National Textile University

Department of Computer Science

Subject:

Operating system

Submitted to:

Sir Naseer

Submitted by:

Ayema

Reg number:

23-NTU-CS-1142

Assignment no. :

02

Semester: 5Th

Part 1

Question 1:

A counting semaphore is initialized to 7. If 10 wait() and 4 signal() operations are performed, find the final value of the semaphore.

Initial value = 7
After 10 wait → 7 − 10 = −3
After 4 signal → −3 + 4 = 1

Final semaphore value = 1

Question 2:

A semaphore starts with value 3. If 5 wait() and 6 signal() operations occur, calculate the resulting semaphore value.

Initial value =3

After 2 wait →3 – 5 = -2

After 6 signal → -2 + 6 = 4

Final semaphore value =4

Question 3:

A semaphore is initialized to 0. If 8 signal() followed by 3 wait() operations are executed, find the final value.

Initial value =0

After 3 wait →0 – 3 = -3

After 8 signal → -3 + 8 = 5

Final semaphore value =5

Question 4:

A semaphore is initialized to 2. If 5 wait() operations are executed:
a) How many processes enter the critical section?
b) How many processes are blocked?

Initial value = 2

After 5 wait → 2-5 =-3

a). 2 process enter critical section

b). 3 process get blocked

Question 5:

A semaphore starts at 1. If 3 wait() and 1 signal() operations are performed:

a) How many processes remain blocked?

b) What is the final semaphore value?

Initial value = 1
Operations: 3 wait(), 1 signal()

Step-by-step:

- wait → 1 → 0 (1 enters)
- wait → blocks (1)
- wait → blocks (2)
- signal → wakes 1 blocked process

Remaining blocked = 1

a) Blocked processes = 1

b) Final semaphore value = 0

Question 6:

semaphore S = 3;

wait(S);

wait(S);

signal(S);

wait(S);

wait(S);

a) How many processes enter the critical section?

b) What is the final value of S?

Operation	S	Result
wait	3 → 2	enter
wait	2 → 1	enter
signal	1 → 2	resource released

wait	2 → 1	enter
wait	1 → 0	enter

Question 7:

semaphore S = 1;

wait(S);

wait(S);

signal(S);

signal(S);

a) How many processes are blocked?
b) Final value of S?

Initial: S = 1

wait → 1 → 0 (enter)
wait → blocks
signal → wakes blocked process
signal → increments S

Result

- No process remains blocked
- Final value = 1

a) 0 blocked processes
b) S = 1

Question 8:

A binary semaphore is initialized to 1. Five wait() operations are executed without any signal(). How many processes enter the critical section and how many are blocked?

- Binary semaphore allows only 1 process
- Remaining processes must block

Result

- Entered CS = 1
- Blocked = 4

Question 9:

A counting semaphore is initialized to 4. If 6 processes execute wait() simultaneously, how many proceed and how many are blocked?

4 processes proceed

2 processes are blocked

Question 10:

S = 2

wait(S);

wait(S);

wait(S);

signal(S);

signal(S);

wait(S);

Step	S	Status
Initial	2	
wait	1	enter
wait	0	enter
wait	blocked	
signal	0	wakes 1
signal	1	
wait	0	enter

- **Max blocked at any time = 1**

Question 11:

A semaphore is initialized to 0. Three processes execute wait() before any signal(). Later, 5 signal() operations are executed:
a) How many processes wake up?
b) What is the final semaphore value?

