Peterson's Algorithm

Process P_0

```
while(1) {
  intendToEnter[0] = 1;          (1)
  turn = 1;                      (2)
  while(intendToEnter[1] &&
        turn == 1);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[0] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Process P_1

```
while(1) {
  intendToEnter[1] = 1;          (1)
  turn = 0;                      (2)
  while(intendToEnter[0] &&
        turn == 0);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[1] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Mutual Exclusion

- For both P_0 and P_1 to be in their Critical Section
  - Both intendToEnter[0] and intendToEnter[1] must be 1 AND
  - turn = 0 and turn = 1 (at the same time) ← Impossible!!!
- Therefore, the mutual exclusion is met

Peterson's Algorithm

Process P_0

```
while(1) {
  intendToEnter[0] = 1;          (1)
  turn = 1;                      (2)
  while(intendToEnter[1] &&
        turn == 1);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[0] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Process P_1

```
while(1) {
  intendToEnter[1] = 1;          (1)
  turn = 0;                      (2)
  while(intendToEnter[0] &&
        turn == 0);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[1] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Progress

- P_0 can be kept out of Critical Section only if stuck in while loop, i.e. intendToEnter[1] = 1 and turn = 1
  - If P_1 is NOT ready to enter Critical Section
    - intendToEnter[1] = 0 → P_0 can then enter its critical section
- If P_1 has set intendToEnter[1] = 1, it is also in its while loop, then either P_0 or P_1 will go depending on value of turn
- Therefore, the progress condition is met

Peterson's Algorithm

Process P_0

```
while(1) {
  intendToEnter[0] = 1;          (1)
  turn = 1;                      (2)
  while(intendToEnter[1] &&
        turn == 1);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[0] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Process P_1

```
while(1) {
  intendToEnter[1] = 1;          (1)
  turn = 0;                      (2)
  while(intendToEnter[0] &&
        turn == 0);              (3)
  CRITICAL SECTION               (4)
  intendToEnter[1] = 0;          (5)
  REMAINDER SECTION              (6)
}
```

Bounded Waiting

- If P_1 enters Critical Section, then turn = 1
  - But will then reset intendToEnter[1] = 0 on exit
  - This allows P_0 to enter Critical Section
- If P_1 tries again, and has time to reset intendToEnter[1] = 1 before P_0 gets to its Critical Section
  - It must also set turn = 0
- Since P_0 is stuck at the while loop, P_0 will get to enter the Critical Section after at most one Critical entry by P_1
- Therefore, the bounded waiting is met

Bounded-Waiting Mutual Exclusion with Test-and-Set

```
do {
    waiting[i] = 1;
    while(waiting[i] && test_and_set(&lock)); // key = test_and_set(&lock)
    waiting[i] = 0;

    // critical section

    // Scan waiting array in the cyclic ordering
    // i+1, i+2, ..., n-1, 0, ..., i-1, and find the first process
    // waiting[j] == 1
    j = (i + 1) % n;
    while((j != i) && !waiting[j])
        j = (j + 1) % n;
    if(j == i)
        lock = 0;
    else
        waiting[j] = 0;

    // remainder section
} while (true);
```