National Textile University, Faisalabad

Department of Computer Science

Name:	Nimra Tanveer
Class:	BSSE A 5th
Registration No:	23-NTU-CS-1201
Assignment:	After-mid Homework-1
Course Name:	Operating Systems – COC 3071
Submitted To:	Sir Nasir Mahmood

Operating System

After-mid Homework-1

Part 1: Semaphore theory

Q # 1: Semaphore = 7

Wait ( ) = 10   (− 1 from semaphore)

Signal ( ) = 4   (+ 1 into semaphore)

Solution:

7 − 10 = −3   (process block)

−3 + 4 = 1   (extra signal)

Semaphore value = 1

Q# 2:

Semaphore = 3

wait ( ) = 5

Signal ( ) = 6

Solution:

3 − 5 = −2 (process block)

−2 + 6 = 4

Semaphore value = 4

Q# 3:

Semaphore value = 0
Signal ( ) = 8
Wait ( ) = 3

$0 + 8 = 8$
$8 - 3 = 5$

Final value = 5

Q# 4:

Semaphore = 2
Wait ( ) = 5

$2 - 5 = -3$

(a) 2 process enter in the critical section.

(b) 3 process are blocked

Q# 5:

Semaphore = 1
Wait ( ) = 3
Signal( ) = 1

(a) $1 - 3 = -2$

Thus, 2 process are blocked.

(b) $-2 + 1 = -1$

Thus, Final value is -1.

Q # 6:

Semaphore = 3

Start = 3
Wait (S) = $3 - 1 \overset{(3-1)}{=} 2$ (Here, 2 blocked and 1 enter in CS)
wait (S) = $2 - 1 = 1$ (Here, 1 blocked and 1 enter in CS)
Signal (S) = $1 + 1 = 2$ (Here, 2 in process and 0 blocked)
wait (S) = $2 - 1 = 1$ (Here, 1 is blocked and 3 enter in CS)
wait (S) = $1 - 1 = 0$ (Here, 4 enter in CS and 1 blocked)

(a) 4 processes are enter in critical section

(b) Final value is 0.

Q # 7:

Start (S) = 1
wait (S) = $1 - 1 = 0$
wait (S) = $0 - 1 = -1$ (blocked)

signal(S) = -1 + 1 = 0
signal (S) = 0 + 1 = 1

(a) 1 process is blocked

(b) Final value is 1.

Q# 8:

Semaphore = 1
wait ( ) = 5
signal ( ) = 0

1 - 5 = -4 (Here, 4 blocked and 1 enter in CS)

(b) 4 processes are blocked
(a) 1 process is enter in critical section.

Q # 9:

Semaphore = 4
wait ( ) = 6

4 - 6 = -2 (blocked)

(a) 4 are in processes
(b) 2 are in blocked

Q # 10:

semaphore (S) = 2

(a) wait (S) = 2 − 1 = 1
wait (S) = 1 − 1 = 0
wait (S) = 0 − 1 = −1 (blocked)
signal (S) = −1 + 1 = 0
signal (S) = 0 + 1 = 1
wait (S) = 1 − 1 = 0

(b) Maximum 1 process is blocked

Q # 11:

Semaphore = 0
wait ( ) = 3
signal ( ) = 5

0 − 3 = −3 (blocked)
5 − 3 = 2

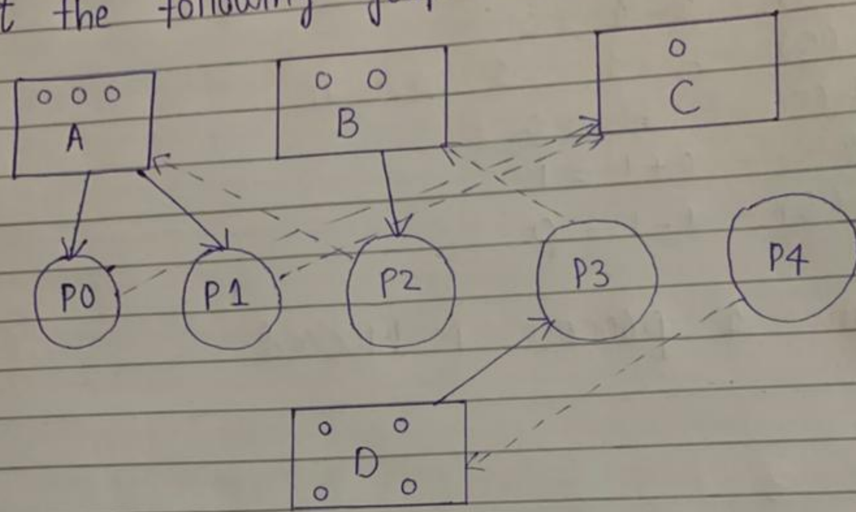
(a) First 3 wake up
Remaining 2 increase semaphore

Thus, 3 processes are wake up

(b) Final value is 2.

