Solution to the Problem:

The deadlock arises in the case where everybody is seated at the table and is trying to acquire his/her left/right chopstick simultaneously.To avoid this **we allow only k-1 people to sit at the table**.(if there are k people).

We do this by using a **counting semaphore** on k-1 people.

If a philosopher is able to sit on the table,he tries to acquire his left chopstick and then the right.All chopsticks are protected by mutex locks.Then he tries to acquire the bowls one by one.This again is protected by mutex locks.

Implementation of Counting Semaphore:

We define our counting semaphore struct with **three** values:
1)The **count**(no of resources)
2)A **mutex_lock** to synchronise changes to the shared variable(count).
3)A **pthread_cond_t** () to implement the blocking version of the counting semaphore.This is the structure which the threads wait on if in case they are blocked. (Also called the **conditional variable**)

sem_create():
This function initializes the semaphore structure.It takes two values:**semaphore structure** and **integer value** we want to initialize with.**pthread_mutex_init()** and **pthread_condt_init()** are used to initialize the lock and conditional variable.

Signal():
This function increases the **count field** in the semaphore **by one**.Since this a critical section,we synchronise this by using the **mutex_lock** in our struct.At the end of the increment,we **signal** all the threads(in case there were threads waiting on the semaphore or more specifically the **pthread_cond_t** struct variable).Signalling is done via **pthread_cond_signal**().

Wait():
This function decreases the count variable by one.As mentioned before since this is a critical section we use mutex_locks.If the count <=0 then the thread execution blocks on the conditional variable.We do this by calling **pthread_cond_wait()**.Only **pthread_cond_signal()** can unblock the thread now.

Reference.