BACK TO COLLEGE

STRUCTURE OF IMPLEMENTATION

In the given situation there are n pharmaceutical companies, m vaccination zones and o students.

Every pharmaceutical company is treated as thread which manufactures the vaccines. At a time, the pharmacy can create any number of batches ranging from 1 to 5 and each batch may contain vaccines ranging from 10 to 20. I have implemented this using rand function and doing the modulus to get the number in the required range. The pharmaceutical company takes 2-5 seconds to manufacture the vaccines. I am using sleep to simulate the time of manufacture.

Each vaccine zone is also a thread, which goes through every pharmaceutical company (stored in an array) and picks the batches of vaccines from company. Then the zone declares some random number of slots, then the zone waits until either the students picks the all the slots of vaccine zone or the waiting students become zero. Now the zone enters vaccination phase. After vaccination the students (who has taken the slots from that zone) the zone thread whether there are vaccines left or not. If yes, then zone again declares the slots. If not the vaccines once again iterate through the pharmacy array and picks up vaccines.

Every Student is a thread .The student comes to campus late (as usually 😃). To simulate this student sleep for some random time. The student arrives and checks if there are any slots for him. If he finds a slot, he takes up the slot and waits until the zone (from where he picked up the slot) enters the vaccination phase. After he got vaccinated, he gets checked for antibodies, if he tests negative, he again goes through the process. If a person gets negative for antibody tests for three times, he will be sent home. If a person tests positive for antibody test, he enters the college (the thread gets exited).

PHARMACEUTICAL COMPANY

The pharmaceutical company gets some random number of batches between 1 to 5 using 1 + rand()%5.The company gets random number of vaccines for each batch between 10 , 20(both inclusive) using 10 + rand()%11. It sleeps for some random time to simulate manufacture time. Then it enters the pharmacy wait function and keeps on waie by the zone doesn't happen at the same time. it waits(busy wait) until all the batches produced by the company are used.After the batches get completely used up the company resumes production. When we are accessing the number of batches in the company we employ a lock so that access by the company and update by zones is syncronized.

```
// Function to let the company thread wait until all the batches are
consumed
void pharmawait(struct pharma * reqcompany, int batches)
{

        while(1)
        {
                pthread_mutex_lock(&(reqcompany->pharmalock)); // Zone
thread will be modifying the value of " num batches "
                //so to synchronize the process lock is set up
```

```
                if(reqcompany->numcompbatch <= 0) // check if there are any
batches left
                {
                        printf("All the vaccines prepared by Pharmaceutical
Company %d are emptied. Resuming production now.\n",reqcompany->pharmaid);
                        break;
                }
                pthread_mutex_unlock(&(reqcompany->pharmalock));
        }
        pthread_mutex_unlock(&(reqcompany->pharmalock));

}
// Pharmacy thread enters here
void * pharmafun(void * args)
{
        struct pharma * company = (struct pharma *)args;

        while(1)
        {
                int w  =  2+ rand()%4; //preparation time

                int batches =  1 + rand()%5; //Get the number of batches to
prepare

                int vaccines = 10 + rand()%11; // Number of vaccines to
prepare

                sleep(w);  // To simulate the manufacture time

                pthread_mutex_lock(&(company->pharmalock)); // lock the
company since no zone should
                                                        // access
before complete filling
                company->numbatch = batches;

                company->numcompbatch = batches;

                company->numvaccines = vaccines;
//              printf("pharacompany %d prepared %d batches each of %d
vaccines\n",company->pharmaid, batches , vaccines);
                printf("Pharmaceutical Company %d has prepared %d batches
of vaccines which have success probability %0.3f . \n Waiting for all the
vaccines to be used to resume production\n",company->pharmaid, vaccines,
probarr[company->pharmaid]);
                pthread_mutex_unlock(&(company->pharmalock)); // Unlock it.

                pharmawait(company,batches); // Wait until all the batches
are consumed
        }

        return NULL;
}
```

I am using the below structure to store the information about the pharmacy companies. I am storing it in an array to let others threads (Zones , students)use it.

```
struct pharma
{
        int pharmaid;// Id to access

        int numbatch; // Number of batches prepared

        int numcompbatch; // Number of batches used up by the zones

        int numvaccines; // Nunber of vaccines in each batch

        int waittime; // Time of manufacture

        pthread_t pid;

        pthread_mutex_t pharmalock;
};
```

