National Textile University

Department of Computer Science

Subject:

Operating System

Submitted To:

Sir Nasir Mehmood

Submitted By:

Hafsa Tayyab

Registration No:

23-NTU-CS-1163

Assignment:

2

Semester:

5th

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.

- The initial value was 7.
- Operations performed include 10 wait(): 10 decrements(-10)
- 4 signal(): 4 increments(+4)
- Final value: 7-10+4 = **1**

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

- The initial value was 3.
- Operations performed include 5 wait(): 5 decrements(-5)
- 6 signal(): 6 increments(+6)
- Final value: 3-5+6 = **4**

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

- The initial value was 0.
- Operations performed include 8 signal(): 8 increments(+8)
- 3 wait(): 3 decrements(-3)
- Final value: 0+8-3 = **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?

If a semaphore is initialized to 2. Following wait() operations would decrement its value by 1;

- 1st Wait() operation: 2-1 = 1 > Semaphore's value is >= 0, process would enter the critical section.
- 2nd Wait() operation: 1-1 = 0 > Semaphore's value is >= 0, process would enter the critical section.
- 3rd Wait() operation: 0-1 = -1 > Semaphore's value is < 0, process would be blocked.
- 4th Wait() operation: -1-1 = -2 > Semaphore's value is < 0, process would be blocked.
- 5th Wait() operation: -2-1 = -3 > Semaphore's value is < 0, process would be blocked.

Results: a) 2 processes will enter the critical section.

b) 3 processes would be blocked (enter the waiting queue).

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?

If a semaphore is initialized to 1;

- 1st Wait() operation: 1-1 = 0 > Semaphore's value is >= 0, process would enter the critical section.
- 2nd Wait() operation: 0-1 = -1 > Semaphore's value is < 0, process would be blocked.
- 3rd Wait() operation: -1-1 = -2 > Semaphore's value is < 0, process would be blocked.
- 1st Signal() operation: -2+1 = -1 > Semaphore's value is incremented; one process would be unblocked (enter the critical section).

Results: a) Only one process would remain blocked (signified by semaphore's value (-1)).

b) Final 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?

Semaphore's initial state: S=3.

- Wait(S) operation: 3-1 = 2; S >= 0, process would enter the critical section.
- Wait(S) operation: 2-1 = 1; S >= 0, process would enter the critical section.
- Signal(S) operation: 1+1 = 2; Semaphore's value is incremented.
- Wait(S) operation: 2-1 = 1; S >= 0, process would enter the critical section.
- Wait(S) operation: 1-1 = 0; S >= 0, process would enter the critical section.

Results: a) 4 processes would enter the critical section.

b) Final Value: 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?

Semaphore's initial state: S=1.

- Wait(S) operation: 1-1 = 0; S >= 0, process would enter the critical section.
- Wait(S) operation: 0-1 = -1; S < 0, process would be blocked.
- Signal(S) operation: -1+1 = 0; Semaphore's value is incremented, one process would be unblocked (enter the critical section).
- Signal(S) operation: 0+1 = 1; Semaphore's value is incremented.

Results: a) No processes are blocked at the end.

b) Final Value: 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?

A binary semaphore enables only one process to use the critical section at a time. If a binary semaphore is initialized to 1. It indicates that the critical section is free and one process can use it at that time so,

- The 1st wait() call will decrement it by 1, S=0, and enter the critical section.
- All the following wait() operations would be terminated and the processes would be blocked.

Results: a) Only one process would enter the critical section.

b) 4 processes would be blocked (enter the waiting queue).

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

- The initial value was 4.
- Operations performed include 6 wait(): 6 decrements(-6)
- Final value: 4-6 = **-2**

This indicates that 4 processes would enter the critical section and 2 processes would be blocked (enter the waiting queue).

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?

Semaphore S initial state = 2.

- Wait(S) operation: 2-1 = 1; S >= 0, process would enter the critical section.
    - No. of processes blocked = 0.
- Wait(S) operation: 1-1 = 0; S >= 0, process would enter the critical section.
    - No. of processes blocked = 0.
- Wait(S) operation: 0-1 = -1; S < 0, process would be blocked.
    - No. of processes blocked = 1.
- Signal(S) operation: -1+1 = 0; Semaphore's value is incremented; one process would be unblocked (enter the critical section).
    - No. of processes blocked = 0.
- Signal(S) operation: 0+1 = 0; Semaphore's value is incremented.

    - No. of processes blocked = 0.
- Wait(S) operation: 2-1 = 1; S >= 0, process would enter the critical section.
    - No. of processes blocked = 0.

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?

