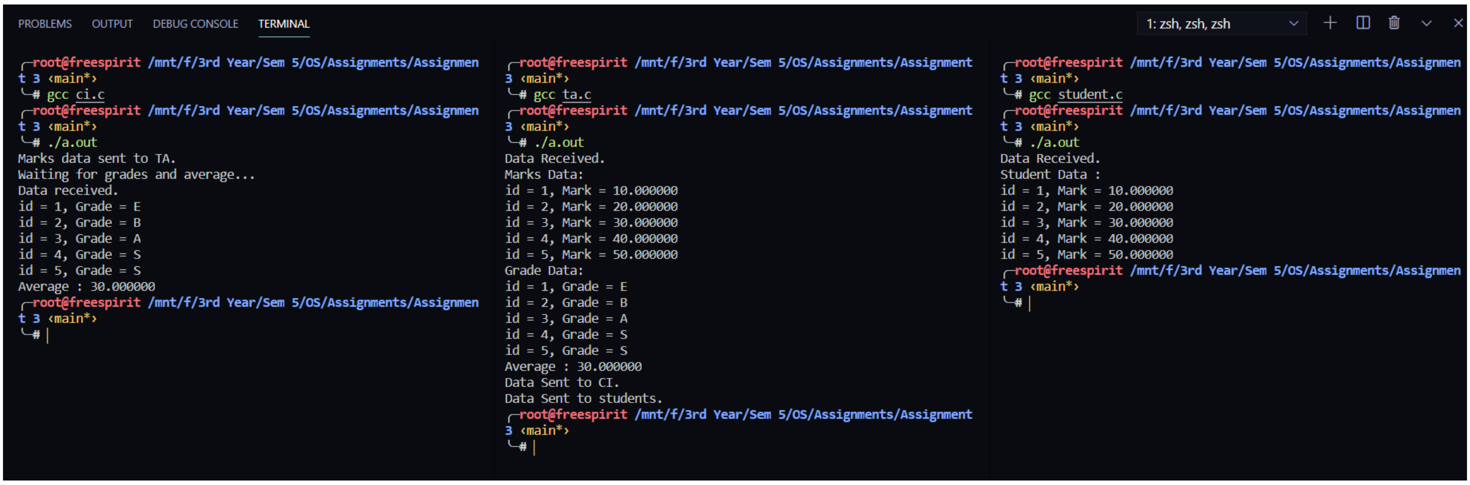
OS Assignment 3

Readme

Meenakshi Madhu – B180390CS

Q1. Message Queue



```
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# gcc ci.c
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# ./a.out
Marks data sent to TA.
Waiting for grades and average...
Data received.
id = 1, Grade = E
id = 2, Grade = B
id = 3, Grade = A
id = 4, Grade = S
id = 5, Grade = S
Average : 30.000000
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# |
```

```
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment
3 ‹main*›
└─# gcc ta.c
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment
3 ‹main*›
└─# ./a.out
Data Received.
Marks Data:
id = 1, Mark = 10.000000
id = 2, Mark = 20.000000
id = 3, Mark = 30.000000
id = 4, Mark = 40.000000
id = 5, Mark = 50.000000
Grade Data:
id = 1, Grade = E
id = 2, Grade = B
id = 3, Grade = A
id = 4, Grade = S
id = 5, Grade = S
Average : 30.000000
Data Sent to CI.
Data Sent to students.
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment
3 ‹main*›
└─# |
```

```
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# gcc student.c
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# ./a.out
Data Received.
Student Data :
id = 1, Mark = 10.000000
id = 2, Mark = 20.000000
id = 3, Mark = 30.000000
id = 4, Mark = 40.000000
id = 5, Mark = 50.000000
┌─root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignmen
t 3 ‹main*›
└─# |
```

Programs:

- ci.c – Course Instructor
- ta.c – Teaching Assistant
- student.c – Student

System calls used:

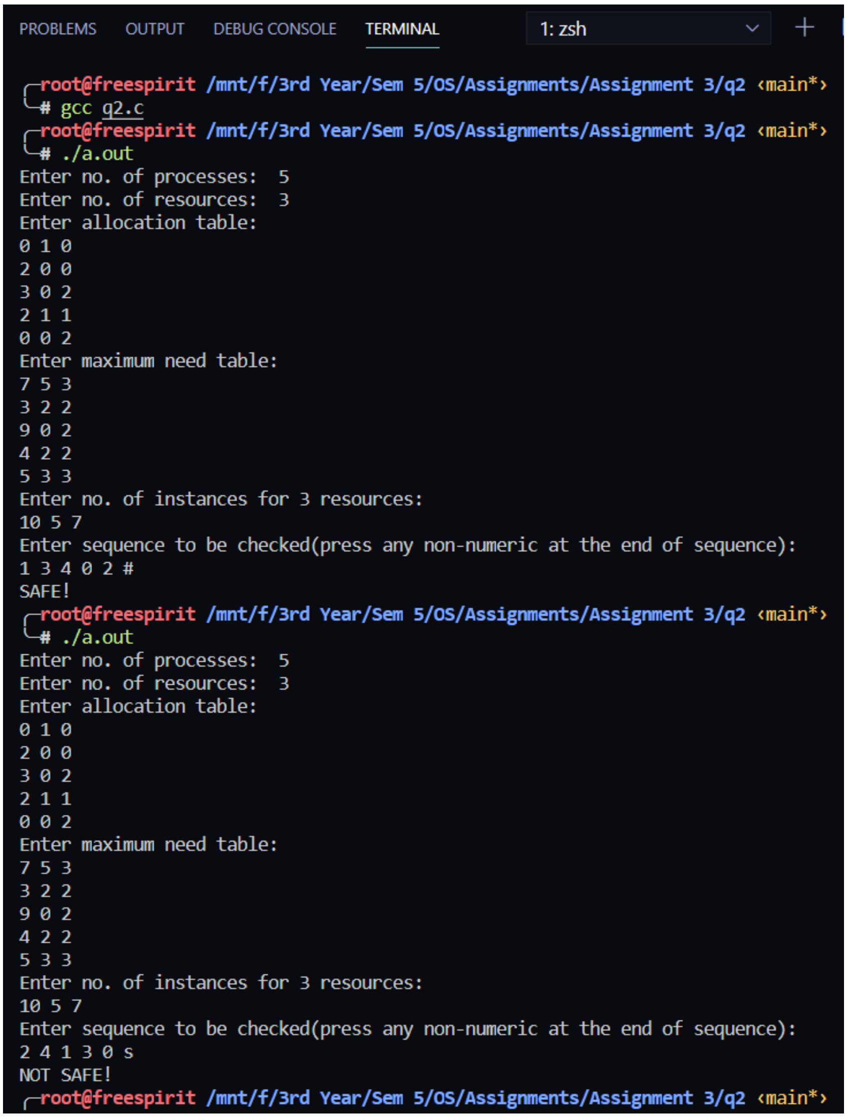
- ftok(): to generate a unique key.
- msgget(): either returns the message queue identifier for newly created message queues or the identifier for an existing queue with the same key value.
- msgsnd(): Data is placed on to a message queue
- msgrcv(): messages are retrieved from a queue.
- msgctl(): It performs various operations on a queue. Here it is used to destroy message queue.

Input array used:-

```
inputmarks[5] = {10, 20, 30, 40, 50};
```

Here, the course instructor(CI) sends these marks to the Teaching assistant (TA) through the message queue, with a message-type 10. The TA receives these marks and calculates grades, and average marks. This new data is again written to the message queue with message-type as 100. The marks are written as well, with message-types 1-5 according to the student-id. The CI receives the grades, and average marks where as the student only receives their marks according to their student-id.

Q2. Banker's Algorithm



```
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q2 ‹main*›
└# gcc q2.c
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q2 ‹main*›
└# ./a.out
Enter no. of processes:  5
Enter no. of resources:  3
Enter allocation table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter maximum need table:
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
Enter no. of instances for 3 resources:
10 5 7
Enter sequence to be checked(press any non-numeric at the end of sequence):
1 3 4 0 2 #
SAFE!
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q2 ‹main*›
└# ./a.out
Enter no. of processes:  5
Enter no. of resources:  3
Enter allocation table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter maximum need table:
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
Enter no. of instances for 3 resources:
10 5 7
Enter sequence to be checked(press any non-numeric at the end of sequence):
2 4 1 3 0 s
NOT SAFE!
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q2 ‹main*›
```