VACCINATION ZONES

The thread uses pharmarr[i]->pharmalock , vaczone[i]->zonelock and waitstd mutexes to lock the critical zones of this section. Picking up vaccine batches from companies is a critical section . No two zones should accesss the same company simultaneously. Some part of declaring slots section is critical section code , since the student thread updates the zonearr[i]->slots and uses the zonearr[i]->invac variable which belongs to the zone thread. Hence to avoid simulatenous access/update we employ locks. After declaring slots the zone thread waits until either the slots are filled or waiting students becomes zero. For the student to register the vaccination zone should be not in vaccination phase and should have atleast one slot left.

```
// Thread representing the zones enters into this function
void * zonefun(void * args)
{
        struct zone * vaczone = (struct zone *)args;
        vaczone->invac = 0; //  invac = 0 says the zone is not in
vaccination phase

        struct pharma * pcompany; // to iterate through the pharma
companies
        while(1)
        {
                while(1)
                {
                        int find = 0;
                        for(int i=0;i<numpharma;i++)
```

```
                        {

                                pcompany = pharmaarr[i];
                                pthread_mutex_lock(&(pcompany-
>pharmalock)); // The zone shouldn't take batches until every aspect is
updated and

// No two zones should access the same company simultaneously

                                if(pcompany->numbatch>0)
                                {
                                        printf("Pharmaceutical Company %d
is delivering a vaccine batch to Vaccination Zone %d which has success
probability %0.3f \n", pcompany->pharmaid,  vaczone->zoneid,
probarr[pcompany->pharmaid]);
                                        find = 1;
                                        pcompany->numbatch--;  //
decreasing the number of batches
                                        vaczone->heldvaccines = pcompany-
>numvaccines; // Number of vaccines taken
                                        vaczone->frompharma = pcompany-
>pharmaid; // To know from which pharmacy the vaccines are taken
//                                      printf("zone %d took %d vaccines
from company %d and remainaing batches are %d\n", vaczone->zoneid,
pcompany->numvaccines, pcompany->pharmaid,pcompany->numbatch);

                                        printf("Pharmaceutical Company %d
has delivered vaccines to Vaccination zone %d, resuming vaccinations
now\n", pcompany->pharmaid,  vaczone->zoneid);
                                        pthread_mutex_unlock(&(pcompany-
>pharmalock));
                                        break;
                                }
                                else
                                {
                                        pthread_mutex_unlock(&(pcompany-
>pharmalock));
                                }

                        }
                        if(find == 1)  // if the zone finds a batch of
vaccines
                        {
                                break;
                        }
                }

                while(1)
                {
                        if(vaczone->invac==0)
                        {
                                pthread_mutex_lock(&waitstd); // waiting
students are updated student thread
```

```
// we shouldnot access a thread when we another thread is changing it
                                int ws = waitingstd ;
                                pthread_mutex_unlock(&waitstd);
                                if(ws>0) // allocate the slots if and only
if when there are waiting students
                                {
                                pthread_mutex_lock(&(vaczone->zonelock));

                                // Number of slots available are decreased
student and student should take slots until evrything
                                // regarding the slots get updated.

                                pthread_mutex_lock(&waitstd);

                                vaczone->slots =waitingstd;

                                pthread_mutex_unlock(&waitstd);

                                int numslots = 1 + rand()%8; // Generating
random number of slots
                                if(vaczone->slots > numslots)
                                {
                                        vaczone->slots =  numslots;
                                }

                                // Number of slots should not be greater
than the number of vaccines held.
                                if(vaczone->slots > vaczone->heldvaccines)
                                {
                                        vaczone->slots =  vaczone-
>heldvaccines;
                                }

                                if(vaczone->slots > 0)
                                {
                                        printf(" Vaccination zone %d is
ready to vaccinate with  %d slots \n", vaczone->zoneid, vaczone->slots);
                                }

                                pthread_mutex_unlock(&(vaczone->zonelock));

                                //Enter into vaccination zone only if the
slots declared are non - zero
                                if(vaczone->slots>0)
                                {
                                        vaccinationzone(vaczone);
                                }
                                /*
                                   if(vaczone->heldvaccines <= 0)
                                   {
                                   pthread_mutex_lock(&(pcompany-
>pharmalock));
                                   pcompany->numcompbatch--;
                                   pthread_mutex_unlock(&(pcompany-
```

```
>pharmalock));

                                        printf("",vaczone->zoneid);

                                        break;
                                        }*/
                                }
                        }
                        else
                        {
                                pthread_mutex_lock(&(vaczone->zonelock));

                                if(vaczone->compstd <= 0)
                                {
                                        vaczone->invac = 0;
                                        vaczone->slots = 0;
//                                      printf("zone  %d
completed\n",vaczone->zoneid);
                                }
                                pthread_mutex_unlock(&(vaczone->zonelock));

                                if(vaczone->heldvaccines <= 0)
                                {
                                        pthread_mutex_lock(&(pcompany-
>pharmalock));
                                        pcompany->numcompbatch--;
                                        pthread_mutex_unlock(&(pcompany-
>pharmalock));

                                        printf("Vaccination zone %d has run
out of vaccines\n",vaczone->zoneid);

                                        break;
                                }

                        }

                }
        }
        return NULL;
}
```

