1. P0  
 1. If (token)

2. CS

3. token = T

~~~~~~~~~~~~~~~~

P1

1. if (token)

~~~~~~~~~~~~~~~~

P0

1. if (token)

~~~~~~~~~~~~~~~~

P1

2. CS

~~~~~~~~~~~~~~~~

P0

2. CS

Therefore we will violate the rule of mutual exclusion with the proposed algorithm.

2. This algorithm must prove both aspects of progress. First, it must ensure that it cannot be in a stuck state. This algorithm cannot get stuck because the turn variable will not allow it. The turn will either be a 0 or 1, therefore one process will always get to the critical section. Second, this algorithm must ensure that it cannot be in a deadlocked state. In the algorithm if the other process wants to run (flag check) then we check to make sure it is their turn. If it is, we set our turn to false. Therefore, we can never enter a deadlocked state.

3. If the wait() and signal() functions were not atomic, it is easy to see that multiple processes can execute the critical section at the same time. If both processes check to see if they are “locked” out at the same time initially, they will both enter the critical section and lock any other processes out, therefore violating the mutual exclusion principle.

4. The first condition of deadlock is mutual exclusion. The dining philosophers concept ensures mutual exclusion between all processes because one philosopher can only own a chopstick at a time. The second condition of deadlock is hold and wait. In the dining philosophers concept a philosopher can own the chopstick to his left and attempt to use the chopstick to his right as well, therefore utilizing the hold and wait principle. The third condition of deadlock is no preemption. In the dining philosophers concept a philosopher cannot forcibly remove a chopstick from another philosopher. The last condition of deadlock is circular wait. In the dining philosophers if all philosophers pick up the stick to the left of them we see an example of circular wait.

If a philosopher could forcibly remove a chopstick from another philosopher, we would be using preemption. This will remove one of the four conditions of deadlock.

5. struct [sem](http://lxr.linux.no/linux+v3.8/+code=sem) {

int [semval](http://lxr.linux.no/linux+v3.8/+code=semval); ***/\* current value \*/***

int [sempid](http://lxr.linux.no/linux+v3.8/+code=sempid); ***/\* pid of last operation \*/***

struct [list\_head](http://lxr.linux.no/linux+v3.8/+code=list_head) [sem\_pending](http://lxr.linux.no/linux+v3.8/+code=sem_pending); ***/\* pending single-sop operations \*/***};