Explanation

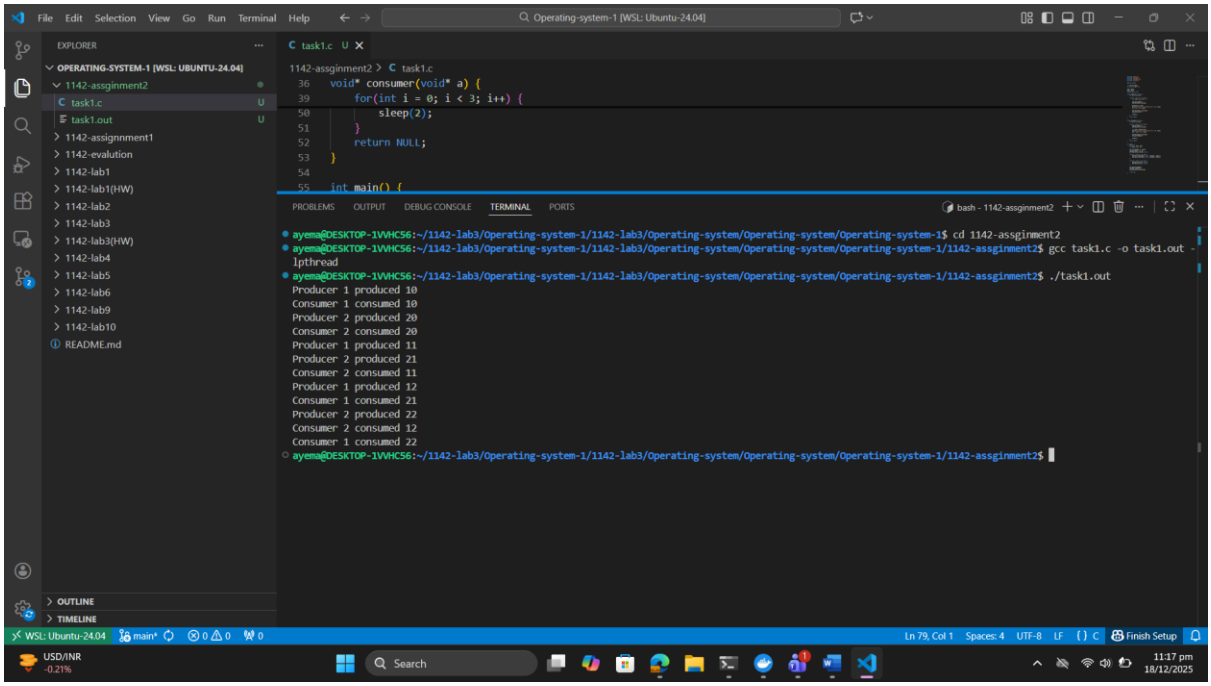
- Initial wait() → all 3 block
- First 3 signal() → wake blocked processes
- Remaining 2 signal() → increase semaphore

Result

- Woken processes = 3
- Final semaphore value = 2

Part 2

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define SIZE 5

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

sem_t empty;
sem_t full;
pthread_mutex_t m;

void* producer(void* a) {
    int id = *(int*)a;

    for(int i = 0; i < 3; i++) {
        int item = id * 10 + i;

        sem_wait(&empty);
        pthread_mutex_lock(&m);

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

        pthread_mutex_unlock(&m);
        sem_post(&full);

        sleep(1);
    }
    return NULL;
}

void* consumer(void* a) {
    int id = *(int*)a;

    for(int i = 0; i < 3; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&m);

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

        pthread_mutex_unlock(&m);
        sem_post(&empty);

        sleep(2);
    }
    return NULL;
}

int main() {
    pthread_t p[2], c[2];
    int id[2] = {1, 2};

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

    for(int i = 0; i < 2; i++) {
        pthread_create(&p[i], NULL, producer, &id[i]);
        pthread_create(&c[i], NULL, consumer, &id[i]);
    }

    for(int i = 0; i < 2; i++) {
        pthread_join(p[i], NULL);
        pthread_join(c[i], NULL);
    }

