GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

SYSTEM PROGRAMMING
CSE 344 - 2020 SPRING

HW5 REPORT

SILA BENGİSU YÜKSEKLİ
1801042877

➢ *Explanation of the way of work and the aim of the program*

User can compile by typing make on terminal. User should give command line argument like “-i filePath” , then it can be run by writing “./floristApp -i filePath” on terminal. This argument refers to the following:

- i: input file. It contains an uncertain number of florists informations which are name of the florist, flower types that they sell, its coordinate and speed. Also, it contains client informations which are the coordinates and flower types that they request.

This program simulates the delivery of an uncertain number of flower orders to the clients by florists. In this scenario, there is as much thread as the number of florists and a central thread that transmits the orders to the florists.

Hence, there is one source files which are floristApp.c with many threads. Helper.c is for helper functions like printing messages, parsing command lines, etc.

➢ *Process (floristApp.c)*

This problem is similar to the classic “producer-consumer” example. Many threads were used to solve this problem. The main thread refers to someone who transmits orders and remaining threads refer to the florists.

Each florist works on the same function with different parameters. When threads are created, each of them takes a parameter which is void pointer. This pointer has a size of one and is held in heap. The element of the array corresponds to the number of the florist(1,2,3…). In this way, every florist knows which flower will be delivered to which customer.

In addition, every florist has its own request queue. In order for orders to be delivered as soon as possible, even if the florist is busy, the main thread holds the orders in the request queue.

➢ *Representing of Request Queue, Florist and Statistic*

All of them are represented by a struct to make things easier.
Florist struct is shown in below :

```
struct florist
{
        char* name;
        float x;
        float y;
        float speed;
        char** flowerTypes;
        int size;
};
```

-> Each florist has its name, location (x, y coordinate), speed of delivery, flower kinds and total number of flower kinds that it sells informations.
I made an array of florist struct. When I parse input file, all the informations are initialized and so it's easier to access information.

Queue struct is shown in below :

```
struct Queue
{
        char** request;
        int offset;
        float* distance;
        char** client_no;
};
```

-> Each queue has its request array, offset information to indicate index when something need to be added into request array, distance between client and florist and client number information to indicate which client requests flower.
I make an array of queue struct. When central thread parses clients, it saves information into this array and florists can acceess this informations by this way.

Statistic struct is shown in below :

```
struct Statistic
{
        float total_time;
        int totalOrder;
};
```

-> Each statistic has total number of flower order and total time to deliver all orders, informations.
Every florist thread returns their statistic informations by making an array of this struct and they return it so that central thread can print datas on the screen.

❖ *Central Thread*

It continues to work until it delivers all the flower orders to florists.Then, it is waiting for all the florists to finish and it terminates after the datas it receives from them is printed on screen.

By turning in a loop, it sequentially saves client’s location and orders into request Queue as shown in below. By doing it, it gets the lock because they are common variables, other threads can change them. It finds the client's distance from the florist. Method Chebyshev Distance is used to find the distance. First, it looks at which florist is selling that flower by using “***if (strcmp(florists[i].flowerTypes[j],tmp3) == 0”***. Then it finds the nearest florist and directs the order to that florist.

```
pthread_mutex_lock(&mutexes[no]);
strcpy(requests[no].request[requests[no].offset],tmp3);
requests[no].distance[requests[no].offset] =  min;
strcpy(requests[no].client_no[requests[no].offset],clientNum);
requests[no].offset +=  1;
++Count[no];
pthread_cond_signal(&full[no]);
pthread_mutex_unlock(&mutexes[no]);
...
```

❖ *Florists*

In my algorithm, all florists are waiting to take orders. This is controlled by the Count variable. If Count is equal to 0, the thread is waiting by using “***pthread_cond_wait(&full[index],&mutexes[index]) “*** in a while loop until the order comes up and the value of Count is increased. Count variable is a array and its size is the number of florists.

Florists reach their own variables with the index value, which is the parameter given to the function. Let's say that the florist who works at the moment is the first florist. Then, using the index value that comes as a parameter, Count [0] is checked to see if any order has arrived.

The delivery time of the order is calculated from x = v * t using the florist's speed information and distance from the customer. The preparation time of the flower is determined by a random number between 1 and 250. The total elapsed time is simulated using usleep. Distance information is given by central thread and added into request array. After the order is delivered, Count[index] variable is decreased.

When central thread is done and florists delivers all the orders, thread exits the loop and return its statistics which shows how many orders are delivered and how much time is spend.

❖ *Exit condition of florists and How it occurs*

After the main thread directs all orders to the appropriate florists, it expects the florists to deliver all the orders in a loop using a conditional variable which is isFinished as shown in below :

```
for (int i = 0; i < threadCount; ++i)
{
        pthread_mutex_lock(&mutexes[i]);
        while(order[i] != requests[i].offset) {
                pthread_cond_wait(&isFinished[i],&mutexes[i]);
        }
        pthread_mutex_unlock(&mutexes[i]);
}
```

When the florists delivers a new order, it sends a signal to the main thread by using "***pthread_cond_signal(&isFinished[index])"*** every time so that main thread can control the condition which is written in loop.

When all the florists are done, central thread checks one more time if they are done. In this time all florists get stuck in the while loop shown in below because no more orders are received.

```
while(Count[index] == 0) {
        pthread_cond_wait(&full[index],&mutexes[index]);
}
```

After that central thread informs all florists that there are no more orders. Florists wait for a signal to continue. It should come out of the loop. So, central thread change the common variable flag as one, increase the Count variable and send a signal to them as shown in below :

```
for (int i = 0; i < threadCount; ++i)
{
        pthread_mutex_lock(&mutexes[i]);
        flag[i] = 1;
        ++Count[i];
        pthread_cond_signal(&full[i]);
        pthread_mutex_unlock(&mutexes[i]);
}
```

By this way, florists understands that there are no more inputs. It checks if the flag is one. If it is one, it decreased the value of Count. Because normally Count is empty. It is increased for prevent stucking. After that it checks if Count is zero and flag is one. According to it, it exits by returning statistics.

Checking flag is important. Because if the main thread will continue to order, Count will increase. In that case, it should not exit.

➢ *Synchronization between main thread and florists*

All Synchronization problems are solved by using conditional variables and mutex. Different mutex and conditional variables were used for each florist. So I made them as an array. The variables used are as follows:

- **pthread_mutex_t* mutexes :** There are as many mutex as the number of florists in the array. İt is used for protect common variables. Because if one thread tries to reach it when another thread changes it, threads may get wrong results and cause deadlock.

- **pthread_cond_t* full :** There are as many full conditional variable as the number of florists in the array. This variable is used to check if the request queue is full. Florists check every time, if the queue has at least one element in it. If it is empty florist waits until some order is transmitted by using this variable.

- **pthread_cond_t* isFinished :** There are as many isFinished conditional variable as the number of florists in the array. It is used for waiting all florists. When central thread has finished its job, it waits for all florists to delivers all orders. When florist delivers a order, it sends a signal every time to check there is someone waiting for it. When the last order is delivered, it sends a signal again. Then main thread gets the lock and checks one more if the current florists is done. Then it continues.