Gebze Technical University
Computer Engineering
CSE344-HW3
Assignment

Mustafa Gurler
171044034

1-)Problem Definition

This project is aimed at multiple processors. The problem of cigarette smokers has been addressed and resolved. The main purpose of this problem is to deliver the limited materials provided by the wholesaler to the required person by means of semaphores, besides the unlimited number of materials. In this way, any deadlock situation is avoided.

2-)Design and Solution

In general, one semaphore is used for each process, besides, one semaphore is used for each material. The "is_done" semaphore was used to suppress the suppressed result after the dessert was made. Two-character data read from the file name received from the terminal is transmitted to the process that needs these materials with the help of pusher processes. For this, 6 processes plus 4 pusher processes were created during the fork. The pushers decide which chef will work. That chefin semaphore is upgraded with a post.

2.1 Reading Arguments

```
int _i_ = 0;
int _n_ = 0;
int c;
char* inputFilePath;
char* nameFile;

while((c = getopt(argv, argc, "i:n:")) != -1){
    switch(c)
    {
        case 'i':
            inputFilePath = optarg;
            _i_ = 1;

            break;
        case 'n':
            nameFile = optarg;
            _n_ = 1;

            break;
        default:
            break;
    }
}
if((!_i_) || (!_n_)){
    perror("No input File");
    exit(EXIT_FAILURE);
}
```

I used just simple getopt function. I got all the requirements

2.2 Initializing Semaphores

```
sem_agent = mmap(NULL, sizeof(*sem_agent), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
sem_agent = sem_open(name,(O_CREAT | O_RDWR | O_EXCL), 0666, 1);
if(sem_agent< 0)
{
    perror("sem_init");
    exit(EXIT_FAILURE);
}
sem_chef0 = mmap(NULL, sizeof(*sem_chef0), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
sem_chef0 = sem_open("sem_chef0",(O_CREAT | O_RDWR | O_EXCL), 0666, 0);
if (sem_chef0 < 0)
{
    perror("sem_init");
    exit(EXIT_FAILURE);
}
sem_chef1 = mmap(NULL, sizeof(*sem_chef1), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
sem_chef1 = sem_open("sem_chef1",(O_CREAT | O_RDWR | O_EXCL), 0666, 0);
if (sem_chef1 < 0)
{
    perror("sem_init");
    exit(EXIT_FAILURE);
}
sem_chef2 = mmap(NULL, sizeof(*sem_chef2), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
sem_chef2 = sem_open("sem_chef2",(O_CREAT | O_RDWR | O_EXCL), 0666, 0);
if (sem_chef2 < 0)
{
    perror("sem_init");
    exit(EXIT_FAILURE);
}
sem_chef3 = mmap(NULL, sizeof(*sem_chef3), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
sem_chef3 = sem_open("sem_chef3",(O_CREAT | O_RDWR | O_EXCL), 0666, 0);
if (sem_chef3 < 0)
{
    perror("sem_init");
    exit(EXIT_FAILURE);
}
```

I used sem_open system call for named semaphores, used sem_init system call for unnamed semaphores.

2.3 SIGINT kill Signal

```
struct sigaction sigterm_action;
memset(&sigterm_action, 0, sizeof(sigterm_action));
sigterm_action.sa_handler = &sigterm_handler;

if(sigaction(SIGINT, &sigterm_action, NULL) != 0){
    perror("sigaction");
}
```

I ended all the processes via SIGINT signal.

