CSE 344 SYSTEM PROGRAMMING:

REPORT:

SERDİL ANIL ÜNLÜ

1801042672

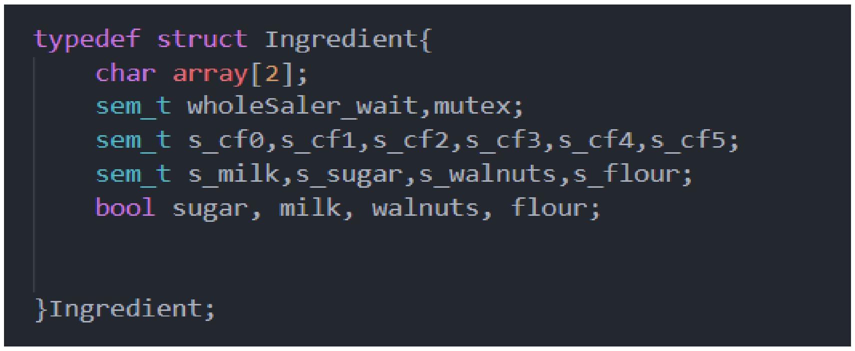
HOW I SOLVED PROBLEM AND MY DESIGN?

UNNAMED.C

-First of all, I am explaining my solution for unnamed semaphore. I used structure for easy access, I have character string, Boolean values and semaphore values in the structure. I use Boolean values in my functions called helper, which I use as pushers. I use the character string to show the values I read from the file and to understand what materials I need according to those values. I opened init_sharedMemory function for unnamed semaphore and in this function I used shm_open, ftruncate and mmap functions respectively. I created a function for each chef, and I also created a pusher function for each material.

I call my helper functions and the functions I wrote for chef in child. In Parent, I perform the wholesaler transaction. According to the letters from the file, whichever chef needs contact with the wholesaler, the wholesaler waits for that chef to pick up the ingredients and produce the product, and then move on to the next chefs. Finally , I killed the child processes by doing kill and wait(null).

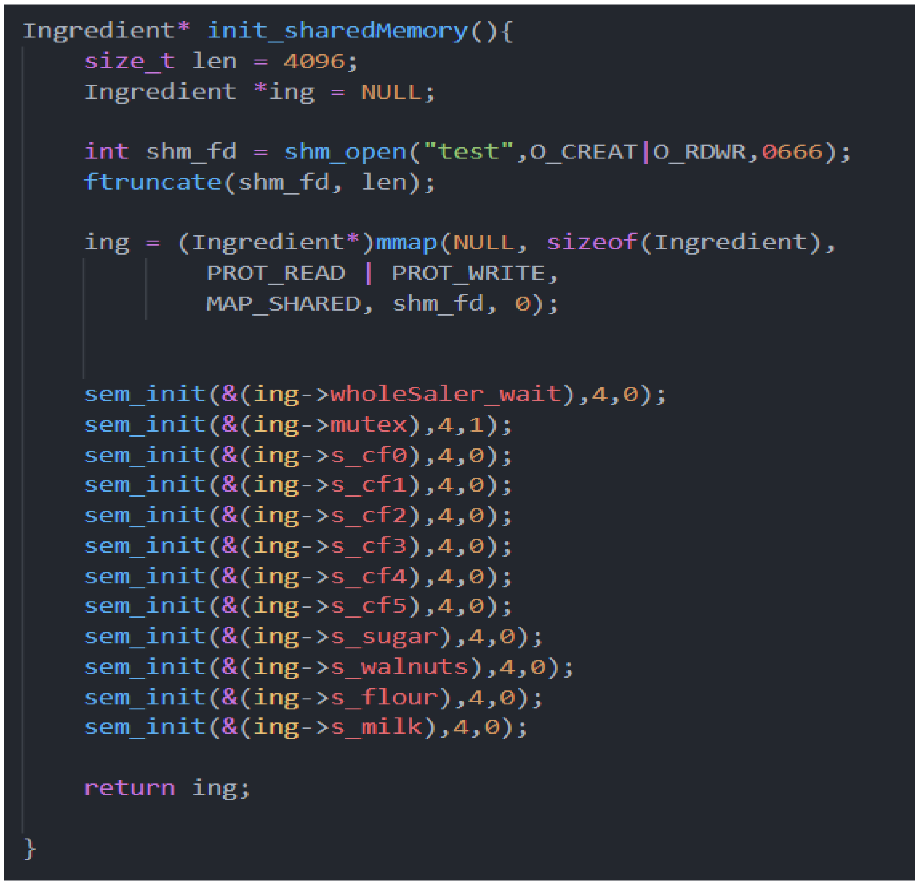
-Some of the screenshots of the functions I wrote in the unnamed.c file are as follows:



```c
typedef struct Ingredient{
    char array[2];
    sem_t wholeSaler_wait,mutex;
    sem_t s_cf0,s_cf1,s_cf2,s_cf3,s_cf4,s_cf5;
    sem_t s_milk,s_sugar,s_walnuts,s_flour;
    bool sugar, milk, walnuts, flour;


}Ingredient;
```

