1. Solution

```
boolean flag[0..1] := false;
integer turn := 0 or 1;

entry(i) {
flag[i] := true;
turn := j;
while flag[j] and turn=j
        do skip;
  }

exit(i) {
flag[i] := false;
  }
```

2. Proof of Safety

It is sufficient to show that when one process enters the CS the other process is not in the CS.

- **Case 1**. Suppose P0 enters CS at time t1. Thus at t1, either flag[1]=false or turn!=1 (note that this takes care the non-atomic if-condition checking).
- **Case 1.1**. Suppose flag[1]=false at t1. Then P1 is not in CS, because P1 sets flag[1] to true in Entry(1) and does not change it in CS, and P1 does not change flag[1].
  end 1.1
- **Case 1.2**. Suppose turn!=1 and flag[1]=true at t1. Then turn=0 at t1, because it is initially 0 or 1 and is set to either 0 or 1. We do a proof by contradiction. Assume P1 is in CS at t1, and let it have last entered the CS at time t2 (< t1). Thus at t2, either flag[0]=false or turn!=0.
  - **Case 1.2.1**. Suppose flag[0]=false at t2. Then P0 is in NCS at time t2. Therefore at some time t3 in the interval (t2, t1), P0 sets turn to 1. From t3 to t1, P0 does not change turn. From t2 to t1, P1 remains in CS, hence P1 cannot set turn to 0. Hence turn should be 1 at t1. Contradiction.
    end 1.2.1
  - **Case 1.2.2**. Suppose turn!=0 and flag[0]=true at t2. Then turn=1 at t2. During (t2, t1), P1 remains in CS, so it does not change turn. P0 cannot set turn to 0. Hence turn would say 1 at t1. Contradiction.
    end 1.2.2
    end 1.2
    end 1
- **Case 2**. Suppose P1 enters CS at time t1. Then we have that P0 is not in CS at t1 by a symmetric argument of Case 1 [i.e., Case 1 with process ids, flag indices, and turn values interchanged].
  end 2

end of safety proof

3. Proof of Progress for P0

Suppose P0 becomes hungry at time t1 (i.e., executes 1).
We need to show that P0 eventually enters the CS, i.e., gets past 3.
At some time t2 (> t1), P0 starts testing the condition in 3 (because 1 and 2 cannot block).

Case 1: P1 is thinking at t2.
Then flag[1]=false at t2, when P0 executes 3, and P0 gets past 3 at t2.
Case 2: P1 is eating at t2.
Then turn=1 and P1 has to wait (recall that we already proved safety).
Eventually at some time t3 (>t2) P1 finishes eating and does 4.
  Case 2.1: P1 stays thinking until P0 tests condition.
  Then P0 gets past 3 (at some time after t3).
  Case 2.2: P1 becomes hungry at time t4 (>t3) before P0 tests condition.
  Then P1 eventually executes 3, setting turn to 0.
  After this turn remains 0 and P0 eventually gets past 3 (at some time after t4).
Case 3: P1 is hungry at t2.
Then flag[1]=true at t2, and turn determines who gets to eat next.
  Case 3.1: P1 was the last to update turn.
  Then turn=0 at t2 and P0 gets past 3.
  Case 3.2: P0 was the last to update turn.
  Then turn=1 at t2, P0 waits, and P1 gets past 3 at some time t3 (>t2).
  At this point, P1 is eating and this case reduces to case 2.
end of progress proof for P0

4. Proof of progress for P1

Symmetric to the argument of progress proof for P0.

5. Comments

- As you can see, the progress proof is more involved than the safety proof. This is typical of concurrent programs: that is, ensuring that an algorithm achieves progress often requires more careful thinking than ensuring that the algorithm does not do anything wrong.
- In the safety proof, we resorted to proof by contradiction in case 1.2. Show this case without resorting to contradiction. You have to handle two sub cases: (1) turn=0 at t1 because P0 set it so at some time in the past; and (2) turn=0 at t1 because it was initially so and P0 has not updated it so far.