Part 3: RAG

Convert the following graph into



Step 1:- Resources

- A ⇒ 3 instances
- B ⇒ 2 instances
- C ⇒ 1 instances
- D ⇒ 4 instances

Step 2:- Processes

- P0
- P1
- P2
- P3
- P4

Step-3: Allocation Edges R → P

From Graph → Solid arrows

- A → P0
- A → P1
- B → P2
- D → P0
- D → P3

Step-4:- Request Edges P → R

- P0 → C
- P1 → B
- P2 → C
- P3 → B
- P4 → D

Step-5: Allocation Matrix

Process	A	B	C	D
P0	2	0	0	1
P1	1	0	0	0
P2	0	1	0	0
P3	0	0	0	1
P4	0	0	0	0

Request Matrix

Process	A	B	C	D
P0	0	0	1	0
P1	0	1	0	0
P2	0	0	1	0
P3	0	1	0	0
P4	0	0	0	1

Step-6: Calculation of Avaliable Resources

Total	Allocated	Resources	Avaliable
A = 3	A = 2+1 = 3	A	0
B = 2	B = 1	B	1
C = 1	C = 0	C	1
D = 4	D = 1+1 = 2	D	2

Avaliable vector: A vector that is calculated by a subtracting allocated instances from total instance of each resource.

Hence, avaliable vector is (0, 1, 1, 2).

Part 4: Banker's Algorithm

Total Existing Resources:

Total			
A	B	C	D
6	4	4	2

Max – Allocation

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	1	1	3	2	1	1				
P1	1	1	0	0	1	2	0	2				
P2	1	0	1	0	3	2	1	0				
P3	0	1	0	1	2	1	0	1				

Questions:

Q# 1: Avaliable Vector

Resources	Allocated	Total	T-A
A	4	6	2
B	2	4	2
C	2	4	2
D	2	2	0

Q#2: Need Matrix

Need matrix = Max – Allocation

Process	A	B	C	D
P0	1	2	0	0
P1	0	1	0	2
P2	2	2	0	0
P3	2	0	0	0

Q# 3: Safety Check

Avalaible vector (2,2,2,0) ⇒ Working Array

- Condition

Need – Process ≤ Available, they are execute

Now, checking Process one by one

For Process P0

Need (1,2,0,0), Avaliable (2,2,2,0)

Result Possible: P0 complete → Return Resources

Banker's Algorithm process does not follow order just follow resoure avalibility.

New Avaliable = (2,2,2,0) + (2,0,1,1) ↳ (Allocation)

= (4,2,3,1)

Safe Sequence: P0

For Process P1

Need (0,1,0,2) , Avaliable (4,2,3,1)

Here, we skip because D is less.

For Process P2

Need (2,2,0,0) , Avaliable (4,2,3,1)

Result Possible: P2 complete → Return Resources

New Avaliable = (4,2,3,1) + (1,0,1,0) ↳ (Allocation)
= (5,2,4,1)

Safe Sequence: P0 → P2

For process P3

Need (2,0,0,0) , Avaliable (5,2,4,1)

Result Possible: P3 complete → Return Resources

New Avaliable = (5,2,4,1) + (0,1,0,1) → Allocation
= 5,3,4,2

Safe Sequence: P0 → P2 → P3

Again For Process P1

Need (0,1,0,2) , Avaliable (5,3,4,2)

New Avaliable = (5,3,4,2) + (1,1,0,0)
= (6,4,4,2)

Safe Sequence : P0 → P2 → P3 → P1

yes, system is in safe state because at every step at least one process have need avaliable resoure is less or equal required that's why every process deadlock is complete without deadlock.

Part 2:

Semaphore Coding Consider the Producer–Consumer problem

using semaphores as implemented in Lab-10 (Lab-plan attached). Rewrite the program in your own coding style, compile and execute it successfully, and explain the working of the code in your own words. Submission Requirements:

- Your rewritten source code
- A brief description of how the code works
- Screenshots of the program output showing successful execution

Working of code

- The buffer has **5 slots**
- empty semaphore counts empty spaces
- full semaphore counts filled spaces
- mutex avoids race condition

Producer:

- Waits if buffer is full
- Locks buffer
- Produces item and stores it
- Unlocks buffer
- Signals consumer

Consumer:

- Waits if buffer is empty
- Locks buffer
- Consumes item
- Unlocks buffer
- Signals producer