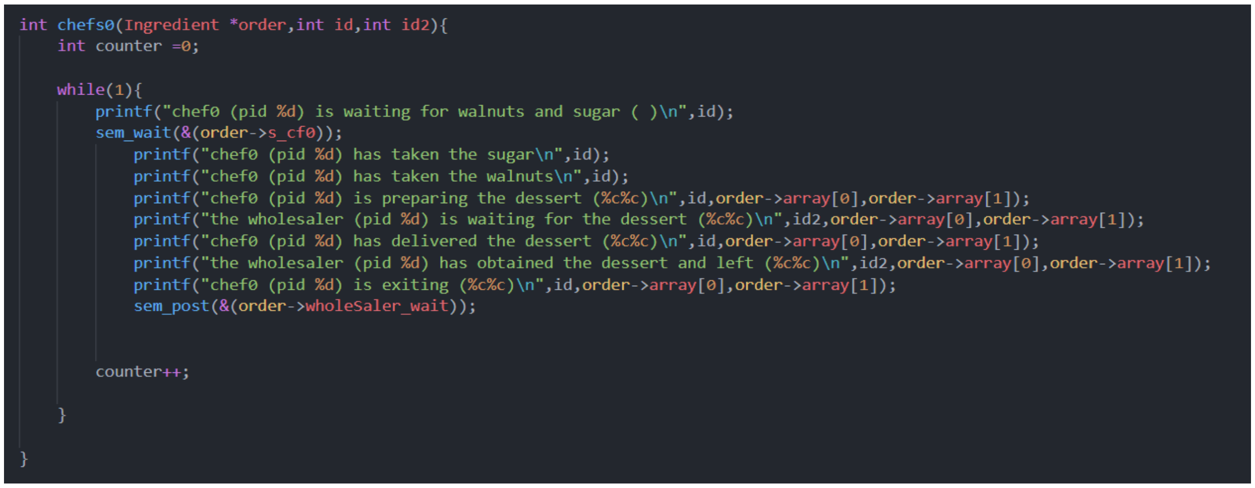
My structure.

The function I set up to do the init process.

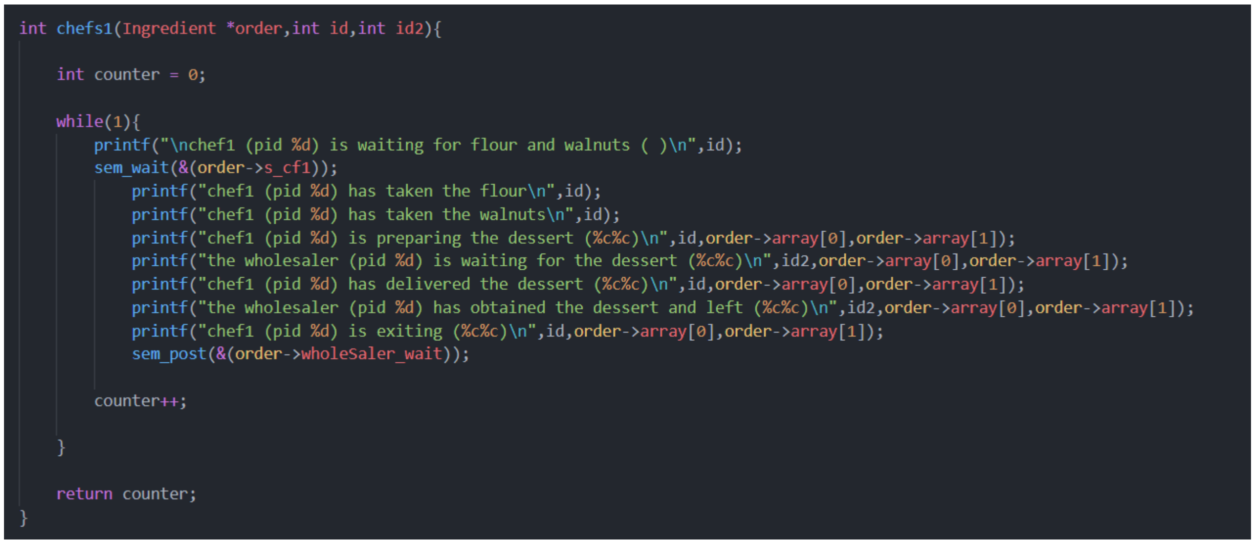


```c
Ingredient* init_sharedMemory(){
    size_t len = 4096;
    Ingredient *ing = NULL;

    int shm_fd = shm_open("test",O_CREAT|O_RDWR,0666);
    ftruncate(shm_fd, len);

    ing = (Ingredient*)mmap(NULL, sizeof(Ingredient),
            PROT_READ | PROT_WRITE,
            MAP_SHARED, shm_fd, 0);


    sem_init(&(ing->wholeSaler_wait),4,0);
    sem_init(&(ing->mutex),4,1);
    sem_init(&(ing->s_cf0),4,0);
    sem_init(&(ing->s_cf1),4,0);
    sem_init(&(ing->s_cf2),4,0);
    sem_init(&(ing->s_cf3),4,0);
    sem_init(&(ing->s_cf4),4,0);
    sem_init(&(ing->s_cf5),4,0);
    sem_init(&(ing->s_sugar),4,0);
    sem_init(&(ing->s_walnuts),4,0);
    sem_init(&(ing->s_flour),4,0);
    sem_init(&(ing->s_milk),4,0);

    return ing;

}
```

