Assignment 4

Name: Vaibhav Girish
Roll number: 209121

For this assignment I implemented my own semaphore using the pthread library and used this to solve the modified dining philosophers problem given to us.
There were two approaches to this:
For both approaches an array of size k would be initialized for this example I used the case k = 5
An array of philosophers, forks and ph each of size k were created.
The array of philosophers was an integer array which stored the integer value corresponding to the particular philosopher.
The array of forks was an array of semaphores for each of the k forks available on the table
The array of ph was an array of threads each corresponding to one of the k philosophers in the question
In addition to this a semaphore array sauce of size 2 corresponds to the two sauce bowls that were being used.

In the main function all of the above were initialized to their respective values.
The semaphore I implemented had an integer count and a pthread mutex which was common to both
It also had an initialization function which would initialize these two values.
The wait and signal function will be discussed below.
The print_signal function would print the current count value
The destroy function would destroy the semaphore ensuring that no other code segment can use the semaphore

1) **Blocking approach:**
   In this approach the wait function would simply block and wait for the signal until the count value becomes 0, it then decrements it and returns. This was used by the philosophers to gain access to forks and sauce bowls, and by setting the count value to 1 I ensured that at a time only one philosopher had access to a particular fork or sauce bowls. This was done by using a conditional pthread, which would wait for a signal.

To avoid deadlocks I used the approach of letting the philosopher at even positions (0,2,4,..) pick up the right fork first and then the left fork whereas the philosophers at odd positions (1,3,5..) would pick up the left fork first and then the right. The philosophers would then try to access the sauce bowls one after another, ensuring that there is no deadlock.
Also once the philosopher is done eating the two adjacent ones tried to eat this ensured that there was no starvation.

2) **Non Blocking approach:**
In the non blocking approach the conditional pthread was removed and instead the wait function would return values .Pthread trywait functions and would return a non zero value if there was no wait available. This way there would not be any blocking. If a philosopher could pick up the right or left fork depending on their position similar to the blocking case they would try for the remaining one and if they could not gain access they would simply drop the fork and try later. The same was the case with sauce bowls. If they could gain access to both only then would they eat otherwise they would simply drop the first one and try later.
This ensures that there is no deadlocks or starvation.

References I used for this assignment were:

1. https://docs.oracle.com/cd/E18752_01/html/816-5137/sync-11157.html
2. https://linux.die.net/man/3/pthread_cond_init
3. https://docs.oracle.com/cd/E19683-01/806-6867/sync-12/index.html