1. A friend thinks that atomic instructions are unnecessary. Here is his solution for mutual exclusion using a global "occupied" flag. Show a scenario (i.e., timeline of actions with TiSj labels) in which it fails.

initially: occupied = 0;

// ---- thread 1 ---- // ---- thread 2 ----

T1S1: while( occupied ) ; // spin T2S1: while( occupied ) ; // spin

T1S2: occupied = 1; T2S2: occupied = 1;

T1S3: // critical section T2S3: // critical section

T1S4: occupied = 0; T2S4: occupied = 0;

T1S1; T2S1; T2S2; T1S2; T1S3 in critical section; T2S3 in critical section

Since both threads are in the critical section simultaneously, this is a failure of mutual exclusion.

2. Here is another solution for mutual exclusion using a global "occupied" flag, but it uses a test\_and\_set atomic instruction. Why does it not fail in the same way as the code in question 2?

initially: occupied = 0;

// ---- thread 1 ---- // ---- thread 2 ----

T1S1: while( test\_and\_set(&occupied) ) ; T2S1: while( test\_and\_set(&occupied) ) ;

T1S2: // critical section T2S2: // critical section

T1S3: occupied = 0; T2S3: occupied = 0;

Occupied

0

T1S1 test\_and\_set <-> 1  
Returns 0 which it   
read; fall out of loop

1 <-> T2S1 test\_and\_set  
Returns 1 which it read;  
stays in loop