Some of my chef functions:

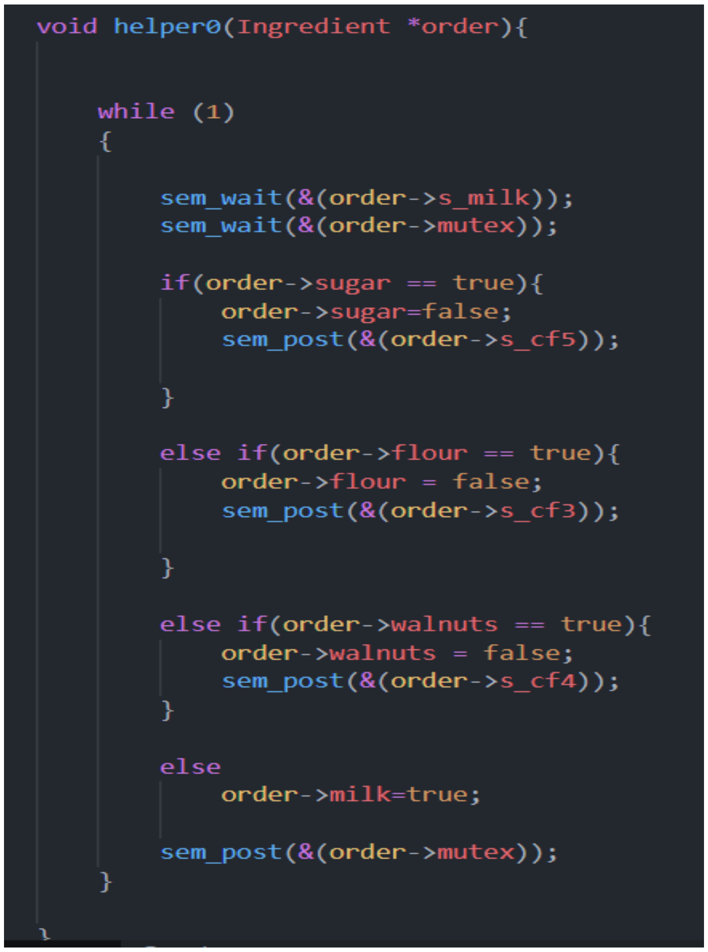


```c
int chefs0(Ingredient *order,int id,int id2){
    int counter =0;

    while(1){
        printf("chef0 (pid %d) is waiting for walnuts and sugar ( )\n",id);
        sem_wait(&(order->s_cf0));
            printf("chef0 (pid %d) has taken the sugar\n",id);
            printf("chef0 (pid %d) has taken the walnuts\n",id);
            printf("chef0 (pid %d) is preparing the dessert (%c%c)\n",id,order->array[0],order->array[1]);
            printf("the wholesaler (pid %d) is waiting for the dessert (%c%c)\n",id2,order->array[0],order->array[1]);
            printf("chef0 (pid %d) has delivered the dessert (%c%c)\n",id,order->array[0],order->array[1]);
            printf("the wholesaler (pid %d) has obtained the dessert and left (%c%c)\n",id2,order->array[0],order->array[1]);
            printf("chef0 (pid %d) is exiting (%c%c)\n",id,order->array[0],order->array[1]);
            sem_post(&(order->wholeSaler_wait));


        counter++;

    }
}
```

