

Report for Q2

Operating Systems

Kushagra Agarwal

2018113012

I used the following logic to solve the problem:

1. Each pharma company is a thread
2. Each vaccination zone is a thread
3. Each student is a thread

Now first of all I initialize threads array for these 3 things. Then I created all the pharma company threads first. The pharma company threads on creation enter this function called `produce_vaccines()`. This function makes each of the **n** pharma companies wait for some time **sleeptime** and then these pharma companies can start producing **r** batches of vaccines with **p** in each batch. Now each pharma company thread enters an infinite loop of waiting: producing: calling `vaccine_ready`: restart.

```
void* produce_vaccines(void * input)
{
    indx * in=(indx*) input;
    int ind=in->inx;

    int p = rand()%11 + 10; // Between 10-20 ( size of batch )
    int r = rand()%5 + 1; // Between 1-5 ( num of batches )

    int sleeptime = rand()%4 + 2; // Between 2-5

    int num=r;
    int i;

    while(1)
    {
        printf("Pharmaceutical Company %d is preparing %d batches of vaccines which have success probability %d\n", ind, r, pharma_infos[ind-1].prob);
        sleep(sleeptime);
        printf("Pharmaceutical Company %d has prepared %d batches of vaccines which have success probability %d\n", ind, r, pharma_infos[ind-1].prob);
        num=r;
        vaccine_ready(ind,p,num);
        printf("All the vaccines prepared by Pharmaceutical Company %d are emptied. Resuming production now\n", ind);
    }
}
```

This `vaccine_ready` function is responsible for allowing any of the zones to get a batch of prepared vaccines. It loops with a condition ($r > 0$), i.e, the number of batches should not be zero, cause as soon as the number of batches becomes zero the code should be taken

back to produce_vaccines(). If the number of batches is non_zero, the function loops over all zones and tries to find out if any of them are free. If yes, it drops a batch of vaccine at that zone and reduces the number of batches by one. Also, a structure was used for each zone, which stored the pharma company from which it got the vaccine. The code makes sure an update is made there securely using mutex3. The mutex1 was used for pharma threads.

```
void vaccine_ready(int ind,int p,int r)
{
    int i;
    while(r>0)
    {
        pthread_mutex_lock(&mutex1);

        //Entering critical section

        for(i=0;i<zonenum;i++)
        {
            if(zone[i]==0)
            {
                zone[i] += p;
                r--;

                printf("Pharmaceutical Company %d is delivering a vaccine ba

                pthread_mutex_lock(&mutex3);
                center_infos[i].which_pharma = ind;
                pthread_mutex_unlock(&mutex3);

                if(r<=0)
                    break;
            }
        }

        pthread_mutex_unlock(&mutex1);
    }
}
```

For vaccination zones, the threads were created in main() and these threads were sent to book_zones function, which has the job of making these zones create slots and move into vaccination phases. Again looping over infinitely we see if our zone has

remaining vaccines. If the vaccine count is greater than zero then we allot some slots to the center which can be used by the students to get vaccinated. Once a slot is created the function vaccination_phase is called.

```
void* book_zones(void * input)
{
    indx * in=(indx*) input;
    int ind=in->inx;

    int range;
    int flag = 0;
    int i,slot;

    while(1)
    {
        flag = 0;
        while(flag!=1)
        {
            pthread_mutex_lock(&mutex1);
            pthread_mutex_lock(&mutex2);

            if(zone[ind-1]>0) // If we find a Zone with vaccines
            {
                range = zone[ind-1]%8 + 1;

                flag = 1;
                slot = rand()%range+1;
                zone_slot[ind-1] += slot;
            }

            if(flag==1)
            {
                printf("Vaccination Zone %d is ready to vaccinate with %d slots\n", ind, zone_slot[ind-1]);
            }

            pthread_mutex_unlock(&mutex2);
            pthread_mutex_unlock(&mutex1);
        }

        vaccination_phase(ind);
    }
}
```

