```
class PetersonAlgorithm implements Lock {
    boolean wantCS[] = {false, false};
    int turn = 1;
    public void requestCS(int i) {
        int j = 1 - i;
        wantCS[i] = true;
        turn = i;
        while (wantCS[j] && (turn == j)) ;
    }
    public void releaseCS(int i) {
        wantCS[i] = false;
    }
}
```

3a) The above shows Peterson's algorithm modified so that the process sets the turn variable to itself. In this case, if we had two processes, process 0 and 1, if they both requested the critical section (requestCS(0) and requestCS(1)), there is a case where they are both in the critical section at the same time. If process 1 sets the turn to itself, it would not wait in the while loop and proceed to enter the critical section. While process 1 is in the critical section, if process 0 sets the turn to itself, it would similarly bypass the while loop and enter the critical section.

Therefore, we have shown a case that violates the safety property.

```
class PetersonAlgorithm implements Lock {
    boolean wantCS[] = {false, false};
    int turn = 1;
    public void requestCS(int i) {
        int j = 1 - i;
        turn = j;
        wantCS[i] = true;

        while (wantCS[j] && (turn == j)) ;
    }
    public void releaseCS(int i) {
        wantCS[i] = false;
    }
}
```

3b) The above shows Peterson's algorithm modified so that the turn variable is set before the wantCS variable. In this case, if we had two processes, process 0 and 1, if they both request the critical section (requestCS(0) and requestCS(1)), there is a case where they both are in the critical section at the same time.

If process 0 enters requestCS and sets turn to 1, then is interrupted by process 1 which sets turn back to

0, wantCS[1] = true, it will then enter the critical section (because wantsCS[0] = false). If process 0 then resumes, wantCS[1] will be true, but turn will equal 0, not 1, so process 0, too, can enter the critical section.

Therefore, we have shown a case which violates the safety property.

4) Starvation is when one process can repeatedly enter the critical section while another process is denied the critical section. If process 0 and 1 both request the critical section and process 1 "wins" (i.e. enters the critical section), turn is set to 1 by process 0. Now, process 0 is waiting because waitCS[1] = true and turn is also equal to 1. But once process 1 releases the critical section, wantCS[1] is now false. Now, if process 0 where to continue executing, it will enter the critical section because wantCS[1] is false. If process 1 requested the critical section again before process 0 were to continue executing, it would wait at the while loop because process 1 has to set turn back to 0. Eventually, the scheduler will give back control to process 0 and it will enter the critical section.