The zone thread busy waits until students to register all the slots of that particular zone i.e vacinzone->compstd >= vacinzone->slots (Compstd represents the number of students registered)Or the waiting students number becomes zero i.e ws <= 0 && (vacinzone->compstd > 0 (Here ws represents the number of waiting students). After any of the above criteria is satisfied the thread changes the status to zone to represnt that the zone is in vaccination phase . It sets invac variable to 1. It decreases the number of vaccines held in the zone. After all the registered students gets vaccinated , the zone checks whether any

vaccines are left or not , if all the vaccines are used up the zone again looks for batches in company. If some vaccines are still left then the zone again declare the slots.

```
// Zone thread busy waits here.

void * vaccinationzone(struct zone * vacinzone)
{ int has = 1;

        if(vacinzone->slots<=0)
        {
//              while(1)
//                      printf("yes\n\n\n");
                return NULL;
        }

        while(1)
        {
                pthread_mutex_lock(&waitstd);
                int ws = waitingstd;
                pthread_mutex_unlock(&(waitstd));

                pthread_mutex_lock(&(vacinzone->zonelock));     // The
invac are accessed by the students students should not access interleaving
an update
                if( ws <= 0 && (vacinzone->compstd > 0) ) // Enters into
vaccination zone since there are no students waiting
                {
                        printf("vaccination zone %d entering vaccination
phase\n", vacinzone->zoneid);
                        vacinzone->invac = 1;
                        vacinzone->heldvaccines -= vacinzone->compstd;
                        // decrease the held vaccines
                        //by the number of students taken up the slots in
that zone
                        pthread_mutex_unlock(&(vacinzone->zonelock));
                        break;
                }
                pthread_mutex_unlock(&(vacinzone->zonelock));


                pthread_mutex_lock(&(vacinzone->zonelock));
                if(vacinzone->compstd >= vacinzone->slots)
                {

                        printf("vaccination zone %d entering vaccination
phase\n", vacinzone->zoneid);
                        vacinzone->invac = 1;

//                      printf("vaccination zome of zone id %d aagamaha\n",
vacinzone->zoneid);
                        vacinzone->heldvaccines -= vacinzone->compstd;  //
decrease the held vaccines
```

```
//by the number of students taken up the slots in that zone

                        //              printf("Slots completed\n");
                        pthread_mutex_unlock(&(vacinzone->zonelock));
                        break;
                }
                pthread_mutex_unlock(&(vacinzone->zonelock));

        }
}
```

I am using the below structure to store the information about the vaccination zones. I am storing it in an array to let others threads (students)use it.

```
struct zone
{
        int zoneid; // Id to access

        int heldvaccines; // Number of vaccines in the zone

        int slots; // NUmber of slots declared

        int frompharma; // The pharmacy company id from which the vaccines
are taken

        int invac; // whether in vaccination phase or not

        int compstd;

        pthread_t zid;

        pthread_mutex_t zonelock;
};
```

STUDENT

Each student is a thread . Each student enters the "studentfun" function . The student thread sleep initially for some random time to simulate random entrance time . A student thread runs until he has tested positive for antibody test or he has tested negative three times for vaccine test. If he was tested negative three times he will be sent home. When a student arrives he increment the value of waiting students after he picks up a slot he decrements it. For the student to register the vaccination zone should be not in vaccination phase and should have atleast one slot left i.e (zoneofvac->compstd < zoneofvac->slots) && (zoneofvac->invac == 0) .We are using a global mutex -Waitstd not to let two threads change the value of waiting studetns simultaneiously.

