Homework - 4 CSE344
System Programming
Hamza YOĞURTCUOĞLU - 171044086

→ **main.c :** I put the **Usage** at first. I seperated command line argument with **getopt().**

Then file format control is done. Wholesaler that every once in a while delivers two **distinct** ingredients.

The Right Format	**The Wrong Format**
MS	**SS** (Ingredients must be distinct)
FM	**ABCA** (wrong characters and number)
WS	**!^!'** (WRONG FILE CONTENT)
SW	**…**
…	**…**
…	**…**

File must have valid content and at **least 10 rows.**

Note : there is newline('\n') after two characters. One correctly formatted file is left in the zip directory.

SHARED MAPPING

In this section, it will be explained why the shared semaphores, array are used.

SIZE 5 SEMAPHORE (5 MUTEX) ARRAY

SIZE 2 INT SHARED VARIABLE ARRAY

All semaphores are used in common. All of them are used like mutex for synchronization.

SEMS 0 → All chefs wait same semaphore. when wholesaler put two distinct ingredients data structure(array). '**sems[0]**' is increased once and a chef is awakened.

SEMS 1 → After putting the wholesaler ingredients into the data structure, it waits until the Güllaç is ready. The chef increases this semaphore after the Güllaç is ready. In this way, wholesaler can continue reading new ingredients.

SEMS 2 → Semaphore used to write 'delivered, wholesaler left' prints sequentially as follows. The chef should always deliver. Then the wholesaler should take the dessert and leave. Then the chef should wait

- → chef3 has delivered the dessert to the wholesaler
- → the wholesaler has obtained the dessert and left to sell it
- → chef3 is waiting for sugar and flour

SEMS 3 → I don't use printf or fprintf in I/0. I did synchronization of writing console with this mutex. You can check print functions :

```
void printIngredient(enum Ingredients item);
void printWaitingChef(struct chefInfo * infoC);
void printDeliverIng(char firstChar,char secondChar);
void printTakenIngs(char firstChar,char secondChar,struct chefInfo * infoC);
void printDeliveredDessert(struct chefInfo * infoC);
void preparingDessertPrint(struct chefInfo * infoC);
void printEndlessSupply(struct chefInfo * infoC);
```

SEMS 4 → It is for sequentially printing order like SEM[2].

Note : All semaphores are common for all threads. There is no threat specific semaphore.

deliveredData 0 → first ingredient to be delivered

deliveredData 1 → second ingredient to be delivered

Note : The ingredients will be delivered to a data structure(array)

→ **DOES WHOLESALER IS UNAWARE OF HOW MANY THREADS/CHEFS ARE WAITING?**

Wholesaler knows **no** information about chefs. Wholesaler does just read the file. Assigning the ingredients to the data structure. Posting the **SEM[0]** (common semaphore that all chefs are waiting for) semaphore. Waiting for a chef to take ingredients.

→ **HAVE ALL SEMAPHORES BEEN SHARED/SAME BETWEEN CHEFS FOR SYNCHRONIZATION?**

Please check **SHARED MAPPING,** the title in page 2. This is what all semaphores are used for. In short, all semaphores are the same among chefs and are used for syncronization.

→ **HOW DOES CHEF WAIT FOR THE PREPARATION OF THE DESSERT?**

Please check the **preparingDessertPrint** function. After the required information message is given. The chef is sleeping for a random number of seconds (1 to 5 inclusive).

→ IF THE WHOLESALER KNOWS NOTHING ABOUT CHEFS, HOW CAN INGREDIENTS ARE TAKEN CORRECTLY BY CORRECT CHEF?

Wholesaler posts **SEM[0]** after assigning the next ingredients to the data structure. In this way, one chef wakes up. The awakened chef checks his own missing ingredients. If not, the awakened chef postes **SEM[0]** and arouses another chef. Finally, if it is its own lack ingredients, the chef takes the necessary ingredients and prepares them.

→ HOW TO WHOLESALER IS DONE, CHEF WILL UNDERSTAND THAT THERE WON'T BE ANYT MORE INGREDIENTS

To be able to deliver wholesaler ingredients . Wholesaler uses the 'enum Ingredients' array. If the wholesaler cannot find any other ingredients in inputFile, it assigns **NOTHING ('N')** to the data structure and wakes up all chefs. The chefs get out of the loop when they see **NOTHING**.

→ WHEN CTRL + C PRESSING IS DONE, ARE ALL THE RESOURCES BEIGN RELEASED ?

Pointers of all the resources received were defined globally. In this way, if a memory allocate is made from a method, these resources can be released in SIGINT(**CTRL+C**) handler. Below you can see the result of **valgrind**:

```
==15164== HEAP SUMMARY:
==15164==     in use at exit: 0 bytes in 0 blocks
==15164==   total heap usage: 12 allocs, 12 frees, 3,446 bytes allocated
==15164==
==15164== All heap blocks were freed -- no leaks are possible
==15164==
==15164== For counts of detected and suppressed errors, rerun with: -v
==15164== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 1 : Freed all resources when ctrl+c pressing and normal execution.

There is no error or warning from any processes.(This image is just a process)

In SIGINT(**CTRL+C**) handler function, terminate is converted to 0, so that the program ends **after the last chef delivering process ends.**

Test Program Examples: The following commands write how you can run process. I tested ingredients up to 1000 lines .

You can execute with Absolute path or relative path for files.

./program ./program -i filePath

./program -i (AbsolutePath)/filePath

inputFile Examples: The file format should be as described on page 1. The file has been checked in the **fileCheck** function.

File must have valid content and at **least 10 rows.**

Note : there is newline(‘\n’) after two characters. One correctly formatted file is left in the zip directory.

…

…

→ Output Example :

```
chef1 has an endless supply of flour and milk but lacks walnuts and sugar
chef1 is waiting for walnuts and sugar
chef2 has an endless supply of walnuts and milk but lacks sugar and flour
chef2 is waiting for sugar and flour
chef3 has an endless supply of sugar and milk but lacks flour and walnuts
chef3 is waiting for flour and walnuts
chef6 has an endless supply of flour and sugar but lacks walnuts and milk
chef6 is waiting for walnuts and milk
chef5 has an endless supply of walnuts and flour but lacks sugar and milk
chef4 has an endless supply of walnuts and sugar but lacks milk and flour
chef4 is waiting for milk and flour
chef5 is waiting for sugar and milk
the wholesaler delivers milk and flour
```

…

…

…

```
the wholesaler delivers flour and sugar
chef2 has taken the flour
chef2 has taken the sugar
chef2 is preparing the dessert
chef2 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
chef2 is waiting for sugar and flour
```

End of ouput. Then All chefs and wholesaler is terminating.

→ Compiler and Run :

Makefiles Commands:

make #that command compiler
make clean #that command cleans all object files

To run :
./program -i filePath

If you want to see outputs. Check above **outputs.**

→ Note : 1) There is newline('\n') after two characters.
2) In SIGINT(CTRL+C) handler function, terminate is converted to 0, so that the program ends after the last chef delivering process ends.

Thanks :)