SYSTEMS PROGRAMMING

#midterm_project

jwan hussein
151044078

parent process :

the parent process is the process which initialize the shared memory object , initialize the counters , semaphores and mutexes .
Then it forking all other processes and wait for them to finish then destroying the semaphores and mutexes and unlinks the shared memory object then terminates .

the shared memory structure :

struct shm{

int KP ; // counter to hold the number of soup in the kitchen
int KC ; // counter to hold the number of main course in the kitchen
int KD ; // counter to hold the number of dessert in the kitchen
int CP ; // counter to hold the number of soup on the counter
int CC ; // counter to hold the number of mian course on the counter
int CD ; // counter to hold the number of dessert on the counter

int KSP ; // counter to hold the total number of soup was delivered to the kitchen
int KSC ; // counter to hold the total number of main course was delivered to the kitchen
int KSD ; // counter to hold the total number of dessert was delivered to the kitchen

int total_served_P ; // counter to hold the number of soup plates has been served by the cooks
int total_served_C ; // counter to hold the number of main course plates has been served by the cooks
int total_served_D ; // counter to hold the number of dessert plates has been served by the cooks

int tables ; // counter to hold the number of empty tables
int Ustudentq ; // counter to hold the number of Ustudent in the queue
int Gstudentq ; // counter to hold the number of Gstudent in the queue

sem_t Kempty ; // semaphore to keep tracking empty spaces in the kitchen , initial value = K
sem_t Cempty ; // semaphore to keep tracking empty spaces in the counter , initial value = S
sem_t Tempty ; // semaphore to keep tracking empty table , initial value = T

sem_t Kmutex ; // binary semaphore to lock and unlock kitchen counters KP , KC and KD , initial value = 1
sem_t Cmutex ; // binary semaphore to lock and unlock counter counters CP , CC and CD , initial value = 1
sem_t Tmutex ; // mutex to lock and unlock the tables counter , initial value = 1 ;

sem_t TSmutex; // mutex to lock and unlock counter counters total_served_P , total_served_C , total_served_D , initial value = 1

sem_t cook1 ; // semaphore to signal the cook who is responsible for soup , initial value = 0

sem_t C1mutex ; // binary semaphore to signal the cook who is responsible for main course to continue can be posted only from the cook who is resposible for soup , initial value = 0

sem_t C2mutex ; // binary semaphore to signal the cook who is responsible for dessert to continue can be posted only from the cook who is resposible for main course , initial value = 0
sem_t C3mutex ; // binary semaphore to signal the cook who is responsible for soup to continue can be posted only from the cook who is resposible for dessert , initial value = 1

sem_t Ssemaph ; // sempafore whose value indicates the number of students that can be served , initial value = 0
sem_t SQ ; // student queue mutex , initial value = 1
sem_t SU ; // Ustudent mutex (gives permission to Ustudent to take food from the counter) , initial value = 0

```
        sem_t SG      ;   // GStudent mutex (gives permission to
                              Gstudent to take food from the
                              counter ) , initial value = 0
        sem_t cont      ;   // mutex for the organizer
        sem_t SQsem     ;   // sempafore whose value indicates the
                                number of students in the queue
                                initial value = 0
};
```

supplier process :

```
detector = 0 ;

while(1){
        sem_wait(&myshared→Kempty);  // checking for empty space in kitchen
        sem_wait(&myshared→Kmutex);  // cheching if the kitchen is empty
        if((r = read(filed,&ch,1)) == 0){  //  checking for EOF , if EOF was reached it means that the supplier
                                              has done his job
                sem_post(&myshared->Kmutex);   // unlocking the kitchen so that cooks can continue there
                                                  job
                return 0 ;
        }
        if(r == -1 ){           // checking for error while reading the file
                perror("ERROR read file :");
                return -1 ;
        }
        if(ch == 'P'){   // if the byte read from file was P (soup);
                myshared→KP++;   // increase the soup counter in the kitchen
                myshared→KSP++; // increase the soup counter this counter will not be
                                        decreased by the cooks
        }else if( ch == 'C'){ // if the byte read from file was C (main course);
                myshared→KC++;   // increase the main course counter in the kitchen
                myshared→KSC++; // increase the main course counter this counter will not be
                                        decreased by the cooks
        }else if( ch == 'D'){  // if the byte read from file was C (main course);
                myshared→KD++;   // increase the dessert counter in the kitchen
                myshared→KSD++; // increase the dessert counter this counter will not be
                                        decreased by the cooks
        }
        /* after the supplier has delivered a plate to the kitchen it assigns the value of the minimum
           counter (KSP , KSC , KSD) to a variable called temp , now if the temp is not equal to the detector ,it
           means that the  supplier has delivered to the kitchen three different plates so it posts the
           semaphore (cook1) to let the cook who is responsible for the soup know that his now allowed to
           start taking plate to the counter .
        */
        temp = minimum(myshared->KSP , myshared->KSC , myshared->KSD );
        if(temp != detector){
                sem_post(&myshared->cook1);
                detector = temp ;
        }
        sem_post(&myshared→Kmutex); //
        counter++ ;
}
```