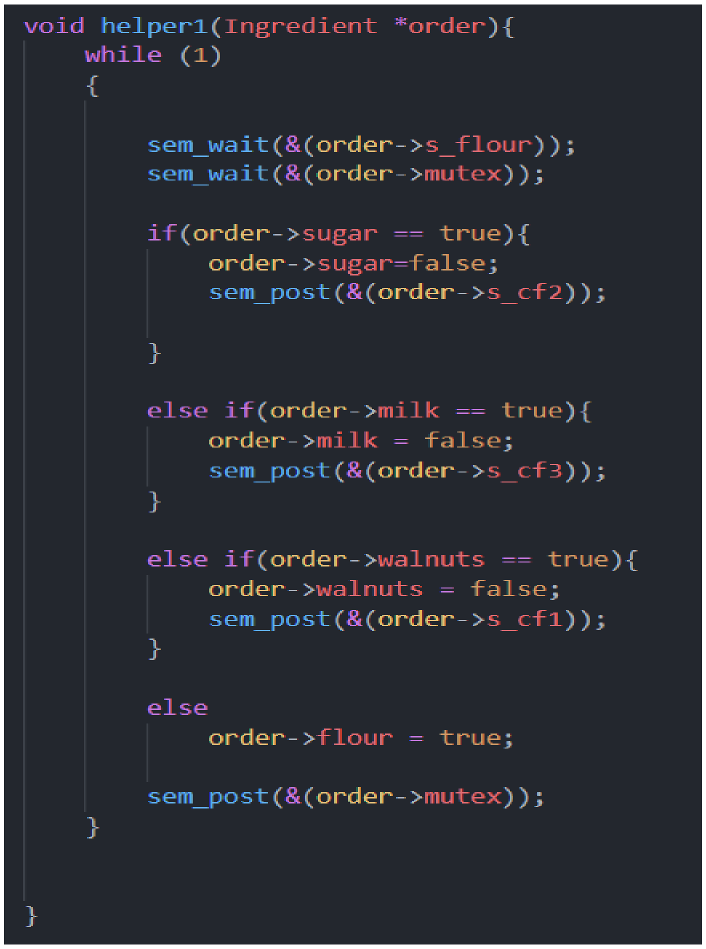


```c
int chefs1(Ingredient *order,int id,int id2){

    int counter = 0;

    while(1){
        printf("\nchef1 (pid %d) is waiting for flour and walnuts ( )\n",id);
        sem_wait(&(order->s_cf1));
            printf("chef1 (pid %d) has taken the flour\n",id);
            printf("chef1 (pid %d) has taken the walnuts\n",id);
            printf("chef1 (pid %d) is preparing the dessert (%c%c)\n",id,order->array[0],order->array[1]);
            printf("the wholesaler (pid %d) is waiting for the dessert (%c%c)\n",id2,order->array[0],order->array[1]);
            printf("chef1 (pid %d) has delivered the dessert (%c%c)\n",id,order->array[0],order->array[1]);
            printf("the wholesaler (pid %d) has obtained the dessert and left (%c%c)\n",id2,order->array[0],order->array[1]);
            printf("chef1 (pid %d) is exiting (%c%c)\n",id,order->array[0],order->array[1]);
            sem_post(&(order->wholeSaler_wait));

        counter++;

    }

    return counter;
}
```

Some of my pusher functions: (there is 4 pusher(helper functions for ingredient)):

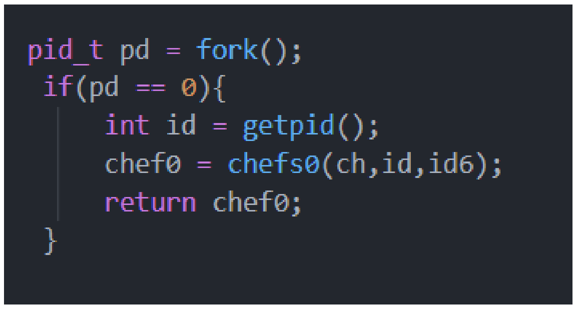


```c
void helper0(Ingredient *order){


    while (1)
    {

        sem_wait(&(order->s_milk));
        sem_wait(&(order->mutex));

        if(order->sugar == true){
            order->sugar=false;
            sem_post(&(order->s_cf5));

        }

        else if(order->flour == true){
            order->flour = false;
            sem_post(&(order->s_cf3));

        }

        else if(order->walnuts == true){
            order->walnuts = false;
            sem_post(&(order->s_cf4));
        }

        else
            order->milk=true;

        sem_post(&(order->mutex));
    }

}
```



