these are my mutexes:

```
thread_mutex_t eights[10000];  // for putting a lock on slots alloted
pthread_mutex_t remcomp[10000];   // for comapany data
pthread_mutex_t waiting_students; //for counting number of waiting students
pthread_mutex_t comp_students;    // number of students vaccinated and tested
positive
```

my structures to store data about student , vaccination zone and company are:

```
typedef struct s
{
    int id;
    int count;              // for indicating student round
    float recent_vacc_prob; // recently vacinated vaccine probability
} s;
typedef struct zone
{

    int total; //for counting total number of vaccines
    int id;
    int eight; //number of slots alloted

} zone;
typedef struct company
{
    int id;
    float prob;
    int batches;//number of batches it  has currently
    int curbatch; // number of vaccines in currently prepared batches
} comp;
comp *input2[10000];
zone *input1[10000];
s *input[10000];
```

my global variables:

```
float prob[10000]; //probaility of vaccines
int waiting;// for counting number of waiting students
int completed; // for counting number of suucessfully or unsuccessfully completed
their vaccination
int student_zone[10000][10000];    //student_zone [x][y]] is one  when a student x
is alloted to vaccination zone  y 0 else
int remcheck[10000];               //for storing how many vaccines a company
deleiverd - how many of them have got used
int zone_comp_count[10000][10000]; //for track of how many vaccines a zone has
recieved from a company
```

pharmacamp thread function:

1)it checks for number of remaining students and starts producing if number of remaining students to be vaccinated is greater than zero

2)it decides some r batches of cnt size size and then prepares in certain amount of time after that it increases number of batches it currently has and total number of vaccines it produced.

```
        pthread_mutex_lock(&remcomp[inputs->id]);
        inputs->curbatch = cnt;
        inputs->batches = 1 + (rand() % 4);
        remcheck[inputs->id] = cnt * (inputs->batches);
        pthread_mutex_unlock(&remcomp[inputs->id]);
```

3) it then does busywaiting till its vaccines are completed or number of remaining students to be vaccinated is zero

```
         while (remcheck[inputs->id] > 0 && (o - completed) > 0)
        {
        }
```

4) after busy waiting it checks the reasono for which it stopped busy waiting.if due to vaccines completion then it starts vaccine production if due to no remaining students to be vaccinated it returns

```
         if (remcheck[inputs->id] == 0)
        {
            printf(CYAN);
            printf("all the vaccines prepared by pharmacetical company %d are
emptied resuming production now\n\n", inputs->id);
            printf(RESET);
        }
        if (o - completed <= 0)
            break;
```

vaccizone thread function:

1)it checks the number of remaining students to be vaccinated then enters in to loop else breaks

2)then it iterates over all companies untill till it gets vaccines or number of remaing students to be vaccinated is zero.it will vaccines from a company if avvailable then it decreases the numeber of batches of that company and breaks from these loops.

```
         while (o - completed > 0 && inputs->total == 0)
        {
            for (int i = 1; i <= n; i++)
            {
                if (input2[i]->batches != 0)
                {

                    inputs->total = input2[i]->curbatch;
                    zone_comp_count[inputs->id][i] = input2[i]->curbatch;
                    pthread_mutex_lock(&remcomp[i]);
                    input2[i]->batches--;

                    pthread_mutex_unlock(&remcomp[i]);
                    printf(GREEN);
                    printf("pharmacetical company %d is delivering a vaccine batch
to vaccination zone %d which has success probabilty %f\n\n", i, inputs->id,
input2[i]->prob);
                    printf(RESET);
                    break;
                }
            }
        }
```

3)it nows allotes slots less than or equal to eight by applying a lock and unlocks it after changing the number of slots.

```
         pthread_mutex_lock(&eights[inputs->id]);

        if (inputs->total <= 8)
        {

            inputs->eight = inputs->total;
        }
        else
        {
            inputs->eight = 8;
        }


        printf(CYAN);
        printf("vaccination zone %d is ready to vaccinate with %d slots\n\n",
inputs->id, inputs->eight);
        printf(RESET);

        int temp = inputs->eight; // number of slots alloted


        pthread_mutex_unlock(&eights[inputs->id]);
```

