Exam 1

Proposed solution for the first exam OS.

Question 1-

1- The term “race condition” means that more than one process are interested by enter in the CS, such that only one may enter. That’s why we’re speaking about “race” as some cars can have a race (the goal is to be the first to reach the “line” which is in our case the CS).
As an example, we can provide the next one:
When we’re starting our computer, a lot of processes have some work to done. Nevertheless, sometime, there is no possibility that two processes act in the same place. This can be explain by the fact that one process may change some value of variable such that the second process is not agree with this, or either doesn’t know that a change have been done, and this may be a terrible issue. Then, we must use synchronous algorithm.

2- The monitor is the process which begin to run the main method of our program. Thus, at the beginning of any program, the monitor which is a process begin to run, and if at any time of the program he needs to create another process (use of fork for example), the monitor will create the first process, and from here there will be two processes which will run the program.
WRONG ANSWER.

3- This question is answered in a paper.

Question 2-

We provide a pseudo code as an answer of the problem.

SHARED:
int count = 1
semaphore mutex = 1

Code:

```
while(1)
{
        if(thread.id == 0)
                await(count % 3 = 1)
        else if (thread.id == 1)
                await(count % 3 = 2)
        else if (thread.id == 2)
                await(count % 3 = 0)
        DOWN(mutex)
        print(thread.id) * count
        count++
        UP(mutex)
}
```

Question 3-

Mutual exclusion:
Assuming for the sake of the contradiction that the algorithm doesn't provide mutual exclusion, our goal is the show that there is no possibility that two processes enter in the CS at the same time.
Assume that 0 is already in the CS. By examining the code we have:
$Write_0$(flag[0] = true) → $Read_0$(flag[1] = false) → CS
Then, assuming 1 is interested by enter in the CS, by examining the code we have:
$Write_1$(flag[1] = true) → $Read_0$(flag[0] = true) → stuck in the while loop, and no access to the CS.
Since that in the while loop of the process 1 there is no possibility to change the value of flag[0], there is no possibility for the process 1 to enter in the CS, during the process 0 is already here.
Now, let's see what if both processes are in the while loop.
Since next either 0 or 1, one of the process will be stuck in the await loop, and the second will enter in the CS.
This is a contradiction of our assumption, then, the algorithm provide mutual exclusion.

Deadlock free:
We will prove something stronger, which is starvation free. Then, there is no need to prove this statement since it's always true when starvation freedom is true.

Starvation free:
Assuming for the sake of the contradiction that the algorithm doesn't provide starvation freedom.
Then, assume that the process 0 is stuck forever in the entry code.
By examining the code we have:
$Write_0$(flag[0] = true) → $Read_0$(flag[1] = true) → stuck in the while loop.
When the assumption that flag[1] = true is trivial because if not the process 0 will enter in the CS.
Then, once again, by examining the code, since we have flag[1] = true, we know that process 1 is also interested by the CS, there is two possibilities:
The first one is that process 1 is already in the CS, then, as we can't assume that process 1 is stuck forever in the CS, he will after a finite amount of time enter in the exit code. By examining the code, we have $Write_1$(next = 0) → $Write_0$(flag[1] = false).
From now if process 0 gets running time he will be able to enter in the CS. If not, and assuming process 1 is interested by enter in the CS again, and he gets running time, he will enter in the wile loop since by examining the code we have:
$Write_1$(flag[1] = true) → $Read_1$(flag[0] = true) → stuck in the while loop.
Since next = 0, by examining the code of the process 1 we have:
$Read_1$(next = 0) → $Write_1$(flag[1] = false) → $Read_1$(next = 0) → stuck in the await loop.
Obviously, process 0 will enter in the CS at the moment he will gets running time.
The second one is that process 1 is also stuck in the while loop. Then, next is either 1 or 2, so, as we saw in the mutual exclusion proof, one of both process will enter in the CS and the second will be stuck in the await loop, then, we are in the same situation that the first one.
Then, there is no possibilities for the process 0 to be stuck forever in the entry code. The proof for the process 1 is the same.