National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Amina

Reg. number: 23-NTU-CS-1136

Lab no.: Assignment

Semester:5th

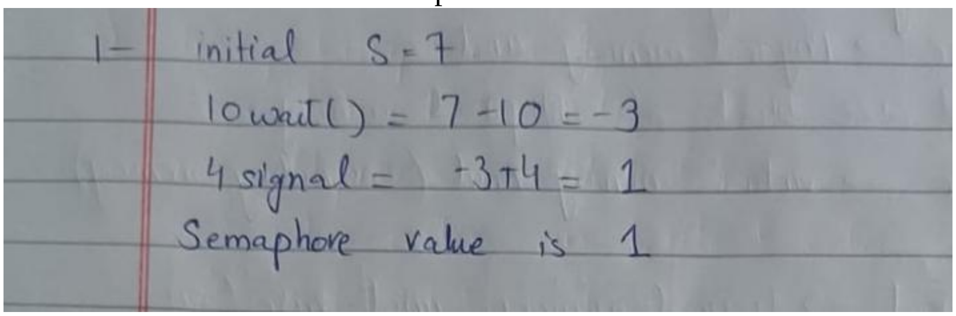
Operating Systems – COC 3071

SE 5th A – Fall 2025

After-mid Homework -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.



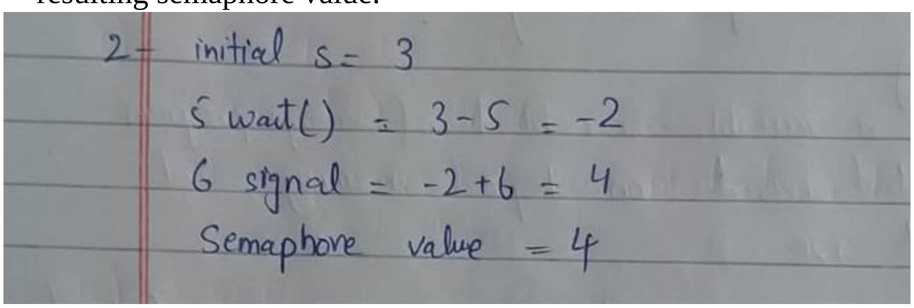
1- initial S = 7

10 wait() = 7 - 10 = -3

4 signal = -3 + 4 = 1

Semaphore value is 1

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



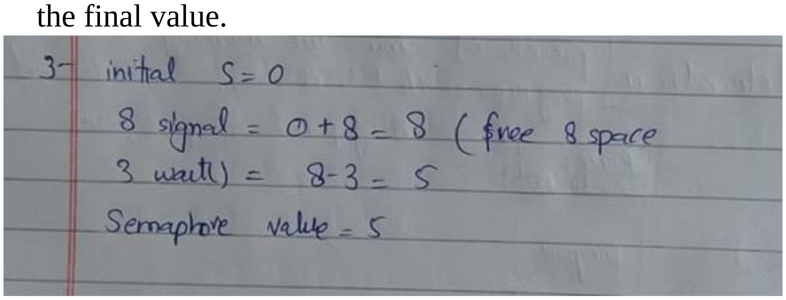
2- initial s = 3

5 wait() = 3 - 5 = -2

6 signal = -2 + 6 = 4

Semaphore value = 4

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



3- initial S = 0

8 signal = 0 + 8 = 8 (free 8 space

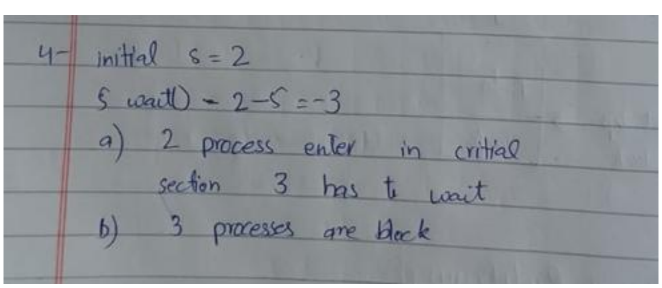
3 wait() = 8 - 3 = 5

Semaphore value = 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?



