System Programming
HW4 Report
Güllaç Chefs Simulation With Threads

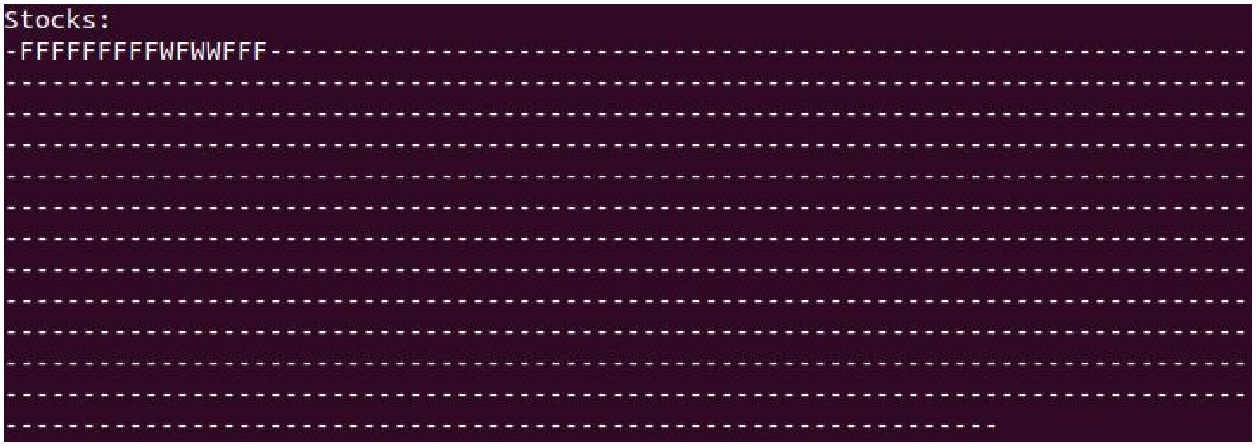
Muhammed Okumuş
151044017

Shared Resource Design

This program simulates a producer-consumer algorithm. There is 1 shared resource called **stock** that holds the ingredients the wholesaler brought. The chefs also take from this stock, if they can find the ingredients they are missing. This data structure is kept as a **character array**. The upper case letters (**M**)ilk, (**F**)lour, (**W**)alnuts and (**S**)ugar represents the ingredients in the stock. '**-**' character represents an empty spot in the stock. Stock size is decided at compile time. It would be better to decide it dynamically by checking the input file size.

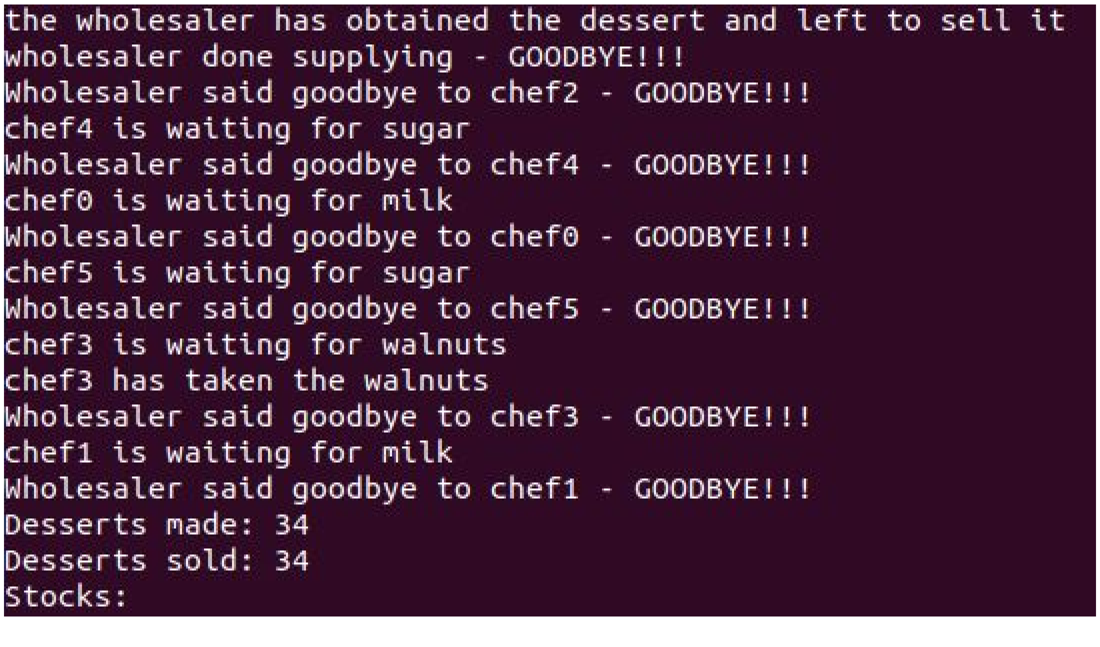
POSIX unnamed semaphores used for controlling the stock access. The semaphore used is declared globally so all threads can access it.

