ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ 2

Ομάδα 8
Γιαννούκος Τριαντάφυλλος Ανάργυρος
Ματζώρος Χρήστος Κωνσταντίνος

2.1 Δυαδικοί Σηματοφόροι

```
typedef struct semaphore{
    int  flag;
    int  is_blocked;
    int  val;
    pthread_mutex_t  mtx;
}mybsem;

void mybsem_init(mybsem *sem, int init_value){
    sem->flag = init_value;
    sem->val = init_value;
    sem->is_blocked = 0;
    if(init_value == 0){
        pthread_mutex_lock(&sem->mtx);
    }
}

void mybsem_destroy(mybsem *sem){
    if(sem->val == 0){
        pthread_mutex_unlock(&sem->mtx);
    }
        pthread_mutex_destroy(&sem->mtx);
}
```

```
void mybsem_down(mybsem *sem){
    if(sem->flag == 0){
        sem->is_blocked ++;
        pthread_mutex_lock(&sem->mtx);
    }
    else if(sem->flag == 1){
        sem->flag--;
        pthread_mutex_lock(&sem->mtx);
    }
}

void mybsem_up(mybsem *sem){
   if( (sem->flag == 0) && (sem->is_blocked > 0 )){
       sem->is_blocked --;
       pthread_mutex_unlock(&sem->mtx);
       pthread_yield();
   }
   else if( (sem->flag == 0) && (sem->is_blocked == 0) ){
       pthread_mutex_unlock(&sem->mtx);
       sem->flag++;
   }
    else{ printf("ERROR semaphore value is already one "); }
}
```

2.2 Υπολογισμός Fractals

```
typedef struct parameters{
   ... (Parameters for the calculation)
   mybsem sem1;
   int done;
}worker;
```

```
void *mandel_foo(void * arg){
   while(1){
      mybsem_down(&my_struct.sem1);
      mandel_Calc();
      mybsem_down(&sem2);
      my_parameters->done = 1;
      mybsem_up(&sem4);
      mybsem_down(&sem3);
      mybsem_up(&sem2);
   }
   return NULL;
}
```

```
int main(int argc, char *argv[]) {
   mybsem_init(&main_results[i].sem1, 0);   for  i = 0, … , N
   mybsem_init(&sem2, 1);
   mybsem_init(&sem3, 0);
   mybsem_init(&sem4, 0);
   Create N threads
   while(1){
      for (i=0; i<nofslices; i++) {
         Assign values to the worker
         mybsem_up(&main_results[i].sem1);
      }
      for (k=0; k<nofslices; k++) {
         mybsem_down(&sem4);
         mybsem_up(&sem3);
         for (i=0; i<nofslices; i++) {
            if(main_results[i].done == 1){
               Draw ith slice
            }
         }
      }
   }
}
```

2.3 Στενή Γέφυρα

```
int main(){
    …
    mybsem_init(&sem[0],1);
    mybsem_init(&sem[1],1);
    mybsem_init(&sem_order,1);
    mybsem_init(&sem_bridge,1);
    mybsem_init(&wait_car,0);
    …
    Destroy the semaphores
}
void *foo(void *arg){
   enter_the_bridge(colour);
   sleep();  // Κρίσιμο Τμήμα
   exit_the_bridge(colour);
}
```

```
void enter_the_bridge(char colour){
   if(colour =='r'){ c = 0; }
   else{ c = 1; }
   mybsem_down(&sem_order);
   mybsem_down(&sem[c]);
   car_count[c]++;
   if(car_count[c] == 1){
      mybsem_down(&sem_bridge);
   }
   if(car_count[c] > N){
      mybsem_up(&sem[c]);
      mybsem_down(&wait_car);
   }
   else{
      mybsem_up(&sem[c]);
   }
   mybsem_up(&sem_order);
}
```

```
void exit_the_bridge(char colour){
   if(colour =='r'){ c = 0; }
   else{ c = 1; }
   mybsem_down(&sem[c]);
   car_count[c]--;
   if(car_count[c] == N){
      mybsem_up(&wait_car);
   }
   if(car_count[c] == 0){
      mybsem_up(&sem_bridge);
   }
   mybsem_up(&sem[c]);
}
```

2.4 Τρενάκι

```
int main(){
   ...
   main_info.counter = 0;
   main_info.nofrides = 0;
   main_info.train_done = 0;

   mybsem_init(&next_ride, 0);
   mybsem_init(&train_loaded, 0);
   mybsem_init(&ready_to_start, 0);
   mybsem_init(&ready_to_disembark, 0);
   mybsem_init(&wait_passengers, 1);
   mybsem_init(&counter_sem, 1);
   mybsem_init(&waiting_room, 1);
   mybsem_init(&leave, 1);
   ...
   Destroy the semaphores
}
```

```
   typedef struct information{
      int N;
      int counter;
      int nofrides;
      int train_done;
   }train_info;
```

```
void *passenger_foo(void *arg){
   enter_the_train(pas_info);

   exit_the_train(pas_info);
}
```

2.4 Τρενάκι

```
int enter_the_train(train_info *pas_info){
    If the train informed me that the ride won't happen
    I have to terminate
  mybsem_down(&waiting_room);
  mybsem_down(&counter_sem);
  pas_info->counter++;
  if(pas_info->counter == 1){
    mybsem_down(&wait_passengers);
    mybsem_up(&counter_sem);
    mybsem_up(&waiting_room);
    mybsem_down(&train_loaded);
    mybsem_up(&wait_passengers);
  }
  else if(pas_info->counter < pas_info->N){
    mybsem_up(&counter_sem);
    mybsem_up(&waiting_room);
    mybsem_down(&wait_passengers);
    mybsem_up(&wait_passengers);
  }
  else if(pas_info->counter == pas_info->N){
    mybsem_up(&counter_sem);
    mybsem_up(&ready_to_start);
  }
  return 0;
}
```

```
void exit_the_train(train_info *pas_info){
  mybsem_down(&counter_sem);
  pas_info->counter--;
  mybsem_down(&leave);
  if(pas_info->counter == 0){
    mybsem_up(&counter_sem);
    mybsem_up(&next_ride);
  }
  else if(pas_info->counter == (pas_info->N)-1){
    mybsem_up(&counter_sem);
    mybsem_down(&ready_to_disembark);
  }
  else{
    mybsem_up(&counter_sem);
  }
  mybsem_up(&leave);
}
```

```
void *train_foo(void *arg){
    If there are not enough people to perform a ride inform them and terminate

while(1){
    mybsem_down(&ready_to_start);
    mybsem_up(&train_loaded);

    sleep(T);    // CS

    mybsem_up(&ready_to_disembark);

    sleep(2);   // return to start point

    mybsem_down(&next_ride);
    if(ride_info->nofrides ==
      ride_number){
      ride_info->train_done = 1;
      mybsem_up(&waiting_room);
      break;
    }
    mybsem_up(&waiting_room);
    ride_number++;
  }
  return NULL;
}
```