Question 1

• Race condition: the result of the computation depends on the relative speed of two or more processes
  – Occur non-deterministically
  – Hard to debug

```c
void insert(struct node *item) {
    item->next = head;
    head = item;
}
```

```
void insert(struct node *item) {
    item->next = head;
    head = item;
}
```
void insert(struct node *item)
{
    item->next = head;
    head = item;
}

• Question: How many states?
• $3^N$

N processes
while(TRUE) {
    while(lock == 1);
    lock = 1;
    critical();
    lock = 0
    non_critical();
}

while(TRUE) {
    while(lock == 1);
    lock = 1;
    critical();
    lock = 0
    non_critical();
}
Question 4

- A uniprocessor system runs one thread at a time
- Concurrency arises from preemptive scheduling
- The scheduler is invoked on a timer interrupt
  - Disabling interrupts disables preemptive scheduling and guarantees atomicity
Question 5

```c
void mutex_lock(bool* lock) {
    if (test_and_set(lock) == 1) {
        sleep();
    }
}

void mutex_unlock(bool* lock) {
    *lock = 0;
    wakeup();
}
```

- The `wakeup()` is lost
Question 6

```c
int count = 0;
#define N 4 /* buf size */
prod() {
    while(TRUE) {
        item = produce()
        if (count == N)
            sleep();
        insert_item();
        count++;
        if (count == 1)
            wakeup(con);
    }
}

con() {
    while(TRUE) {
        if (count == 0)
            sleep();
        remove_item();
        count--; 
        if (count == N-1)
            wakeup(prod);
    }
}

Concurrent uncontrolled access to the buffer
```
Question 6

int count = 0;
#define N 4 /* buf size */

prod() {
    while(TRUE) {
        item = produce()
        if (count == N)
            sleep();
        insert_item();
        count++;
        if (count == 1)
            wakeup(con);
    }
}

con() {
    while(TRUE) {
        if (count == 0)
            sleep();
        remove_item();
        count--;
        if (count == N-1)
            wakeup(prod);
    }
}

Concurrent uncontrolled access to the counter