each time when the supplier delivers the three kind of meals the (cook 1) semaphore is being posted once so in total it will be posted for M*L time (M is the the number of student , L is The number of round a student does before going home)

cook processes :

since there will be at least three cooks so ,
all the cooks that have there (id%3 == 0),will be responsible for taking soup plates,
all the cooks that have there (id%3 == 1), will be responsible for taking main course plates,
all the cooks that have there (id%3 == 2),will be responsible for taking dessert plates,
so the cooks that

so the cooks that are responsible for dessert will be waiting cook1 semaphore to be posted by supplier then they wait for C3mutex (C3mutex is a binary semaphore , can be posted only by the cooks that are responsible for dessert plates , and initial value = 1) this cooks are also responsible for posting C1mutex

in case of cooks that are responsible for main course don't wait for cook1 they just wait for C1mutex (C1mutex is a binary semaphore that can be posted from cooks that are responsible for soup) this cooks are also responsible for posting C2mutex

and the cooks that are responsible for dessert waits for C2mutex (C1mutex is a binary semaphore that can be posted from cooks that are responsible for main course) this cooks are also responsible for posting C3mutex and Ssemaph (Ssemaph is a semaphore which it's value indicates the number of student that can be served)

Tsmutex is just to lock and unlock the counters total_served_p total_served_C total_served_D

```
while(1){
        sem_wait(&myshared→TSmutex);
        if((id%3) == 0){
                if(myshared->total_served_P == lm){       /* all cooks that are responsible for soup exits here if they
                                                                finish there job  */
                        sem_post(&myshared->TSmutex);
                        return 0 ;
                }
                else{
                        myshared->total_served_P++ ;
                        sem_post(&myshared->TSmutex);
                        sem_wait(&myshared→cook1);   /* here cooks responsible for soup wait for cook1 semaphore to
                                                                be posted by supplier */
                        sem_wait(&myshared→C3mutex);  /* here  here cooks responsible for soup wait for C3mutex
                                                                which initial value is 1 and can be posted only by cooks
                                                                responsible for desserts   */
                }
        }else if((id%3) == 1){
                if(myshared->total_served_C == lm){      /* all cooks that are responsible for main course exits here if
                                                                they finish there job  */
                        sem_post(&myshared->TSmutex);
                        return 0 ;
                }
                else{
                        myshared->total_served_C++ ;
                        sem_post(&myshared->TSmutex);
                        sem_wait(&myshared→C1mutex);   /* here  here cooks responsible for main course wait for
                                                                C1mutex  which initial value is 0 and can be posted only
                                                                by cooks responsible for soup   */

                }
        }else{
                if(myshared->total_served_D == lm){       /* all cooks that aare responsible for dessert exits here if they
                                                                finish there job  */
                        sem_post(&myshared->TSmutex);
                        return 0 ;
                }
                else{
                        myshared->total_served_D++ ;
                        sem_post(&myshared->TSmutex);
                        sem_wait(&myshared→C2mutex);  /* here  here cooks responsible for dessert wait for C2mutex
                                                                which initial value is 0 and can be posted only by cooks
                                                                responsible for main course   */
                }
        }
        sem_wait(&myshared->Kmutex );
        if(id%3 == 0){
                myshared->KP--;
        }else if(id%3 == 1){
                myshared->KC--;
        }else{
                myshared->KD--;
        }
        sem_post(&myshared→Kempty);     /* posting Kempty which means that the empty spaces in kitchen has been
                                                increased */
        sem_post(&myshared→Kmutex);    /* the cooks exits the kitchen  */
        sem_wait(&myshared→Cempty);    /* and try to enter the counter */
```

```
        sem_wait(&myshared→Cmutex);

        if(id%3 == 0){
                myshared->CP++;
                sem_post(&myshared→C1mutex);  /* cooks responsible for soup posts C1mutex */
        }else if(id%3 == 1){
                myshared->CC++;
                sem_post(&myshared→C2mutex);  /* cooks responsible for main course posts C2mutex */
        }else{
                myshared->CD++;
                sem_post(&myshared→C3mutex); /* cooks responsible for main course posts C3mutex */
                sem_post(&myshared→Ssemaph); /* also posts Ssemaph , now for sure there is the all three kind of plates
                                                        on the counter */
        }
        sem_post(&myshared->Cmutex);

}
```

