Part 1:

F

1) 13 – one thread for each class

12- 5 threads for each class

2)?

Part 2:

First solution:

· Avoiding contagion: yes / no

Explain:

Let’s assume two customers made up(customers) so they are waiting for the chef and also made down(chef) because chef is initialized to 10 this means they both got in line for the meal which is 0. Now let’s assume the chef process is running and will make both prepareMeal() so both meals are in line, and this will cause a possible situation in which both customers make down(meal) because meal is 2 and both can be together running getMeal().

· Progress: yes / no

Explain:

By the fact customer is waiting to meal (down(meal)) it means the chef was available since chef>0 and this customer is waiting in down(meal), the chef is is making down(customer) since at least the above customer exists, he will prepareMeal() and put up(meal). And eventually The OS will give the above customer’s process some CPU time and he/ or any other customers which are in down(meal) will get the meal that the chef made.

· Efficiency: yes / no

Explain:

The while loop is just for the making sure the preparation process keep going. We see that the chef will ‘wait’ for a customer in the down(customer) line and by the fact we are using Semaphores we know it will be efficient.

· No starvation: yes / no

Explain:

We see that the chef makes meals as much as needed meaning for every customer that makes up customer, the chef makes down(customer and makes a meal). So it suggests the customers will always have a meal to take. By the fact we are using Semaphores we now that the customer will eventually get his meal and this is true because the meal>0 if a customer is waiting for it.

· Adhering to the purple badge: yes / no

Explain:

By the fact the Semaphore for customers doesn’t have a sealing it will not stop at 10.   
Let’s assume we have 11 customers that one after the other will only execute the up(customer) line, this will lead the customer>10 and will now be aligned with the “purpule badge”.

Second solution:

For each of the following properties, mark if it holds or not and explain.

· Avoiding contagion: yes / no

Explain:

Same as before.   
Let’s assume two customers made to the up(customers) line, this is possible because the cnt\_customers<10 so they are waiting for the chef and also made down(chef) because chef is initialized to 10 this means they both got in line for the meal which is 0. Now let’s assume the chef process is running and will make both prepareMeal() so both meals are in line, and this will cause a possible situation in which both customers make down(meal) because meal is 2 and both can be together running getMeal().

· Progress: yes / no

Explain:

Same as before.  
By the fact customer is waiting to meal (down(meal)) it means the chef was available since chef>0 and this customer is waiting in down(meal), the chef is is making down(customer) since at least the above customer exists, he will prepareMeal() and put up(meal). We can see that the change in the logic of adding a cnt\_customers will not affect the progress since the chef will make the numbers of meals as suppose to the customer counter, which is aligned with amount of customers waiting for the meal. And eventually The OS will give the above customer’s process some CPU time and he/ or any other customer which are in down(meal) will get the meal that the chef made.

· Efficiency: yes / no

Explain:

Nothing changed in the chef logic.

· No starvation: yes / no

Explain:

As said above, we can see that the change in the logic of adding a cnt\_customers will not affect the fact that the chef will make the amount of meals needed because every customer that is waiting for a meal did up(customer) and this means the chef will make a meal according to the number of customers that made up(customer), this means that for every customer in line down(meal) will eventually have a meal. And we know that the Semaphores are fair so it will be starvation free.

· Adhering to the purple badge: yes / no

Explain:

By the fact the cnt\_customrs is not protected with Semaphore or any other tool, it will not make sense and it could lead to any number. For examples 11 process(customer), one after the other will execute the if steatment, and another process will get CPU time, this will lead to 11 process to pass the is section with out even increasing the cnt\_customers

Part 3:

a.Busy waiting is happening every time there’s a sleeping process, waiting to be awakened.

No. Busy waiting is [synchronization](https://www.baeldung.com/cs/semaphore" \l "process-synchronization) technique in which a process/task waits and constantly checks for a condition to be satisfied before proceeding with its execution.  
unlike waiting to be awakened, busy waiting can be inefficient because the looping procedure is a waste of computer resources

b.Semaphores can provide the same functionality as mutex locks. True / False

c.Assuming we need to run 2 concurrent tasks, the only benefit concurrent threads have on concurrent processes is a much more efficient communication between them.

d.Single core processor computers do not need semaphores, monitors and other locks’ as they do not run anything in parallel.