

Homework 4

Elif Akgün

1801042251

1. Problem Definition

In this homework, there are 7 actors; the wholesaler and 6 chefs. It assumed that one needs exactly 4 ingredients for preparing g \ddot{u} llaç: milk (M), flour (F), walnuts (W), and sugar (S).

There are 6 chefs in a street, and each has an endless supply of two distinct ingredients and lacks the remaining two. For example, chef1 has an endless supply of milk and flour but lacks walnuts and sugar; chef2 has an endless supply of milk and sugar but lacks flour and walnuts, etc.

Every chef sits in front of her/his store and waits for the remaining two ingredients to arrive in order to go and prepare g \ddot{u} llaç. The wholesaler 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. Once the g \ddot{u} llaç is ready, the wholesaler must take it for sale and the chefs must wait again for the next visit of the wholesaler.

The wholesaler reads the ingredients to deliver from a file denoted by *filePath* containing two letters at each line, representing the ingredients to deliver. The ingredients are delivered to a data structure (e.g. array, stack, etc), located at the heap and thus shared among all involved threads. Once the wholesaler is done, notifies the chefs that there won't be any more ingredients.

2. Implementation

I implement every actor as a separate threads. I benefited from cigarette smokers problem. There are 12 threads in my program: 6 chef threads, 4 pusher threads, a agent thread, and also main thread. The wholesaler is main thread and it waits for other threads with *pthread_join*.

2.1 Wholesaler

In main thread, firstly, semaphores are initialized then input file denoted by *filePath* is opened. It checks if the file is less than 10 lines, if it is, it prints error messages and exits. Then creates all threads. In while loop, it reads the input file to end of the file. Before reading file, wholesaler waits *gullac* semaphore. So one of the chefs before preparing g \ddot{u} llaç, wholesaler can not put new ingredients. If read system call returns 0 then wholesaler notifies that there is no more ingredient in the file to pusher threads and chef threads with *semChef* and *semPusher* semaphores. If there are no more elements in the file, wholesaler posts these semaphores. If there are some elements in the file, wholesaler puts ingredients to ingredients

array that read from file: M, F, W, or S. After read, it notifies agent that *ingredients* array is full now, so agent can post these ingredients now. If there are no more elements in the file, breaks the loop. Waits for thread with *pthread_join*. Last of all, destroys the semaphores and deallocates memory then exits.

2.2 Chefs

All chefs execute same function(*chef*). In this function each chef waits for relevant ingredients:

- chef1 waits for milk and flour
- chef2 waits for milk and walnuts
- chef3 waits for milk and sugar
- chef4 waits for flour and walnuts
- chef5 waits for flour and sugar
- chef6 waits for walnuts and sugar

Each chef waits for different combination ingredients so they wait for different semaphores based on milk, flour, walnuts and sugar. For this purpose, chef1 uses *MF*, chef2 uses *MW*, chef3 uses *MS*, chef4 uses *FW*, chef5 uses *FS*, and chef6 uses *WS* semaphores. After take ingredients that they wait, they prepare the güllaç. Chefs simulate dessert preparation by sleeping for a random number of seconds (1 to 5 inclusive). After preparing güllaç, chefs notify to parent desert is ready (with *gullac* semaphore) and wait for wholesaler to read file again. For this purpose chefs use *semChef* semaphore. If file does not contain any ingredients anymore, break the loop and exit.

2.3 Agent

Agent thread includes a while loop that goes on infinity. First of all, agent waits *semAgent* semaphore. When wholesaler reads the ingredients from file, posts the *semAgent* semaphore. Then agent can go on.

Agent checks ingredients array's elements. If array includes 'M' (milk), posts milk semaphore, else if array includes 'F' (flour), posts flour semaphore, else if array includes 'W' (walnuts), posts walnuts semaphore, else array includes 'S' (sugar), posts sugar semaphore.

2.4 Pusher

There are 4 ingredients to prepare güllaç, so there are 4 pushers. Pusher1 waits *milk* semaphore, Pusher2 waits *flour* semaphore, Pusher3 waits *walnuts* semaphore, and Pusher4 waits *sugar* semaphore.