4- initial s = 2

5 wait() = 2 - 5 = -3

a) 2 process enter in critical section 3 has to wait

b) 3 processes are block

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?

5- S → 1
3 wait() = 1-3 = -2
1 signal() = -2 +1 = -1
a) 1 process remain block
b) final semaphore = -1

6.
semaphore S = 3;
wait(S); wait(S);
signal(S);
wait(S); wait(S);

6- S = 3
wait(S) perform wait operation on S
S = S-1
• wait (S) = 3-1 = 2
a) 1 process enter in critical Section
b) = S = 2

• wait(S) = 2-1 = 1
a) 1 proces enter in critical Section
b) S = 1

• signal (S) = 1+1 = 2 (free 2 space)
• wait (S) = 2-1 = 1
a) 1 process enter in critical section
b) S = 1
• wait(S) = 1-1 = 0
a) No process enter in critial sectio
b) = S = 0
⇒ Process enter in critical section = 4
⇒ S = 0

a) How many processes enter the critical section?
b) What is the final value of S?

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?

7- $S = 1$

wait(S) = 1-1 = 0 (1 enter in critical section)

wait(S) = 0-1 = -1 process blocked

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

signal(S) = 0 + 1 1 free space

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?

8) binary semaphore = 1

5 wait() = 1-5 = -4

- 4 process are blocked
- 1 process enter in critical sectio

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

9) $S = 4$

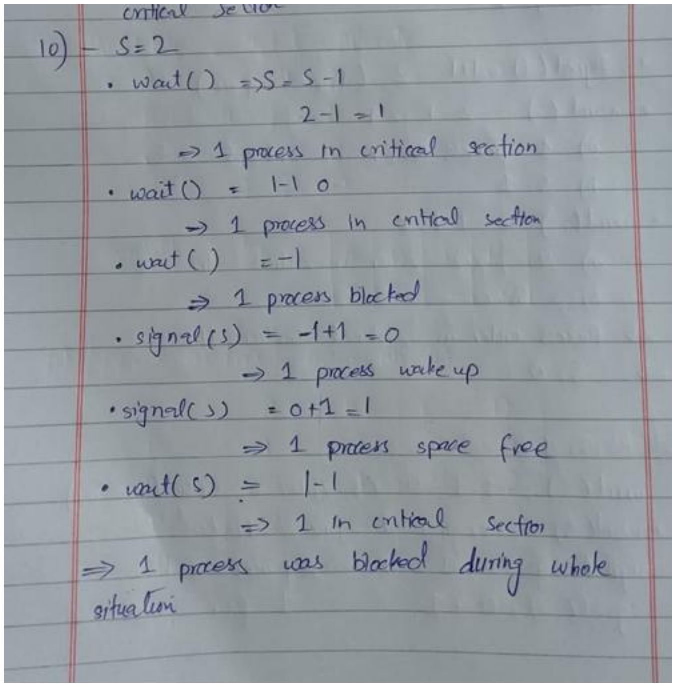
6 wait() = 4-6 = -2

- 2 p-rocesses are blocked, rest in critical section

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?



critical section

10) – S = 2

• wait() ⇒ S = S − 1
2 − 1 = 1
⇒ 1 process in critical section

• wait() = 1 − 1 0
⇒ 1 process in critical section

• wait() = −1
⇒ 1 process blocked

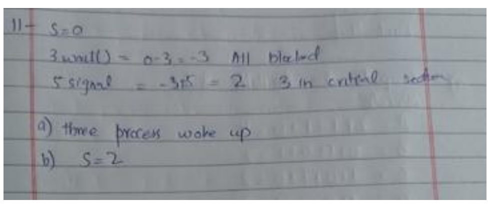
• signal(s) = −1 + 1 = 0
⇒ 1 process wake up

• signal(s) = 0 + 1 = 1
⇒ 1 process space free

• wait(S) = 1 − 1
⇒ 1 in critical section

⇒ 1 process was blocked during whole situation

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?