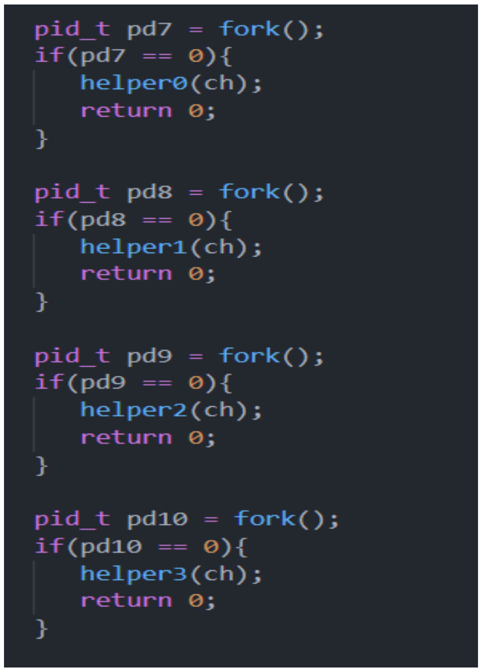
```
void helper1(Ingredient *order){
    while (1)
    {

        sem_wait(&(order->s_flour));
        sem_wait(&(order->mutex));

        if(order->sugar == true){
            order->sugar=false;
            sem_post(&(order->s_cf2));

        }

        else if(order->milk == true){
            order->milk = false;
            sem_post(&(order->s_cf3));
        }

        else if(order->walnuts == true){
            order->walnuts = false;
            sem_post(&(order->s_cf1));
        }

        else
            order->flour = true;

        sem_post(&(order->mutex));
    }


}
```

This is how I call the chief functions in children.



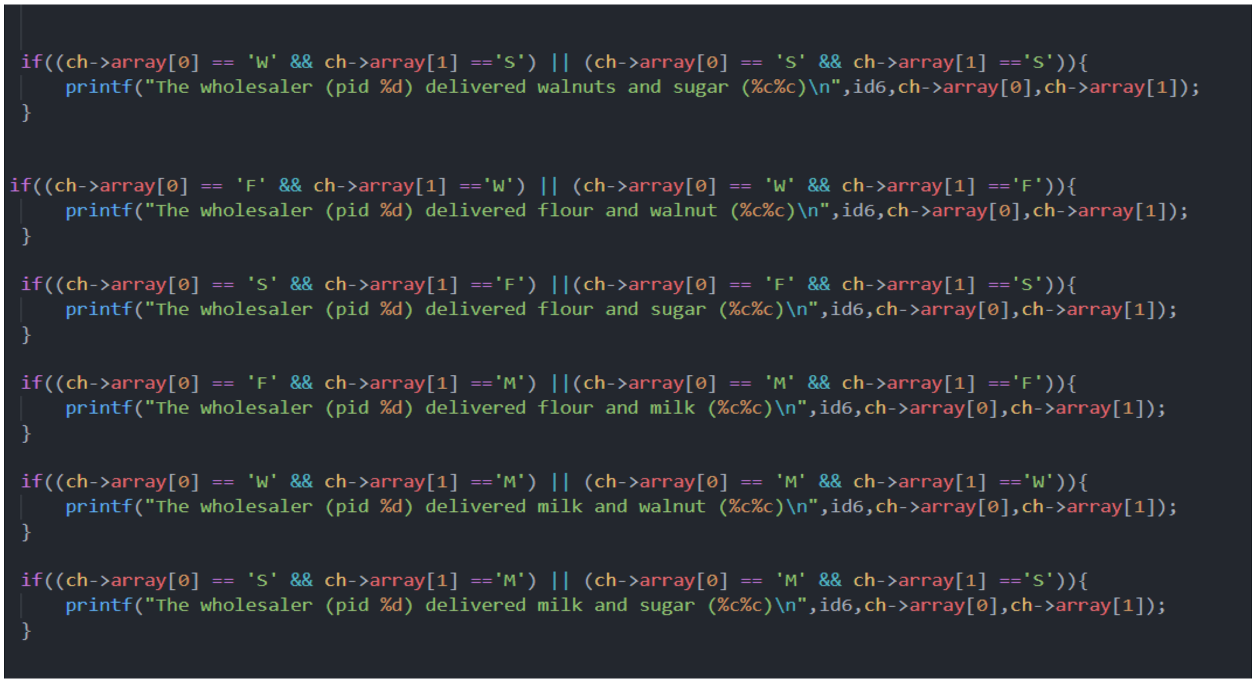
```
pid_t pd = fork();
 if(pd == 0){
     int id = getpid();
     chef0 = chefs0(ch,id,id6);
     return chef0;
 }
```

- I also used children for my Helper(pusher) functions.