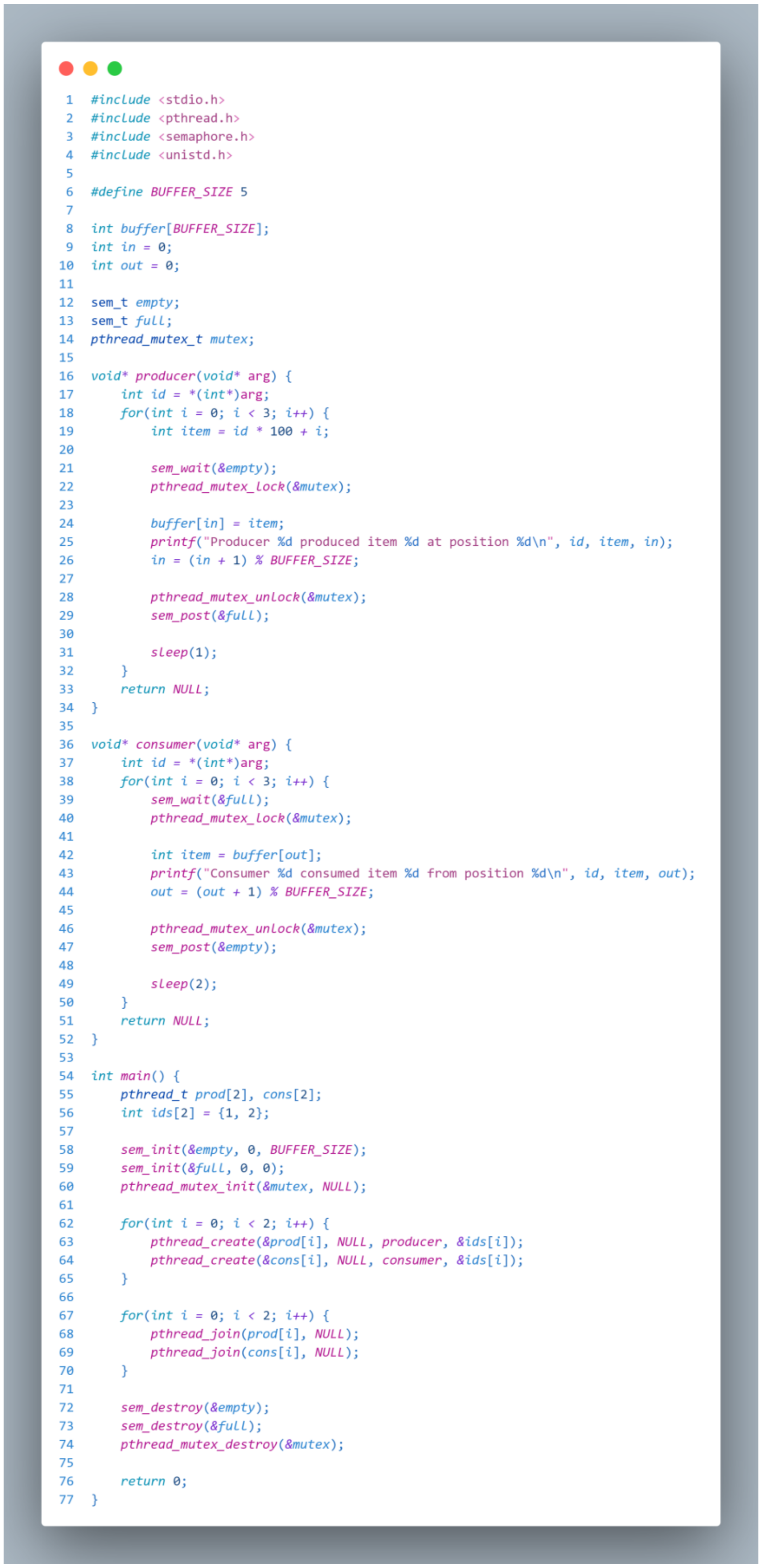
- Execution of 3 wait() would decrement the value 3 times: S = -3, indicating three processes are blocked
- Later 5 signal() operations would increment the value 5 times: S = -3+5 = 2.
- This indicates that all three blocked processes would wake up and resulting semaphore is incremented.
- The final semaphore's value is: S = 2.

Part 2: Semaphore Coding

Consider the Producer–Consumer problem using semaphores as implemented in Lab-10 (Lab-plan attached).