Pusher1 waits for milk and after agent posts *milk* semaphore, takes the *mutex* to block other pushers. Then checks other ingredients with *isFlour*, *isWalnut* and *isSugar*. If there is flour, also there is milk, it means that chef that waiting for milk and flour can prepare güllaç. So Pusher1 posts *MF* semaphore. If there are walnuts, also there is milk, it means that chef waiting for milk and walnuts can prepare güllaç. So Pusher1 posts *MW* semaphore. If there is sugar, also there is milk, it means that chef that waiting for milk and sugar can prepare güllaç. So Pusher1 posts *MS* semaphore. If there are no other ingredients, there is

just milk, Pusher1 makes *isMilk* 1 to notify there is milk in the array. Then posts to *mutex* to unblock other pushers.

Pusher2 waits for flour and after agent posts *flour* semaphore, takes the *mutex* to block other pushers. Then checks other ingredients with *isMilk*, *isWalnut* and *isSugar*. If there is milk, also there is flour, it means that chef that waiting for milk and flour can prepare g  lla  . So Pusher2 posts *MF* semaphore. If there are walnuts, also there is flour, it means that chef waiting for flour and walnuts can prepare g  lla  . So Pusher2 posts *FW* semaphore. If there is sugar, also there is flour, it means that chef waiting for flour and sugar can prepare g  lla  . So Pusher2 posts *FS* semaphore. Else, if there are no other ingredients, there is just flour, Pusher2 makes *isFlour* 1 to notify there is flour in the array. Then posts to *mutex* to unblock other pushers.

Pusher3 waits for walnuts and after agent posts *walnuts* semaphore, takes the *mutex* to block other pushers. Then checks other ingredients with *isMilk*, *isFlour* and *isSugar*. If there is milk, also there are walnuts, it means that chef waiting for milk and walnuts can prepare g  lla  . So Pusher3 posts *MW* semaphore. If there is flour, also there are walnuts, it means that chef waiting for flour and walnuts can prepare g  lla  . So Pusher3 posts *FW* semaphore. If there is sugar, also there are walnuts, it means that chef waiting for walnuts and sugar can prepare g  lla  . So Pusher3 posts *WS* semaphore. If there are no other ingredients, there is just walnuts, Pusher3 makes *isWalnuts* 1 to notify there are walnuts in the array. Then posts to *mutex* to unblock other pushers.

