Peterson's Solution for Critical Section

Peterson's Solution is used for mutual exclusion and allows two processes to share a single-use resource without conflict. It uses only shared memory for communication. Peterson's Solution originally worked only with two processes, but hence been generalized for more than two.

Before using the shared variables (i.e before entering its critical region), each process calls enter_region with its own process number 0 or 1, as parameter. This call will cause it to wait, if need be until it is safe to enter.

ANSI C Code

```
Sdefine FALSE 0
Sdefine TRUE = 1
#define N
 int turn;
INT interested [N]
 Void enter_region (int Process).
{  int other
    other = 1-process;
   interested[process = TRUE;
    turn = process;
while(turn == process&& interested[other]==TRUE)
}
 Void leave_region (int process)
 { interested f process -f ALSE
 }
```

After it has finished with the shared variables, the process calls Leave_region to indicate that it is done and to allow the other

process to enter.

→ process 0 calls enter^region by setting its array element and sets 'turn' to 0.

→ If process 1 now makes a call to enter^region, it will hang there until interested[0] goes to FALSE, an event that only happens when process 0 calls Leave^region to exit the critical region.

Now Consider the case that both processes call enter-region almost simultaneously. Then both will store their process no. in turn. Which ever store is done last is the one that counts. the first one is overwritten and lost. Suppose that process 1 stores last, so turn is 1, When both processes come to the while statement, process 0 executes it zero times and enters its critical region.
Process 1 loops and does not enter its critical region until process 0 exists its critical region.