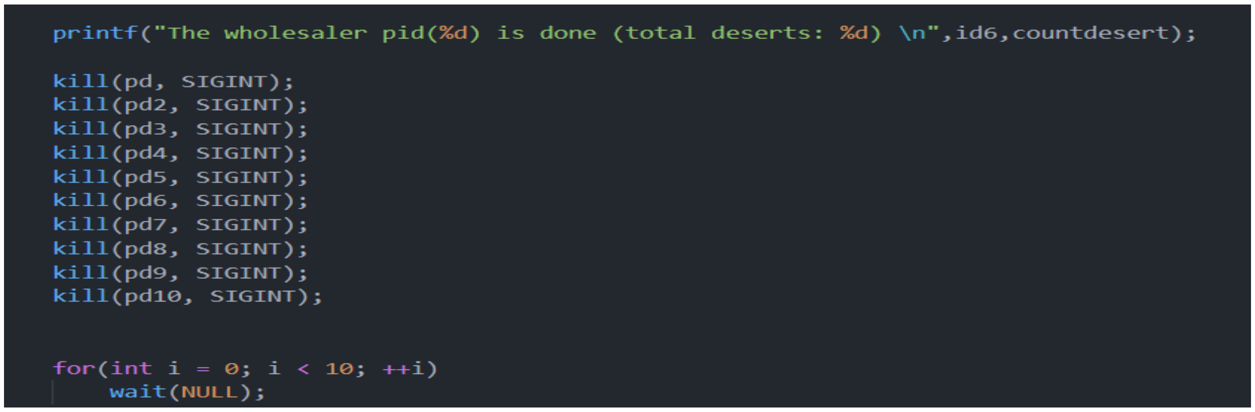


```c
pid_t pd7 = fork();
if(pd7 == 0){
    helper0(ch);
    return 0;
}

pid_t pd8 = fork();
if(pd8 == 0){
    helper1(ch);
    return 0;
}

pid_t pd9 = fork();
if(pd9 == 0){
    helper2(ch);
    return 0;
}

pid_t pd10 = fork();
if(pd10 == 0){
    helper3(ch);
    return 0;
}
```

- I also do wholesaler transactions on parent.



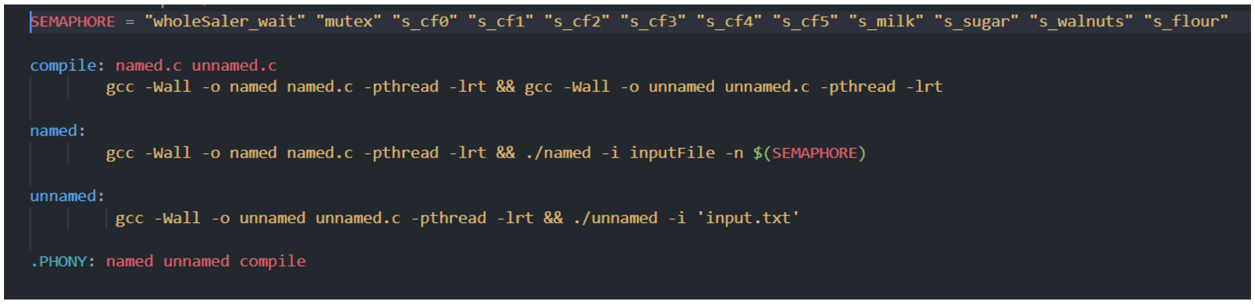
```c
 if((ch->array[0] == 'W' && ch->array[1] =='S') || (ch->array[0] == 'S' && ch->array[1] =='S')){
     printf("The wholesaler (pid %d) delivered walnuts and sugar (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }


if((ch->array[0] == 'F' && ch->array[1] =='W') || (ch->array[0] == 'W' && ch->array[1] =='F')){
     printf("The wholesaler (pid %d) delivered flour and walnut (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }

 if((ch->array[0] == 'S' && ch->array[1] =='F') ||(ch->array[0] == 'F' && ch->array[1] =='S')){
     printf("The wholesaler (pid %d) delivered flour and sugar (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }

 if((ch->array[0] == 'F' && ch->array[1] =='M') ||(ch->array[0] == 'M' && ch->array[1] =='F')){
     printf("The wholesaler (pid %d) delivered flour and milk (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }

 if((ch->array[0] == 'W' && ch->array[1] =='M') || (ch->array[0] == 'M' && ch->array[1] =='W')){
     printf("The wholesaler (pid %d) delivered milk and walnut (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }

 if((ch->array[0] == 'S' && ch->array[1] =='M') || (ch->array[0] == 'M' && ch->array[1] =='S')){
     printf("The wholesaler (pid %d) delivered milk and sugar (%c%c)\n",id6,ch->array[0],ch->array[1]);
 }
```



```c
printf("The wholesaler pid(%d) is done (total deserts: %d) \n",id6,countdesert);

kill(pd, SIGINT);
kill(pd2, SIGINT);
kill(pd3, SIGINT);
kill(pd4, SIGINT);
kill(pd5, SIGINT);
kill(pd6, SIGINT);
kill(pd7, SIGINT);
kill(pd8, SIGINT);
kill(pd9, SIGINT);
kill(pd10, SIGINT);


for(int i = 0; i < 10; ++i)
    wait(NULL);
```