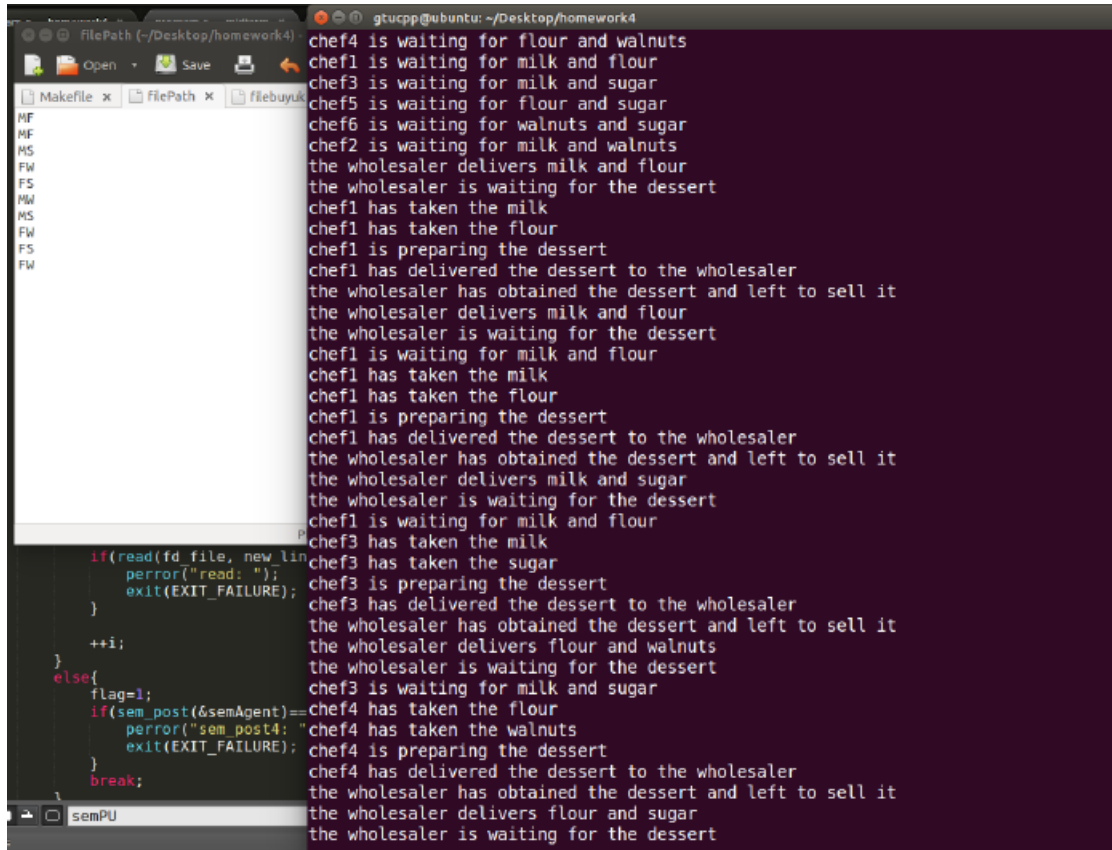
Pusher4 waits for sugar and after agent posts *sugar* semaphore, takes the *mutex* to block other pushers. Then checks other ingredients with *isMilk*, *isFlour* and *isWalnuts*. If there is milk, also there is sugar, it means that chef waiting for milk and sugar can prepare g  lla  . So Pusher4 posts *MS* semaphore. If there is flour, also there is sugar, it means that chef waiting for flour and sugar can prepare g  lla  . So Pusher4 posts *FS* semaphore. If there are walnuts, also there is sugar, it means that chef that waiting for walnuts and sugar can prepare g  lla  . So Pusher4 posts *WS* semaphore. If there are no other ingredients, there is just sugar, Pusher4 makes *isSugar* 1 to notify there is sugar in the array. Then post to *mutex* to unblock other pushers.

After pushing, pushers wait for wholesaler to read file again with *semPusher* semaphore, if file does not contain any ingredients anymore, break the loop and exit.

An example of the program running: Let's wholesaler read "MF" chars form file. Then ingredients[0]='M' , ingredients[1]='F' now. Agent checks the array then post milk and flour semaphores. Pusher1 waited for milk and Pusher2 waited for flour. Let's Pusher1 takes milk then takes mutex. Checks the isFlour, isWalnuts and isSugar. None of them. So isMilk=1 now. Pusher1 posts mutex. Then Pusher2 takes flour and mutex. Checks for isMilk, isWalnuts and isSugar. isMilk=1. So chef that waiting for milk and flour (chef1) can prepare g  lla  . Pusher1 posts MF semaphore then chef1 thread continues to run and prepares g  lla  . Then waits for milks and flour again.

3. Tests

3.1 Sample Input/Output 1



```

chef4 is waiting for flour and walnuts
chef1 is waiting for milk and flour
chef3 is waiting for milk and sugar
chef5 is waiting for flour and sugar
chef6 is waiting for walnuts and sugar
chef2 is waiting for milk and walnuts
the wholesaler delivers milk and flour
the wholesaler is waiting for the dessert
chef1 has taken the milk
chef1 has taken the flour
chef1 is preparing the dessert
chef1 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers milk and flour
the wholesaler is waiting for the dessert
chef1 is waiting for milk and flour
chef1 has taken the milk
chef1 has taken the flour
chef1 is preparing the dessert
chef1 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers milk and sugar
the wholesaler is waiting for the dessert
chef1 is waiting for milk and flour
chef3 has taken the milk
chef3 has taken the sugar
chef3 is preparing the dessert
chef3 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers flour and walnuts
the wholesaler is waiting for the dessert
chef3 is waiting for milk and sugar
chef4 has taken the flour
chef4 has taken the walnuts
chef4 is preparing the dessert
chef4 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers flour and sugar
the wholesaler is waiting for the dessert