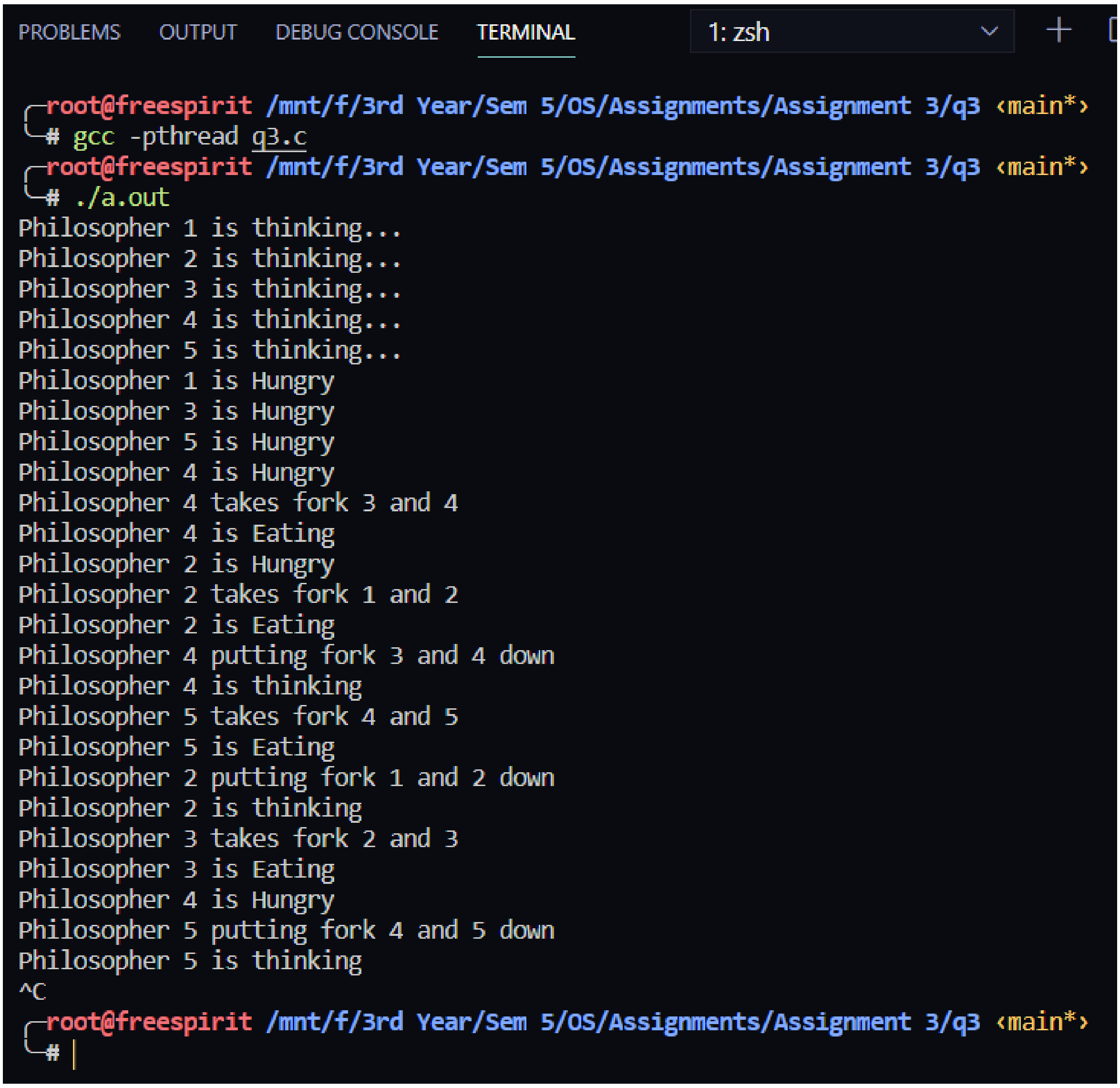
Input:

- No. of processes – n
- No. of resources – m
- Allocation table – AllocTable[n][m]
- Maximum Need table – MaxTable[n][m]
- Total no. of instances for m resources – Total[m]
- Sequence of processes

Using the available data, remaining resources needed for each process is calculated. According to this resource request for processes in the sequence, the program checks whether the request can be granted or not.

If for a particular process, the available resources are not enough, the given sequence is not a SAFE STATE. Else is it a SAFE STATE.

Q3. Dining Philosopher’s problem using semaphores



```
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q3 ‹main*›
└# gcc -pthread q3.c
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q3 ‹main*›
└# ./a.out
Philosopher 1 is thinking...
Philosopher 2 is thinking...
Philosopher 3 is thinking...
Philosopher 4 is thinking...
Philosopher 5 is thinking...
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 4 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
^C
┌root@freespirit /mnt/f/3rd Year/Sem 5/OS/Assignments/Assignment 3/q3 ‹main*›
└# |
```

Here, the basic algorithm for an instance of the Dining Philosopher’s problem is implemented.

No. of philosophers = 5

An infinite loop is generated.