Write a programm to implement

1.Producer-consumer problem using semaphores

2.Dining philosopher problem

Producer-consumer problem using semaphores

```
#include <stdio.h>
#include <stdlib.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 item;
   while (1) {
      item = rand() % 100;

      sem_wait(&empty);
      pthread_mutex_lock(&mutex);

      buffer[in] = item;
      printf("Produced: %d\n", item);
      in = (in + 1) % BUFFER_SIZE;
```

```
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
       sleep(1);
    }
}

void* consumer(void* arg) {
    int item;
    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        sleep(2);
    }
}

int main() {
    pthread_t prod_thread, cons_thread;

    // Initialize semaphores and mutex
    sem_init(&empty, 0, BUFFER_SIZE);
```

```
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    // Create threads
    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);

    // Wait for threads to finish
    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);

    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Dining philosopher problem

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_PHILOSOPHERS 5

sem_t chopsticks[NUM_PHILOSOPHERS];

void* philosopher(void* num) {
    int id = *(int*)num;
    int left = id;
    int right = (id + 1) % NUM_PHILOSOPHERS;

    while (1) {
        printf("Philosopher %d is thinking.\n", id);
        sleep(rand() % 3);

        printf("Philosopher %d is hungry.\n", id);

        if (id % 2 == 0) {
            sem_wait(&chopsticks[left]);
            sem_wait(&chopsticks[right]);
        } else {
            sem_wait(&chopsticks[right]);
            sem_wait(&chopsticks[left]);
        }
```

```
        printf("Philosopher %d is eating.\n", id);
        sleep(rand() % 2);

        sem_post(&chopsticks[left]);
        sem_post(&chopsticks[right]);

        printf("Philosopher %d finished eating and put down chopsticks.\n", id);
        sleep(rand() % 2);
    }

    return NULL;
}

int main() {
    pthread_t threads[NUM_PHILOSOPHERS];
    int ids[NUM_PHILOSOPHERS];


    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_init(&chopsticks[i], 0, 1);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, philosopher, &ids[i]);
    }
```

```
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(threads[i], NULL);
    }


    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_destroy(&chopsticks[i]);
    }

    return 0;
}
```

```
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 3 is hungry.
Philosopher 3 is eating.
Philosopher 2 is hungry.
Philosopher 1 is hungry.
Philosopher 0 is hungry.
Philosopher 0 is eating.
Philosopher 4 is hungry.
Philosopher 0 finished eating and put down chopsticks.
Philosopher 2 is eating.
Philosopher 3 finished eating and put down chopsticks.
Philosopher 4 is eating.
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 3 is hungry.
Philosopher 0 is hungry.
Philosopher 4 finished eating and put down chopsticks.
Philosopher 1 is eating.
```