1. Following C program creates a global array with size 1000 and initialises the array with random values. Then creates 3 threads to perform find average, minimum and maximum values in the global array and prints out the result.

   Below is the program together with it's output:

   q1.c

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

// definition(s)
#define N 1000
#define NOF_THREADS 3

// global variable(s)
int array[N];
pthread_t tid[NOF_THREADS];
int avg_val, min_val, max_val;

void *calcStats(void *arg)
{

    pthread_t id = pthread_self();
    if (pthread_equal(id, tid[0]))
    {
        // first thrad: avg()
        int sum = 0;
        for (int i = 0; i < N; i++)
        {
            sum += array[i];
        }
        avg_val = sum / N;
        pthread_exit(&avg_val);
    }
    else if (pthread_equal(id, tid[1]))
    {
        // second thread: min()
        min_val = array[0];
        for (int i = 1; i < N; i++)
        {
            if (array[i] < min_val)
            {
                min_val = array[i];
            }
```

```
41          }
42          pthread_exit(&min_val);
43      }
44      else
45      {
46          //  third thread: max()
47          max_val = array[0];
48          for (int i = 1; i < N; i++)
49          {
50              if (array[i] > max_val)
51              {
52                  max_val = array[i];
53              }
54          }
55          pthread_exit(&max_val);
56      }
57
58      return NULL;
59  }
60
61  int main(void)
62  {
63      int err;
64      int *ptr[NOF_THREADS];
65
66      // populate the global array with random numbers [0, N)
67      for (int j = 0; j < N; j++)
68      {
69          array[j] = (rand() % N);
70      }
71
72      for (int i = 0; i < NOF_THREADS; i++)
73      {
74          err = pthread_create(&(tid[i]), NULL, &calcStats, NULL);
75          if (err != 0)
76              printf("\n ERROR: cannot creat the thread [%s]", strerror(err));
77          else
78              printf("\n SUCCESS: thread creation\n");
79      }
80
81      pthread_join(tid[0], (void **)&(ptr[0]));
82      pthread_join(tid[1], (void **)&(ptr[1]));
83      pthread_join(tid[2], (void **)&(ptr[2]));
84
85      printf("Average value of the random array: %d\n", *ptr[0]);
86      printf("Minimum value of the random array: %d\n", *ptr[1]);
87      printf("Maximum value of the random array: %d\n", *ptr[2]);
88
```

```
    return 0;
}
```

make
./q1

```
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw3/q1$ make
gcc -pthread -o q1 q1.c
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw3/q1$ ./q1

 SUCCESS: thread creation

 SUCCESS: thread creation

 SUCCESS: thread creation
Average value of the random array: 499
Minimum value of the random array: 0
Maximum value of the random array: 999
```

2. Shared memory practice for writing a producer and consumer programs which will write and read from the shared memory allocated for the struct student.

```
// Question 2
// @author: Zeynep Cankara
// @version: 1.0 07/03/2021

// import libraries
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>

typedef struct student
{
    int id;
    char name[128];
    char lastname[128];
    int age;
    double cgpa;
} student;

void *createSharedMemory(size_t size)
{
    // read and writable buffer
```

```
25      int accessRights = PROT_READ | PROT_WRITE;
26
27      // shared access to the memory
28      int visibilityRights = MAP_SHARED | MAP_ANONYMOUS;
29
30      return mmap(NULL, size, accessRights, visibilityRights, -1, 0);
31  }
32
33  // Create a student
34  struct student *createStudent(int id, char *name, char *lastname, int age, double cgpa)
35  {
36      struct student *newStudent = malloc(sizeof(struct student *));
37      newStudent->id = id;
38      strcpy(newStudent->name, name);
39      strcpy(newStudent->lastname, lastname);
40      newStudent->age = age;
41      newStudent->cgpa = cgpa;
42      return newStudent;
43  }
44
45  // print the student information
46  void printStudent(struct student *s)
47  {
48      printf("id: %i\n", s->id);
49      printf("age: %i\n", s->age);
50      printf("cgpa: %.2f\n", s->cgpa);
51      printf("name: %s\n", s->name);
52      printf("lastname: %s\n", s->lastname);
53  }
54
55  int main(int argc, char **argv)
56  {
57      // define the students
58      struct student *s1 = createStudent(1, "John", "Fish", 21, 3.2);
59      struct student *s2 = createStudent(2, "William", "Smith", 22, 4.0);
60      struct student *s3 = createStudent(3, "Alice", "Keys", 21, 3.9);
61
62      // create the shared memory
63      void *shmem1 = createSharedMemory(sizeof(s1));
64      void *shmem2 = createSharedMemory(sizeof(s2));
65      void *shmem3 = createSharedMemory(sizeof(s3));
66
67      // child acts as a producer and parent as the consumer
68      int pid = fork();
69
70      if (pid == 0)
71      {
72          // write information to the shared memory
```

