GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

SYSTEM PROGRAMMING
CSE 344 - 2020 SPRING

HW4 REPORT

SILA BENGİSU YÜKSEKLİ
1801042877

➢ *Explanation of the way of work and the aim of the program*

User can compile by typing make on terminal. User should give command line arguments like “-i filePath” , then it can be run by writing “./program -i filePath” on terminal.

This argument of filepath is needed for understanding what type of ingredients will be delivered.

The contents of the file must consist of the letters M(milk), F(flour), S(sugar) and W(walnut). The user can change the input file as desired, but it should contain at least ten rows and every line must have two letters to represent a missing group.

The program provides a simulation of making a dessert by six chefs with a wholesaler. In this scenario, there is a wholesaler that delivers the ingredients to street and there are chefs who take the ingredients from the wholesaler to make dessert. There are four types of ingredients which are milk, flour, walnut and sugar. Each cook has four ingredients, but the remaining two are missing. Therefore, each of them waits for two ingredients that the wholesaler will distribute. After the wholesaler distributes the ingredients, it waits until it gets a dessert and program continues until wholesaler delivers all the ingredients.

Each actor was represented as a thread. Hence, there is one process which is program with seven threads.

➢ *Process (program.c)*

This problem is similar to the classic “cigarette smokers” example. Seven threads were used to solve this problem. The main thread refers to the wholesaler and remaining six threads refer to the chefs.

Each chef works on the same function with different parameters. When threads are created, each one them takes a parameter which is void pointer. This pointer has a size of one and is held in heap. The element of the array corresponds to the number of the chef(1,2,3…). By this way, every chef knows which the ingredients it waits for by entering into the correct if block using chef number.

Also, there is an array data structure which is called ingredients. This array has a size of two and is held in heap so that it can be shared among all threads. Ingredients that are read from the file line by line, are pushed into this array. Thus, all threads can see the incoming ingredients when they work.

❖ *WholeSaler*

It continues to work until it delivers all the ingredients. Each line of the file indicates two missing ingredients. In turn, two ingredients in each row are pushed into shared array and accordingly, it goes into an if block and distributes the necessary ingredients to the street as seen in the code snippet below :

```
if ((ing1 == 'M' && ing2 == 'F')  ||  (ing1 == 'F' && ing2 == 'M'))
{
        wholesaler_Delivering("milk","flour");
        if (semop(semid,sops1,2) == -1)
        {
                ...
                exit(1);
        }
}
```

Distribution is carried out by the line starting with the semop. For example, first line is MF. Then wholesaler increments the value of milk and flour semaphore. By this way, chefs can understand that milk and flour are available and can continue to work. The wholesaler waits for the dessert to be ready after distributing.

After the operation is finished, the main thread (wholesaler) informs all other threads(chefs) that there will be no more ingredients. For this purpose pthread_cancel function is used. This function sends a cancellation request to the thread that we want to terminate. Its cancel type is set as asynchronous that means cancellation can be at any time. By this way, all threads are canceled when all the ingredients is distributed. When a thread is terminated, it releases its resources, but the operating system keeps an entry for that thread, thinking that it can be joinable at any time. So, main thread(wholesaler) waits for all threads(chefs) to avoid threads being a zombie by using pthread_join function. The exit status of threads which are canceled, is PTHREAD_CANCELED and the main thread collects them.

❖ *Chefs*

In my algorithm, the missing ingredients of each chef are as follows:
Chef1 : milk and flour
Chef2 : milk and walnut
Chef3 : milk and sugar
Chef4 : flour and walnut
Chef5 : flour and sugar
Chef6 : walnut and sugar

All of them are waiting for the missing ingredients to make dessert as seen in the below :

```
if (cook == 1) {
        if (semop(semid,sop1,2) == -1)
        {
                ...
                exit(1);
        }

        cook_hasTaken(1,"milk");
        ingredients[0] = 'x';
        cook_hasTaken(1,"flour");
        ingredients[1] = 'x';

        random = rand() % 5;
        cook_isPreparing(1);
        sleep(random);
        cook_hasDelivered(1);
  }
```

Waiting is carried out by the line starting with the semop. For example,let the first line of the file be MF. Then chef decrease the value of milk and flour semaphore. By this way, chefs can wait for the wholesaler to supply. Once the ingredients is delivered, chef takes them from shared array. The elements of the array are changed to x to indicate that the ingredients have been received. Preparing dessert is simulated by sleep system call. It sleeps for a random seconds between zero and six. After it prepares the dessert, it continues to wait for the ingredients again.

➢ *Synchronization between wholesaler and chefs*

SysV semaphores are used to provide synchronization between threads. Intercommunication is necessary, because there are many common sources such as milk, flour etc. The threads need each other to keep working. For example, the chef has to wait for the ingredients from the wholesaler. They cannot make dessert without ingredients and the wholesaler has to wait for the dessert from the chef. So they have to communicate with each other to keep each other informed.

Semaphores are used for this kind of waiting operations. One semaphore set is created with the size of five. The contents of this set are listed below:

- **güllaç** : It corresponds to dessert. It is the first element of the set. The wholesaler decreases the value of this semaphore to block itself until the

dessert is ready and the chefs increase the value of this semaphore to inform the wholesaler that the dessert is ready.

- **milk** : It is the second element of the set. If the current ingredient is milk, then wholesaler increases the value of this semaphore to inform chefs. The correct chef works and uses this source. Then it decreases its value again.
- **flour :** It is the third element of the set. If the current ingredient is flour, then wholesaler increases the value of this semaphore to inform chefs. The correct chef works and uses this source. Then it decreases its value again.
- **walnut** : It is the fourth element of the set. If the current ingredient is walnut, then wholesaler increases the value of this semaphore to inform chefs. The correct chef works and uses this source. Then it decreases its value again.
- **sugar** : It is the fifth element of the set. If the current ingredient is sugar, then wholesaler increases the value of this semaphore to inform chefs. The correct chef works and uses this source. Then it decreases its value again.

➢ *Why SysV semaphores ?*

In this problem, every chef has to take two missing ingredients at the same time. If they don't take them at the same time, they can cause deadlock. For example more than one chefs may need milk. But only one chef needs those two ingredients at the same time. Let's say the wholesaler supply both milk and flour. If one of them takes flour and another chef takes milk, every thread will remain pending. Because, every chef waits for the remaining ingredient, so they can't make dessert and wholesaler can't supply more ingredients because it waits for dessert to be ready. So there will be a deadlock situation.

To avoid this, every chef should take elements at the same time so that no one can interrupt. With sysV semaphores, this operation is easy. By using semop function, duties of the semaphores can be set and operation can be carried out at the same time. In POSIX, this operation is more complicated and we need extra more threads for this. That's way I used sysV semaphores.