//Solving Producer Consumer Synchronization Problem using Semaphore (multithreading)

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<sys/types.h>
#include<unistd.h>
int buf[5],f=0,r=0;
sem_t mutex,full,empty;
void *produce(void *arg)
{
   int i;
   for(i=0;i<5;i++)
   {
      sem_wait(&empty);
      sem_wait(&mutex);
      printf("produced item is %d\n",i);
      buf[(++r)%5]=i;
     sleep(1);
      sem_post(&mutex);
            sem_post(&full);
      //printf("full %d\n",full);
   }
}


void *consume(void *arg)
{
      int item,i;
      for(i=0;i<5;i++)
      {
            sem_wait(&full);
            sem_wait(&mutex);
            item=buf[(++f)%5];
            printf("consumed item is %d\n",item);
           sleep(1);
            sem_post(&mutex);
            sem_post(&empty);
      }
}
void main()
{
   pthread_t tid1,tid2;
```

```
  sem_init(&mutex,0,1);
  sem_init(&full,0,0);
  sem_init(&empty,0,5);
  pthread_create(&tid1,NULL,produce,NULL);
    pthread_create(&tid2,NULL,consume,NULL);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
}
```

Dining Philosopher Problem Solution (multithreading)

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        // state that eating
        state[phnum] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",
                      phnum + 1, LEFT + 1, phnum + 1);

        printf("Philosopher %d is Eating\n", phnum + 1);

        // sem_post(&S[phnum]) has no effect
        // during takefork
```

```
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}

// take up chopsticks
void take_fork(int phnum)
{

    sem_wait(&mutex);

    // state that hungry
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // eat if neighbours are not eating
    test(phnum);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

    sem_wait(&mutex);

    // state that thinking
    state[phnum] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}
```

```
void* philospher(void* num)
{

    while (1) {

        int* i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{

    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)

        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++) {

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }

    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```