Q1

Assume we have 3 threads $A,\ B,\ C$ and they all try to enter level 1 together at the following order:

$$\text{write}_A\left(\text{level}\left[A\right]=1\right)\to\text{write}_A\left(\text{victim}\left[1\right]=A\right)\to\text{write}_B\left(\text{level}\left[B\right]=1\right)\to\text{write}_B\left(\text{victim}\left[1\right]=B\right)\to$$
$$\to\text{write}_C\left(\text{level}\left[C\right]=1\right)\to\text{write}_C\left(\text{victim}\left[1\right]=C\right)$$

so clearly, $D_A\to D_B$, now we can examine the following possible chain of events:

$$\to\text{read}_B\left(\text{victim}\left[1\right]=C\right)\to\text{write}_B\left(\text{level}\left[B\right]=2\right)\to\text{write}_B\left(\text{victim}\left[2\right]=B\right)\to$$
$$\to\text{read}_B\left(\text{level}\left[A\right]=1\right)\to\text{read}_B\left(\text{level}\left[C\right]=1\right)$$

since all threads but B are at level ≤ 2 , it's possible for B to enter it's critical section:

$$\to CS_B\to\text{write}_B\left(\text{level}\left[B\right]=0\right)$$

It's only natural to have B trying to enter level 1 again:

$$\to\text{write}_B\left(\text{level}\left[B\right]=1\right)\to\text{write}_B\left(\text{victim}\left[1\right]=B\right)$$

Notice that nothing keeps C from reading $\text{victim}\left[1\right]=B$ before A does, so it's possible to continue with

$$\to\text{read}_C\left(\text{victim}\left[1\right]=B\right)$$

and we can let C reach (and finish) it's CS before A reads $\text{victim}\left[1\right]\neq A$:

$$\to CS_C\to\text{write}_C\left(\text{level}\left[C\right]=0\right)$$

In the same manner as above we can send B and C interchangeably to finish an unbounded number of "re-wind $\to$level-up$\to$ CS" execution cycles before having A reaching the event $\text{read}_A\left(\text{victim}\left[1\right]\neq A\right)$. So although $D_A\to D_B$, we get that for any $k\geq 0$ it's possible for ${CS_B}^{1+k}\to CS_A$ to occur.

Q2

Lemma. *Two threads performing only their first event will cause the shared memory state to be the same, regardless of order*

Proof. *If The first operation in lock() is a* read*: reading obviously can't change the shared memory state.*
If The first operation in lock() is a write*: because it is the first operation, it means that the data that is written is constant, and it always writes to the same register. This is because the thread didn't read anything, including any information that can cause it to change the data written or location to write to. Therefore, order of writing has no effect.*

We will let T1 perform the first action, then T2 will perform the first action. Then, we will let one of the threads run until it unlocks (It is deadlock-free so this must happen). We claim that regardless of the choice of thread to run, it will behave the same as the other thread will behave. This is because the shared memory state is identical regardless of choice of the first thread(Lemma). Therefore, it doesn't matter which thread started first, a contradiction to strong FCFS.

Q3

1. Normalizing the (total) sequential execution time of the program to be 1, while the execution time is comprised out of M which is 40% of the total time - $\frac{2}{5}$ then the unparallelizable portion of the program is $1-p=\frac{2}{5}$ thus $p=\frac{3}{5}$. Using Amdahl's formula we get

$$\begin{aligned}S&=\frac{1}{\frac{2}{5}+\frac{3/5}{n}}\\&=\frac{1}{\left(\frac{1}{5}\right)\left(2+\frac{3}{n}\right)}\\&=\frac{5n}{2n+3}\end{aligned}$$

   which gives us an upper bound of 2.5 folds of overall speedup.

2. Assume the sequential execution time of the program is T.. Since the rest of the program is completely parallelizable, we can express the total running time on n processors as $\frac{3}{10}T + \frac{\frac{7}{10}T}{n}$.

   denote M's speedup by S', then $\frac{\frac{3T}{10}}{S'} + \frac{\frac{7T}{10}}{n} = \frac{\frac{3T}{10} + \frac{\frac{7T}{10}}{n}}{2}$.

$$\begin{aligned}
\frac{T}{10}\left(\frac{3}{S'} + \frac{7}{n}\right) &= \frac{T}{10}\frac{3+\frac{7}{n}}{2} \\
&\Downarrow \\
\frac{3}{S'} + \frac{7}{n} &= \frac{3+\frac{7}{n}}{2} \\
&\Downarrow \\
\frac{3}{S'} &= \frac{3-\frac{7}{n}}{2} \\
&\Downarrow \\
S' &= \frac{3\cdot 2}{3-\frac{7}{n}} = \frac{6n}{3n-7}
\end{aligned}$$

3. Denote M's part in the overall execution time by $1-p$, given that $\frac{1}{\frac{1-p}{3}+\frac{p}{n}} = \frac{2}{1-p+\frac{p}{n}}$ (twice the speedup compared to running on n processors without M being sped up), solution is $1-p = \frac{3}{n+3}$.

Q4

1. Without loss of generality, B finished the lock() code after A did (finishing is running Line 11 when turn $==$ me). Let's mark the last time thread A executed Line 8 as t1, and similarly t2 for B. Because B finished after A, t1<t2. Because t1 is the last time A executed Line 8, A must finish Line 9 before B runs Line 10. So in the (immediately) next time A runs Line 11 "turn" is not equal to "me", and A runs Line 8 one more time - a contradiction for t1 being the last time A ran Line 8.

2. Counter example: A sets "turn", A sets "busy" to TRUE, B sets "turn", A checks if it's his turn (it's not) so it loops in the inner "while" until someone else calls unlock(). However, B is now also stuck in the loop because "busy" is true. And actually any other thread that wants to lock will get stuck in the loop, and no-one can finish the lock(), so no-one can unlock().

3. Starvation-free implies deadlock-free.

Q5

If we have k MRSW registers and N threads, if $k < N$, then by the pigeonhole principle, we have at least two threads that will write to the same register before entering CS. By the definition of MRSW,