Rüzgar Ayan 21801984

2020-2021 SPRING CS342 HOMEWORK 3 Report

Question 1.

The code for question 1 is below:

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

#define SIZE 1000

int arr[SIZE];
int aver = 0, min = 0, max = 0;

void *findMax (void *param);
void *findMin (void *param);
void *findAver (void *param);

int main(int argc, char *argv[]) {
    srand(time(NULL));
    for (int i = 0; i < SIZE; i++)
    {
        arr[i] = rand();
    }

    pthread_t tid1, tid2, tid3; /* id of the created thread */
    pthread_attr_t attr1, attr2, attr3;  /* set of thread attributes */

    pthread_attr_init (&attr1);
    pthread_attr_init (&attr2);
    pthread_attr_init (&attr3);

    pthread_create (&tid1, &attr1, findMax, NULL);
    pthread_create (&tid2, &attr2, findMin, NULL);
    pthread_create (&tid3, &attr3, findAver, NULL);

    pthread_join (tid1, NULL);
    pthread_join (tid2, NULL);
    pthread_join (tid3, NULL);

    printf("Average=%d\nMin=%d\nMax=%d\n", aver, min, max);

    return 0;
}

void *findMax (void *param)
{
    int maxSoFar = arr[0];
    for (int i = 0; i < SIZE; i++)
        if (arr[i] > maxSoFar)
            maxSoFar = arr[i];

    max = maxSoFar;

    pthread_exit(0);
}

void *findMin (void *param)
```

```
{
    int minSoFar = arr[0];
    for (int i = 0; i < SIZE; i++)
        if (arr[i] < minSoFar)
            minSoFar = arr[i];

    min = minSoFar;

    pthread_exit(0);
}

void *findAver (void *param)
{
    long long int sum = 0;
    for (int i = 0; i < SIZE; i++)
        sum += arr[i];

    aver = sum / SIZE;

    pthread_exit(0);
}
```

Output of the program is below:

```
zzlawlzz@zzlawlzz-VirtualBox:~/Desktop/hw3$ ./q1
Average=1093188737
Min=970885
Max=2145025706
```

Question 2.

For this question, I have taken a lot of help from the example code shared in Moodle. First file below is the “q2shareddefs.h” file.

```
struct student {
    int id;
    char name[64];
    char lastname[64];
    int age;
    double cgpa;
};

#define SHAREDMEM_NAME "/q2"
#define SHAREDMEM_SIZE 1024
```

Next file is “q2producer.c”

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "q2shareddefs.h"

int main(int argc, char *argv[]) {

    int fd;
    void *shm_start;
    struct shared_data *sdata_ptr;
    fd = shm_open(SHAREDMEM_NAME, O_RDWR | O_CREAT, 0660);
    if (fd < 0)
    {
        printf("Could not create shared memory, terminating the program.\n");
        exit(1);
    }
    else printf("Created shared memory.\n");

    ftruncate(fd, SHAREDMEM_SIZE);
    shm_start = mmap(NULL, SHAREDMEM_SIZE,
             PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm_start < 0)
    {
        printf("Could not map shared memory, terminating the program.\n");
        exit(1);
    }
    else printf("Mapped shared memory.\n");
    close(fd);

    struct student *student1, *student2, *student3;

    student1 = (struct student *) shm_start;
    student1->id = 1045;
    strcpy (student1->name, "Ali");
    strcpy (student1->lastname, "Demir");
    student1->age = 18;
    student1->cgpa = 3.42;
    printf("Written to the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
cgpa=%f)\n",
        student1->id, student1->name, student1->lastname, student1->age, student1->cgpa);
```

```
    shm_start += sizeof(struct student);

    student2 = (struct student *) shm_start;
    student2->id = 3253;
    strcpy (student2->name, "Umut");
    strcpy (student2->lastname, "Duru");
    student2->age = 20;
    student2->cgpa = 2.50;
    printf("Written to the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
cgpa=%f)\n",
        student2->id, student2->name, student2->lastname, student2->age, student2->cgpa);

    shm_start += sizeof(struct student);

    student3 = (struct student *) shm_start;
    student3->id = 2366;
    strcpy (student3->name, "Mert");
    strcpy (student3->lastname, "Çetin");
    student3->age = 22;
    student3->cgpa = 2.99;
    printf("Written to the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
cgpa=%f)\n",
        student3->id, student3->name, student3->lastname, student3->age, student3->cgpa);

    printf("End of the producer program\n");

    return 0;
}
```

