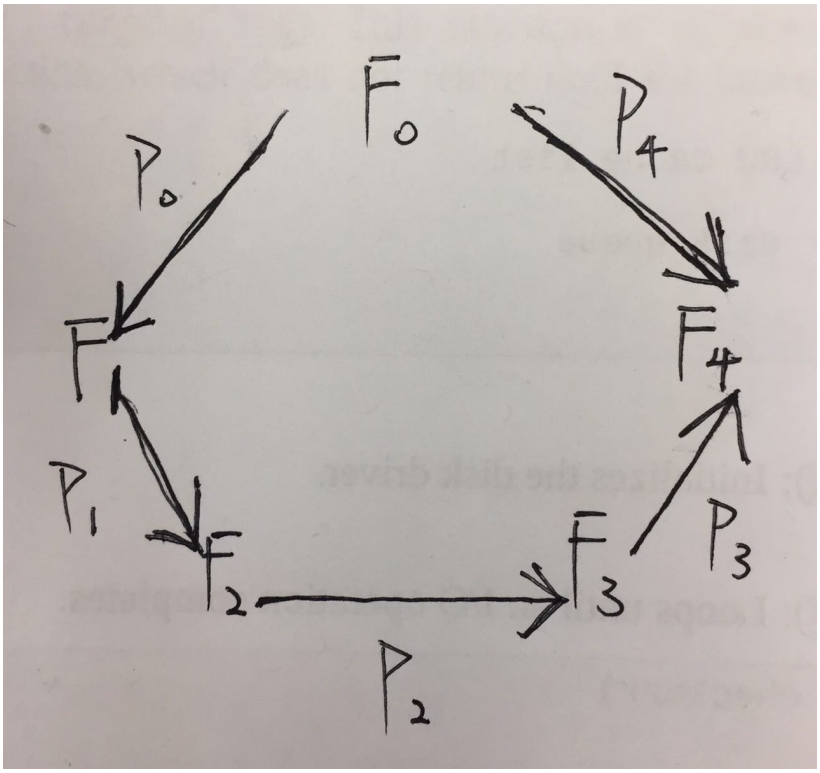
Composed by Yu Zhang(001259692)

Question 1:

As the following diagram shows, philosopher **P4** is forced to grab the fork in a **"first right later left"** order, while others keep in the **"first left later right"** order.

The reason why it solves the deadlock is that it breaks at P4 the circle in the resource dependency graph.

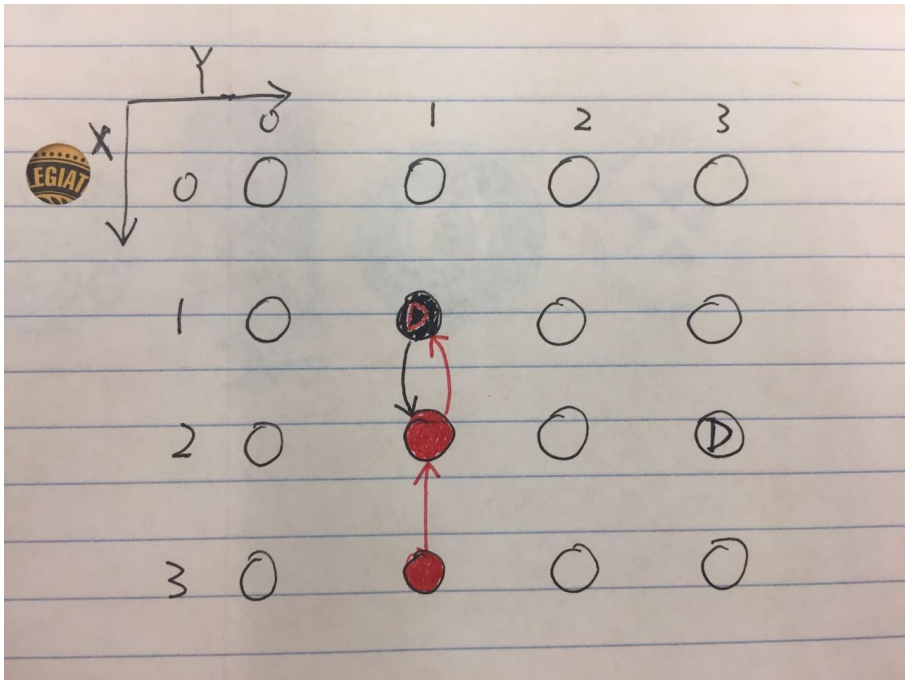


Question 2:

a. See q2.c
b. The program finally reaches the deadlock where packet 1 take as its next node the current node of packet 2 as well as packet 2 wait for packet 1 to leave its current node.
   See the output below,

```
hw8 — q2 — 80×24
packet 0: now at 2/1, next stop is 1/1
packet 0: reaches destination 1/1

packet 1: starts at 3/1, destination is 1/1
packet 1: now at 3/1, next stop is 2/1
packet 1: now at 2/1, next stop is 1/1
packet 0: starts at 3/1, destination is 1/1
packet 1: reaches destination 1/1

packet 0: now at 3/1, next stop is 2/1
packet 0: now at 2/1, next stop is 1/1
packet 1: starts at 3/1, destination is 1/1
packet 0: reaches destination 1/1

packet 1: now at 3/1, next stop is 2/1
packet 1: now at 2/1, next stop is 1/1
packet 1: reaches destination 1/1

packet 1: starts at 3/1, destination is 1/1
packet 0: starts at 1/1, destination is 2/3
packet 1: now at 3/1, next stop is 2/1
packet 0: now at 1/1, next stop is 2/1
packet 1: now at 2/1, next stop is 1/1
```

c. Deadlock grid graph:
The red route represents packet 1, the black route represents packet 0. It is obvious that the resource dependency said at b) contains circle at 2/1->1/1 and 1/1->2/1



d. Eliminate the deadlock:
   A simple strategy to avoid deadlock is keep p0 and p1 in contact, when p0 knows they are in a resource dependency circle (could be implemented by using condition variable to carry the node information), always let p0 release its lock. Restart p0 at the node it just yielded after p1 moves out of the circle.