11 – S = 0
3 wait() = 0 − 3 = −3 All blocked
5 signal = −3 + 5 = 2 3 in critical section

a) three process wake up
b) S = 2

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

```
#include <stdio.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;
```

```c
pthread_mutex_t mutex;
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;
}
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;
    }
```

```
compilation terminated.
amina@DESKTOP-SEP18NK:~/OSLabs$ cd ./lab10/
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ gcc h
/usr/bin/ld: cannot find h: No such file or directory
collect2: error: ld returned 1 exit status
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ gcc ./homeTask.c -o Q1 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ ./Q1
Producer 1 produced item 100 at position 0
Consumer 1 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 1 consumed item 101 from position 2
Consumer 2 consumed item 201 from position 3
Producer 1 produced item 102 at position 4
Producer 2 produced item 202 at position 0
Consumer 1 consumed item 102 from position 4
Consumer 2 consumed item 202 from position 0
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$
```

How It Works:

Buffer:

- A small array where producers put items and consumers take items.
- in tells where the next item goes, out tells where the next item is taken from.

Producer:

- Waits if the buffer is full.
- Locks the buffer to safely put an item.
- Unlocks the buffer and tells consumers that there’s a new item.

Consumer:

- Waits if the buffer is empty.
- Locks the buffer to safely take an item.
- Unlocks the buffer and tells producers that there’s space.

Synchronization:

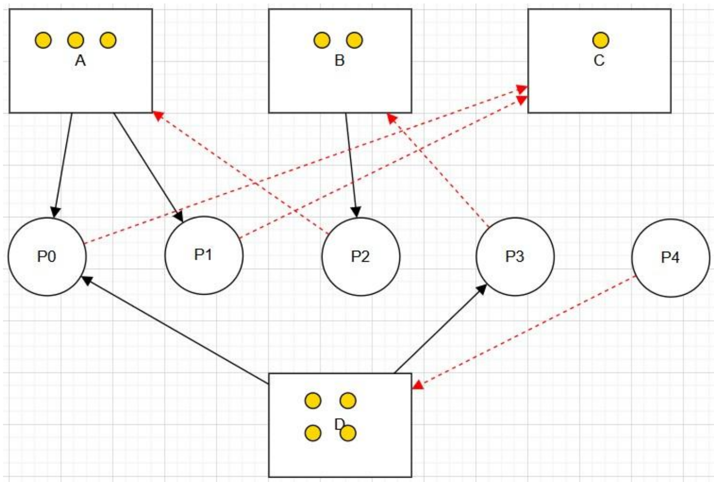
- Semaphore empty -> Counts empty spaces.
- Semaphore full ->Counts items in the buffer.
- Mutex-> Ensures only one thread touches the buffer at a time.

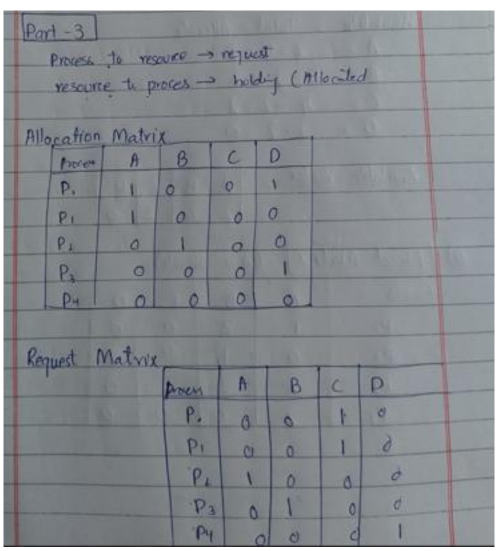
Circular Buffer:

When in or out reach the end of the array, they go back to the start.

