Part 2 - Theoretical

A)

i) Incorrect ! - I will provide a counter example:

1. Thread 1 starts: id = 1.
    1. value = 1
    2. Thread 1 performs swap: value = lock = 0, lock = 1.
    3. Enters Critical Section.
2. Thread 2 starts, id = 2.
    1. value = 2
    2. performs swap: value = lock = 1, lock = 2.
    3. Does not enter CS, return to the while, value = 2
    4. performs swap: value = lock = 2, lock = 2.
    1. enters Critical Section - No Mutual Exclusion -

ii) Correct !

We'll assume there are 2 threads deadlocked inside the while loop. That means that 'lock' must be 0 to satisfy the while condition after swap.

But, if we are deadlocked inside the while loop, and lock is kept 0, that means that value must become zero somewhere, thus breaking the while condition, leading to entrance of the Critical Section and making, in negate with the assumption that lock = 0 (as it entered the Critical Section).

iii) Incorrect ! - i will provide a counter example:

1. Thread 1 runs and enters Critical section.
2. Thread 2 runs and stuck inside the while loop. (same to section 'i' above)
3. Thread 1 finishes, lock = 0.
4. Thread 2 DOES NOT finish, Thread 1 enters "do" again, entered Critical Section again.

Continuing this steps over and over again, retaining same values and Thread 2 stuck at the while loop.

B)

a) False

Deadlock is a private case of Starvation, a counter example is given at Part A - i (above), for a deadlock without starvation.

b) False

When there are several executions (e.g. calculations / file manipulation / other operations relevant) the processor can manage another thread to work while the first thread waits for. Therefor increasing the time working on the job the program tried to do.

So is it not correct that it will <u>always</u> yield better results single threaded.

c) False

For start we'll note that a critical section is not necessarily an atomic procedure.

Therefor, as long as the context switch is done when critical section is not Atomic, it's possible. (and not possible when CS is Atomic).

C) Next Page

Part C

I will use 5 counting semaphores.
Initial values:
S1 = 2.
S2 = S3 = S4 = S5 = 0.

Process 1:

```
while(true)
{
    down(s1)
    down(s1)
    // Run process 1
    up(s2)
}
```

Process 2:

```
while(true)
{
    down(s1)
    // Run process 2
    up(s3)
    up(s4)
}
```

Process 3:

```
while(true)
{
    down(s3)
    // Run process 3
    up(s5)
}
```

Process 4:

```
while(true)
{
    down(s4)
    // Run process 4
    up(s1)
}
```

Process 5:

```
while(true)
{
    down(s5)
    // Run process 5
    up(s1)
}
```