```

```

if(read(fd_file, new_line, 1024) < 0)
    perror("read: ");
    exit(EXIT_FAILURE);
}
++i;
}
else{
    flag=1;
    if(sem_post(&semAgent)==0)
        perror("sem_post4: ");
        exit(EXIT_FAILURE);
    }
    break;
}
semPU

```

3.2 Sample Input/Output 2

The screenshot shows a C++ program running in a terminal window. The program simulates a sequence of events between chefs and a wholesaler. The output is as follows:

```

chef2 is preparing the dessert
chef5 is waiting for flour and sugar
the wholesaler is waiting for 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 milk and walnuts
the wholesaler delivers milk and sugar
the wholesaler is waiting for the dessert
chef3 has taken the milk
chef3 has taken the sugar
chef3 is preparing the dessert
chef3 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers flour and walnuts
chef3 is waiting for milk and sugar
chef4 has taken the flour
chef4 has taken the walnuts
chef4 is preparing the dessert
the wholesaler is waiting for the dessert
chef4 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
chef4 is waiting for flour and walnuts
the wholesaler delivers flour and sugar
the wholesaler is waiting for the dessert
chef5 has taken the flour
chef5 has taken the sugar
chef5 is preparing the dessert
chef5 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
the wholesaler delivers flour and walnuts
chef4 has taken the flour
chef4 has taken the walnuts
chef4 is preparing the dessert
the wholesaler is waiting for the dessert
chef5 is waiting for flour and sugar
chef4 has delivered the dessert to the wholesaler
the wholesaler has obtained the dessert and left to sell it
chef4 is waiting for flour and walnuts

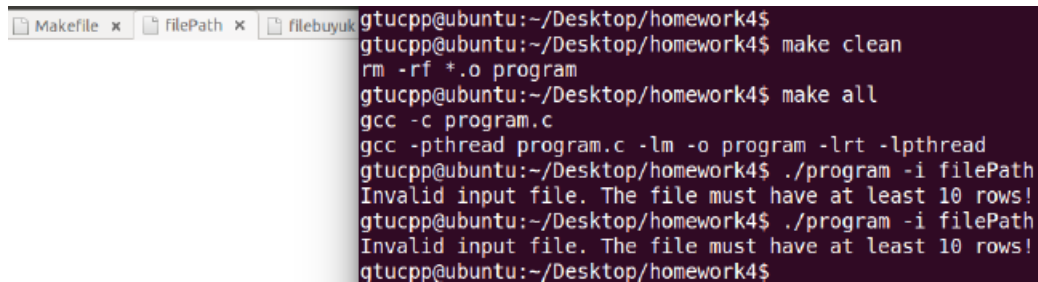
```

The program code is visible in the background, showing a loop that reads input from a file and processes it. The code includes error handling for file reading and semaphore operations.

3.3 Testing Command Line Arguments

```
gtucpp@ubuntu:~/Desktop/1801042251_CSE344_HW44/1801042251_CSE344_HW4$ ./program -i
Wrong command line arguments!
Usage: ./program -i filePath
-i represents the file that contains ingredients to deliver
gtucpp@ubuntu:~/Desktop/1801042251_CSE344_HW44/1801042251_CSE344_HW4$ ./program
Wrong command line arguments!
Usage: ./program -i filePath
-i represents the file that contains ingredients to deliver
gtucpp@ubuntu:~/Desktop/1801042251_CSE344_HW44/1801042251_CSE344_HW4$ ./program -i filePath k
Wrong command line arguments!
Usage: ./program -i filePath
-i represents the file that contains ingredients to deliver
```

3.4 Testing Input File



```
gtucpp@ubuntu:~/Desktop/homework4$
gtucpp@ubuntu:~/Desktop/homework4$ make clean
rm -rf *.o program
gtucpp@ubuntu:~/Desktop/homework4$ make all
gcc -c program.c
gcc -pthread program.c -lm -o program -lrt -lpthread
gtucpp@ubuntu:~/Desktop/homework4$ ./program -i filePath
Invalid input file. The file must have at least 10 rows!
gtucpp@ubuntu:~/Desktop/homework4$ ./program -i filePath
Invalid input file. The file must have at least 10 rows!
gtucpp@ubuntu:~/Desktop/homework4$
```