Εργασία 2η

Ομάδα 10
Αξελού Ολυμπία 2161
Τσιτσοπούλου Ειρήνη 2203

2.1 Δυαδικοί σηματοφόροι

```
typedef struct{
   pthread_mutex_t mtx;
   pthread_mutex_t waitingCS;
   int value;
}my_bsem;
```

Η υλοποίηση αυτή δεν εγγυάται δικαιοσύνη επειδή βασιζόμαστε στην εσωτερική υλοποίηση των pthread_mutexes όσον αφορά το ποιος ξεμπλοκάρει πρώτος σε μία κλήση της unlock. Η ίδια η βιβλιοθήκη των pthread_mutexes δεν εγγυάται δικαιοσύνη.

my_bsem_init

```
//initialize attr for init

//initialize mtx and CS mutexes

value = init_value;

//init(sem,0)
if (!init_value){ lock(mtx); }
```

my_bsem_up

```
lock(waitingCS);

if (value == 1) //sem already up
  unlock(waitingCS);
  return 1;

value++;

If (value <= 0) //sb is blocked
  unlock(mtx);

unlock(waitingCS);
return 0;
```

my_bsem_down

```
lock(waitingCS);

value--;

if (value < 0) {
  unlock(waitingCS);
  lock(mtx);
}
else if (value == 0){
  unlock(waitingCS);
}
```

my_bsem_destroy

```
lock(waitingCS); //to make sure that
                 //none is using mtx

if (value != 1){
  unlock(mtx);
}

unlock(waitingCS);

//destroy mtx and CS mutexes
```

2.2 Παράλληλος υπολογισμός fractals

```
volatile int *res;                bsem *sem_args;   init(sem_args[i], 0);
volatile mandel_Pars * slices;    bsem *sem_draw; init(sem_assign[i], 0);
volatile int maxIterations;       bsem *sem_assign;       init(sem_draw[i], 1);
```

MAIN THREAD

```
. . .
//allocate res, slices, semaphores
for(#nofslices){
  create thread;
  down(sem_args[i]); //wait for it to take its args
}
. . .
while(1){
  //create #nofslices jobs and assign to workers
  for(#nofslices) up(sem_assign[i]); //notify workers
  while(1){
    for(#nofslices)
       if (up(sem_draw[i] == 0) //find a just-finished worker
          break; //and draw its slice
    if(all workers done) break; //-> next job
   }
  //take coordinates (next job)
 } . . .
```

WORKER THREAD

```
//retrieve arguments
up(sem_args[my_no]); //notify main that arguments have been
retrieved

while(1){
        down(sem_assign[my_no]); //wait for main to assign job
        perform Mandelbrot computation
        down(sem_draw[my_no]); //notify main to draw my slice
        }
```

```
bsem mtx;           init(mtx, 1);
bsem waitq[2];      init(waitq[i], 0);
int waiting[2] = {0,0}, onbridge[2] = {0,0}, carsPassing[2] = {-1,-1};
```

2.3 Στενή Γέφυρα

BRIDGE ENTER

```
down(mtx);
if(carsPassing[mcolor] >= 0){  //is there sb of other color waiting
   carsPassing[mcolor]++;
}

//full bridge or other color (waiting too long || on bridge)
if(onbridge[!mcolor] || (onbridge[mcolor] >= bridgeCapacity) ||
carsPassing[mcolor] > MAX_PASSING){
     waiting[mcolor]++;
     if(carsPassing[!mcolor] == -1){ carsPassing[!mcolor] = 0; }
     up(mtx);

    down(waitq[mcolor]);

    down(mtx);

    if(carsPassing[!mcolor] >= 0){carsPassing[!mcolor] = -1;}
    if((waiting[mcolor] > 0) && (onbridge[mcolor] < bridgeCapacity)){
       waiting[mcolor]--; onbridge[mcolor]++;
       up(waitq[mcolor]);   //chain unblocking
    }
else{ onbridge[mcolor]++; }
up(mtx);
```

CAR

```
bridge_enter( );
// on bridge
bridge_exit ( );
```

BRIDGE EXIT

```
down(mtx);
onbridge[mcolor]--;         //car out of bridge

//no one left on bridge while other color waiting
if(onbridge[my_color] == 0 && waiting[!my_color] > 0 ){
   waiting[!my_color]--; onbridge[!my_color]++;
   up(waitq[!mcolor]);
}
//bridge not full anymore: allows same-color cars to pass
else if(onbridge[my_color] >= bridgeCapacity - 1 &&
waiting[my_color] > 0 ){
   waiting[my_color]--;
   onbridge[my_color]++;
   up(waitq[my_color]);
}

up(mtx);
```

2.4 Τρενάκι

```
bsem mtx;             bsem wait_to_fill;        init (mtx,1);                init (wait_to_fill, 0);
bsem train_start;     bsem wait_to_empty;       init (train_start, 0);       init (wait_to_empty, 0);
bsem pas_entering;    bsem pas_exiting;         init (pas_entering, 0);      init (pas_exiting, 0);
int waiting = 0, onboard = 0;
N = trainCapacity;
```

passenger

```
train_enter( );
// on ride
train_exit ( );
```

train_enter( )

```
down(mtx);
waiting++;
if(waiting == N){up(wait_to_fill);} //train can start filling up
up(mtx);

down(pas_entering);

down(mtx);
if(onboard < N){ waiting--; onboard++; up(pas_entering); }
else if(onboard == N){ up(train_start); } //last passenger notifies
up(mtx);
```

train_exit( )

```
down(pas_exiting);
down(mtx);
if(onboard > 0){ onboard--; up(pas_exiting); } //chain unblocking
else if (onboard == 0) { up(wait_to_empty); } //last one exiting
up(mtx);
```

train

```
down(mtx);
if (waiting <= 2*trainCapacity){
  up(mtx); down(wait_to_fill); down(mtx); }

onboard++; waiting--:

up(pas_entering);  //notify first passenger to get in
up(mtx);

down(train_start); //wait for everyone to get in
//on ride

down(mtx);
onboard--;
up(pas_exiting);     //notify first passenger to get out
up(mtx);
down(wait_to_empty);
```