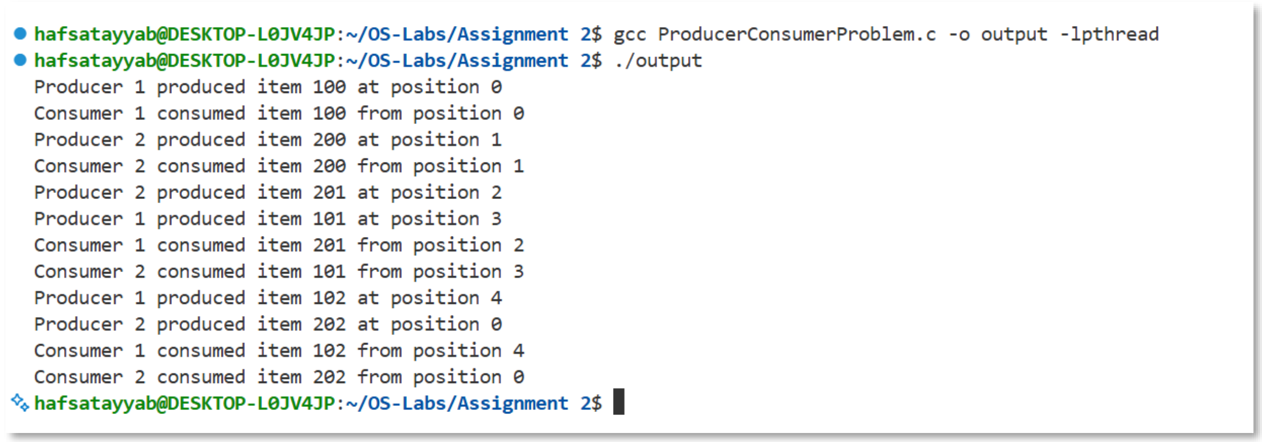
- **Rewrite the program in your own coding style, compile and execute it successfully, and explain the working of the code in your own words.**

Source Code:



```c
#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;
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;
}
```

Output:



```
● hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Assignment 2$ gcc ProducerConsumerProblem.c -o output -lpthread
● hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Assignment 2$ ./output
  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 2 produced item 201 at position 2
  Producer 1 produced item 101 at position 3
  Consumer 1 consumed item 201 from position 2
  Consumer 2 consumed item 101 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
  hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Assignment 2$ █
```

Remarks:

This program solves the **Producer-Consumer problem** using three synchronization primitives:

1. **Empty Semaphore (empty)**: This is initialized to the size of the buffer (5). It keeps track of how many spaces are left. The producer must "wait" for an empty slot before adding an item.
2. **Full Semaphore (full)**: This is initialized to 0. It tracks how many items are currently in the buffer. The consumer must "wait" for this to be greater than 0 before it can take an item.
3. **Mutex Lock (mutex)**: This acts as a key to the "Critical Section." Since both the producer and consumer modify the shared buffer and its indices (in and out), only one thread is allowed to access that section at a time to prevent data corruption.

The Workflow:

- The **Producer** finds an empty spot, locks the buffer, places an item, unlocks, and then signals that the buffer is now "fuller" by 1 unit.
- The **Consumer** waits for a full spot, locks the buffer, removes the item, unlocks, and then signals that the buffer is now "emptier" by 1 unit.

Part 3: RAG (Recourse Allocation Graph)

• Convert the following graph into matrix table

Process	**Allocation (A, B, C, D)**	**Request (A, B, C, D)**
P0	(1, 0, 0, 1)	(0, 0, 1, 0)
P1	(1, 0, 0, 0)	(0, 0, 1, 0)
P2	(0, 1, 0, 0)	(1, 0, 0, 0)
P3	(0, 0, 0, 1)	(0, 1, 1, 0)
P4	(0, 0, 0, 0)	(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:

• Snapshot at the initial time stage:

• Questions:

1. **Compute the Available Vector:**

• Calculate the available resources for each type of resource.

The Available vector is calculated as: Total - sum(Allocation)

- Total Resources: A=6, B=4, C=4, D=2
- Current Allocation 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
- Available: (6-4, 4-2, 4-2, 2-2) = [2, 2, 2, 0].

2. **Compute the Need Matrix:**

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

Process	Max (A B C D)	Allocation (A B C D)	Need (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.

Current Work/Available: **[2, 2, 2, 0]**

1. **Check P0**: Need (1 2 0 0) Work (2 2 2 0)? Yes, it is safe to run.
   - P0 finishes. New Work = [2, 2, 2, 0] + [2, 0, 1, 1] = [4, 2, 3, 1].
2. **Check P1**: Need (0 1 0 2) Work (4 2 3 1)? No, Need is not being fulfilled because Resource D is insufficient.
3. **Check P2**: Need (2 2 0 0) Work (4 2 3 1)? Yes, it is safe to run.
   - P2 finishes. New Work = [4, 2, 3, 1] + [1, 0, 1, 0] = [5, 2, 4, 1].
4. **Check P3**: Need (2 0 0 0) Work (5 2 4 1)? Yes, it is safe to run.
   - P3 finishes. New Work = [5, 2, 4, 1] + [0, 1, 0, 1] = [5, 3, 4, 2].
5. **Check P1**: Need (0 1 0 2) Work (5 3 4 2)? Yes, it is safe to run.
   - P1 finishes. New Work = [5, 3, 4, 2] + [1, 1, 0, 0] = [6, 4, 4, 2].

Safe Sequence: P0, P2, P3, P1

The system is in a safe state.

---

*_ALHUMDULILAH_*