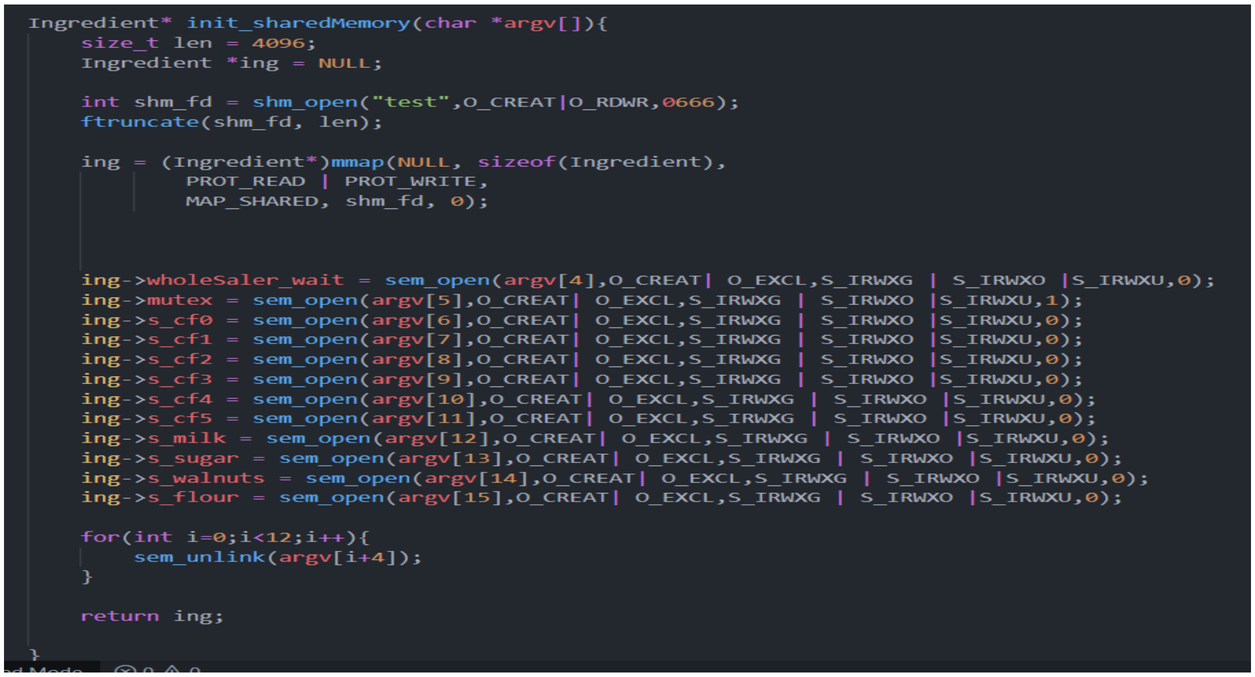
named.c file :

- I wrote all the semaphores in the makefile to run the named semaphore in the terminal, and if you want to run it in a short way, you can both compile and run it by directly by writing “make named” on terminal.

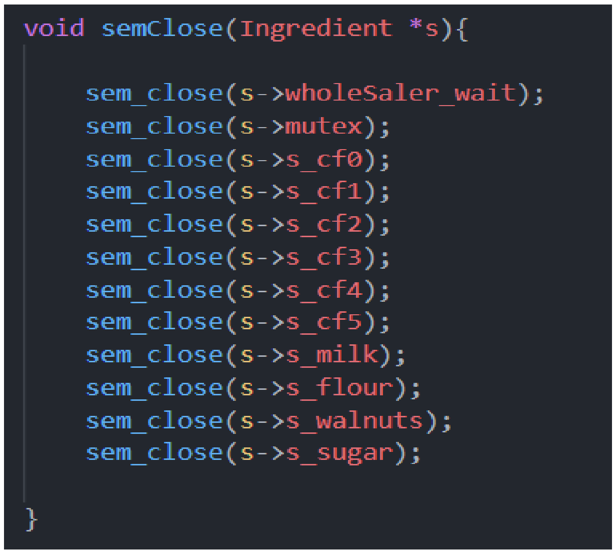


```makefile
SEMAPHORE = "wholeSaler_wait" "mutex" "s_cf0" "s_cf1" "s_cf2" "s_cf3" "s_cf4" "s_cf5" "s_milk" "s_sugar" "s_walnuts" "s_flour"

compile: named.c unnamed.c
        gcc -Wall -o named named.c -pthread -lrt && gcc -Wall -o unnamed unnamed.c -pthread -lrt

named:
        gcc -Wall -o named named.c -pthread -lrt && ./named -i inputFile -n $(SEMAPHORE)

unnamed:
         gcc -Wall -o unnamed unnamed.c -pthread -lrt && ./unnamed -i 'input.txt'

.PHONY: named unnamed compile
```

- Unlike the unnamed file in the named.c file, I used the sem_open function instead of the the init function. I also did not neglect to use the unlink function after making sem_open. I also made sem_close inside the children.

