National Textile University, Faisalabad

Department of Computer Science

Name	Dawood Saif
Class	SE-5th (A)
Reg. No.	23-NTU-CS-1145
Course	Operating system
Submitted To	Sir Nasir Mehmood
Submission Date	18/12/2025
Assignment	After Mid Home Task 1

After Mids Home Work 1

Part 1: Semaphore theory

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

Answer:

Let the semaphore be represented as keys. The semaphore starts at 7, which means 7 keys are available.

- Each wait () operation takes 1 key, so 10 wait() operations mean −10 keys.
- Each signal () operation returns 1 key, so 4 signal() operations mean +4 keys.

So, Final value=Initial keys−keys taken+keys returned =7−10+4=1
Final semaphore value is 1.

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

Answer:

Let the semaphore be represented as keys.

- The semaphore starts with **3 keys**.
- Each **wait ()** operation takes **1 key**, so **5 wait () operations mean −5 keys**.
- Each **signal ()** operation returns **1 key**, so **6 signal () operations mean +6 keys**.

Now combine everything: Final value=3−5+6=4
The resulting semaphore value is 4.

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

Answer:

Let the semaphore be represented as keys.

- The semaphore starts at 0, which means no keys are available.
- Each signal() operation adds 1 key, so 8 signal() operations add +8 keys.
- Each wait() operation takes 1 key, so 3 wait() operations take −3 keys.

Now combine everything: Final value=0+8−3=5

The final semaphore value is 5.

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?**

Answer:

Let the semaphore be represented as keys.

- The semaphore starts with 2 keys, meaning 2 processes can enter the critical section.
- The first 2 wait() operations get the keys and enter.
- The remaining 3 wait() operations find no keys available, so they must wait (blocked).

A) Number of processes that enter the critical section: 2

B) Number of processes that are blocked: 3

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?**

Answer:

The semaphore starts with 1 key, meaning 1 process can enter the critical section.

- The first wait () operation takes the key , Process 1 enters.
- The next 2 wait () operations find no keys available ,Processes 2 and 3 are blocked.
- Then 1 signal () operation returns a key ,Process 2 gets the key and enters.
- So 3rd process remains blocked.

A) Processes that remain blocked: 1 (Process 3)

B) Final semaphore value: -1.

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?**

Answer:

Semaphore starts with 3 keys, meaning 3 processes can enter.

1. First wait(S):

   Process 1 takes 1 key so,

S = 3 − 1 = 2

- Process 1 enters

2. Second wait(S):

- Process 2 takes 1 key so,

S = 2 − 1 = 1

- Process 2 enters
- signal(S):
- A key is returned so ,

S = 1 + 1 = 2 (increased by 1)

3. Third wait(S):

   Process 3 takes 1 key so

S = 2 − 1 = 1

- Process 3 enters

4. Fourth wait(S):

   Process 4 takes 1 key so

S = 1 − 1 = 0

- Process 4 enters

a) Processes that enter the critical section: 4
b) Final value of S: 0

7. **Semaphore S = 1; wait(S); wait(S); signal(S); signal(S); a) How many processes are blocked? b) What is the final value of S?**

Answer:

Semaphore starts with 1 key, meaning 1 process can enter.

1. First wait(S):

Process 1 takes 1 key so ,

S = 1 − 1 = 0

- Process 1 enters

2. Second wait(S):

   Process 2 tries to take a key so,no key available

   a. Process 2 is blocked so, S = 0 − 1 = -1
   b.

3. First signal(S):
   a. A key is returned
   b. Blocked process 2 gets the key and enters so, S = -1 + 1 = 0

4. Second signal(S):
   a. The key is returned so,
   b. S = 0 + 1 = 1

a) Processes that were blocked: 1 (Process 2)
b) Final value of S: 1

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?

Answer:

Binary semaphore = 1 key, meaning only 1 process can enter at a time.

1. First wait ():

Process 1 takes the key so , enters the critical section

Semaphore value = 1-1=0

2. Next 4 wait ():

No key is available

Processes 2, 3, 4, 5 are blocked

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

Answer:

Semaphore = 4 keys, meaning 4 processes can enter the critical section at the same time.

