CS350-hw7

easyabi

November 2020

1 Question 1

1.1 a

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \times (2^{1/n} - 1)$$

$$\frac{4.5}{19} + \frac{4.5}{22} + \frac{3}{9} \leq 3 \times (2^{1/3} - 1) \longrightarrow 0.775 \leq 0.78$$

The inequality holds. Therefore, the task set is schedulable under RM

1.2 b

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \times (2^{1/n} - 1)$$

$$\frac{4.5}{19} + \frac{4.5}{22} + \frac{3}{9} + \frac{1}{6} \leq 4 \times (2^{1/4} - 1) \longrightarrow 0.94 \leq 0.76$$

The above inequality does not hold and therefore it does yield not any conclusion; Hence, we need to draw the system to see if it is schedulable or not. According to the following figure the taskset is schedulable under RM because no task misses deadline.

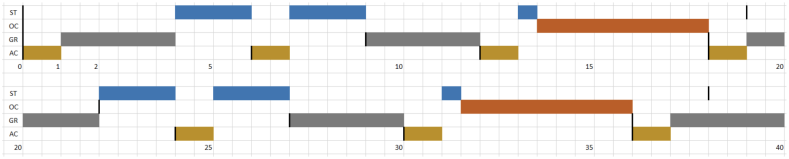

Figure 1:

1.3 c

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \times (2^{1/n} - 1)$$

$$\frac{4.5}{19} + \frac{4.5}{22} + \frac{3}{9} + \frac{1}{6} + \frac{4}{15} \leq 5 \times (2^{1/5} - 1) \longrightarrow 1.2 \leq 0.74$$

The above inequality does not hold and therefore it does not yield any conclusion; Hence, we need to draw the system to see if it is schedulable or not. According to the following figure the taskset is not schedulable because at least one task misses its deadline.

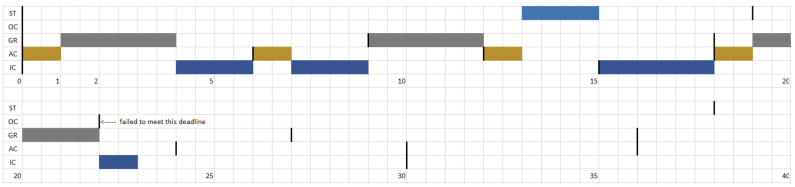

Figure 2:

1.4 d

We just need to keep the utilization of each CPU less than $n \times (2^{1/n} - 1)$

CPU 1: star tracking, orbit correction, Atomic clock sampling

CPU 2: ground-station, inter-satellite communication

1.5 e

The taskset is schedulable as shown in the following figure:

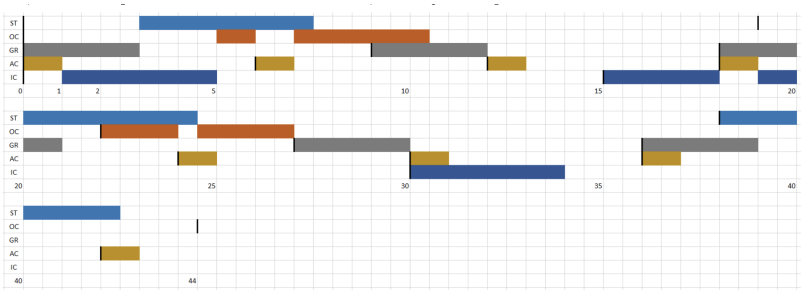

Figure 3:

1.6 d

Global RM would chooses up to two task with the shortest period. The task set is schedulable under this policy as shown below:

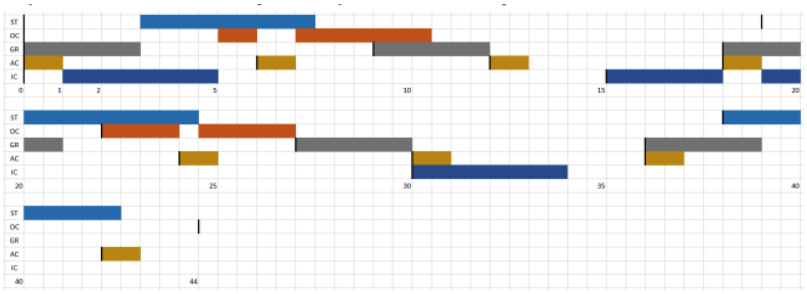

Figure 4:

2 Question 2

2.1 a

All items in the flag array must be initialized to False.

2.2 b

It has no impact. Process i still can access the critical section

2.3 c

No, The first 6 lines can be produced, but line 7 cannot be produced. Line 7 says j can go to critical section while falg[i] is already set to true in line 4, which cannot happen according the code

2.4 d

Yes, this will result in both processes to be in critical section at the same time as shown in the following figure:

Step	Process i	Process j
1	flag[i] = true	
2	print_line("Process i: Entering CS")	
3	while(flag[j]) <--- this is false	
4	print_line("Process i: Inside CS")	
5		flag[j] = true
6		print_line("Process j: Entering CS")
7		while (flag[i])
8		if (turn == i)
9		flag[j] = true while (turn == i)
10	turn = j	
11	flag[i] = false	
12	flag[i] = true	
13	print_line("Process i: Entering CS")	
14		flag[j] = true
15		print_line("Process j: Inside CS")
16		turn = i
17		flag[j] = false

Figure 5:

2.5 e

```
Semaphore mutEx = 1;
Process i:
    Repeat:
        /* REMAINDER SECTION */
        wait(mutEx)

        /* CRITICAL SECTION — BEGIN */
        ...
        print_line("Process i: Inside CS");
        ...
        /* CRITICAL SECTION — END */

        signal(mutEx)

        /* REMAINDER SECTION */
    Forever

Process j:
    Repeat:
        /* REMAINDER SECTION */
        wait(mutEx)

        /* CRITICAL SECTION — BEGIN */
        ...
        print_line("Process j: Inside CS");
        ...
        /* CRITICAL SECTION — END */

        signal(mutEx)

        /* REMAINDER SECTION */
    Forever
```

Figure 6:

3 Question 3

3.1 a

Listing 1: roommate

```

semaphore plates = p;
semaphore pizza = 8;
while (true) {
    wait(plate);
    wait(pizza);
    eat_pizza();
    signal(plate)
}
```

3.2 b

Listing 2: roommate

```

semaphore plates = p;
semaphore pizza = 8;
semaphore knife = 3;
semaphore fork = 5;
while (true) {
    wait(plate);
    wait(knife);
    wait(fork);
    wait(pizza);
    eat_pizza();
    signal(fork)
```

```
13      signal(knife)
14      signal(plate)
15  }
```

3.3 c

Listing 3: roommate

```
1
2   semaphore plates = p;
3   semaphore pizza = 8;
4   semaphore knife = 3;
5   semaphore fork = 5;
6   semaphore MamaJane = 0;
7   semaphore mutex = 1;
8   int num_pizza = 0; // the number of eaten slices
9
10  process roommate:
11      while (true) {
12          wait(plate);
13          wait(knife);
14          wait(fork);
15          wait(pizza);
16          wait(mutex)
17          num_pizza ++;
18          if (num_pizza == 8) {
19          signal(MamaJane)
20          }
21          signal(mutex);
22          eat_pizza();
23          signal(fork)
24          signal(knife)
25          signal(plate)
26      }
27
28  process MamaJane:
29  while (true) {
30          wait(MamaJane);
31          wait(mutex)
32          for(int i = 0; i < 8; i++) {
33          signal(pizza)
34          }
35          num_pizza = 0
36          signal(mutex)
37      }
```