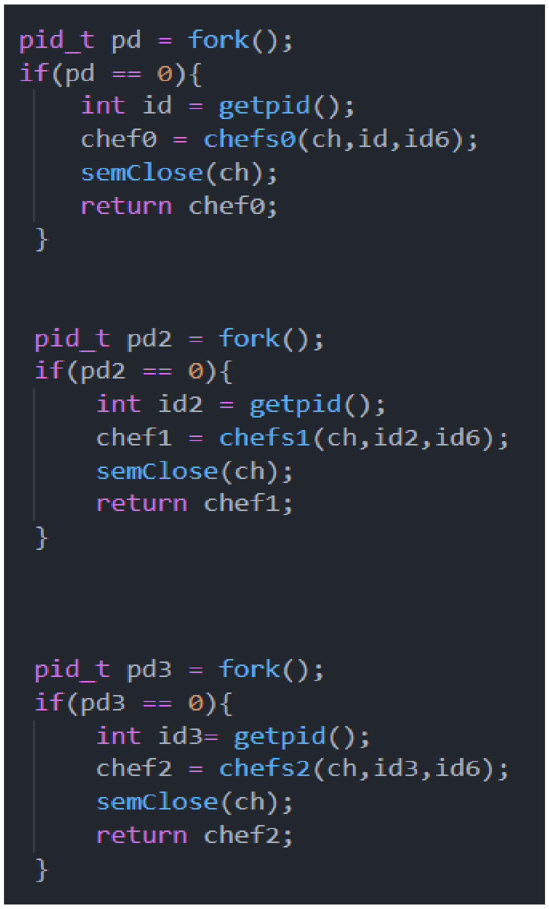


```c
Ingredient* init_sharedMemory(char *argv[]){
    size_t len = 4096;
    Ingredient *ing = NULL;

    int shm_fd = shm_open("test",O_CREAT|O_RDWR,0666);
    ftruncate(shm_fd, len);

    ing = (Ingredient*)mmap(NULL, sizeof(Ingredient),
            PROT_READ | PROT_WRITE,
            MAP_SHARED, shm_fd, 0);


    ing->wholeSaler_wait = sem_open(argv[4],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->mutex = sem_open(argv[5],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,1);
    ing->s_cf0 = sem_open(argv[6],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_cf1 = sem_open(argv[7],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_cf2 = sem_open(argv[8],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_cf3 = sem_open(argv[9],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_cf4 = sem_open(argv[10],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_cf5 = sem_open(argv[11],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_milk = sem_open(argv[12],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_sugar = sem_open(argv[13],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_walnuts = sem_open(argv[14],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);
    ing->s_flour = sem_open(argv[15],O_CREAT| O_EXCL,S_IRWXG | S_IRWXO |S_IRWXU,0);

    for(int i=0;i<12;i++){
        sem_unlink(argv[i+4]);
    }

    return ing;

}
```



```c
void semClose(Ingredient *s){

    sem_close(s->wholeSaler_wait);
    sem_close(s->mutex);
    sem_close(s->s_cf0);
    sem_close(s->s_cf1);
    sem_close(s->s_cf2);
    sem_close(s->s_cf3);
    sem_close(s->s_cf4);
    sem_close(s->s_cf5);
    sem_close(s->s_milk);
    sem_close(s->s_flour);
    sem_close(s->s_walnuts);
    sem_close(s->s_sugar);

}
```

- I closed my semaphores with sem_close inside the semClose function, then I call the semClose function inside the children inside the main.



```
pid_t pd = fork();
if(pd == 0){
    int id = getpid();
    chef0 = chefs0(ch,id,id6);
    semClose(ch);
    return chef0;
 }


 pid_t pd2 = fork();
 if(pd2 == 0){
     int id2 = getpid();
     chef1 = chefs1(ch,id2,id6);
     semClose(ch);
     return chef1;
 }



 pid_t pd3 = fork();
 if(pd3 == 0){
     int id3= getpid();
     chef2 = chefs2(ch,id3,id6);
     semClose(ch);
     return chef2;
 }
```

WHICH REQUIREMENTS I ACHIEVED AND FAILED?

I complete all tasks, including the latest updates. I wrote a helper function for each material under the name of pusher functions , and I also use the character string to print it on each line in the terminal.