NATIONAL TEXTILE

UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Subject

Operating System

SUBMITTED BY:

Fatima Waseem: 23-NTU-CS1155

SECTION SE: 5th (A)

SUBMITTED TO:

Sir Nasir

Assignment

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.**

    - Initial value of semaphore = 7
    - Number of wait() operations = 10
    - Number of signal() operations = 4

Rule:

- Each wait() operation decreases the semaphore by 1.
- Each signal() operation increases the semaphore by 1.

Calculation:

Final value = Initial value – wait() + signal().

Final value = 7 – 10 + 4

Final value = 1

Final value of semaphore = 1.

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

Rule:

- Each wait() operation decreases the semaphore by 1.
- Each signal() operation increases the semaphore by 1.

Calculation:

Final value = Initial value – wait() + signal().

Final value = 3 – 5 + 6

Final value = 4

Final value of semaphore = 4.

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

    - Initial value of semaphore = 0
    - Number of wait() operations = 3
    - Number of signal() operations = 8

Rule:

- Each wait() operation decreases the semaphore by 1.
- Each signal() operation increases the semaphore by 1.

Calculation:

- Final value = Initial value + signal () - wait().
- Final value = 0 + 8 - 3
- Final value = 5

Final value of semaphore = 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?**

    - Initial value of semaphore = 2
    - Number of wait() operations = 5

    Rule:

- Each wait() operation allows a process to enter the critical section **only if semaphore > 0**
- When semaphore becomes **0**, further wait() operations are **blocked.**

Calculation:

- Initial semaphore value = 2
- First 2 wait() operations:
    - Semaphore decreases to 0
    - 2 processes enter the critical section.

- Remaining 5 − 2 = 3 wait() operations:

- Semaphore is 0.
- These 3 processes are blocked.

a) **2 processes enter the critical section.**
b) **3 processes blocked.**

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 of semaphore = 1
- Number of wait() operations = 3
- Number of signal() operations = 1

Apply wait() operations:

- Initial value = 1.
- 1st wait() semaphore becomes 0.
- 1 process enters critical section.

Remaining wait() operations:

- 3-1 = 2.
- These 2 processes are blocked.
- Apply signal() operation:
- One signal releases one blocked process.

Answer:

a) **Processes remaining blocked:**
   - Initially blocked = 2.
   - One released by signal() = 1.
   - Remaining blocked = 1.
b) **Final semaphore value:**
   - Semaphore value = 0.

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

Operation	Semaphore Value	Explanation
Initial value	3	—
wait(S)	2	Process enters
wait(S)	1	Process enters
signal(S)	2	Resource released

Operation	Semaphore Value	Explanation
wait(S)	1	Process enters
wait(S)	0	Process enters

a) **4** processes enter the critical section.
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?**

Operation	Semaphore Value	Notes
Initial	1	—
wait(S)	0	1 process enters CS
wait(S)	0	Semaphore = 0 → 1 process blocked
signal(S)	0	Releases 1 blocked process → process immediately acquires the semaphore.
signal(S)	1	Increments semaphore → no blocked process

When signal() is called and there is a blocked process, the blocked process is immediately released and acquires the semaphore. The semaphore may not always increase by 1 because the blocked process consumes it.

a) **Processes blocked: 0**
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?**

Operation	Semaphore Value	Notes
Initial	1	—
wait(S) 1	0	First process enters CS
wait(S) 2	0	Semaphore = 0 → process blocked

Operation	Semaphore Value	Notes
wait(S) 3	0	Semaphore = 0 → process blocked
wait(S) 4	0	Semaphore = 0 → process blocked
wait(S) 5	0	Semaphore = 0 → process blocked

- Processes entered critical section: 1.
- Processes blocked: remaining 4(wait 2,3,4,5).

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

Operation	Semaphore Value	Notes
Initial	4	—
wait(S) 1	3	Process 1 enters CS
wait(S) 2	2	Process 2 enters CS
wait(S) 3	1	Process 3 enters CS
wait(S) 4	0	Process 4 enters CS
wait(S) 5	0	Semaphore = 0 → Process 5 **blocked**
wait(S) 6	0	Semaphore = 0 → Process 6 **blocked**

Processes that proceed: 4.
Processes 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?

Operation	Semaphore Value	Notes
Initial	2	—
wait(S) 1	1	Process 1 enters CS
wait(S) 2	0	Process 2 enters CS
wait(S) 3	0	S = 0 → Process 3 blocked
signal(S) 1	0	Releases 1 blocked process (Process 3 enters CS immediately)

Operation	Semaphore Value	Notes
signal(S) 2	1	No blocked processes → S increments by 1
wait(S) 4	0	Process 4 enters CS

c) **Processes blocked at any time:** 1

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?

