Assignment 2

Gülbarin Maçin

69163

Problem 1)



a.1) FCFS

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  | P4 |  | P5 |

0 8 10 22 24 28 30 32 34 44

a.2) SJF

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P4 |  | P3 |  | P1 |  | P5 |  | P2 |

0 2 4 8 10 18 20 30 32 44

a.3) Non-preemptive Priority

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P5 |  | P2 |  | P4 |  | P1 |  | P3 |

0 10 12 24 26 28 30 38 40 44

a.4) RR (TQ=3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  | P4 |  | P5 |  | P1 |  | P2 |  | P3 |  | P5 |  | P1 |  | P2 |  | P5 |  | P2 |  | P5 |

0 3 5 8 10 13 15 17 19 22 24 27 29 32 34 35 37 40 42 44 46 49 51 54 56 59 61 62



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FCFS | SJF | Non-preemptive | RR |
| P1 | 0 | 10 | 30 | 34 (44-8) |
| P2 | 10 | 32 | 12 | 47 (59-12) |
| P3 | 24 | 24 | 44 | 31 (35-4) |
| P4 | 30 | 0 | 26 | 15 (17-2) |
| P5 | 34 | 20 | 0 | 52 (62-10) |
| Average | 19.6 | 17.2 | 21.6 | 45.8 |

SJF has the minimal waiting avreage.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FCFS | SJF | Non-preemptive | RR |
| P1 | 8 | 18 | 38 | 44 |
| P2 | 22 | 44 | 24 | 59 |
| P3 | 28 | 8 | 44 | 35 |
| P4 | 32 | 2 | 28 | 17 |
| P5 | 44 | 30 | 38 | 62 |
| Average | 26.8 | 20.4 | 34.4 | 43.4 |

Problem 2)



a.1) FCFS

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  | P4 |  | P5 |

0 10 10.5 25.5 26 31 31.5 34 34.5 47

CPU utilisation= 47/ (47+4\*0.5) = 0.95

a.2) SJF

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P4 |  | P3 |  | P1 |  | P5 |  | P2 |

0 2.5 3 8 8.5 18.5 19 31.5 32 47

CPU utilisation= 47/ (47+4\*0.5) = 0.95

a.3) non-preemptive

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P5 |  | P2 |  | P4 |  | P1 |  | P3 |

0 12.5 13 28 28.5 31 31.5 41.5 42 47

CPU utilisation= 47/ (47+4\*0.5) = 0.95

a.4) RR (TQ=3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  | P4 |  | P5 |  | P1 |  | P2 |  | P3 |  | P5 |  | P1 |  | P2 |  | P5 |  | P1 |  | P2 |  | P5 |  | P2 |  | P5 |

0 3 3.5 6.5 7 10 10.5 12.5 13 16 16.5 19.5 20 23 23.5 25.5 26 29 29.5 32.5 33 36 36.5 39.5 40 41 41.5 44.5 45 48 48.5 51.5 52 52.5