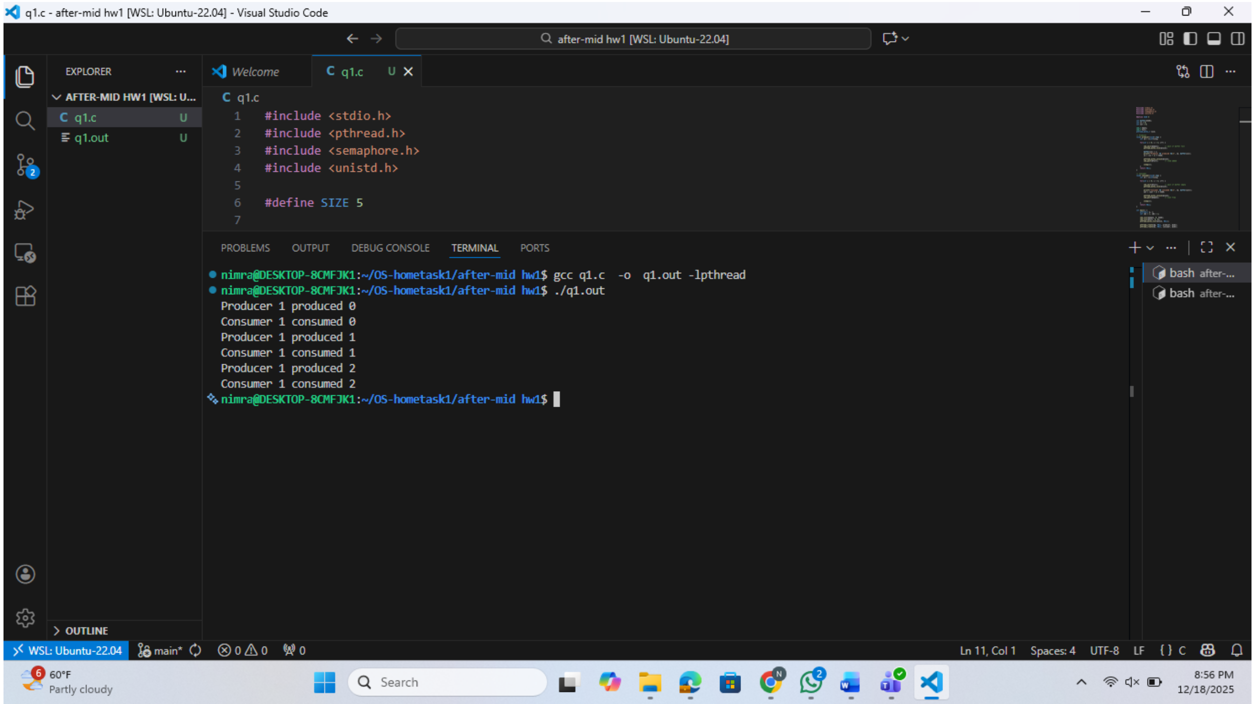
Code

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>


#define SIZE 5
```

```
int buffer[SIZE];
int in = 0;
int out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t lock;

// Producer
void* producer(void* arg) {
    int id = *(int*)arg;

    for(int i = 0; i < 3; i++) {

        sem_wait(&empty);      // wait if buffer full
        pthread_mutex_lock(&lock);

        buffer[in] = i;
        printf("Producer %d produced %d\n", id, buffer[in]);
        in = (in + 1) % SIZE;

        pthread_mutex_unlock(&lock);
        sem_post(&full);       // item added

        sleep(1);
```

```
    }
    return NULL;
}

// Consumer
void* consumer(void* arg) {
    int id = *(int*)arg;

    for(int i = 0; i < 3; i++) {

        sem_wait(&full);      // wait if buffer empty
        pthread_mutex_lock(&lock);

        printf("Consumer %d consumed %d\n", id, buffer[out]);
        out = (out + 1) % SIZE;

        pthread_mutex_unlock(&lock);
        sem_post(&empty);     // slot free

        sleep(1);
    }
    return NULL;
}

int main() {
    pthread_t p, c;
```

```c
    int id1 = 1, id2 = 1;

    sem_init(&empty, 0, SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&lock, NULL);

    pthread_create(&p, NULL, producer, &id1);
    pthread_create(&c, NULL, consumer, &id2);

    pthread_join(p, NULL);
    pthread_join(c, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&lock);

    return 0;
}
```


q1.c - after-mid hw1 [WSL: Ubuntu-22.04] - Visual Studio Code
after-mid hw1 [WSL: Ubuntu-22.04]
EXPLORER
AFTER-MID HW1 [WSL: U...
q1.c U
q1.out U
Welcome
q1.c U
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define SIZE 5
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/after-mid hw1$ gcc q1.c -o q1.out -lpthread
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/after-mid hw1$ ./q1.out
Producer 1 produced 0
Consumer 1 consumed 0
Producer 1 produced 1
Consumer 1 consumed 1
Producer 1 produced 2
Consumer 1 consumed 2
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/after-mid hw1$
bash after-...
bash after-...
OUTLINE
WSL: Ubuntu-22.04
main*
Ln 11, Col 1 Spaces: 4 UTF-8 LF { } C
60°F
Partly cloudy
Search
8:56 PM
12/18/2025