

Spencer Shimko
 CMSC 421
 Homework #2

1. Exercise problem 6.3 from the text (page 185) [20 points]

a)

FCFS:

P1	P2	P3	P4	P5
0-10	11	12-13	14	15-19

SJF:

P2	P4	P3	P5	P1
0-1	1-2	3-4	5-9	10-19

Non-preemptive Priority:

P2	P5	P1	P3	P4
0-1	2-6	7-16	17-18	19

RR:

P1	P2	P3	P4	P5	P1	P3	P5	P1	P5	P1	P5	P1	P5	P1...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15-19

b)

FCFS:

-P1 = 10
 -P2 = 11
 -P3 = 13
 -P4 = 14
 -P5 = 19

SJF:

-P1 = 19
 -P2 = 1
 -P3 = 4
 -P4 = 2
 -P5 = 9

Priority:

-P1 = 16
 -P2 = 1
 -P3 = 18
 -P4 = 19
 -P5 = 6

RR:

-P1 = 19
 -P2 = 2
 -P3 = 7
 -P4 = 4
 -P5 = 14

c)

FCFS:

-P1 = 0
 -P2 = 10
 -P3 = 11
 -P4 = 13
 -P5 = 14

SJF:

-P1 = 9
-P2 = 0
-P3 = 2
-P4 = 1
-P5 = 4

Priority:

-P1 = 6
-P2 = 0
-P3 = 16
-P4 = 18
-P5 = 1

RR:

-P1 = 9
-P2 = 1
-P3 = 5
-P4 = 3
-P5 = 9

d)

SJF: 3.2

2. Exercise problem 6.4 from the text (pages 185–186) [20 points]

a)

$$(8 + (12-0.4) + (13-1)) / 3 = 10.5333$$

b)

$$(8 + (9-1) + (13-0.4)) / 3 = 9.5333$$

c)

6.86

3. Moonbucks coffee shop serves two kinds of coffee: hot-coffee and cold-coffee. Mr. Moon who makes coffee serves only one customer at a time and remaining customers will have to wait. When there are no customers, Mr. Moon sleeps. After serving a hot-coffee-customer, a cold-coffee customer gets served next only if there is no hot-coffee customer waiting. Similarly after serving a cold-coffee customer, a hot-coffee customer gets served only if there is no cold-coffee customer waiting. If Mr. Moon is asleep, the customer wakes him up. Write a monitor to coordinate Mr. Moon, the hot-coffee customers and the cold-coffee customers. [20 points]

```
monitor coffeeshop
    boolean busy, noCustomers;
    condition hot, cold;
    Queue coldCust;
    Queue hotCust;
    enum {HOT,COLD} custType;

    void serveCustomers ( first ) {
        if (first == HOT){
            while ( !hotCust.isEmpty() ){
                serveHot();
                stopServeHot();
            }
        }
    }
}
```

```

    }
    } else
        while ( !coldCust.isEmpty() ){
            serveCold();
            stopServeCold();
        }
        serveCustomers( HOT);
    }
    sleep( );
}
void serveHot ( ){
    if (busy)
        hot.wait()
    busy = true;
}
void stopServeHot ( ) {
    busy = false;
    hot.signal();
    hotCust.dequeue();
}
void serveCold ( ){
    if (busy)
        cold.wait()
    busy = true;
}
void stopServeCold ( ) {
    busy = false;
    cold.signal();
    coldCust.dequeue();
}
void sleep ( ){
    while ( noCustomers );
    newCustomer ( customer );
    serve(customer.type);
}
void newCustomer( customer ){
    if (customer.wantsHot)
        hotCust.enqueue (customer);
    if (customer.wantsCold)
        coldCust.enqueue (customer);
}
}

```

4. Consider the following snap shot of a system:

Process	Allocation					Maximum					Available				
	P0	1	0	2	1	1	1	1	2	1	3	0	0	x	1
P1	2	0	1	1	1	2	2	2	1	1					
P2	1	1	0	1	0	2	1	3	1	0					
P3	1	1	1	1	0	1	1	2	2	1					

When $x = 10$, show that this is safe state. [5 points]

Need (Max[i, j] - Allocation[i, j])

P0	0	1	0	0	2
P1	2	2	1	0	0
P2	1	0	3	0	0
P3	0	0	1	1	1

This is a safe state because there exists a safe sequence:
<P3, P2, P1, P0>

What is the smallest value of x for which this is a safe state? Prove. [15 points]

Need (Max[i, j] - Allocation[i, j])

P0	0	1	0	0	2
P1	2	2	1	0	0
P2	1	0	3	0	0
P3	0	0	1	1	1

So this need chart indicates that the highest requirement for X is 3. Let's start with X as low as possible and work up from there as required.

Assume X=0

-Given the above safe sequence we can see that P3 runs first and releases it's resources back to the system. It needs X=1 to run.

Assume X=1

-P3 allocates the 1 from X and then releases this X back to the system along with P3's pre-allocated "1". So after P3 runs X=2.

-Now we try to run P2 but P2 needs X=3 to run. So we must bump X to 3.

Answer: X=3 since P2 requires the highest X that is not available at it's execution time.