National Textile University

Department of Computer Science

Subject:
Operating System

Submitted to:
Sir Nasir Mehmood

Submitted by:
Ghulam Mohyuddin

Reg number:
23-NTU-CS-FL-1158

Assignment Number:
2

Semester:
5th

Part#1

1. A counting semaphore is initialized to 7. If 10 wait() and 4 signal() operations are performed, find the final value of the semaphore.
   Solution:
   Applying wait():
   S=7-10
   S=-3
   Applying signal():
   S=-3+4
   S=1
   Final Answer:
   S=1

2. A semaphore starts with value 3. If 5 wait() and 6 signal() operations occur, calculate the resulting semaphore value.
   Solution:
   Applying wait():
   S=3−5
   S=−2
   Applying signal():
   S=-2+6
   S=4
   Final Answer:
   S=4

3. A semaphore is initialized to 0. If 8 signal() followed by 3 wait() operations are executed, find the final value?
   Solution:
   Applying wait():
   S=0+8
   S=8
   Applying signal():
   S=8-3
   S=5
   Final Answer:
   S=5

4. A semaphore is initialized to 2. If 5 wait() operations are executed?
   Solution:
   S=2-1=1
   S=1-1=0
   S=0-1=-1
   S=-1-1=-2
   S=-2-1=-3
   (a) How many processes enter the critical section?
   Just first 2 process enter critical section, beacaue to enter in critical section semaphore must be <=0
   (b) How many processes are blocked?
   Remaining 3 process are blocked

5. <u>A semaphore starts at 1. If 3 wait() and 1 signal() operations are performed?</u>
Solution:
Applying wait():
S=1-1=0
S=0-1=-1
S=-1-1=-2
Applying signal():
S=-2+1=-1

(a) **How many processes remain blocked?**
1 process.
(b) **What is the final semaphore value?**
<u>S=-1</u>

6. <u>Semaphore S = 3; wait(S); wait(S); signal(S); wait(S); wait(S)?</u>
Solution:
Applying wait():
S=3-1=2
S=2-1=1
Applying signal():
S=1+1=2
Applying wait():
S=2-1=1
S=1-1=0
(a) **How many processes enter the critical section?**
4 processes.
(b) **What is the final value of S?**
<u>S=0</u>

7. <u>Semaphore S = 1; wait(S); wait(S); signal(S); signal(S).</u>
Solution:
Applying wait():
S=1-1=0
S=0-1=-1
Applying signal():
S=-1+1=0
S=0+1=1
(a) **How many processes are blocked?**
1 Process.
(b) **What is the final value of S?**
<u>S=1</u>

8. <u>A binary semaphore is initialized to 1. Five wait() operations are executed without any signal().</u>
Solution:
Applying wait():
S=1->0
S=0
S=0
S=0
S=0
How many processes enter the critical section?
Only one process, because to enter in critical section semaphore must be <=0.
How many are blocked?

Remaining all are blocked (4 process).

9. A counting semaphore is initialized to 4. If 6 processes execute wait() simultaneously?
Solution:
Applying wait():
S=4-1=3
S=3-1=2
S=2-1=1
S=1-1=0
S=0-1=-1
S=-1-1=-2
How many proceed?
First 4 process.
How many are blocked?
Last 2 process.

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.
Solution:

Semaphore Value	**Process State**	**Blocked?**
2	NULL	N0
2-1=1	Critical Section	N0
1-1=0	Critical Section	N0
0-1=-1	Blocked	Yes
-1+1=0	Unblocks	N0
0+1=1	NULL	N0
1-1=0	Critical Section	N0

(b) How many processes were blocked at any time?
Only 1 process is blocked at wait(3).

11. Semaphore is initialized to 0. Three processes execute wait() before any signal(). Later, 5 signal() operations are executed.
Solution:
Applying wait():
S=0-1=-1
S=-1-1=-2
S=-2-1=-3
Applying signal():
S=-3+1=-2
S=-2+1=-1
S=-1+1=0
S=0+1=1
S=1+1=2
(a) How many processes wake up?
3 Blocked process waked up at signal().
(b) What is the final semaphore value?
S=2

