CS314 Operating Systems
Spring 2024
Exercise Questions on February 20th, 2024 (PART I)

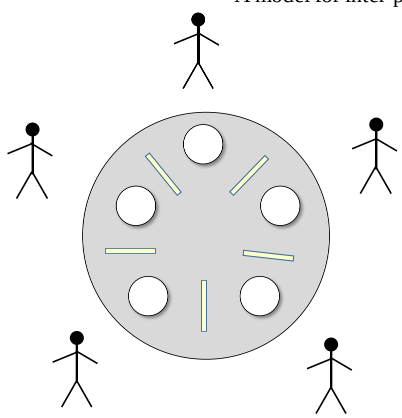
EXERCISE #1

“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.

The Dining Philosophers

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”.

SOLUTION #1

```
#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
    }
}
```

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?

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.

<u>EXERCISE #2</u>

The following is another solution ("SOLUTION #2") for "the dining philosopher problem".

SOLUTION #2

```
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
       }
}
```

```
void test(int i)
{
   if ((state[i] = HUNGRY) & (state[LEFT] != EATING)
                           & (state[RIGHT] != EATING))
   {
         state[i] = EATING;  // I start eating
         signal (philosopher[i]);
    }
}
```

```
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
     }
}
```

```
void put_forks(int i)
{
    wait(MUTEX);

   state[i] = THINKING;   // I don't need forks
    test(LEFT);           // my left waiting for me?
    test(RIGJT);          // my right waiting for me?

    signal(MUTEX);
}
```

Assume:

- N = 5
- semaphore philosopher [N]; // binary semaphore for each dining philosopher
- semaphore MUTEX; // binary semaphore for mutex

- Each philosopher semaphore = 0; // each set to '0'
- The mutex semaphore = 1; // set to '1'

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.