Midterm Project Report

Tugay Apaydın - 1801042081

I started implementing my project with implementing supplier and cooker processes. Since cooks and supplier are sharing the same memory (kitchen), there was some required synchronization processes.

1. When the kitchen is empty, cooks must wait for a plate; In reverse, if the kitchen is full, supplier must wait for an empty room. I easily solved this synchronization by adding 2 semaphores, empty and full.

Cooker:

```
wait(full)

post(empty)
```

Supplier;

```
wait(empty)

post(full)
```

2. In my algorithm, cooks are making some calculation in order to know if there is certain

required plate that should be put on the counter. For example, if the counter size is 4 and there are 2 soups in the counter, then cooks must deliver desert or course and if there is

no desert or course in the kitchen, then they should wait supplier to bring a course or desert.

So the process basically block supplier (because while making calculations supplier may change

values), makes calculations and then waits for supplier to bring a specific plate,if needed, or unblocks supplier. Of course while blocking the supplier, cooks cannot wait supplier to bring a plate so a cook can unblock supplier for a short time to let him deliver the food.

Cooker:

```
wait(supp)

if a place is required and not in the kitchen

post(supp)

wait(plate)
```

```
post(supp)
Deliver a plate to the counter
```

```
Supplier:
wait(supp)
if a place is required from a cook
post(plate) //If only needed
post(supp)
```

I firstly thought that since cookers and supplier are waiting for empty room or plate, and waiting each other before making calculations there should not be any other synchronization. But that was a terrible mistake. I faced a problem and it took long time to find that what caused that error.

In cooker process, I was unblocking the supplier before taking a plate from kitchen. In other words, I was unblocking the supplier right after calculation of a required specific plate.

But in the rest of the process, cooker is taking something from the kitchen and supplier is delivering something to the kitchen. What happens if they do this work at the same time? In that case my counter variables are set to wrong values. So I blocked supplier process until the end of the cooker process.

```
Cooker:
wait(supp)
if a place is required and not in the kitchen
post(supp)
wait(plate)

Deliver a plate to the counter
post(supp)
```

Supplier:

```
wait(supp)
if a place is required from a cook
post(plate) //If only needed
post(supp)
```

After implementing cook process, I implemented student process. Student should take the 3 plates at one time so I needed a helper process which is called perfectGuy. perfectGuy simply delivers 3 type of food at the same time to a student.

3. perfectGuy simply waits for each type of food and gives a tray to the student (it is a pseudo process, not exactly giving a tray). After taking the food, student finds a table and sit.

Student:

```
post(perfectGuy->wait)
wait(tray)
wait(table)
post(table)
```

perfectGuy:

```
wait(perfectGuy->wait)
wait(P)
wait(C)
wait(D)
post(counter->empty)
post(tray)
```

4. In order to avoid calculation errors, cook process must be blocked while perfectGuy is delivering the food to the student. Because, counter variables are changed at this time.

```
Student:
post(perfectGuy->wait)
wait(tray)
post(counter->let) //Unblocks cooks
wait(table)
post(table)
```

```
perfectGuy:
wait(perfectGuy->wait)
wait(P)
wait(C)
wait(D)
wait(counter->let) //Blocks cooks
post(counter->empty)
post(tray)
```

These were the synchronizing problems that I faced while implementing the project.