April 20th, 2022

CSE344 – System Programming - Homework #3 - v4
Semaphores

Objective
You will prepare a small bakers simulation inspired by a classic synchronization problem. Let's assume for the sake of simplicity that one needs exactly 4 ingredients for preparing güllaç:
- milk (M)
- flour (F)
- walnuts (W)
- and sugar (S).

6 x Chef
There are 6 chefs in the main street of Gebze, and each has an endless supply of two distinct ingredients and lacks the remaining two:

chef0 has an endless supply of milk and flour but lacks walnuts and sugar,
chef1 has an endless supply of milk and sugar but lacks flour and walnuts,
chef2 has an endless supply of milk and walnuts but lacks sugar and flour,
chef3 has an endless supply of sugar and walnuts but lacks milk and flour,
chef4 has an endless supply of sugar and flour but lacks milk and walnuts,
chef5 has an endless supply of flour and walnuts but lacks sugar and milk.

Every chef sits in front of her/his store and waits for the remaining two ingredients to arrive in order to go and prepare güllaç. Once the güllaç is ready they deliver it to the wholesaler and continue to wait for more ingredients. In case of termination, each chef process will return (and by "return" I actually mean return, not print on screen) the total number of desserts it has prepared so far.

1 x Wholesaler
There is also a wholesaler that every once in a while delivers two distinct ingredients out of the four to the street of the chefs, lets the chefs know that the ingredients have arrived and then waits for the dessert to be ready.

However, do not assume that **the wholesaler knows which chef needs which ingredients**. So do not make the mistake of signaling **directly from the wholesaler only the chef corresponding** to the delivered ingredients. Example: if the wholesaler delivers F and M, don't just signal/sem_post the chef waiting for F and M.

Once the güllaç is ready, the wholesaler takes it for sale and the chefs must wait again for the next visit of the wholesaler.

The wholesaler will read the ingredients to deliver from an input file containing two capital letters at each line, representing the ingredients to deliver; e.g.
MS
FM
WS
SM
etc

Assume the file has valid content and at least 1 row. The ingredients will be stored in a character array taking place in a shared memory segment; if the wholesaler for instance delivers SM, the array will contain 'S' and 'M'. After the wholesaler is done (i.e. he has no more ingredients to deliver, he must notify the chefs that the procedure is over – how? That's up tp you.) It will collect the return values of the chef processes, and print the total number of desserts delivered to him.

How

You will implement the program as the chefs and the wholesaler in the form of 6+1 processes that print on screen their activities. You will implement two version of the program, one solving it with named semaphores and another solving it with unnamed semaphores.

```
./hw3named -i inputFilePath -n name
./hw3unnamed -i inputFilePath
```

-i: denotes the input file path (absolute or relative)
hw3named: must use named semaphores for synchronization (you can include additional args if necessary).
hw3unnamed: must use unnamed semaphores for synchronization.

Output

Chefs (print the contents of the character array containing the ingredients at the end of every output line):

Initially (replace the bold parts; e.g. i becomes the chef id, ingredient becomes Milk, etc):

```
chefi (pid XYZ) is waiting for ingredient1 and ingredient2
```

After obtaining the ingredients (replace the bold parts):

```
chefi (pid XYZ) has taken the ingredient1
chefi (pid XYZ) has taken the ingredient2
chefi (pid XYZ) is preparing the dessert
chefi (pid XYZ) has delivered the dessert
```

At exit (replace the bold parts):

```
chefi (pid XYZ) is exiting
```

The wholesaler (replace the bold parts):

```
the wholesaler (pid XYZ) delivers ingredient1 and ingredient2
the wholesaler (pid XYZ) is waiting for the dessert
the wholesaler (pid XYZ) has obtained the dessert and left
the wholesaler (pid XYZ) is done (total desserts: 34)
```

Evaluation rules

1) Compilation error: grade set to 1; if the error is resolved during the demo, then evaluation continues.
2) Compilation warning (with respect to the **-Wall** flag); -1 for every warning until 10. -20 points if there are more than 10 warnings; no chance of correction at demo.
3) No makefile: -30
4) No pdf report submitted (or submitted but insufficient, e.g. 3 lines of text with no design explanation, etc): -20. Other file formats are not admissible.
5) If the required command line arguments are missing/invalid, your program must print usage information and exit. Otherwise: -10

6) The program crashes or doesn’t produce the expected output with valid input: -50 for each of the two versions.
7) The program doesn’t satisfy a requirement: -100
8) Presence of memory leak (regardless of amount – checked with valgrind) -30
9) Late submissions will not be accepted
10) In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message. Otherwise: -10
11) Cleanup after the children processes, no zombie processes, -50 otherwise.
12) Although I’ve said it repeatedly in class, apparently I also have to write it: DON’T USE BUSY WAITING. EVER. FOR NO MATTER WHAT REASON. -100 otherwise.

Is my homework submission valid?
It is valid if given a correct input file, it creates all necessary processes.

Submission rules:

- Your source files, your makefile and a report; place them all in a directory with your student number as its name, and zip the directory.
- Your report must contain: how you solved this problem, your design decisions, which requirements you achieved and which you have failed.
- The report must be in English.
- Your makefile must only compile the program, not run it!
- Do not submit any binary executable files. The TAs will compile them on their own boxes.
- Your code will be compared against online sources; you are free to be inspired, but you are also expected to write your own code.
- Homeworks are individual tasks. Proven cases of plagiarism will be punished to the full extent.

Hints:

- Plan/design carefully.
- Check for memory leaks with valgrind before submitting.
- Compile and run your program on more than one POSIX systems to ensure portability, if you can.
- Control whether you satisfy each and every one of the requirements prior to submission.
- Test your programs with arbitrarily sized input files.

Good luck.