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

    return 0;
}
```



Explain:

This code lets a producer and a consumer share a buffer safely. The producer adds an item only when space is available, and the consumer removes an item only when one exists. A semaphore controls space and items, while a mutex makes sure only one thread uses the buffer at a time. This keeps everything safe and synchronized.

If we remove mutex

Producer aur consumer ek sath buffer access karenge race condition, wrong output.

If we remove empty semaphore

Producer buffer full hone ke bawajood item add karega buffer overflow.

If we remove full semaphore

Consumer empty buffer se item uthayega buffer underflow.

If we change buffer size

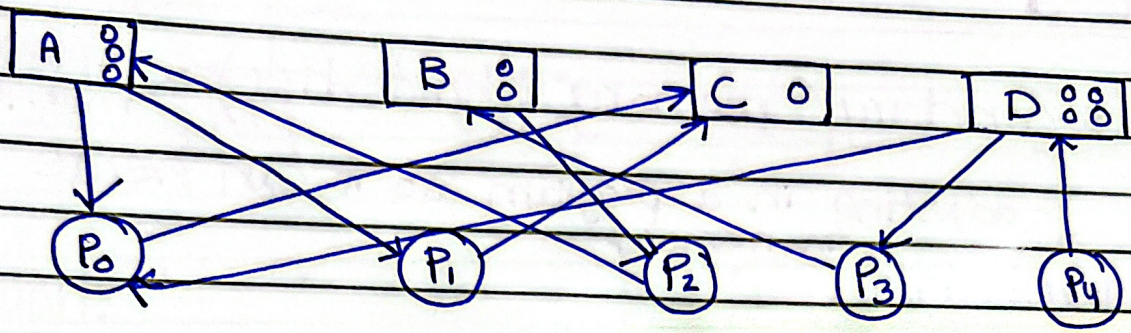
Program phir bhi kaam karega, sirf items zyada ya kam store honge.

If we change sleep time

Speed change hogi, logic same rahega.

Part- (3)

- RAG



- Allocation matrix

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

- Request matrix

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

Total Instances:

A = 3 B = 2 C = 1 D = 4

Part - 4

Total resources

A	B	C	D
6	4	4	2

- Initial matrices:

	A	B	C	D
P_0	2	0	1	1
P_1	1	1	0	0
P_2	1	0	1	0
P_3	0	1	0	1

(Allocation matrix)

	A	B	C	D
P_0	3	2	1	1
P_1	1	2	0	2
P_2	3	2	1	0
P_3	0	1	0	1

(Maximum)

	A	B	C	D
P_0	1	2	0	
P_1	0	1	0	
P_2	2	2	0	
P_3	2	0	0	

(C - A)
chain-al

- Available ~~matrix~~ vector:

Available vectors: total - (sum of all allocation)
process

R	Sum
A	2+1+1+0 = 4
B	0+1+0+1 = 2
C	1+0+1+0 = 2
D	1+0+0+1 = 2

Total

A	6
B	4
C	4
D	2

Available = Total - sum

	A	B	C	D
Resources	6-4	4-2	4-2	2-2
	2	2	2	0

• Safety check: need ≤ work

Initial work = 2, 2, 2, 0

$P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$

P_0

Initial work (available resource)	Need	Checking
2	1	$1 \leq 2$
2	2	$2 \leq 2$
2	0	$0 \leq 2$
0	0	$0 \leq 0$

all are safe, so P_0 can execute.

• New work = work + allocated

New work	old work + allocated
4	2+2
2	2+0
3	2+1
1	0+1

P_3:

Current work	Need	Check
4	2	$2 \leq 4$
2	0	$0 \leq 2$
3	0	$0 \leq 3$
1	0	$0 \leq 1$

New work	work + allocation
4	4+0
3	2+1
3	3+0
2	1+1

Safe to execute.

P_1:

Current work	Need	check
4	0	$0 \leq 4$
3	1	$1 \leq 3$
3	0	$0 \leq 3$
2	2	$2 \leq 2$

So, safe to execute

New work	work + Allocation
5	4+1
4	3+1
3	3+0
2	2+0

P_2:

current work	Need	check
5	2	$2 \leq 5$
4	2	$2 \leq 4$
3	0	$0 \leq 3$
2	0	$0 \leq 2$

So, safe to execute.

New work	Work + allocation
6	5+1
4	4+0
4	3+1
2	2+0

- $P_0 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$

- wait : 1-1=0
  (It means no blocked and 0 semaphor enter)
- wait = 0-1 = -1
  (It means 1 blocked and sempahore value=-1)
- Signal = -1+1=0
  (0 blocked & 0 sempahore)
- signal = 0+1=1
  (0 blocked and 1 sempahore)
- wait = 1-1=0
  (0 blocked & 0 sempahore)

(b)

In this scenario only onces (1 blocked at a time)