4)it will then does busy waiting untill waiting students becomes zero or number of slots becomes . after busy waiting by the value of remaing it can know number of slots it has alloted exactly.so that we can decrease that value from total vaccines as those many vaccines will be used for this phase.

```
        pthread_mutex_unlock(&eights[inputs->id]);
        while (waiting != 0 && inputs->eight != 0)
        {

        }

        pthread_mutex_lock(&eights[inputs->id]);
        int temp2 = temp - inputs->eight; //number of vaccines willbe used in a
vaccination phase
        inputs->total -= (temp2);
        inputs->eight = 0;
        pthread_mutex_unlock(&eights[inputs->id]);
```

5)now this is in the vaccination phase . now this will check the every student and sees who got slots for this zone then it iterates over companies then finds a company whose vaccines are with it .then it will vaccinate student and decreses the vaccine count for company and itself

```
         for (int i = 1; i <= o; i++)
        {
            if (student_zone[i][inputs->id] == 1)
            {
                for (int j = ind; j <= n; j++)
                {
                    if (zone_comp_count[inputs->id][j] > 0)
                    {
                        pthread_mutex_lock(&remcomp[inputs->id]);
                        remcheck[inputs->id]--;
```

```
                        pthread_mutex_unlock(&remcomp[inputs->id]);
                        zone_comp_count[inputs->id][i]--;
                        pthread_mutex_lock(&color);
                        printf(GREEN);
                        printf("student %d on vaccination zone %d has been
vaccinated which has success probaiblity %f\n\n", i, inputs->id, prob[j]);
                        printf(RESET);
                        input[i]->recent_vacc_prob = prob[j];
                        student_zone[i][inputs->id] = 0;
                        pthread_mutex_unlock(&color);
                        sleep(1);
                        ind = j;
                        continue;
                    }
                }
            }
        }
```

6)after that it completes vaccination phase then checks number of reaming students to know whether to continue or not

student thread function:

1)student will first check whether his round is less than 3 or not if it is he will increment wait otherwise he exits;

2)then untill he gets vaccinated he iterate over zones and knows which zone is currently available then he will decrement waiting if a zone is available and does busy waiting untill the zone will changes the flag .after changing the flag will be send back to second round based on probbility otherwise he will leave leave the zone

```
               while (ok)
        {
            for (int i = 1; i <= m; i++)
            {
                pthread_mutex_lock(&eights[i]);


                if (input1[i]->eight > 0)
                {


                    input1[i]->eight--;
                    pthread_mutex_lock(&color);
                    printf(MAGENTA);
                    printf("Student %d assigned a slot on vaccination zone %d and
waiting to be vaccinated\n\n", inputs->id, i);
                    printf(RESET);
                    pthread_mutex_unlock(&color);
                    pthread_mutex_unlock(&eights[i]);
                    pthread_mutex_lock(&waiting_students);
                    waiting--;
                    pthread_mutex_unlock(&waiting_students);
                    student_zone[inputs->id][i] = 1;
```

```
                    while (student_zone[inputs->id][i])
                    {

                    }
                    ok = 0;
                    sleep(4);
                    float p = inputs->recent_vacc_prob;
                    int round = p * 100;
                    int decider = 1 + rand() % 100;
                    if (decider < round)
                    {
                        pthread_mutex_lock(&color);
                        printf(GREEN);
                        printf("student %d has tested positive for antibodies\n\n",
inputs->id);
                        printf(RESET);
                        inputs->count = 5;
                        pthread_mutex_unlock(&color);
                        pthread_mutex_lock(&comp_students);
                        completed++;
                        pthread_mutex_unlock(&comp_students);
                    }
                    else
                    {
                        pthread_mutex_lock(&color);
                        printf(RED);
                        printf("student %d has tested negative for antibodies\n\n",
inputs->id);
                        printf(RESET);
                        pthread_mutex_unlock(&color);
                        inputs->count += 1;
                    }
                    //pthread_mutex_unlock(&queue[i]);
                }
                else
                {

                    pthread_mutex_unlock(&eights[i]);
                }

after all threads got completed we print simulation is over
```