The function vaccination_phase has the job of returning when the slots of a certain phase are over. It then goes back to the function book_zones which then again allots slots the zone. So it is a simple implementation using an array zone_slot for all zones which

stores the number of available slots. If the value at our index of interest is 0 then we flag the loop and come out of it.

```
void vaccination_phase(int ind)
{
    printf("Vaccination Zone %d entering Vaccination Phase\n", ind);

    int flag = 0;

    while(flag != 1)
    {
        pthread_mutex_lock(&mutex2);

        // Entering Critical Section now

        if(zone_slot[ind-1]==0) // Do not let this zone get new students til
        {
            flag=1;
            printf("Vaccination Zone %d leaving Vaccination Phase\n", ind);
        }

        pthread_mutex_unlock(&mutex2);
    }
}
```

Next, we move to the student's logic implementation. Student threads are created and they are sent to the student() function described below. This function simply adds 1 to the visit number for each student and calls the wait_for_slot() function.

```
void* student(void* input)
{
    indx * in=(indx*) input;
    int ind=in->inx,z=0;

    int student_sleep = rand() % 5;
    sleep(student_sleep);

    pthread_mutex_lock(&mutex4);
    student_visits[ind]++;
    printf("Student %d has arrived for his round %d of Vaccination\n",ind, student_visits[i]);
    pthread_mutex_unlock(&mutex4);

    wait_for_slot(ind);
}
```

The wait_for_slot() function is the main logic of this code.

```
void wait_for_slot(int ind)
{
    int flag = 0;
    int i;

    printf("Student %d is waiting to be allocated a slot on a Vaccination Zone\n", ind);

    while(flag!=1)
    {
        pthread_mutex_lock(&mutex1);
        pthread_mutex_lock(&mutex2);

        //Entering Critical Section

        for(i = 0; i < zonenum; i++)
        {
            if(zone_slot[i]>0)
            {
                zone_slot[i]--;
                flag=1;

                printf("Student %d assigned a slot on the Vaccination Zone %d and waiting to be vaccinated\n", ind, i+1);
                sleep(0.1);

                int probability;

                pthread_mutex_lock(&mutex3);
                probability = pharma_infos[center_infos[i].which_pharma -1].prob;
                pthread_mutex_unlock(&mutex3);

                printf("Student %d on Vaccination Zone %d has been vaccinated which has success probability %d\n", ind, i+1, probability);
                zone[i]--;


                pthread_mutex_lock(&mutex4);
                int visits_stud = student_visits[ind];
                pthread_mutex_unlock(&mutex4);

                int chance = rand()%100 + 1;
                if(chance <= probability)
                {
                    printf("Student %d has tested positive for antibodies\n", ind);
                    break;
                }
            }
            else if(visits_stud == 3)
            {
                printf("Student %d has tested negative for antibodies but completed 3rd attempt, hence being sent back\n", ind);
                break;
            }
            else
            {
                printf("Student %d has tested negative for antibodies\n", ind);
                flag = 0;

                pthread_mutex_lock(&mutex4);
                visits_stud = ++student_visits[ind];
                pthread_mutex_unlock(&mutex4);

                printf("Student %d has arrived for his round %d of Vaccination\n", ind, visits_stud);
            }
        }

        pthread_mutex_unlock(&mutex2);
        pthread_mutex_unlock(&mutex1);
    }
}
```



Basically, when a student is free and is waiting to get a zone, this function loops over all zones and verifies if any zone has a slot available. If yes then it reduces 1 from the available slots in that zone and flags it as 1. Now we want to find out if the student got antibodies. So for this, I used a struct to store the values of probabilities for each pharma company and as each zone is associated with the pharma company it got vaccines from, I combined the two to get the probability of student getting positive in the antibody test. I created a random chance generator which if lesser than or equal to the probability (out of 100) ends up making the student have the antibodies and vice versa. I used mutex locks carefully to make sure no deadlock situations can take place. Now three cases arise. If the student got antibodies by the vaccination, then we let the student thread gets destroyed. If this was not the case and the student had appeared for vaccination less than 3 times then we ask him to come again. If neither of the above 2 was true, i.e the student had come for the vaccination 3 times and yet could not get antibodies then we ask him to go home. We print the appropriate message and end the function implementation.