Visualization of the stock at the end of the program:



```
Stocks:
-FFFFFFFFFWFWWFFF---------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------------------------------
```

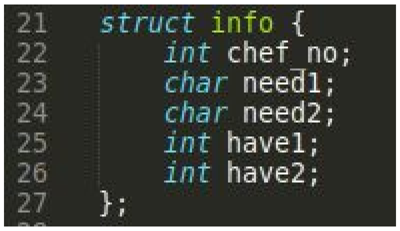
The items left in the stocks is caused by all chefs has taken 1 ingredient they need and waiting for the other. Even tough chefs are waiting, the wholesaler is done so they will exit too:



```
the wholesaler has obtained the dessert and left to sell it
wholesaler done supplying - GOODBYE!!!
Wholesaler said goodbye to chef2 - GOODBYE!!!
chef4 is waiting for sugar
Wholesaler said goodbye to chef4 - GOODBYE!!!
chef0 is waiting for milk
Wholesaler said goodbye to chef0 - GOODBYE!!!
chef5 is waiting for sugar
Wholesaler said goodbye to chef5 - GOODBYE!!!
chef3 is waiting for walnuts
chef3 has taken the walnuts
Wholesaler said goodbye to chef3 - GOODBYE!!!
chef1 is waiting for milk
Wholesaler said goodbye to chef1 - GOODBYE!!!
Desserts made: 34
Desserts sold: 34
Stocks:
```

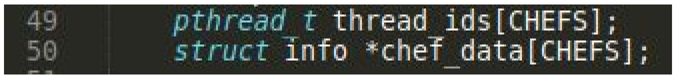
Creating and Passing Data to Threads

The data to be passed to threads is designed as C struct. It holds chef number, first ingredient needed, second ingredient needed, first ingredient is acquired and second ingredient is acquired fields:



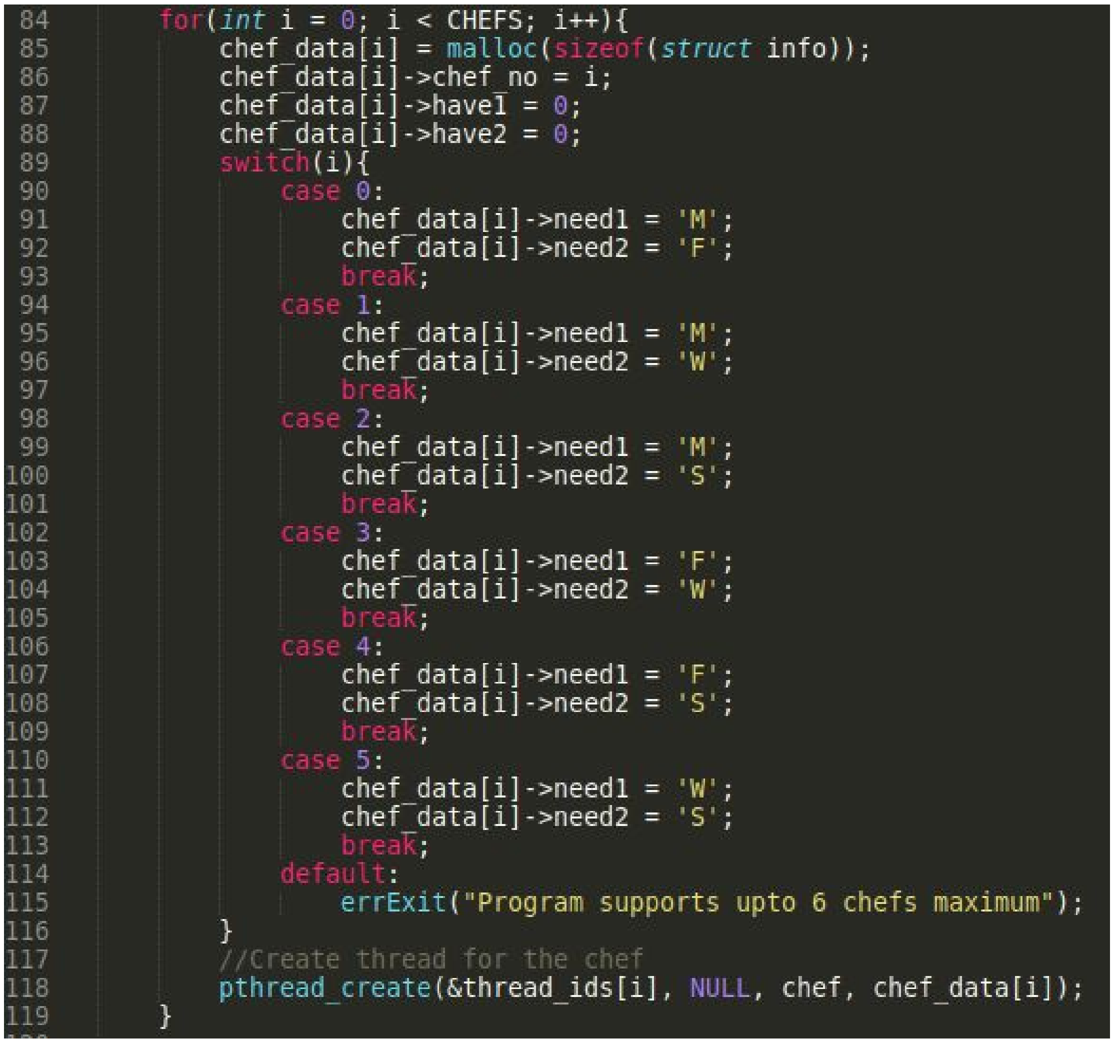
```c
struct info {
    int chef_no;
    char need1;
    char need2;
    int have1;
    int have2;
};
```

