Pınar Yücel
21802188

CS342 Operating Systems - Spring 2021
Homework #3

---

1. Code of the C program:

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

#define ARR_SIZE 1000

int arr[ARR_SIZE];

void initializeArray()
{
	srand(time(NULL));

	int i;
	for(i = 0; i < ARR_SIZE; i++)
	{
		// Generate random numbers between 0 and 1000
		arr[i] = rand() % 1000;
	}
}

void* findAvg (void *avg)
{
	int sum = 0;

	int i;
	for(i = 0; i < ARR_SIZE; i++)
	{
		sum = sum + arr[i];
	}

	if(sum == 0)
	{
		*(double*)avg = 0;
	}
	else
	{
		printf("sum: %d\n", sum);
		*(double*)avg = (double)sum / ARR_SIZE;
	}

	pthread_exit(0);
}
```

```
void* findMin(void *min)
{
	*(int*)min = arr[0];

	int i;
	for(i = 1; i < ARR_SIZE; i++)
	{
		if(arr[i] < *(int*)min)
		{
			*(int*)min = arr[i];
		}
	}

	pthread_exit(0);
}

void* findMax(void *max)
{
	*(int*)max = arr[0];

	int i;
	for(i = 1; i < ARR_SIZE; i++)
	{
		if(arr[i] > *(int*)max)
		{
			*(int*)max = arr[i];
		}
	}

	pthread_exit(0);
}

int  main()
{
	initializeArray();
	double avg;
	int min;
	int max;

	pthread_t tid1;
	pthread_t tid2;
	pthread_t tid3;

	pthread_attr_t attr;
	pthread_attr_init(&attr);

	pthread_create (&tid1, &attr, findAvg, &avg);
	pthread_create (&tid2, &attr, findMin, &min);
	pthread_create (&tid3, &attr, findMax, &max);

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

```
        pthread_join (tid3, NULL);

        printf ("avg = %0.3f\n", avg);
        printf ("min = %d\n", min);
        printf ("max = %d\n", max);
}
```

2. Code of the producer program:

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

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

int main()
{
        const int SIZE = 4096;
        const char *name = "Students";

        int shm_fd;
        void *ptr;

        shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
        ftruncate(shm_fd, SIZE);

        ptr = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

        if (ptr == MAP_FAILED) { printf("Map failed\n"); return -1; }

        struct student *sp;

        sp = (struct student *) ptr;
        sp->id = 100;
        strcpy(sp->name, "Bob");
        strcpy(sp->lastname, "Karlson");
        sp->age = 20;
        sp->cgpa = 3.00;
```

```
        ptr = ptr + sizeof(struct student);
        sp = (struct student *) ptr;
        sp->id = 200;
        strcpy(sp->name, "Nea");
        strcpy(sp->lastname, "Ming");
        sp->age = 21;
        sp->cgpa = 3.30;

        ptr = ptr + sizeof(struct student);
        sp = (struct student *) ptr;
        sp->id = 300;
        strcpy(sp->name, "Yui");
        strcpy(sp->lastname, "Kimura");
        sp->age = 22;
        sp->cgpa = 3.50;

        return 0;
}
```

Code of the consumer program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>

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

int main()
{
        const char *name = "Students";
        const int SIZE = 4096;
        int shm_fd;
        void *ptr;

        shm_fd = shm_open(name, O_RDONLY, 0666);
        if (shm_fd == -1) { printf("shared memory failed\n"); exit(-1); }

        ptr = mmap(0,SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
        if (ptr == MAP_FAILED) {printf("Map failed\n"); exit(-1); }

        struct student* sp = (struct student *) ptr;
```

```
        printf("%d\n", sp->id);
        printf("%s\n", sp->name);
        printf("%s\n", sp->lastname);
        printf("%d\n", sp->age);
        printf("%.2f\n\n", sp->cgpa);

        ptr = ptr + sizeof(struct student);
        sp = (struct student *) ptr;
        printf("%d\n", sp->id);
        printf("%s\n", sp->name);
        printf("%s\n", sp->lastname);
        printf("%d\n", sp->age);
        printf("%.2f\n\n", sp->cgpa);

        ptr = ptr + sizeof(struct student);
        sp = (struct student *) ptr;
        printf("%d\n", sp->id);
        printf("%s\n", sp->name);
        printf("%s\n", sp->lastname);
        printf("%d\n", sp->age);
        printf("%.2f\n", sp->cgpa);

        if (shm_unlink(name) == -1) {printf("Error removing %s\n",name); exit(-1);}
}
```

