```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.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);
}


#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()
{
  int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);
  printf("\n1.Producer\n2.Consumer\n3.Exit");
  while(1)
  {
    printf("\nEnter your choice:");
    scanf("%d",&n);
    switch(n)
    {
      case 1: if((mutex==1)&&(empty!=0))
                producer();
              else
                printf("Buffer is full!!");
              break;
case 2: if((mutex==1)&&(full!=0))
                consumer();
              else
                printf("Buffer is empty!!");
              break;
      case 3:
              exit(0);
              break;
    }
  }

  return 0;
}

int wait(int s)
{
  return (--s);
}
```

```
int signal(int s)
{
   return(++s);
}

void producer()
{
   mutex=wait(mutex);
   full=signal(full);
   empty=wait(empty);
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);
}

void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);
}
```

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

#define NUM_CUSTOMERS 3

sem_t mutex;

void *customer(void *customerID) {

    while (1) {

        sem_wait(&mutex);
        int customerNum = *((int *) customerID);
        // 顾客到达并尝试获取桌子
        printf("顾客 %d 到达。\n", customerNum);

        // 顾客占用桌子
        printf("顾客 %d 占用桌子。\n", customerNum);

        // 模拟用餐时间
        sleep(2); // 假设为两小时的自助晚餐时间

        // 顾客用餐结束并释放桌子
        printf("顾客 %d 用餐结束并释放桌子。\n", customerNum);
        sem_post(&mutex);
    }
}
```

```
int main() {
    pthread_t customers[NUM_CUSTOMERS];
    int customerIDs[NUM_CUSTOMERS] = {1, 2, 3};

    sem_init(&mutex, 0, 1);

    // 创建顾客线程
    for (int i = 0; i < NUM_CUSTOMERS; i++) {
        pthread_create(&customers[i], NULL, customer, &customerIDs[i]);
    }

    // 等待顾客线程结束
    for (int i = 0; i < NUM_CUSTOMERS; i++) {
        pthread_join(customers[i], NULL);
    }

    sem_destroy(&mutex);

    return 0;
}
```