Thread ID's and thread datas are hold in their type arrays:



```c
    pthread_t thread_ids[CHEFS];
    struct info *chef_data[CHEFS];
```

The data is initialized and and threads are created in a for loop:



```c
    for(int i = 0; i < CHEFS; i++){
        chef_data[i] = malloc(sizeof(struct info));
        chef_data[i]->chef_no = i;
        chef_data[i]->have1 = 0;
        chef_data[i]->have2 = 0;
        switch(i){
            case 0:
                chef_data[i]->need1 = 'M';
                chef_data[i]->need2 = 'F';
                break;
            case 1:
                chef_data[i]->need1 = 'M';
                chef_data[i]->need2 = 'W';
                break;
            case 2:
                chef_data[i]->need1 = 'M';
                chef_data[i]->need2 = 'S';
                break;
            case 3:
                chef_data[i]->need1 = 'F';
                chef_data[i]->need2 = 'W';
                break;
            case 4:
                chef_data[i]->need1 = 'F';
                chef_data[i]->need2 = 'S';
                break;
            case 5:
                chef_data[i]->need1 = 'W';
                chef_data[i]->need2 = 'S';
                break;
            default:
                errExit("Program supports upto 6 chefs maximum");
        }
        //Create thread for the chef
        pthread_create(&thread_ids[i], NULL, chef, chef_data[i]);
    }
```

Wholesaler Design(Producer and consumer)

The wholesaler reads 2 characters and a newline from the input file at each iteration until the end of the file. These 2 characters are placed into the stock[] array using the transfer() function. Synchronization is handled in the transfer function by waiting/posting sem_stock_access which the chef threads also use.

Wholesaler also waits a little bit longer after it’s done placing all the ingredients for chefs to finish delivering desserts.

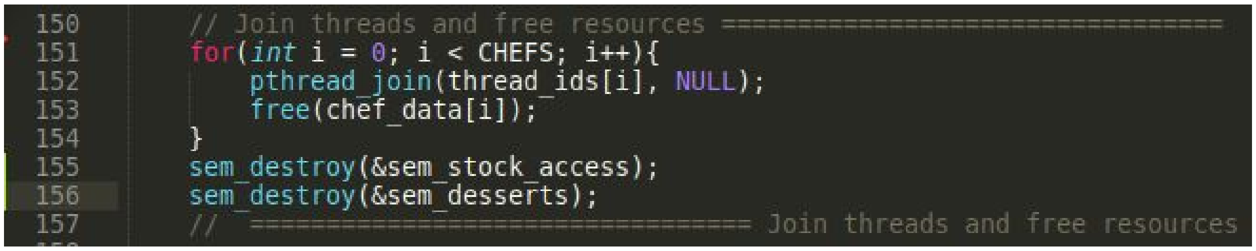
Chef Design(Consumer and producer)

Chefs threads priority is to sell the dessert they made first. If they don’t have a dessert ready, they wait to get stock access. After getting the stock access they check for ingredients they are missing and take them if it exists in the stock. Then they release the stock access(sem_stock_access).

They work until the wholesaler notices them that he is leaving.

Freeing Resources

Threads are joined and semaphores are destroyed at the end of the program.



```
    // Join threads and free resources ==================================
    for(int i = 0; i < CHEFS; i++){
        pthread_join(thread_ids[i], NULL);
        free(chef_data[i]);
    }
    sem_destroy(&sem_stock_access);
    sem_destroy(&sem_desserts);
    //  ================================== Join threads and free resources
```