ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΕΡΓΑΣΙΑ 2

Σαπουντζή Αθανασία Δέσποινα

Θείου Βασιλης

Ασκηση 1

```
int mysem_create(int init) {
    int semid;
    semid = semget(IPC_PRIVATE, 1, S_IRWXU);
    semctl(semid, 0, SETVAL, init);
    return(semid);
}

void mysem_down(int semid) {
    op.sem_op = -1;
    semop(semid, &op, 1);
}

int mysem_up(int semid) {
    value = semctl(semid, 0, GETVAL);
    if(value) { return; }

    op.sem_op = 1;
    semop(semid, &op, 1);
    return;
}

void mysem_destroy(int semid) {
    semctl(semid, 0, IPC_RMID);
}
```

Ασκηση 2

MAIN THREAD:

```
mtx = create(1);
waitq = create(0);
terminate = create(0);
work = create(0);
while(1)  {
  Read input and place it into the buffer
}
for(i=0; i< num_workers; i++) {Create workers}

while (assigns < size) {
    down(mtx);
    if(waiting > 0) {
        waiting--; assigns++;
        up(mtx); up(waitq);
    }
    else { up(mtx); }
}
down(mtx);
if(waiting < num_workers) {
    up(mtx); down(work);
}
else { up(mtx); }
for(i = 0; i< num_workers; i++) { up(waitq); }

down(terminate);

destroy binary semaphores
return;
```

```
volatile int mtx;
volatile int waitq;
volatile int terminate;
volatile int work;
```

WORKER THREAD:

```
while(1) {
    down(mtx);
    waiting++;
    if(waiting == num_workers && jobs_done == size) {
        up(work);
    }
    up(mtx);
    down(waitq);

    down(mtx);
    if(jobs_done == size) {
        waiting--;
        if( waiting == 0) {
            up(mtx);  up(terminate);
            return;
        }
        else { up(mtx); return; }
    }
    else {
        jobs_done++;
        result = primetest();
        up(mtx);
        print result;
    }
}
```

Άσκηση 3:

CAR:

```
volatile int mtx;
volatile int b_q, r_q, join;
```

enter_bridge :

```
down(mtx);
if(car_color == BLUE) {
   if(r_bridge > 0 || b_bridge == bridge_size || b_lim >= cars_lim
      || b_waiting > 0) {
      b_waiting++;  up(mtx);  down(b_q);
      down(mtx);
      if(b_waiting > 0 && b_bridge < bridge_size && b_lim < cars_lim) {
         b_waiting--;  b_bridge++;  b_lim++;
         up(mtx);  up(b_q);
      }
      else { up(mtx); }
   }
   else { b_bridge++;  b_lim++;  up(mtx); }
}
if(car_color == RED) {
   if(b_bridge > 0 || r_bridge == bridge_size || r_lim >= cars_lim
      || r_waiting > 0) {
      r_waiting++;  up(mtx);  down(r_q);
      down(mtx);
      if(r_waiting > 0 && r_bridge < bridge_size && r_lim < cars_lim) {
         r_waiting--;  r_bridge++;  r_lim++;
         up(mtx);  up(r_q);
      }
      else { up(mtx); }
   }
   else { r_bridge++;  r_lim++;  up(mtx); }
}
```

exit_bridge:

```
down(mtx);
if(car_color == BLUE) {
   b_bridge--;  num_cars++;
   if(b_bridge == 0 && r_waiting > 0) {
      b_lim =0;   r_waiting--;   r_bridge++;   r_lim++;
      up(mtx);  up(r_q);
   }
   else if(b_waiting > 0 && (b_lim < cars_lim || r_waiting == 0)) {
         b_waiting--;  b_bridge++;  b_lim++;
         up(mtx);  up(b_q);
   }
   else {
         if(num_cars == input) { up(mtx);   up(join); }
         else { up(mtx); }
   }
}
if(car_color == RED ) {
    r_bridge--;  num_cars++;
    if(r_bridge == 0 && b_waiting > 0) {
       r_im = 0;  b_waiting--;  b_bridge++;  b_lim++;
       up(mtx);  up(b_q);
    }
    else if(r_waiting > 0 && (r_lim < cars_lim || b_waiting == 0)) {
       r_waiting--;  r_bridge++;   r_lim++;
       up(mtx);  up(r_q);
    }
    else {
       if(num_cars == input) { up(mtx);  up(join); }
       else { up(mtx); }
    }
}
return;
```

MAIN:

```
mtx = create(1);
b_q = create(0);
r_q = create(0);
join = create(0);
cars_lim = 2*bridge_size;
for(i = 0; i < input; i++) {
   create car threads
}

if(b_q > 0 || r_q > 0) {
    down(join);
}
destroy binary semaphores
return;
```

Ασκηση 4:

```
volatile int mtx;
volatile int wait_line, exit_train;
volatile int train_full, train_empty,
volatile train_start,join;
```

PASSENGER:

passenger_enter:

```
down(mtx);
if( waiting == train_capacity  &&  boarding == 0) {
   up(train_full);
 }
waiting++;
up(mtx);  down(wait_line);

down(mtx);
if(boarding < train_capacity) {
    waiting--;  boarding++;
    up(mtx);  up(wait_line);
 }
else if(boarding == train_capacity) {
    up(mtx);  up(train_start);
 }
```

passenger_exit:

```
down(exit_train);
down(mtx);
If(boarding == 0) { up(mtx);  up(train_empty); }
else if(boarding  > 0) {
    boarding--;
    up(mtx);  up(exit_train);
}
return;
```

TRAIN:

```
while(1) {
    down(mtx);
    if(waiting < train_capacity) {
        up(mtx);
        down(train_full);
    }
    else { up(mtx); }
    down(mtx);
    waiting--;  boarding++;
    up(mtx);  up(wait_line);
    down(train_start);

    sleep(ride_duration); //RIDE

    down(mtx);
    boarding--;
    up(mtx); up(exit_train);
    down(train_empty);
}
return;
```

MAIN:

```
mtx = create(1);
wait_line = create(0);
exit_train= create(0);
train_full = create(0);
train_empty = create(0);
train_start = create(0);
join = create(0);

Create train
for(i= 0; i < num_passengers i++ ) {
    Create passenger
}

down(join);

Destroy binary semaphores
return;
```