Implementation of Semaphore:

I create my_semaphore structure using pthread library.

```
pthread_cond_t cond -> a conditional variable to sync threads
pthread_mutex_t mtx  ->used to sync mutual exclusion between threads
volatile unsigned counter -> contains the value of counter used in
original semaphore
```

```
wait(struct my_samaphore *ps) -> used to decrement the counter.
In this function I used pthread_mutex_lock to lock the semaphore mutex
while decrementing the counter value to ensure that no interrupt will
occurs while performing this operation and then pthread_mutex_unlock to
unlock the semaphore mutex and then pthread_cond_signal to signal the
threads that now this is free. if counter value is less than zero then
thread still remains in waiting state unless counter becomes greater
than zero.
```

```
signal(struct my_samaphore *ps) -> used to increment the counter.
In this function I used pthread_mutex_lock to lock the semaphore mutex
while incrementing the counter value to ensure that no interrupt will
occurs while performing this operation and then pthread_mutex_unlock to
unlock the semaphore mutex and then pthread_cond_signal to signal the
threads that now this is free.
```

Working of Code:

I used my_semaphore structure to solve the dining philosopher problem. First I create a mutex of type my_semaphore which acts as a bowl and assigns the counter value with 1. And then N philosopher with initial counter value of 0.To share data between all threads(which is equal to total no philosopher) I used global variables.

To avoid deadlock I used a state array which stores the state of every philosopher i.e. whether it is eating(0), thinking(2)or hungry(1).

Whenever the philosopher comes to eat its state changes to hungry and it will check whether its left and right fork is free or not. If it is free then its state changes to eating, otherwise thinking. If not free then it waits until another thread sends a signal that fork and bowl are free.In non-blocking variant I used infinite while loop which waits until the counter becomes greater than zero.

Note:

I'm treating a pair of bowls as a single entity.