Άσκηση 1(primes)

```
struct info_t{
        int array[nthreads];
        enum boolean flag[ntheads];
};
```

Master_thread

```
while(number > 1){
        scanf();
        if(number <= 1){
                lock(mtx);
                term = true; // for all
                if(all sleeping){
                        nthreads signal(job); break;
                }
                else{
                        wait(finish) //for all
                        nthreads – 1 signal(job);
                        break;}
        }
        lock(mtx);
        counter++; available--;
        if(no one available){ wait(ready);}
        find pos to write in a loop; working++;
        if(one or more sleeping){ signal(job);}
        unlock(mtx);
        yield();
}
if(not all exited){ wait(last_thread);}
unlock(mtx);
```

```
struct thread_t{
        pthread_t id;
        enum boolean tem;
        int status, number, nthreads;
        int *counter, available, flag;
        int* working, exited;
        struct info_t* info;
};
```

```
conds{ ready,working,
        finish, available,
        job, last_thread}

mutexes{ mtx  }
```

Worker

```
while(1){
        lock(mtx);
        working--;
        //if no other number, term=true, last worker available{
                signal(finish); }
        //if no job {  wait(job); }
        //if no numbers, term = true {    break; }
        //if flag = 1 (valid pos) give number, flag = 0
        unlock(mtx);
        //primetest;
        lock(mtx);
        counter--; available++; status = 1;
        //if first available{   signal(ready); }
        unlock(mtx);
        yield();
}
exited++;
//if every thread exited{   signal(last_thread); }
unlock(mtx);
pthread_exit();
```

Άσκηση 2(bridge) (*let them enter = update all counters)

conds{ redq, blueq**}**

mutexes{ mtx **}**

arrive_red

```
lock(mutex);
//check if my colour == bridge colour{
        1)enough space on bridge
                curr_bridge++; red_passed++;
                while(enough space && red_waiting > 0){
                        //let them enter (>=2N →break)
                        signal(redq);
                }
                unlock(mutex);
        2)max cars on bridge || 2N
                red_waiting++;
                wait(redq); unlock(mutex);
}
//my colour != bridge colour{
        1)no one on bridge, not blue waiting
                colour=red; curr_bridge++; red_passed=1;
                unlock(mutex);
        2)blue on bridge
                red_waiting++;
                wait(redq);
                while(enough space && red_waiting > 0){
                        //let them enter (>=2N →break)
                        signal(redq);
                }
                unlock(mutex);
}
```

leave_red

```
lock(mutex);
//full bridge{
        curr_bridge--;
        1)if red waiting
                if(red_passed<2N&&no bluewaiting){
                        //let them enter
                        signal(redq);
                } unlock(mutex);
        2)unlock(mutex); }
//I am the last one to exit{
        curr_bridge--;
        1)if opposite colour cars wait
                red_passed=blue_passed=0;
                while(enough space&&blue_waiting > 0){
                        //let them enter (>=2N →break)
                        colour = blue;
                        signal(blueq);
                } unlock(mutex);
        2)if my colour cars wait
                red_passed = 0;
                while(enough space&&red_waiting > 0){
                        //let them enter (>=2N →break)
                        signal(redq);
                } unlock(mutex);
        3)unlock(mutex); }
//random leave{
        curr_bridge--;
        unlock(mutex);
}
```

Άσκηση 3(train)

passenger

```
lock(mtx);
if(enough space && waiting == 0){
        curr++; entering(); in++;
        if(in == max){
                status = 1; signal(ride);
        }
}
else{
        waiting++;
        wait(enter);
        while(enough space && waiting > 0){
                //let them enter(all counters updated)
                signal(enter);
                in++; entering();
                if(in == max){ status = 1; signal(ride);}
        }
        unlock(mtx);
        //exit section
        lock(mtx);
        if(flag != 1){ sleeping++; wait(finish); }
        if(sleeping > 0){ sleeping--; signal(finish);}
        out++;
        if(out == max){ out = 0; signal(last_pass);}
        unlock(mtx);
}
```

```
struct ride_t{
        int max_pass, curr_pass, waiting, flag;
        int sleeping, out, status, in;
};
```

conds{ ride, enter, finish, last_pass**}**
mutexes{ mtx **}**

train

```
while(1){
        lock(mtx);
        if(in < max || status == 0){
                wait(ride);
        }
        unlock(mtx);
        //ride for 1 sec
        lock(mtx);
        flag = 1;
        if(sleeping > 0){
                sleeping--;
                signal(finish);
        }
        if(out < max){ wait(last_pass); }
        in = status = curr_pass = 0;
        if(waiting > 0 && enough space){
                //let one enter(all counters updated)
                signal(enter);
        }
        flag = 0;
        unlock(mtx);
}
```

Άσκηση 4(CCR)

CCR_DECLARE

```
conds{ q1, q2, enter};
mutexes{ mtx };
int n1, n2, sleeping, status;
```

CCR_INIT

```
mutex_init(mtx);
cond_init(q1);
cond_init(q2);
cond_init(enter);
n1= n2 = sleeping = 0;
status = 1;
```

train

```
while(1){
        EXEC(1, (curr == max), printf("max pass on train\n"));
        //ride for 2 seconds
        EXEC(1, 1, flag = 1);
        EXEC(1, (out == max), curr = 0; out = 0; flag = 0; printf("exiting\n"));
}
```

Passenger

```
EXEC(1, (curr < max), curr++; entering());
EXEC(1, (flag == 1), out++; exiting());
```

CCR_EXEC

```
lock(mtx);
sleeping++;
if(status == 0 || sleeping > 1){
        wait(enter);
}
else{ sleeping--; }
status = 0;
while(!cond){
        n1++;
        if(n2 > 0){ n2--; signal(q2); }
        else{
                status = 1;
                if(sleeping > 0){
                        sleeping--; signal(enter);
                }
        }
        wait(q1); n2++;
        if(n1 > 0){ n1--; signal(q1); }
        else{ n2--; signal(q2); }
        if(n2 >= 1){ wait(q2); }
}
//CS
if(n1 > 0){ n1--; signal(q1); }
else if(n2 > 0){ n2--; signal(q2); }
else{
        status = 1;
        if(sleeping > 0){
                sleeping--; signal(enter);
        }
}
unlock(mtx);
```