OS ESE Lab:-

-------------------------

Name: Yash Oswal

SRN: 201901226

Div: B Roll no:- 38

--------------------------

Que. Implement Producer Consumer Problem using threads and semaphores

**ALGORITHM:-**

Using semaphore and thread:

1. Declare two thread variables of pthread\_t structure.

2. Create two threads associated with producer and consumer processes using pthread\_create function.

3. Declare two semaphore variables of sem\_t structure bfull and bempty.

4. Initialize the semaphore variable empty to 0 and full to 1 using sem\_init function.

5. Join two threads using pthread\_join function.

6. Initiate two processes.

7. Destroy both the semaphore variables using sem\_destroy function.

8. Stop.

Producer ( )

1. If semaphore variable bfull is 1, then the producer will wait on empty using the sem\_wait function.

2. Else the producer will produce the string and execute the critical section. Then signal the bfull semaphore variable using the sem\_post function on full and unblock the consumer thread.

Consumer ( )

1. If there is no data to consume, then the thread waits on the semaphore variable bfull using the sem\_wait function.

2. Else the consumer executes the critical section and consumes the data. Then the consumer signals the bempty variable using sem\_post function and unblocks the producer thread.

**CODE:-**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#define Iterations 3

#define Size 3

sem\_t empty;

sem\_t full;

sem\_t lock;

int in = 0;

int out = 0;

int buffer[Size] = {0};

void show()

{

for (int i = 0; i < Size; i++)

printf("%d ", buffer[i]);

printf("\n");

}

void \*prod(void \*prodnum)

{

int item;

for (int i = 0; i < Iterations; i++)

{

sleep(rand() % 3);

item = 1 + rand() % 9;

sem\_wait(&empty);

sem\_wait(&lock);

buffer[in] = item;

printf("\n\n\nProducer ID : %d", \*((int \*)prodnum));

printf("\nProduced : %d", buffer[in]);

printf("\nBuffer Status : ");

show();

if (buffer[0] != 0 && buffer[1] != 0 && buffer[2] != 0)

printf("\n\nProducer Blocked as Buffer is Full\n\n");

in = (in + 1) % Size;

sem\_post(&full);

sem\_post(&lock);

}

}

void \*cons(void \*connum)

{

for (int i = 0; i < Iterations; i++)

{

sleep(rand() % 5);

sem\_wait(&full);

sem\_wait(&lock);

int item = buffer[out];

buffer[out] = 0;

printf("\n\n\nConsumer ID : %d", \*((int \*)connum));

printf("\nConsumed : %d", item);

printf("\nBuffer Status : ");

show();

if (buffer[0] == 0 && buffer[1] == 0 && buffer[2] == 0)

printf("\n\nConsumer Blocked as Buffer is Empty\n\n");

out = (out + 1) % Size;

sem\_post(&empty);

sem\_post(&lock);

}

}

int main()

{

pthread\_t pro[3], con[3];

int p[3] = {1, 2, 3};

int c[3] = {1, 2, 3};

sem\_init(&lock, 0, Size);

sem\_init(&empty, 0, Size);

sem\_init(&full, 0, 0);

pthread\_create(&pro[0], NULL, prod, &p[0]);

pthread\_create(&pro[1], NULL, prod, &p[1]);

pthread\_create(&pro[2], NULL, prod, &p[2]);

pthread\_create(&con[0], NULL, cons, &c[0]);

pthread\_create(&con[1], NULL, cons, &c[1]);

pthread\_create(&con[2], NULL, cons, &c[2]);

pthread\_join(pro[0], NULL);

pthread\_join(pro[1], NULL);

pthread\_join(pro[2], NULL);

pthread\_join(con[0], NULL);

pthread\_join(con[1], NULL);

pthread\_join(con[2], NULL);

sem\_destroy(&lock);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

**OUTPUT:-**

