SYSTEMS PROGRAMMING

#HW5

jwan hussein
151044078

structures :

```
struct POSITION{                                //   hold information about 2D point
        double x ;
        double y ;
};

struct FLOWER{
        char name[32];
};

struct FLORIST{                                 // hold information about a florist
        int kinds ;                             // the number of flower kind it have
        char name[32];                          // it’s name
        double speed ;                          // its speed
        struct POSITION pos ;                   // the florist position
        struct FLOWER *flowers ;                // name of the flowers
};

struct sales {                                  // hold information about Sale statistics
        int totalsale ;
        double totaltime ;
};

struct REQUEST {
        char name[32];                          // costumer name
        int dis ;                               // the distance between the costumer and the closest florist
        char flower[32]                         // the requested flower name
        struct POSITION pos ;                   // the position of the costumer
};

struct REQUEST_QUEUE{
        struct REQUEST req[QUEUE_SIZE];
        int pointer_Producer ;                  // points to an index in the queue which where the new request will be pushed by the main thread
        int pointer_Consumer ;                  // points to an index in the queue which where the request will be read from by the florist thread
        int used ;                              // holds the number of requests in the queue
};
```

main thread :

The main, thread first parses the command line and check if it is true , the program terminates in case of invalid command line .

Secondly, the main thread opens the input file , the program terminates in case of any error occurs in the step .

Thirdly, the main thread handle SIGINT by attaching it with handler function

Fourthly, the main thread read the florists block off the file then store all information about all florists in the florists structure , also for each florists it create a mutex and two condition variables (full,empty).

fifthly , allocating a queue and a sales structure for each florist.

Sixthly , the main thread creates a thread for each florist and gives each thread a unique id starting from zero up until the number of the florists which going to represent the index of which the thread can access the florists structure that has been allocated earlier.

And then the main thread starts to read the requests from the file then find the closest florist and push the read request to the queue .

pseudo code

```
while(!done){
        req = get a request from the file ;
        if(no more request are in the the file ){  //
                done = 1 ;
        }
        else{
                send_to =  the id of the closest florist ;
                if(send_to == -1){
                        unknown kind of flower was requested  , the request was ignored .
                }
                else{
                        lock( mutex[send_to] );
                        while(!interrupt and the queue of the closest florist is full ){
                                cond_wait( empty[send_to] , mutex[send_to] );
                        }
                        if(the program was interrupted ){
                                unlock( mutex[send_to] );
                                free(req);
                                break ;
                        }
                        queue[send_to].used++;
                        push the request into the queue
                        queue[send_to].pointer_Producer++;
                        broadcast( &full[send_to] );
                        unlock( &mutex[send_to] );
                        req_num++;                          /*  counting the requests */
                }
                free(req);
        }
}
```

then the main thread checks if it has been interrupted while it was processing the requests ,in case of an interrupt then waits for all other threads to returns and free all heap space allocated and terminates the program .

If no interrupt signal was caught till this point of execution then the main thread is waiting for all other threads to finish processing the requests

pseudo code :

```
mutex_lock(&S_barrier);
while(number of requests has been processed by all other threads < req_num && !interrupt){
        cond_wait( barrier_cond , S_barrier );
}
cancel = 1 ;                        /*  means that all requests has been processed by the florists */
cond_broadcast(&barrier_cond);
for (int i = 0; i < number_of_florists; ++i){
        cond_broadcast(&full[i]);
}
mutex_unlock(&S_barrier);
```

then the main thread call pthread_join on all florists threads , freeing the allocated heap space and close the file then exit

florist threads :

pseudo code :

```
id = thread id ;
while(1){
        mutex_lock( mutex[id] );
        while(!interrupt &&  !cancel && the queue is empty){
                cond_wait( full[id] , mutex[id] );
                if(interrupt || cancel){
                break;
                }
        }
        if(interrupt){
                mutex_unlock(&mutex[id]);
                break;
        }
        if(cancel){
                mutex_unlock(mutex[id]);
                break;
        }
        queue[id].used--;
        ms = random [1 : 250]
        ms += distance / speed;
        s[id].totaltime += ms;
        s[id].totalsale += 1;

        cond_broadcast(empty[id]);
        mutex_unlock(mutex[id]);

        usleep(ms*1000);
        print the msg ;

        pthread_mutex_lock(S_barrier);
        queue[id].pointer_Consumer++;
        pthread_cond_broadcast(barrier_cond);
        pthread_mutex_unlock(S_barrier);

}
mutex_lock(S_barrier);
finish++;
if(finish < number_of_florists && !interrupt){
        cond_wait(barrier_cond,S_barrier);
}else if(!interrupt){
        printf("All requests processed .\n\n");
        pthread_cond_broadcast(&barrier_cond);
        printf("%s closing shop . \n",florists[id].name );
}else{
        pthread_cond_broadcast(&barrier_cond);
}
pthread_mutex_unlock(&S_barrier);
```

All florists wait here until the main thread update the value of cancel to become one

Main thread also has access to queue[id].pointer_Consumer
When it calculate the number of processed requests,
This is why S_barrier is lock before increasing it's value

the florists wait each other to close their shops

briefly , the relation between the main thread and the florists threads is producer consumer problem , the main thread read the requests from the file and delegate the request to the closest florist by pushing the request into it's queue if there is a space in the queue or the main thread waits until the queue is available to be push with the request then push the request and reads another request and so on , after all requests was delegated the main thread then waits until all requests were processed by the florists and then exit ,
each florist thread, in infinite loop first checks if the queue associated with it's id is empty or not , in case if the queue was not empty it then start to process the request and calculating the preparation time and etc , or if the queue was empty so it is waiting for the main thread to provide it with a request or as mentioned before the main thread update the value of (cancel) variable which means that all florists can break the infinite loop .
the relation between the florist threads is as synchronization barrier we saw in the lecture slides which is used to print closing shop messages as in the example given in the assignment .