<u>Part#2</u>

Code:

```
#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, full;
pthread_mutex_t lock;

void* producer(void* arg) {
   for(int i=0; i<5; i++) {
      int item = i+1;
      sem_wait(&empty);
      pthread_mutex_lock(&lock);
      buffer[in] = item;
      printf("Producer produces %d at %d\n", item, in);
      in = (in+1) % SIZE;
      pthread_mutex_unlock(&lock);
      sem_post(&full);
      sleep(1);
   }
   return NULL;
}

void* consumer(void* arg) {
   for(int i=0; i<5; i++) {
      sem_wait(&full);
      pthread_mutex_lock(&lock);
      int item = buffer[out];
      printf("Consumer consumes %d from %d\n", item, out);
      out = (out+1) % SIZE;
      pthread_mutex_unlock(&lock);
      sem_post(&empty);
      sleep(2);
   }
   return NULL;
}

int main() {
   pthread_t prod, cons;
   sem_init(&empty, 0, SIZE);
   sem_init(&full, 0, 0);
   pthread_mutex_init(&lock, NULL);

   pthread_create(&prod, NULL, producer, NULL);
   pthread_create(&cons, NULL, consumer, NULL);

   pthread_join(prod, NULL);
   pthread_join(cons, NULL);
```

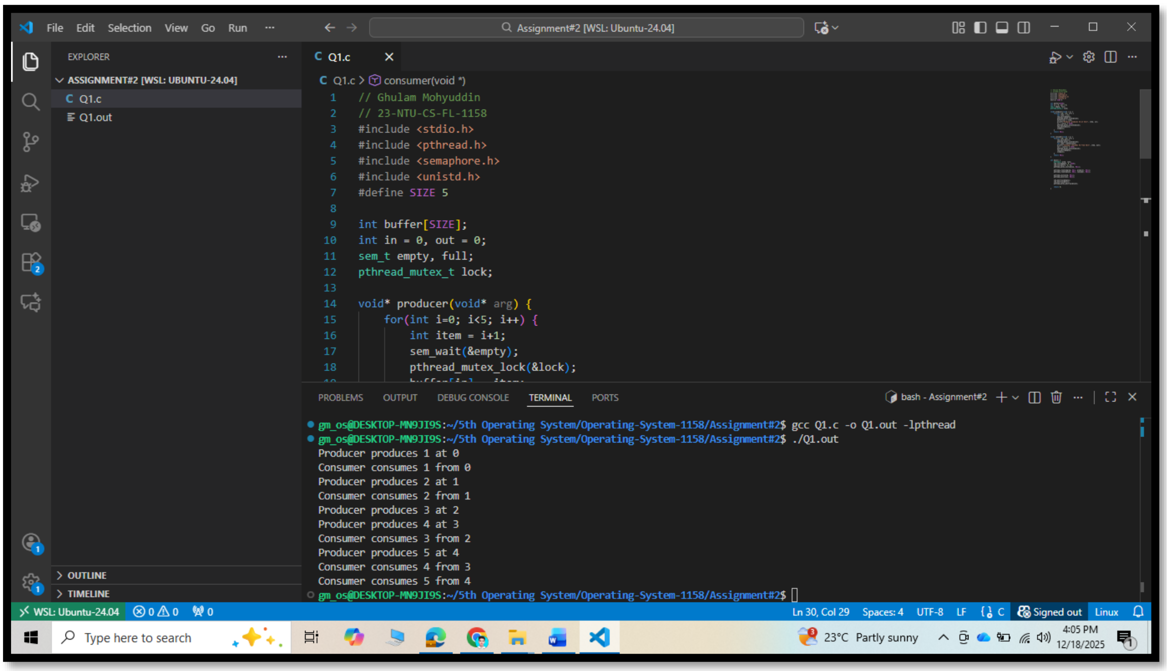
```
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&lock);

    return 0;
}
```

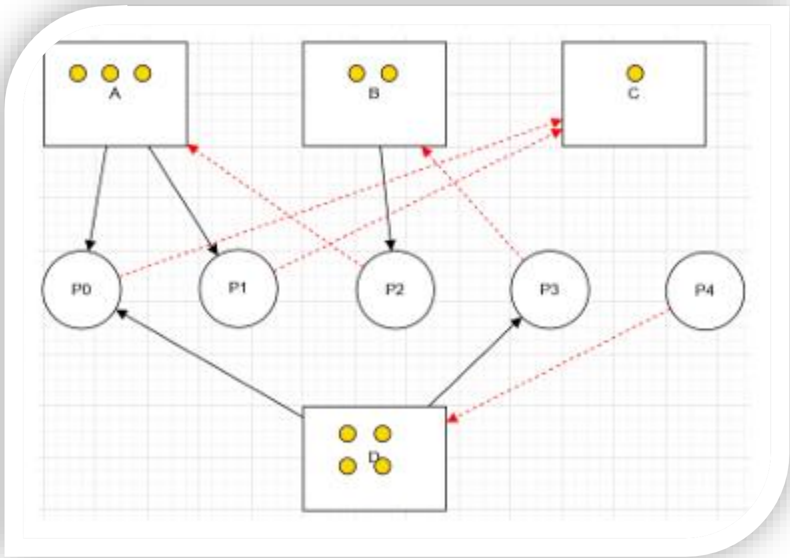
Description:

- ❖ Create a 2 function for producer and consumer.
- ❖ Producer() create 1 item.
- ❖ Use sem_wait(&empty) check the space is empty or not, if the space is empty it put them in buffer, otherwise wait for it empty.
- ❖ Then use pthread_mutex_lock to apply synchronization.
- ❖ The item is placed in 'in' index of buffer.
- ❖ Then update the 'in' index of buffer.
- ❖ Unlock the buffer using pthread_mutex_unlock(&lock).
- ❖ Then signal the consumer by sem_post(&full).
- ❖ Consumer() takes 1 item at a time from the buffer.
- ❖ Use sem_wait(&full) to check if the buffer has an item.
- ❖ If an item exists, it is taken from the buffer.
- ❖ If not, the consumer waits until an item is produced.
- ❖ Lock the buffer using pthread_mutex_lock(&lock).
- ❖ The item from the 'out' index of the buffer.
- ❖ Update the out index for the next item (out = (out + 1) % SIZE).
- ❖ Unlock the buffer using pthread_mutex_unlock(&lock).
- ❖ Then signal the producer by sem_post(&full).
- ❖ In main(), Initialize semaphore and mutex lock.
- ❖ Then create a producer and consumer object.
- ❖ At last, destroy both semaphore and mutex.

Output



Part#3
Covert the following graph into matrix table:



Allocation Matrix:

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

Request Matrix:

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

Part#4

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				

-

Allocation matrix:

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

Max Matrix

	A	B	C	D
P0	3	2	1	1
P1	1	2	0	2
P2	3	2	1	0
P3	2	1	0	1

1. Availible vector:

A.V = Total - Sum of allocated

Sum of allocated:

$A = 2+1+1+0 = 4$

$B = 0+1+0+1 = 2$

$C = 1+0+1+0 = 2$

$D = 1+0+0+1 = 2$

Now, apply availible vector:

$A = 6-4 = 2$

$B = 4-2 = 2$

$C = 4-2 = 2$

$D = 2-2 = 0$

2. Need Matrix:

Need = Max − Allocation

	A	B	C	D
P0	3-2	2-0	1-1	1-1
P1	1-1	2-1	0-0	2-0
P2	3-1	2-0	1-1	0-0
P3	2-0	1-1	0-0	1-0

Now, need matrix is:

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

3. Saftey Check:

Availible = [2, 2, 2, 0]

Step 1:

Check P0: Need(1,2,0,0) ≤ Availible (2,2,2,0)

Availible += Availible[P0] = [2,2,2,0] + [2,0,1,1] = [4,2,3,1]

Finished [P0] = True

Sequence = [P0]

Step 2:

Check P1: Need [0,1,0,2]
≤ Availible [4,2,3,1]

∴ as above decision is false, so we skip it.

Check P2: Need [2,2,0,0] ≤
Availible [4,2,3,1]

∴ now above decision is true

Availible += Allocation [P2]

Availible = [5,2,4,1]

Finished [P2] = True

Sequence = [P0, P2]

Step 3:

Check P1 again: Same process repeat as above.

Check P3: Need [2,0,0,0] ≤
Availible [5,2,4,1]

∴ the above decision is true.

Availible += Allocation [P3]

Availible = [5,3,4,2]

Finished [P3] = True

Sequence = [P0, P2, P3]

Step 4:

Now only P1 left:

Need[0,10,2] ≤ Availible [5,3,4,2]

∴ The above decision is true.

Availible += Allocation [P1]

Availible = [6,4,4,2]

Finished [P1] = True.

Sequence = [P0, P1, P2, P3]

Results:

Now system is purely safe.

Safe sequence is:

P0 → P1 → P2 → P3