And the last file below is “q2consumer.c”.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "q2shareddefs.h"

int main(int argc, char *argv[]) {

    int fd;
    void *shm_start;
    struct shared_data *sdata_ptr;
    fd = shm_open(SHAREDMEM_NAME, O_RDWR, 0660);
    if (fd < 0)
    {
        printf("Could not create shared memory, terminating the program.\n");
        exit(1);
    }
    else printf("Opened shared memory.\n");

    shm_start = mmap(NULL, SHAREDMEM_SIZE,
            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm_start < 0)
    {
        printf("Could not map shared memory, terminating the program.\n");
        exit(1);
    }
    else printf("Mapped shared memory.\n");
    close(fd);

    struct student *student1, *student2, *student3;

    student1 = (struct student *) shm_start;
    printf("Read from the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
```

```
cgpa=%f)\n",
        student1->id, student1->name, student1->lastname, student1->age, student1->cgpa);


    shm_start += sizeof(struct student);
    student2 = (struct student *) shm_start;
    printf("Read from the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
cgpa=%f)\n",
        student2->id, student2->name, student2->lastname, student2->age, student2->cgpa);

    shm_start += sizeof(struct student);
    student3 = (struct student *) shm_start;
    printf("Read from the shared memory: student(id=%d, name=%s, lastname=%s, age=%d,
cgpa=%f)\n",
        student3->id, student3->name, student3->lastname, student3->age, student3->cgpa);

    if (shm_unlink(SHAREDMEM_NAME) < 0)
    {
        printf("Could not remove shared memory, terminating the program.\n");
        exit(1);
    }
    else printf("Removed shared memory.\n");
    printf("End of the consumer program\n");


    return 0;
}
```

When I first run the producer, it writes the student information into the shared memory and exits. Then, when we run the consumer, it reads the information, prints it and then unlinks the shared memory. The execution outputs are as follows.

```
zzlawlzz@zzlawlzz-VirtualBox:~/Desktop/hw3$ ./q2p
Created shared memory.
Mapped shared memory.
Written to the shared memory: student(id=1045, name=Ali, lastname=Demir, age=18, cgpa=3.420000)
Written to the shared memory: student(id=3253, name=Umut, lastname=Duru, age=20, cgpa=2.500000)
Written to the shared memory: student(id=2366, name=Mert, lastname=Çetin, age=22, cgpa=2.990000)
End of the producer program
zzlawlzz@zzlawlzz-VirtualBox:~/Desktop/hw3$ ./q2c
Opened shared memory.
Mapped shared memory.
Read from the shared memory: student(id=1045, name=Ali, lastname=Demir, age=18, cgpa=3.420000)
Read from the shared memory: student(id=3253, name=Umut, lastname=Duru, age=20, cgpa=2.500000)
Read from the shared memory: student(id=2366, name=Mert, lastname=Çetin, age=22, cgpa=2.990000)
Removed shared memory.
End of the consumer program
```

Question 3.

The formula for maximum speedup is as follows:

$$speedup \leq \frac{1}{S + \frac{1-S}{N}}$$

If we put the values S=0.25 and N = 8, the maximum possible speedup becomes

$$speedup \leq \frac{1}{0.25 + \frac{0.75}{8}} \approx \mathbf{2.909}$$

The limit is (when the number of processors go to infinity):

$$speedup \leq \lim_{N \to \infty} \frac{1}{0.25 + \frac{0.75}{N}} = \frac{1}{0.25 + 0} \approx \mathbf{4.000}$$

Question 4.

a) RR scheduling with q = 30 ms:

Process	Finish Time	Total Waiting Time
A	80	30
B	220	125
C	200	125
D	130	55
E	240	125

b) RR scheduling with q = 10 ms:

Process	Finish Time	Total Waiting Time
A	100	50
B	240	145
C	180	105
D	150	75
E	230	115

c) RR scheduling with very very small q:

For this part, we have to assume that there is no overhead due to the context switching. Otherwise, these would take infinite time as q gets closer to 0ms. Also, I had to do some rounding in the results.

Process	Finish Time	Total Waiting Time
A	144	94
B	240	145
C	191	116
D	151	76
E	229	114

d) SRJR:

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

e) FCFS:

Process	Finish Time	Total Waiting Time
A	50	0
B	130	35
C	170	95
D	190	115
E	240	125

Question 5.

We are given,

$$\alpha = 0.4,$$

$$t_0 = 24\,ms,$$

$$t_1 = 18\,ms,$$

$$t_2 = 30\,ms,$$

$$\tau_0 = 20\,ms.$$

Then, we calculate the estimations as follows:

$$\tau_1 = 0.4 * 24 + 0.6 * 20 = 21.6\,ms$$

$$\tau_2 = 0.4 * 18 + 0.6 * 21.6 = 20.16\,ms$$

$$\tau_3 = 0.4 * 30 + 0.6 * 20.16 = \mathbf{24.096\ ms}$$

The estimation for the next cpu burst length is then 24.096 ms.