Initial S = 0:

- wait() 1 → S = 0 → process 1 blocked
- wait() 2 → S = 0 → process 2 blocked
- wait() 3 → S = 0 → process 3 blocked

Signal operations (5 signals):

- signal() 1 → releases process 1 → S remains 0 after process acquires it
- signal() 2 → releases process 2 → S remains 0
- signal() 3 → releases process 3 → S remains 0
- signal() 4 → no blocked process → S increments → S = 1
- signal() 5 → no blocked process → S increments → S = 2

a) How many processes wake up?

- 3 processes were blocked and all 3 wake up after first 3 signals.

b) Final semaphore value

- After 5 signals and all blocked processes released → 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

```
fatima@DESKTOP-3BA3T21:~/Assignment2 code$ gcc Prod_cons.c -o output1
fatima@DESKTOP-3BA3T21:~/Assignment2 code$ ./output1
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
```

Description:

Producer Steps:

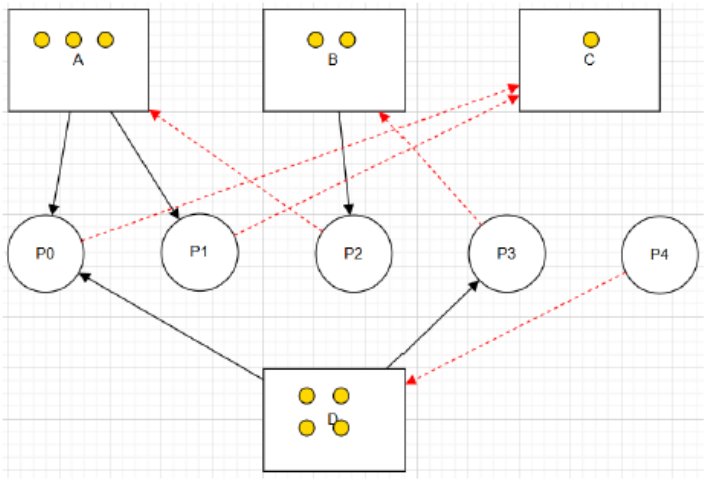
1. Producer checks for empty space using sem_wait(&empty).
2. Producer locks the buffer using mutex.
3. Producer puts an item in the buffer.
4. Producer unlocks the buffer.
5. Producer signals that the buffer is full using sem_post(&full).

Consumer Steps:

1. Consumer checks for available item using sem_wait(&full).
2. Consumer locks the buffer using mutex.
3. Consumer removes an item from the buffer.
4. Consumer unlocks the buffer.
5. Consumer signals that the buffer is empty using sem_post(&empty).

Part 3: RAG (Recourse Allocation Graph)

- Convert the following graph into matrix table ,



RAG:

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

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				

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

Total Existing Resources:

Total			
A	B	C	D
6	4	4	2

Allocated Resources:

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 Available Resources: Existing – Allocated.

A:6-4 = 2.

B:4-2 = 2.

C:4-2 = 2.

D:2-2= 0.

Available Resources vector: **[2, 2, 2, 0]**

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

2. Compute the Need Matrix: Claim/Max – Allocation.

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

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.

3. Safety Check:

- **Step 1: P0**
  - Can P0 be satisfied? Need (1, 2, 0, 0) < Available (2, 2, 2, 0). **Yes.**
  - P0 terminates and releases its allocation.
  - **New Available** = (2, 2, 2, 0) + (2, 0, 1, 1) = (4, 2, 3, 1).
- **Step 2: P2**
  - Can P2 be satisfied? Need (2, 2, 0, 0) < Available (4, 2, 3, 1). **Yes.**
  - P2 terminates and releases its allocation.
  - **New Available** = (4, 2, 3, 1) + (1, 0, 1, 0) = (5, 2, 4, 1)
- **Step 3: P3**
  - Can P3 be satisfied? Need (2, 0, 0, 0) < Available (5, 2, 4, 1). **Yes.**
  - P3 terminates and releases its allocation.
  - **New Available** = (5, 2, 4, 1) + (0, 1, 0, 1) = (5, 3, 4, 2).
- **Step 4: P1**
  - Can P1 be satisfied? Need (0, 1, 0, 2) < Available (5, 3, 4, 2). **Yes.**
  - P1 terminates and releases its allocation.
  - **New Available** = (5, 3, 4, 2) + (1, 1, 0, 0) = (6, 4, 4, 2).

- **Conclusion:** The system is in a Safe State.
- **Safe Sequence**: P0 -> P2 -> P3 -> P1