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

   $S = 0.25$

   $N = 8$

   $speedup \leq \frac{1}{0.25 + 0.75/8} = \frac{1}{0.34375} \simeq 2.91$

   Maximum speed up that we can achieve for that program if we use 8 processors is approximately equal to 2.91

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

   $S = 0.25$

   $N = \infty$

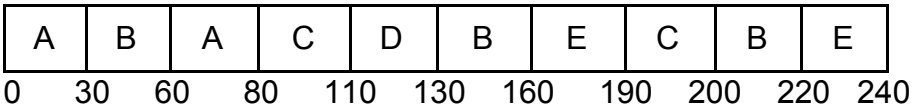
   $speedup \leq \frac{1}{0.25 + 0.75/\infty} = \frac{1}{0.25} = 4$

   Limit is 4

4.

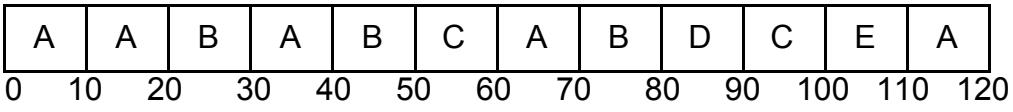
a) RR scheduling with q = 30 ms

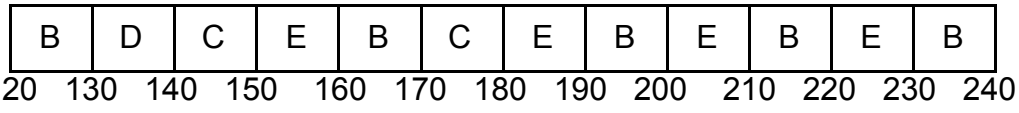
	Arrival time	CPU time	Finish time	Waiting time
A	0	50	80	30
B	15	80	220	125
C	35	40	200	125
D	55	20	130	55
E	65	50	240	125



b) RR scheduling with q = 10 ms

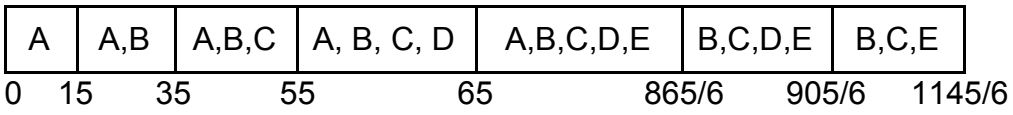
	Arrival time	CPU time	Finish time	Waiting time
A	0	50	120	70
B	15	80	240	145
C	35	40	180	105
D	55	20	140	65
E	65	50	230	115

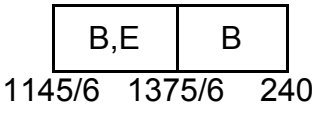




c) RR scheduling with q = very very small

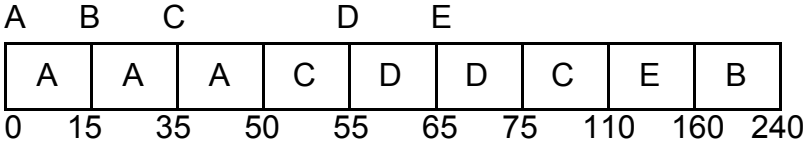
	Arrival time	CPU time	Finish time	Waiting time
A	0	50	865/6	565/6
B	15	80	240	145
C	35	40	1145/6	695/6
D	55	20	905/6	455/6
E	65	50	1375/6	685/6





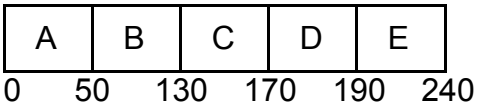
d) SRJF

	Arrival time	CPU time	Finish time	Waiting time
A	0	50	50	0
B	15	80	240	145
C	35	40	110	35
D	55	20	75	0
E	65	50	160	45



e) FCFS

	Arrival time	CPU time	Finish time	Waiting time
A	0	50	50	0
B	15	80	130	35
C	35	40	170	95
D	55	20	190	115
E	65	50	240	125



5. $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

$\tau_1 = 0.4*24 + 0.6*20 = 21.6$
$\tau_2 = 0.4*18 + 0.6*21.6 = 20.16$
$\tau_3 = 0.4*30 + 0.6*20.16 = 24.096$

$\tau_3 = 24.096$