Gebze Technical University
CSE 344 – System Programming
(2022 - Spring)
HW 3 Report

Muhammed Bedir ULUCAY – 1901042697

In this homework we are simulating a bakery – wholesaler system between 6 chef and 1 wholesaler.
Each chef has needs 4 ingredients to make a dessert 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.

When a wholesaler arrive the chefs street he has 2 ingredient to deliver but its not deliver the directly the chef leaves the middle of the street. And Chefs checks the ingredients are what I need to make a dessert. And there is also a rule. A chef takes the either two of them or nothing. So that we are match the chef number and all possible 2 ingredient.
Combination (4 2) = 6 and there are 6 chef so each wholesaler arrive the street one of the chef must get to be ingredients. And chef getting these ingredients from the street. And making dessert and He also leaving the dessert middle of the street and wholesaler getting the dessert from there.

To manage these synchronization we need a inter-layer called pusher. To make the force the rule which A chef takes the ingredients either two of them or nothing. So we can easily manage the control the these role.

In the implementation we have a base main.c file and it takes arguments and creating a wholesaler.c child process. Then all the chef.c and pusher.c creating from the wholesaler.

note: wholesaler just creating the chef.c not talking them just take the total returned dessert number.

Ex:

```
// flour - walnut - sugar - milk
char* argv_pusher0[] = {
    "./pusher",
    "Flour",
    "8",                    // produtc semaphore index
    "1", "2", "3",          // chefs semaphore indexes
    "1","2","3","0",        // boolean queue indexes
    SHARED_MEM,
    SHARED_BOOL,
    SEMAPHORE_SHARED_MEM,
    NULL
};
```

```
char* argv_chef0[] = {
    "./chef",
    "0",
    "Walnuts",
    "Sugar",
    SHARED_MEM,
    SHARED_BOOL,
    SEMAPHORE_SHARED_MEM,
    NULL
};
```

Sending arguments are related the process task.

Wholesaler:

```
while((read_byte = read(ingredient_fd, tmp_addr, sizeof(char) * 3)) && (errno == EINTR) ){
    if(errno == EINTR)
        continue;
    perror("read");
    exit(1);
}
```

Take the argument from data file and store them in shared memory. Then according to the argument He pushes the related ingredients.

```
char first_ing = tmp_addr[0];
switch (first_ing)
{
    case 'F':  sem_post(flour_sem);      break;
    case 'S':  sem_post(sugar_sem);      break;
    case 'M':  sem_post(milk_sem);       break;
    case 'W':  sem_post(walnut_sem);     break;
    default:
        fprintf(stderr," 1 +%c+ Unknown ingredient\n", first_ing);
}
```

Pusher:
A generic C file can be 4 different pusher after setting their arguments.

```
int ingredient_offset = atoi(argv[2]);

int chef1_offset = (int) argv[3][0] - '0';
int chef2_offset = (int) argv[4][0] - '0';
int chef3_offset = (int) argv[5][0] - '0';

int bool1_offset = (int) argv[6][0] - '0';
int bool2_offset = (int) argv[7][0] - '0';
int bool3_offset = (int) argv[8][0] - '0';
int bool4_offset = (int) argv[9][0] - '0';

char* shm_bool_name = argv[11];
char* shm_sem_name = argv[12];      You, 3
```

Using this offsets value we can get the position of the wanted shared memory variable.

After getting the variable our pusher is ready to use.

```
sem_wait(ingredient_sem);
sem_wait(mutex);        You, 3 days ago

if(*bool_1_ing == True){
    *bool_1_ing = False;
    sem_post(chef_1_sem);
}else if(*bool_2_ing == True){
    *bool_2_ing = False;
    sem_post(chef_2_sem);
}else if(*bool_3_ing == True){
    *bool_3_ing = False;
    sem_post(chef_3_sem);
}else{
    *bool_4_ing = True;
}

sem_post(mutex);
```

Using a additional a Boolean shared memory we can manage the semaphore for the related chef.

note:
chef_1_sem is generic name it could be street's chef_5, chef_6 according to the getting argument in the creation process. It is same for the Boolean variables.

Chef:

Chef is waiting the waken up by the pusher after the wake up its getting the ingredients from the shared memory and put it back to street so wholesaler can take the dessert from the street.

```
sem_wait(chef_sem);

if(chef_sigint)
    break;

pop_shared_memory(shared_memory, nth);
counter++;

sem_post(wholesaler_sem);
```

It also returned the counter variable to count the how many dessert he made.

This values of all chefs getting by the wholesaler and print the total gathering dessert on the terminal.

```
int sum = 0;
int ret_val;
for(int i=0; i<CHEF_SIZE; ++i){
    waitpid(chef_pids[i], &ret_val, 0);
    if(ret_val == -1){
        perror("waitpid");
        exit(1);
    }
    sum += WEXITSTATUS(ret_val);
}

fprintf(stdout, "The wholesaler (pid %d) is done (total dessert : %d)\n", getpid(), sum);
```

Example Result: