Jay Patel

CS 352

HW4

Professor Johnny Wang

5.8

Answer: The three conditions are being satisfied by the algorithm of mutual exclusion.

The three conditions are as follows:

a) The use of the flag and turn variables are being ensured b the mutual exclusion. Let's say if both the processes have true as their flag, only one would succeed, namely whose turn it is. The waiting time can only entre its critical section when the other process updates, the value of turn.

b) Progress is provided through the turn variables and flag, but it would not provide strict alternation for this particular algorithm. Instead of that, if the process gets to the access with their critical section, it can set the flag variable to true and enter their critical section. It will repeat the process of entering its section if the process wants to enter the critical section before the other process.

c) The use of the turn variable is bounded waiting is preserved through the use of the turn variable. Let's say there are two processes that wants to enter their respective sections. They both set their value of flag as true, but only the thread whose turn it can proceed and other has to wait. If they are not preserved, it

would have to wait indefinitely while the first process repeatedly entered and exited.

But Dekker's algorithm has a process set the value of turn to the other process, therefore ensuring that the other process will enter its critical section next.

5.11

Answer: Disabling interrupts only prevents other processes from executing on the processor in which interrupts were disables because Interrupts are not sufficient in multiprocessor systems. The process disabling interrupts cannot guarantee mutually exclusive access to program state because there are no limitations on what processes could be executing on other processors.

5.16 A queue should be made so that instead of busy waiting it is able to execute code in order to avoid the busy waiting.

```
1
2   acquire()
3   {
4       if (!available)
5       {
6           enqueue
7       }
8
9       available = false;
10  }
11
12
13
14  release ()
15  {
16      if(queue not emplty)
17      {
18          move processs at the head of queue
19      }
20
21      available = true;
22  }
```

5.29

Answer: The signal() operations that are with monitors is not because of the following: if a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember the face that the signal took place.
If a subsequent wait operation is performed, then the correspond thread simply blocks in semaphores, also every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. Because of the earlier increment a future wait operation would immediately succeed.

5.32

Answer:

```
//5.32

monitor fs
{
    make an enum that has 3 values; thinking, waiting and reading with a state[N]
    condition self [N];
    int sum;

    void unlocked(int i)
    {
        make state at i as waiting
        check if the total + i is greater than N
        if it is then make self at i wait
        else have the state at i as reading
        increment the total with i
    }

    void locked(int i)
    {
        make state at i as thinking
        decrement total with i
        loop from N - total - 1 to x greater than or equal to 0 with x decrementing
        check if the state at x is watiing
        if it is then make self at x as signal
        and break
    }

    intialization.code()
        loop from 0 to n with i in increasing order
        and make state at i as thinking
        outside the loop;
        have total is 0

}
```