Assignment 20

State Aravind's algorithm. List its properties and prove each of them.

SHARED:
inter[n] = {false}
Date[n], Date[i] = i
Stage[n] = {false}

Code for process i.

```
inter[i] = true
repeat:
        Stage[i] = false
        await (for all j != i: inter[j] == false or Date[i] < Date[j])
        Stage[i] = true
until (for all j != i: Stage[j] == false)
<CS>
Date[i] = 1 + max(Date[1], …, Date[n])
Stage[i] = false
inter[i] = false
```

Mutual exclusion:

Assume for the sake of the contradiction that the Aravind's algorithm doesn't provide mutual exclusion. Our goal is to show that two processes can't be in the CS at the same time.
Let A be a set of processes, and let a ∈ A be a process in the CS.
Since a is in the CS, by examining the code we have:
Write_a(inter[a] = true) → Write_a(stage[a] = false) →Read_a(for all j != a: inter[j] == false or Date[a] < Date[j]) → Write_a(stage[a] = true) → Read_a(for all j != a: stage[j] == false)
Now, let b ∈ A be a process interested by enter in the CS.
Since b is interested by enter in the CS, by examining the code we have:
Write_b(inter[b] = true) → Write_b(stage[b] = false)
Then, the process be enter in the await command. The process will be able to continue only if: Read_b(for all j != b: inter[j] == false or Date[b] < Date[j]). Let assume that Date[b] < Date[j] which not bring any contradiction. Then, we must have Read_b(for all j != b: stage[j] == false), but recall stage[a] = true, and obviously a ≠ b, then, the process be must repeat the loop, and this, until the process a enter in the exit code and state Stage[a] as false. Trivially, this scenario is true for all process, and all set of processes interesting by enter in the CS, such as a is already in the CS.
Then, there is no possibilities for b to enter in the CS, contradicting our assumption. Then, the claim is proved.

■

Starvation freedom:

Assume for the sake of the contradiction that the Aravind's algorithm doesn't provide starvation freedom. Let A be a set of processes, and let a ∈ A be a process in the CS. By assumption, there exist a scenario in which a is interested by enter in the CS but he is stuck forever in the entry code.'
Let's show that there is no such a scenario.

Since a is interested to enter in the CS, by examining the code we have:
Write_a(inter[a] = true) $\rightarrow$ Write_a(stage[a] = false) $\rightarrow$ Read_a(for all j != a: inter[j] == false or Date[a] < Date[j])
Let's see if a can be block forever in the previous await command.
Let assume that from all the processes, a have the bigger date, such that at the moment where a is interested, he gets blocked by all the other date. Since in the exit code we have Write_i(Date[i] = 1 + max(Date[1], …, Date[n])) recall Date[a] maximum, Date[i] = Date[a] + 1, then, after a finite amount of time, we will have Date[a] with a minimum value, and then, the process a will be able to pass the await statement at anytime, until to arrive in the exit code (then, he will obviously enter in the CS before).
Continuing the examination of the code we have:
$\rightarrow$ Write_a(stage[a] = true) $\rightarrow$ Read_a(for all j != a: stage[j] == false)
Let's see if a can be block forever in the previous Read command.
As we already shown, we have Date with the shorter value, then, all the interested process may act like:
Write_i(stage[i] = false) $\rightarrow$ Read_a(for all j != i: inter[j] == false or Date[i] < Date[j])
And all the processes may be stuck in the await command since Date[a] is the shorted than all the Date, then we have all the Stage[i] = false. Then, the process a can enter in the CS at the moment he will get running time. Contradicting the assumption the a is stuck forever in the entry code, and proving the claim.

∎