EXERCISE #1

“Dining Philosopher Problem” is one of the classic IPC problems for CS majors to learn IPC using semaphores, as well as the concepts of process starvations and process deadlocks.

In “Dining Philosopher Problem”, there are N processes on a dining table (as shown in the figure below), each of which represents “a (dining) philosopher”. Each dining philosopher repeats the following activities: thinking and eating (dining). For a philosopher to start dining, he (or she) needs two forks (his left and right forks). Note that philosophers share forks.

A model for inter-process-communications (IPC)

- There are $N (N = 5)$ philosophers on a dining table
- Each philosopher has his/her own dish
- Each philosopher needs two forks (left and right)

The following is a solution (“SOLUTION #1”) for “the dining philosopher problem”.
A solution for the Dining Philosophers problem is shown with the following code:

```c
#define N 5 // the number of dining philosophers on a table
semaphore fork[N]; // each fork is represented by a semaphore (a bad design)

void philosopher(int i)
{
    while (TRUE)
    {
        think(); // a philosopher thinks
        wait(fork[i]); // grab the left fork
        wait(fork[(i+1) % N]); // grab the right fork
        eat(); // start eating
        signal(fork[i]); // release the left fork
        signal(fork[(i+1) % N]); // release the right fork
    }
}
```

Questions: Regarding the solution, answer the following questions:

Q1: Is process deadlock possible?
Q2: If yes, how? If no, why not?
Q3: Is process starvation possible?
Q4: If yes, how? If no, why not?
EXERCISE #2

The following is another solution (“SOLUTION #s”) for “the dining philosopher problem”.

**SOLUTION #2**

```c
void philosopher(int i)
{
    while (TRUE)
    {
        think();            // a philosopher thinks
        take_fork(i);  // grab the two forks
        eat();                // start eating
        put_forks(i);    // release the forks
    }
}
```

```c
void take_forks(int i)
{
    wait(MUTEX);
    state[i] = HUNGRY;   // I'm hungry
    test(i);                        // try to get the forks
    signal(MUTEX);
    wait(philosopher[i]);  // I am eating now
}
```

```c
void put_forks(int i)
{
    wait(MUTEX);
    state[i] = THINKING;   // I don't need forks
    test(LEFT);                    // my left waiting for me?
    test(RIGHT);                  // my right waiting for me?
    signal(MUTEX);
}
```

**Questions:** Regarding the solution answer the following questions:

Q1: Is process starvation possible?

Q2: If yes, show how process starvations can happen. If no, explain how process starvations will not happen.

Q3: Is process deadlock possible?

Q4: If yes, show how process deadlock can happen. If no, explain how process deadlocks will not happen.