1. First 4 processes take the keys so , enter the critical section

Semaphore value = 0

2. Remaining 2 processes try to enter so , no keys available

They are blocked

- Processes that proceed: 4
- Processes that are blocked: 2

10. A semaphore S is initialized to 2. wait(S); wait(S); wait(S); signal(S); signal(S); wait(S); a) Track the semaphore value after each operation. b) How many processes were blocked at any time?

Answer:

wait(S) → Process 1

- Process 1 takes 1 key
- Semaphore value = 2 − 1 = 1
- Process 1 enters the critical section

wait(S) → Process 2

- Process 2 takes 1 key
- Semaphore value = 1 − 1 = 0
- Process 2 enters the critical section

wait(S) → Process 3

- No keys left (S = 0)
- Process 3 blocked
- Semaphore value = 0 − 1 = -1

signal(S) → A key returned

- Key is returned
- Process 3 wakes up and takes the key
- Semaphore value = -1 + 1 = 0

signal(S) → Another key returned

- No one is waiting now
- Semaphore value = 0 + 1 = 1

wait(S) → Process 4

- Process 4 takes 1 key

- Semaphore value = 1 − 1 = 0
- Process 4 enters the critical section

a) Semaphore values after each step: **2 → 1 → 0 → -1 → 0 → 1 → 0**
b) Processes blocked at any time: 1 (Process 3).

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?

Answer:

Semaphore starts at 0 keys, meaning no process can enter yet.

3 wait() operations

- Process 1 does wait() so, no key available and P1 is blocked :
  S = -1
- Process 2 does wait() so , no key available and P2 is blocked:
  S = -2
- Process 3 does wait() so ,no key available and P3 is blocked:
  S = -3

All 3 processes are now blocked.

5 signal() operations

Each signal() adds 1 to the semaphore:

1. Signal 1 : wakes Process 1 so :
   S = -3 + 1 = -2
2. Signal 2 : wakes Process 2 so :
   S = -2 + 1 = -1
3. Signal 3 : wakes Process 3 so :
   S = -1 + 1 = 0

After this, all blocked processes are awake, and S = 0 , so next two are available now

4. Signal 4 → S = 1
5. Signal 5 → S = 1 +1 =2

a) Processes that wake up: 3 (all initially blocked processes)

b) Final semaphore value: 2 (2 keys available for future processes)

(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**

Code:

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

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

   for (int i = 0; i < 3; i++)
   {
      int item = id * 100 + i;

      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
```

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

        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        sleep(1);
    }
    return NULL;
}

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

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

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

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        sleep(2);
    }
    return NULL;
}

int main()
{
    pthread_t prod[2], cons[2];
    int ids[2] = {1, 2};

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);
```

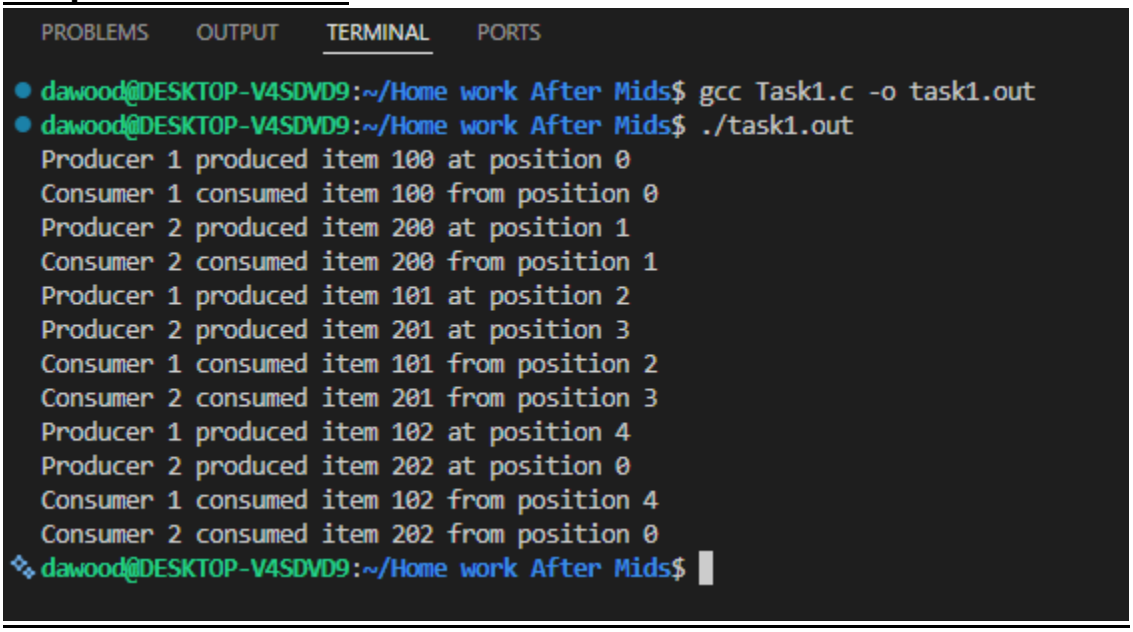
```
    for (int i = 0; i < 2; i++)
    {
        pthread_create(&prod[i], NULL, producer, &ids[i]);
        pthread_create(&cons[i], NULL, consumer, &ids[i]);
    }

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

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

    return 0;
}
```