Part 3: RAG (Recourse Allocation Graph)

- Convert the following graph into matrix table ,





Part -3

Process to resource → request
resource to process → holding (Allocated

Allocation Matrix

Process	A	B	C	D
P_0	1	0	0	1
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	0	1
P_4	0	0	0	0

Request Matrix

Process	A	B	C	D
P_0	0	0	1	0
P_1	0	0	1	0
P_2	1	0	0	0
P_3	0	1	0	0
P_4	0	0	0	1

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**:

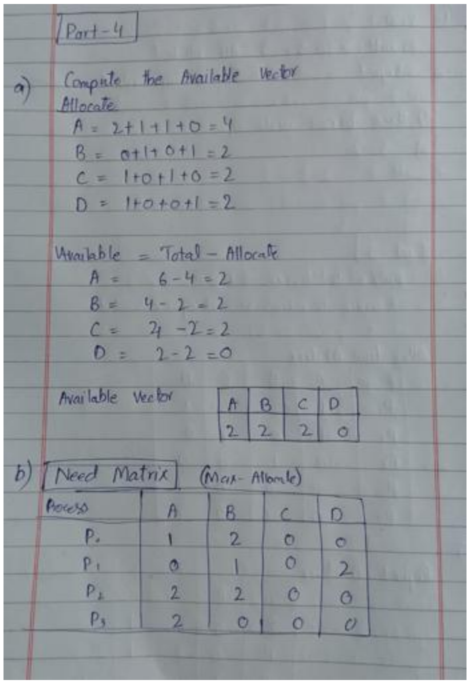
    - Calculate the available resources for each type of resource.

2. **Compute the Need Matrix**:

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

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.



Part-4

a) Compute the Available Vector

Allocate

A = 2+1+1+0 = 4

B = 0+1+0+1 = 2

C = 1+0+1+0 = 2

D = 1+0+0+1 = 2

Available = Total - Allocate

A = 6-4 = 2

B = 4-2 = 2

C = 4-2 = 2

D = 2-2 = 0

Available Vector

A	B	C	D
2	2	2	0

b) Need Matrix (Max- Allocate)

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

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:

Safe when Need ≤ Available
Initially available vector=**(2,2,2,0)**
For P0:
Need(1,2,0,0) ≤Available(2,2,2,0)
thus P0 process can be completed
Now once completed,
Available vector= Previously Available+ AllocationP0
=(2,2,2,0)+(2,0,1,1)=(4,2,3,1)
For P1:
Need(0,1,0,2) ≤Available(4,2,3,1) (X)
thus P1 process cant be completed
We need D(2) available=1

THUS THE GIVEN SEQUENCE IS NOT FEASIBLE THE CORRECT SEQUENCE IS,

Safe when Need ≤ Available
Initially available vector=**(2,2,2,0)**
For P0:
Need(1,2,0,0) ≤Available(2,2,2,0)
thus P0 process can be completed
Now once completed,
Available vector= Previously Available+ AllocationP0

=(2,2,2,0)+(2,0,1,1)=(4,2,3,1)

For P2:

Need(2,2,0,0) ≤Available(4,2,3,1)

thus P2 process completed

Now once completed,

Available vector= Previously Available+ AllocationP2

=(4,2,3,1)+(1,0,1,0)=(5,2,4,1)

For P3:

Need(2,0,0,0) ≤Available(5,2,4,1)

thus P3 process completed

Now once completed,

Available vector= Previously Available+ AllocationP2

=(5,2,4,1)+(0,1,0,1)=(5,3,4,2)

For P1:

Need(0,1,0,2) ≤Available(5,3,4,2)

thus P1 process completed

Now once completed,

Available vector= Previously Available+ AllocationP2

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

Thus the system is in safe state

P0->P2->P3->P1

Submission Guidelines:

- Ensure all answers are well-explained and calculations are shown step-by-step.
- Submit your assignment on MS Team and GitHub in a PDF format.
- VIVA based Evaluation so Develop your own solution after getting help.