P.S :

there is more wait() and post() in the code source file , which were used for printing the messages , since the messages includes the values of the counters .

students processes :

there is two types of student processes :

1) **the graduates**
2) **the undergraduates**

since graduate students have priority at the counter over undergraduates, another actor had to be involved in order to organize the queue on the counter the **Queue_organizer** process

the Queue_organizer process

```
while(counter < total){
        sem_wait(&myshared→SQsem);     /*   waits here if there is no student in the queue */
        sem_wait(&myshared→Ssemaph);  /*  first it makes sure the all three kind of plate are on the counter */
        sem_wait(&myshared→SQ);            /*  set a lock on students queue */
        if(myshared->Gstudentq > 0){            // if there is graduate in the queue then let them pass first
                myshared→Gstudentq--;        // decrease the queue counter
                sem_post(&myshared→SG);  // post SG semaphore which graduates are waiting for */
                sem_post(&myshared→SQ);  // unlock  students queue
                sem_wait(&myshared→cont); // waits here until the student leaves the counter
        }else if (myshared->Ustudentq > 0){  // if there no graduate then check if there is undergraduates
                myshared→Ustudentq--;       // decrease the queue counter
                sem_post(&myshared→SU); // post SU semaphore which undergraduates are waiting for */
                sem_post(&myshared→SQ); // unlock students queue
                sem_wait(&myshared→cont); // waits here until the student leaves the counter
        }
        counter++ ;
}
```

graduate student process

the graduate students first enter the student queue
then wait for SG semaphore which should be posted by the organizer
after entering the counter the student takes the meal and exit the counter searching for a table
and each graduate student repeats this process for L times

```
while(counter < l){
        sem_wait(&myshared→SQ);
        myshared→Gstudentq++;          // enter the queue
        sem_post(&myshared→SQsem); // increase the semaphore value
        sem_post(&myshared->SQ);
        sem_wait(&myshared→SG);       // wait until the organizer post SG
        sem_wait(&myshared->Cmutex );

        myshared->CP--;
        myshared→CC--;                        //  taking the meal
        myshared->CD--;

        sem_post(&myshared->Cempty );
        sem_post(&myshared->Cempty );
        sem_post(&myshared->Cempty );
        sem_post(&myshared->cont);
        sem_post(&myshared->Cmutex );

        sem_wait(&myshared->Tempty );  // wait for empty table
        sem_wait(&myshared->Tmutex );
        myshared→tables--;                     // decrease tables counter
        sem_post(&myshared->Tmutex );

        /*  the students is eating now */

        sem_post(&myshared->Tempty );
        sem_wait(&myshared->Tmutex );
        myshared→tables++;                  // increasing tables counter
        sem_post(&myshared->Tmutex );

        if(counter+1 < l){
                /* Student left table to eat again */
        }else {
                /* Student is done eating */
                return 0 ;
        }

        counter++;
}
```

P.S :

there is more wait() and post() in the code source file , which were used for printing the messages , since the messages includes the values of the counters .

and its the same with undergraduates with one difference , undergraduates are waiting for SU semaphore