CPU utilisation= 47/ (47+16\*0.5) = 0.85

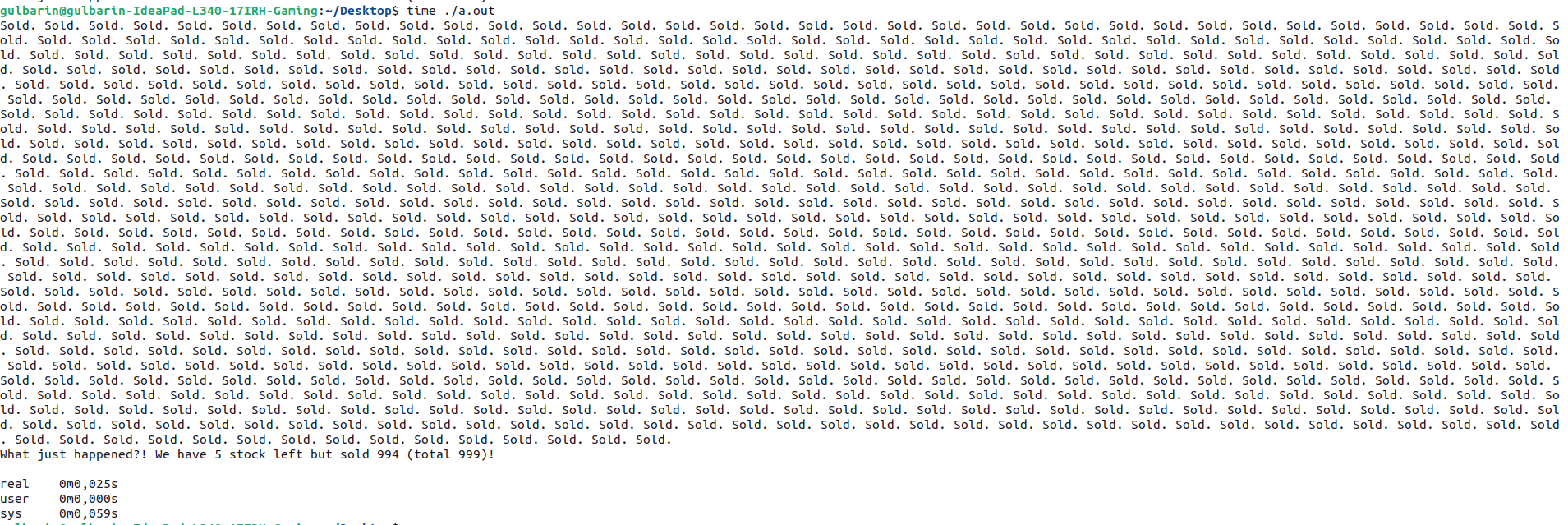
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FCFS | SJF | Non-preemptive | RR |
| P1 | 0 | 8.5 | 31.5 | 31 |
| P2 | 10.5 | 32 | 13 | 36.5 |
| P3 | 25.5 | 3 | 42 | 20.5 |
| P4 | 31.5 | 0 | 28.5 | 10 |
| P5 | 34.5 | 19 | 0 | 40 |
| Average | 20.4 | 12.5 | 23 | 27.6 |

Average waiting time is less than from problem 1 (b). Because the context switch overhead is less than 1. So, it affects the waiting time.

Problem 3)

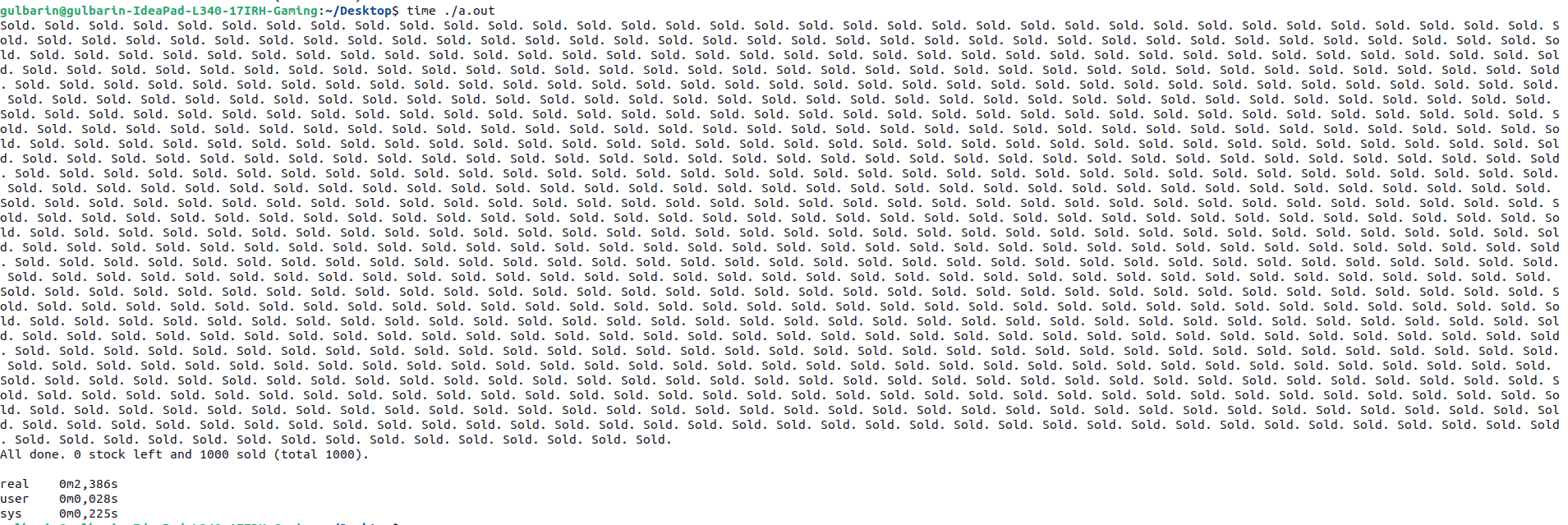
The code part is implemented on the problem3.c file.