2.4 Chef Process

```
if(i == 0 && children_pid[i] == 0){
    int count=0;
    while(1){

        fprintf(stdout, "chef%d (pid %d) is waiting for %c and %c\n", i,getpid(), istrueflag->arr[0], istrueflag->arr[1]);
        fflush(stdout);
        if(sigtermInterrupt == 1){
            break;
        }
        sem_wait(sem_chef0);
        if(sigtermInterrupt == 1){
            break;
        }
        fprintf(stdout,"chef%d (pid %d) has taken the %c\n", i, getpid(), istrueflag->arr[0]);
        fprintf(stdout,"chef%d (pid %d) has taken the %c\n", i, getpid(), istrueflag->arr[1]);
        fprintf(stdout,"chef%d (pid %d) is preparing the dessert\n", i, getpid());
        fflush(stdout);
        if(sigtermInterrupt == 1){
            break;
        }
        sem_post(sem_agent);
        if(sigtermInterrupt == 1){
            break;
        }
        sem_post(is_done);

        count++;
        fprintf(stdout,"chef%d (pid %d) has delivered the dessert\n", i, getpid());
        fflush(stdout);
        if(sigtermInterrupt == 1){
            break;
        }
    }
    fprintf(stdout,"chefi (pid %d) is exiting\n", getpid());
    return count;
```

Chef process waits the signal of sem_chef from pusher. After that it prepares the dessert and increase the number of dessert which he has made it so far. If any SIGINT signal arrives it breaks the loop and returns the total number of dessert which has been made by chef.

2.5 Pusher Process

```
}else if(i == 7 && children_pid[i] == 0){
    while(1){
        if(sigtermInterrupt == 1){
            break;
        }
        sem_wait(sugar);
        if(sigtermInterrupt == 1){
            break;
        }
        sem_wait(sem_mux);
        if(sigtermInterrupt == 1){
            break;
        }

        if(istrueflag->isFlour == 1){
            istrueflag->isFlour = 0;
            if(sigtermInterrupt == 1){
                break;
            }
            sem_post(sem_chef2);
            if(sigtermInterrupt == 1){
                break;
            }
        }else if(istrueflag->isMilk == 1){
            istrueflag->isMilk = 0;
            if(sigtermInterrupt == 1){
                break;
            }
            sem_post(sem_chef5);
            if(sigtermInterrupt == 1){
                break;
            }
        }else if(istrueflag->isWalnuts == 1){
            istrueflag->isWalnuts = 0;
            if(sigtermInterrupt == 1){
                break;
            }
            sem_post(sem_chef0);
            if(sigtermInterrupt == 1){
                break;
            }
        }
        if(sigtermInterrupt == 1){
            break;
        }
        sem_post(sem_mux);
        if(sigtermInterrupt == 1){
```

It checks two of ingredient that matches and signal to chef. It also waits SIGINT signal to end process.

2.6 WholeSaler Process

```c
/*read csv from the file and add all the number to buffer*/
unsigned char buf[3] = {""};

/*Read csv file*/
size_t bytes_read;
int readFd = open(inputFilePath, O_RDONLY);

do{
    sem_wait(sem_agent);

    bytes_read = read(readFd, buf, sizeof(buf));
    istrueflag->arr[0] = buf[0];
    istrueflag->arr[1] = buf[1];
    if(buf[1] == 'M'){
        istrueflag->isMilk = 1;
    }else if(buf[1] == 'F'){
        istrueflag->isFlour = 1;
    }else if(buf[1] == 'W'){
        istrueflag->isWalnuts = 1;
    }else if(buf[1] == 'S'){
        istrueflag->isSugar = 1;
    }else{
        fprintf(stderr, "Wrong Ingredient!!!\n");
        exit(EXIT_FAILURE);
    }
    fprintf(stdout,"the wholesaler (pid %d) delivers %c and %c\n", getpid(), buf[0], buf[1]);
    postIng(buf[0]);
    fprintf(stdout, "the wholesaler (pid %d) is waiting for the dessert\n", getpid());
    sem_wait(is_done);
    fprintf(stdout,"the wholesaler (pid %d) has obtained the dessert and left\n", getpid());

}while(bytes_read == sizeof(buf));

int ret;
int wstatus[10];
int returned[10];
for(int i=0 ; i<10 ; i++){
    ret = kill(children_pid[i], SIGINT);
    if (ret == -1) {
        perror("kill");
        exit(EXIT_FAILURE);
    }

}
```

It reads the csv file and changes the flag true. After that it signals to pusher. That is the start process and after the chef made the dessert and all the ingredients comes end, total dessert number has been calculated and terminates the chef processes.

5. Which requirements I achieved?

I made all the requirements of assignment. You can try any of part you want. Thank you.