Output Screenshot:



Remarks:

In this code, A buffer of size 5 is created to store items. Two indexes are used:

- in → position where producer puts an item
- out → position where consumer takes an item

Then, two semaphores are used:

- empty: keeps track of empty spaces in the buffer
- full: keeps track of filled spaces in the buffer

Now, A mutex is used to ensure that only one thread accesses the buffer at a time.

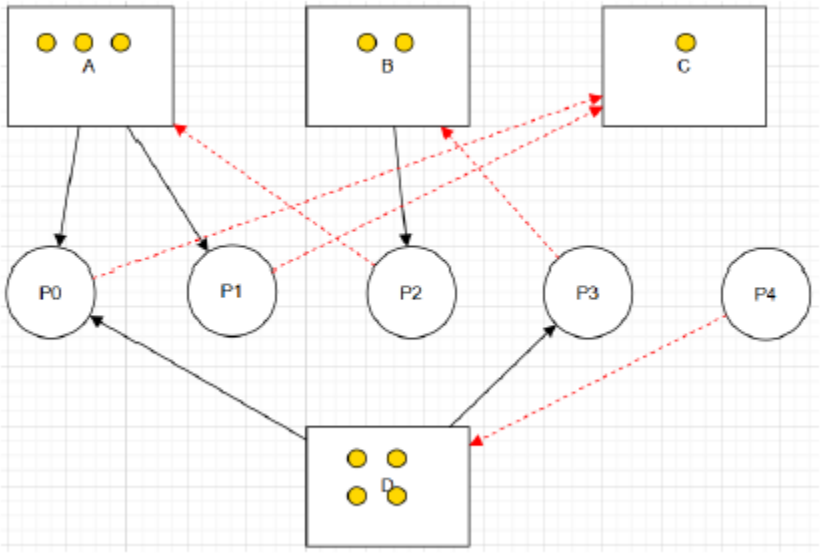
Also, I created Two producer threads and two consumer thread

Here Each producer produces 3 items and puts them into the buffer and Each consumer consumes 3 items from the buffer.

- Producers wait if the buffer is full.
- Consumers wait if the buffer is empty.

Part 3: RAG (Recourse Allocation Graph)

Convert the following graph into matrix table:



Answer:

a) Allocation Matrix

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

b) Request Matrix

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

c) Available Resource vector.

A: 3 (total) - 2 allocated = 1 available
B: 2 (total - 1 allocated = 1 available
C: 1 (total) - 0 allocated = 1 available
D 4 (total) - 2 allocated = 2 available

Process:	A	B	C	D
Total Allocated:	2	1	0	2

Recource	A	B	C	D
Available	1	1	1	2

Part 4: Banker's Algorithm

System Description:

- The system comprises five processes (P0–P3) and four resources (A,B,C,D).

- Total Existing Resources:

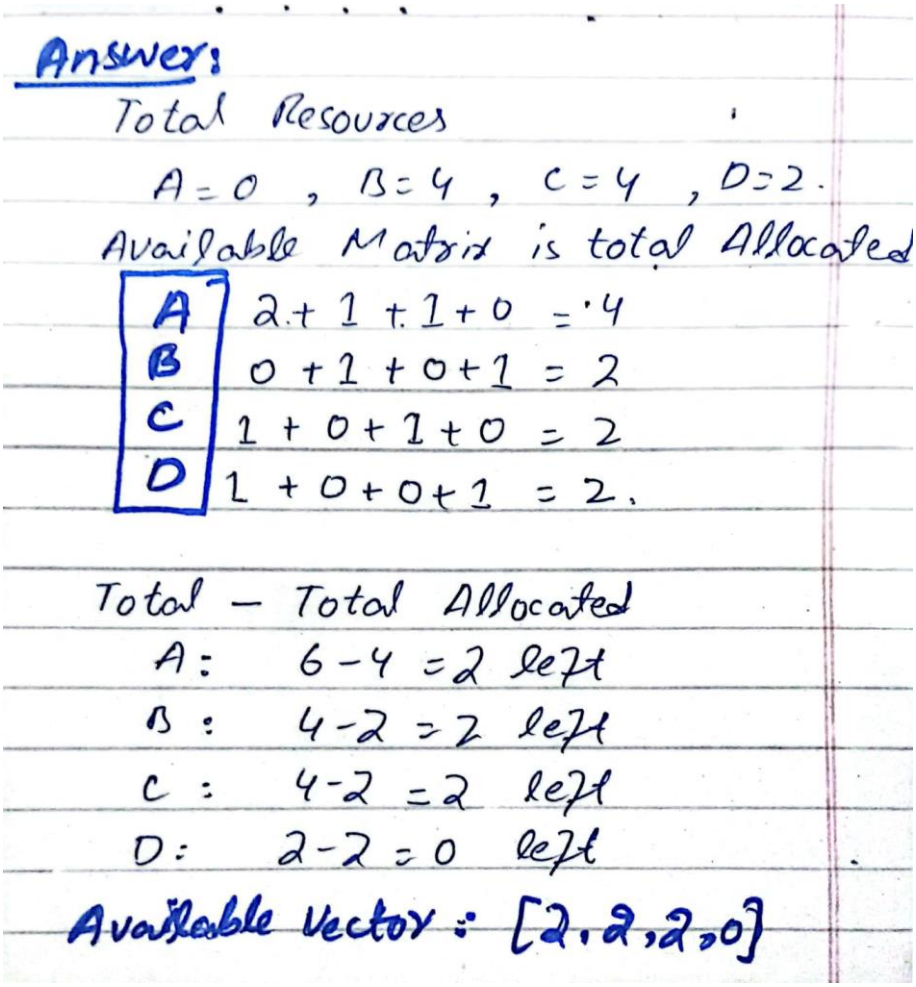
Total			
A	B	C	D
6	4	4	2

- Snapshot at the initial time stage:

	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				

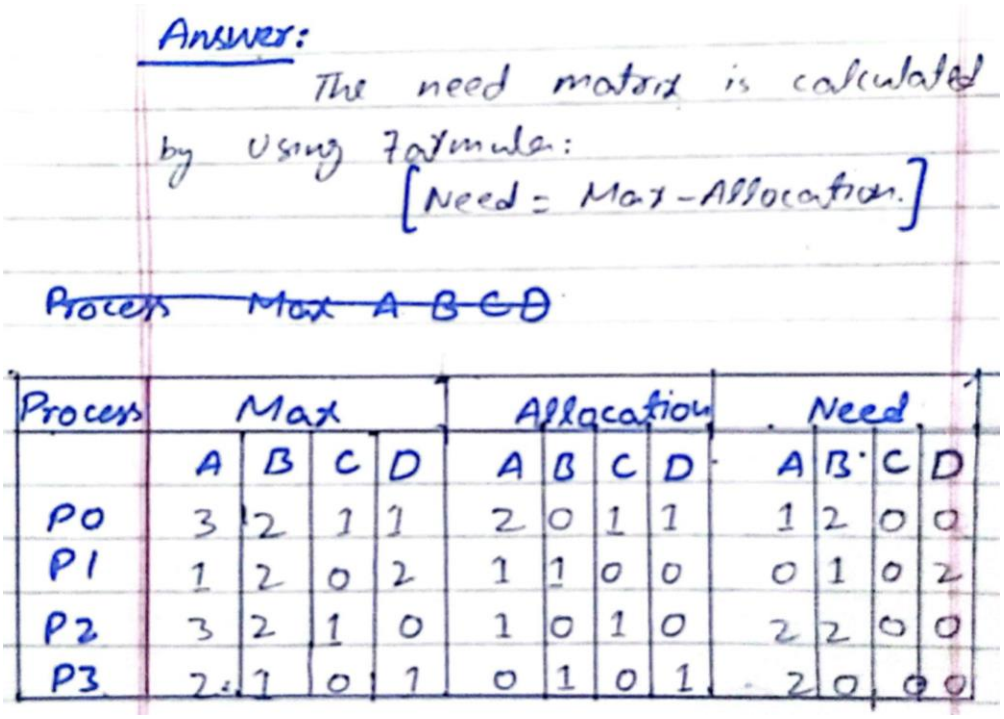
Questions:

1. **Compute the Available Vector**:
   Calculate the available resources for each type of resource.



2. **Compute the Need Matrix**

Determine the need matrix by subtracting the allocation matrix from the maximum matrix



Answer:

The need matrix is calculated by using formula:

[Need = Max - Allocation.]

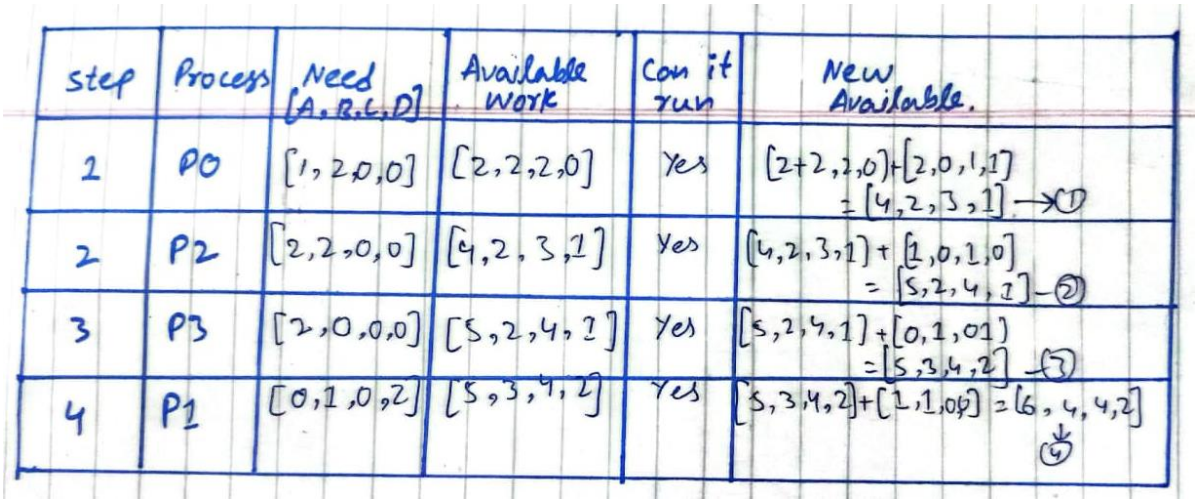
~~Process Max A B C D~~

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

3. **Safety Check**:
   - Determine if the current allocation state is safe. If so, provide a safe sequence of the processes.
   - Show how the Available (working array) changes as each process terminates.

Answer:

To determine if the state is safe, we find a sequence where each process's need available. Once a process finishes, it releases its Allocation back to the Available pool.



step	Process	Need [A,B,C,D]	Available work	Can it run	New Available.
1	P0	[1,2,0,0]	[2,2,2,0]	Yes	[2+2,2,0]+[2,0,1,1] = [4,2,3,1] → (1)
2	P2	[2,2,0,0]	[4,2,3,1]	Yes	[4,2,3,1]+[1,0,1,0] = [5,2,4,1] → (2)
3	P3	[2,0,0,0]	[5,2,4,1]	Yes	[5,2,4,1]+[0,1,0,1] = [5,3,4,2] → (3)
4	P1	[0,1,0,2]	[5,3,4,2]	Yes	[5,3,4,2]+[1,1,0,0] = [6,4,4,2] (4)

Yes. Safe Sequence: {P0, P2, P3, P1}