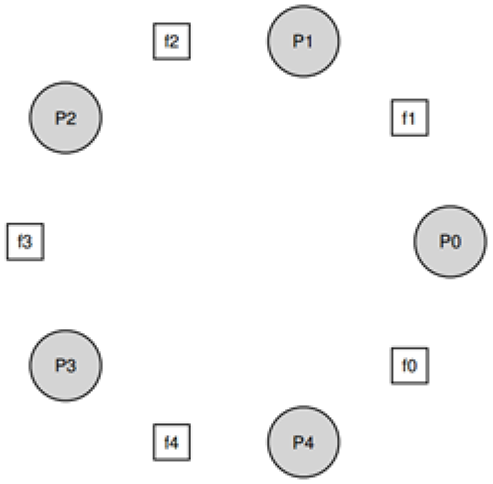
Tutorial Exercise
Deadlocks

UtorID:		Name:	
UtorID:		Name:	
UtorID:		Name:	

1) In the famous dining philosopher's problem, there are 5 philosophers seated at a table with one fork between each of them, as in the picture below.

A philosopher needs two forks to eat, so it has to pick up both the fork on his left and the one on his right!
Each philosopher acts using the following algorithm:

```
// p is the philosopher's ID
void act(int p) {
     while (1) {
         think();
         get_fork(left(p));
         get_fork(right(p));
         eat();
         put_fork(left(p));
         put_fork(right(p));
     }
}
```



Explain the assumptions we need to make about this problem to satisfy the necessary and sufficient conditions for deadlock:

a) Mutual exclusion

b) Hold-and-wait

c) No preemption

d) Circular wait (give a scenario where this would occur)

2) How would you change the algorithm to prevent deadlock?

3) Consider that some philosophers always pick up their left forks first (name these philosophers "lefties"), while other philosophers always pick up their right forks first (we'll call them "righties). Let's also consider that there is at least one "lefty" and at least one "righty" at the table. Can deadlock occur? Is starvation possible (you can assume that we have a fair scheduling policy)?

4) Deadlock avoidance and the Banker's algorithm.
Suppose that we have a system with 8 pages of memory and 3 processes: A, B, and C, which need 4, 5, and 5 pages to complete respectively (Assume no eviction).

If they take turns requesting one page each, and the system grants requests in order, the system will deadlock.

A (needs 4)	1	1	1	2	2	2	3	3	3	W	W
B (needs 5)	0	1	1	1	2	2	2	3	3	3	W
C (needs 5)	0	0	1	1	1	2	2	2	W	W	W
Total allocated	1	2	3	4	5	6	7	8	8	8	8

First, explain to your partner(s) why this is a deadlock.

Which allocation is the last safe state? In other words, which is the last state where there is still some allocation that would allow the processes to complete. Remember that once a process has acquired all of its required resources (pages in this case), it will eventually release them all.