row;
} ThreadData;

void *multiply_row(void *arg)
{
   ThreadData *data = (ThreadData *)arg;
   int row = data->row;

   for (int j = 0; j < SIZE; j++)
   {
       data->result[row][j] = 0;
       for (int k = 0; k < SIZE; k++)
       {
           data->result[row][j] += data->matrixA[row][k] *
data->matrixB[k][j];
       }
   }
   return NULL;
}

int main()
{

   int matrixA[SIZE][SIZE] = {
       {1, 2, 3},
       {4, 5, 6},
       {7, 8, 9}};
```

```
    int matrixB[SIZE][SIZE] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}};

    int result[SIZE][SIZE];

    pthread_t threads[SIZE];
    ThreadData threadData[SIZE];

    for (int i = 0; i < SIZE; i++)
    {
        threadData[i].matrixA = matrixA;
        threadData[i].matrixB = matrixB;
        threadData[i].result = result;
        threadData[i].row = i;

        pthread_create(&threads[i], NULL, multiply_row, &threadData[i]);
    }

    for (int i = 0; i < SIZE; i++)
    {
        pthread_join(threads[i], NULL);
    }

    printf("Resultant Matrix:\n");
    for (int i = 0; i < SIZE; i++)
    {
        for (int j = 0; j < SIZE; j++)
        {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:


root@xit:/media/xit/2nd/Test# gcc t4.c -lpthread
root@xit:/media/xit/2nd/Test# ./a.out
Resultant Matrix:
30 36 42
66 81 96
102 126 150
root@xit:/media/xit/2nd/Test#