```
        memcpy(shmem1, s1, sizeof(*s1));
        printf("Write student 1 from shared memory: %p\n", shmem1);
        memcpy(shmem2, s2, sizeof(*s2));
        printf("Write student 2 from shared memory: %p\n", shmem2);
        memcpy(shmem3, s3, sizeof(*s3));
        printf("Write student 3 from shared memory: %p\n", shmem3);
    }
    else
    {
        wait(NULL); // wait children write
        printf("Read student 1 from shared memory: %p\n", shmem1);
        printStudent(shmem1);
        printf("Read student 2 from shared memory: %p\n", shmem2);
        printStudent(shmem2);
        printf("Read student 3 from shared memory: %p\n", shmem3);
        printStudent(shmem3);
    }
    wait(NULL);
    return 0;
}
```

make
./q2

```
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw3/q2$ ./q2
Write student 1 from shared memory: 0x7f74c82f2000
Write student 2 from shared memory: 0x7f74c82c5000
Write student 3 from shared memory: 0x7f74c82c4000
Read student 1 from shared memory: 0x7f74c82f2000
id: 1
age: 21
cgpa: 3.20
name: Ece
lastname: Keys
Read student 2 from shared memory: 0x7f74c82c5000
id: 2
age: 22
cgpa: 4.00
name: Tom
lastname: Will
Read student 3 from shared memory: 0x7f74c82c4000
id: 3
age: 21
cgpa: 3.90
name: Can
lastname: Smith
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw3/q2$
```

3. According to the *Amdahl's Law* following equation holds with the parameters S as time to execute serial part and N as the parallelizable portion ehich allows maximum speedup of *2.909* for $S = 0.25$ and $N = 8$ values.

$$speedup \leq \frac{1}{S + [\frac{(1-S)}{N}]} \tag{1}$$

$$speedup \leq \frac{1}{0.25 + [\frac{0.75}{8}]} \tag{2}$$

$$speedup \leq 2.909 \tag{3}$$

4. Analysis on the Finishing and Waiting time of the following scheduling algorithms on different 5 processes outlined in the homework description:

RR scheduling with q = 30 ms		
Process	Finish	Waiting
A	160ms	110ms
B	240ms	145ms
C	200ms	125ms
D	110ms	35ms
E	220ms	105ms

RR scheduling with q = 10 ms		
Process	Finish	Waiting
A	120ms	70ms
B	240ms	145ms
C	180ms	105ms
D	140ms	65ms
E	230ms	115ms

RR scheduling with q = very very small		
Process	Finish	Waiting
A	144.15ms	94.15ms
B	240ms	160ms
C	190.83ms	150.83ms
D	150.83ms	130.83ms
E	229.17ms	179.17ms

SRJF		
Process	Finish	Waiting
A	50ms	0ms
B	240ms	145ms
C	110ms	35ms
D	75ms	0ms
E	160ms	45ms

FCFS		
Process	Finish	Waiting
A	50ms	0ms
B	130ms	45ms
C	170ms	95ms
D	190ms	115ms
E	240ms	125ms

5. Estimate after the three bursts in milliseconds (ms) with $\tau_0 = 20ms$ and bursts of lengths 24, 18, and 30, with alpha = 0.4:

$$\tau_0 = 20ms \tag{4}$$

$$\tau_1 = (0.4) \cdot 24 + (0.6) \cdot 20 = 21.6ms \tag{5}$$

$$\tau_2 = (0.4) \cdot 18 + (0.6) \cdot \tau_1 = 20.16ms \tag{6}$$

$$\tau_3 = (0.4) \cdot 30 + (0.6) \cdot \tau_2 = 24.096ms \tag{7}$$