Quiz 3, Q&A

Q. What is the Mutual Exclusion?

A. Mutual exclusion is a mechanism to ensure that only one process (or person) is doing certain things at one time, thus avoid data inconsistency. All others should be prevented from modifying shared data until the current process finishes

Q Explain the Strict Alternation Solution.

A. Strict Alternation (see Fig. 1)

the two processes strictly alternate in entering their CR

the integer variable turn, initially 0, keeps track of whose turn is to enter the critical region

busy waiting, continuously testing a variable until some value appears, a lock that uses busy waiting is called a spin lock

both processes are executing in their noncritical regions

process 0 finishes its noncritical region and goes back to the top of its loop

unfortunately, it is not permitted to enter its CR, turn is 1 and process 1 is busy with its nonCR

this algorithm does avoid all races

but violates condition 3 (Fault tolerance-processes running outside their CR should not block with others accessing the CR)

```
while (TRUE) {
    while (turn != 0)          /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)          /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Figure 1 A proposed solution to the CR problem. (a) Process 0, (b) Process 1

Q Explain the Petersons's solution.

A. Petersons's solution (see Fig. 2)

does not require strict alternation

this algorithm consists of two procedures

before entering its CR, each process calls enter_region with its own process number, 0 or 1

after it has finished with the shared variables, the process calls leave_region to allow the other process to enter

consider the case that both processes call enter_region almost simultaneously

both will store their process number in turn . Whichever 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 enters its critical region

process 1 loops until process 0 exists its CR

no violation, implements mutual exclusion

burns CPU cycles (requires busy waiting), can be extended to work for n processes, but overhead, cannot be extended to work for an unknown number of processes, unexpected effects (i.e.,priority inversion problem)

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 - process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

Figure 2: Peterson' solution for achieving mutual exclusion