Part A)



Part B)

I used lock/unlock in the if section when the stock and sell is changed. Because this part is the critical section so we should lock and unlock.



The time ./a.out results is also in the screenshots.

Problem 4)

monitor deliveryMonitor {

struct order {

int id;

int dist;

int size;

};

int available\_cars; //number of available delivery guys

struct order waiting\_orders[M];

int num\_waiting; //number of orders currently waiting

// !you can introduce shared variables

int available\_cars;

sem\_t mutex = 1;

!implement this function

void request\_delivery(int order\_id, int distance, int size) {

waiting\_orders[num\_waiting].id = order\_id;

waiting\_orders[num\_waiting].dist = distance;

waiting\_orders[num\_waiting].size = size;

wait(&mutex);

num\_waiting++;

signal(&mutex);

waiting\_orders.sort();

available\_cars--;

wait(&mutex);

num\_waitiing--;

signal(&mutex);

}

void release\_car() {

available\_cars++;

broadcast(available\_cars);

}

void initialize(){

available\_cars = 5;

num\_waiting = 0;

}

}

1. Prioritization of distance policy creates risk. The long distance will be more important than the short distance delivery. It makes the short distance delivery failing so it creates a risk.

Problem 5)

1. Max - Allocation

|  |  |
| --- | --- |
|  | Need |
| P1  P2  P3  P4  P5 | A B C D  0 6 4 2  0 0 2 0  1 0 0 2  0 0 0 0  0 7 5 0 |

1. Safety Control

Work = Available

For P1: Need: (0, 6, 4, 2)

Work: (1, 5 ,2, 0)

Need <= Work = False

So, the system ill move to the next process.

For P2: Need: (0, 0, 2, 0)

Work: (1, 5, 2, 0)

Need <= Work = True

Work = Work + Allocation

= 1, 5, 2, 0 + 0, 6, 3, 2

= 1, 11, 5, 2 (New Work)

For P3: Need (1, 0, 0, 2)

Work: ( 1, 11, 5, 2)

Need <= Work = True

Work = Work + Allocation

= 1, 11, 5, 2 + 2, 3, 5, 6

= 3, 14, 10, 8 (New Work)

For P4: Need (0, 0, 0, 0)

Work: (3, 14, 10, 8)

Need <= Work = True

Work = Work + Allocation

= 3, 14, 10, 8 + 0, 0, 1, 2

= 3,14,11,10 (New Work)

For P5: Need: (0, 7, 5, 0)

Work: (3,14,11,10)

Need <= Work = True

Work = Work + Allocation

= 3, 14, 11, 10 + 1, 7, 5, 0

= 4,21,16,10 (New Work)

All the system process is allocated to the safe state. So, we can say the system is in a safe state.

1. Request: (0, 3, 3, 0)

Available: (1, 5, 2, 0)

If request <= available, the process request granted immediately

The request <= available = False so the request is not granted immediately.