After the student picks up a slot he goes and busy waits until the zone (from where he picked up the slot) enters into vaccination phase (zoneofvac->invac == 1). When the student access a vaccination zone data represented by zonearr[i] I lock that i th zone with zonearr[i]->zone_lock which is a mutex lock for concurrency so that no other entity accesses it at the same time creating issues. Once the student enters he stops busy wait and get vaccinated. He will be tested for antibodies. If the testresult is positive he exits(thread exits) else he increments the waiting students value and again goes back for another slot. I have kept track of number of vaccinations done for a student in the student->status attribute . If the attribute is less than or equal to -3 he will be sent to home .

```
// Every student thread  enters here
void * studentfun(void * args)
{
        int enter = 0;
        struct student * shishya = (struct student * )args;
        status[shishya->stid] = 0; // He wasn't tested yet
        int waitsfor = 1 + rand()%4; //To simulate the situation that
students enter in random times
        sleep(waitsfor);
        pthread_mutex_lock(&waitstd);
        waitingstd++;
        pthread_mutex_unlock(&waitstd);


        int i = 0;
        int got = 0; // variable to know whether he has found a slot or
not.
        int have  = -3;
        while( status[shishya->stid] > -3 && status[shishya->stid] <= 0 )
        {
                struct zone * zoneofvac;
                int ctt = 0;
                while(got == 0)
                {

                        if(ctt == 0)
                        {
                        if(status[shishya->stid]==0)
                        {
                                printf("Student %d has arrived for his 1st
round of vaccination\n ",shishya->stid);
                        }
                        if(status[shishya->stid]==-1)
                        {
                                printf("Student %d has arrived for his 2nd
round of vaccination\n ",shishya->stid);
                        }
                        if(status[shishya->stid]==-2)
                        {
                                printf("Student %d has arrived for his 3rd
round of vaccination\n ",shishya->stid);
```

```
                        }
                        }

                        got =0;
                        i  = (i+1)%numzones; // to iterate through zones.
                        if(ctt == 0)
                        printf("Student %d is waiting to be allocated a
slot on a Vaccination Zone\n", shishya->stid);
                        zoneofvac = zonearr[i];
                        pthread_mutex_lock(&(zoneofvac->zonelock)); //
since a zone should not be accessed by two student threads simultaneously

                        if( (zoneofvac->compstd < zoneofvac->slots) &&
(zoneofvac->invac == 0) ) // For the student to register the vaccination
zone should be not in vaccination phase and should have atleast one slot
left
                        {

                                got = 1; // To break from the loop
                                have =  zoneofvac->slots;
                                zoneofvac->compstd++; // Increase the
number of registered students of s zone
                                pthread_mutex_lock(&waitstd); // Global
lock for waiting students
                                waitingstd--;
                                pthread_mutex_unlock(&waitstd);
                                printf("Student %d assigned a slot on the
Vaccination Zone  %d and waiting to be vaccinated\n ",shishya->stid ,
zoneofvac->zoneid);
                        }

                        pthread_mutex_unlock(&(zoneofvac->zonelock));
                /*      if(got)
                        {

                                pthread_mutex_lock(&waitstd);
                                waitingstd--;
                                pthread_mutex_unlock(&waitstd);
                        }*/
                        ctt++;
                }
                ctt = 0;
                got = 0;

                while(1)
                {
                        pthread_mutex_lock(&(zoneofvac->zonelock));
                        int zoc = zoneofvac->invac;
                        pthread_mutex_unlock(&(zoneofvac->zonelock));
                        //After the student picks up a slot he goes and
busy waits until the zone (from where he picked up the slot ) enters into
vaccination phase.
                        if(zoc==1)
                        {
```

```
                                pthread_mutex_lock(&(zoneofvac->zonelock));
// To signal the zone that student got vaccinated.
                                zoneofvac->compstd--;
                                pthread_mutex_unlock(&(zoneofvac-
>zonelock));

                                printf("Student  %d on Vaccination Zone %d
has been vaccinated which has success probability %0.3f \n", shishya->stid
, zoneofvac->zoneid , probarr[zoneofvac->frompharma]);

                                int a = (int)(probarr[zoneofvac-
>frompharma]*1000); // To simulate randomness and probability
                                if((rand()%1000) > a ) // tests Negative
                                {


printf("Student %d has tested negative for antibodies.\n",shishya->stid);

                                        status[shishya->stid]--;
                                        pthread_mutex_lock(&waitstd);
                                        waitingstd++;                   //
He again has to wait for another slot.
                                        pthread_mutex_unlock(&waitstd);



                                }
                                else // tests positve
                                {
                                        printf("Student %d has tested
positive for antibodies.\n",shishya->stid);
                                        status[shishya->stid] = 1;    // He
tests positive and can happily enter the college.
                                }
                                if(status[shishya->stid] == -3) // if a
student test negative for 3 times.
                                {

                                        pthread_mutex_lock(&waitstd);
                                        waitingstd--;
                                        pthread_mutex_unlock(&waitstd);
                                        printf("Student %d is sent home .
He is sad :(\n", shishya->stid);

                                        cnttt++;
                                }

                                break;
                        }
                }
        }

        return NULL;
}
```

This the structure of the Student to store essential information about the student

```
struct student
{
        int stid; // Id to access
        int status; // To know how many times tested negative for antibody
test
        pthread_t sid;
        pthread_mutex_t studentlock;

};
```