ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ 3

Ομάδα 8
Γιαννούκος Τριαντάφυλλος Ανάργυρος
Ματζώρος Χρήστος Κωνσταντίνος

3.1 Υπολογισμός Fractals με Condition Variables

```
typedef struct parameters{
   ...  (Parameters for the calculation)
   pthread_mutex_t mtx1;
   pthread_cond_t cond1;
   int done;
   int flag;
}worker;

void *mandel_foo(void * arg){
   while(1){
      pthread_mutex_lock(&my_parameters->mtx1);
      my_parameters->flag--;
      if(my_parameters->flag < 0){
         pthread_cond_wait(&my_parameters->cond1, &my_parameters->mtx1);
      }
      pthread_mutex_unlock(&my_parameters->mtx1);

      mandel_Calc();

      pthread_mutex_lock(&mtx2);
      my_parameters->done = 2;
      signal_counter++;
      if(signal_counter <= 0){
         pthread_cond_signal(&cond2);
      }
      pthread_mutex_unlock(&mtx2);
   }
   return NULL;
}
```

```
int main(int argc, char *argv[]) {
   create N threads
   while (1) {
      for (i=0; i<nofslices; i++) {
         pthread_mutex_lock(&main_results[i].mtx1);
         Assign values to the workers
         main_results[i].done = 1;
         main_results[i].flag++;
         if(main_results[i].flag <= 0){
            pthread_cond_signal(&main_results[i].cond1);
         }
         pthread_mutex_unlock(&main_results[i].mtx1);
      }
      for (k=0; k<nofslices; k++) {
         pthread_mutex_lock(&mtx2);
         signal_counter--;
         if(signal_counter < 0){
            pthread_cond_wait(&cond2, &mtx2);
         }
         pthread_mutex_unlock(&mtx2);
         for (i=0; i<nofslices; i++) {
            if(main_results[i].done == 2){
               Draw ith slice
            }
         }
      }
   }
}
```

3.2 Στενή Γέφυρα με Condition Variables

```
void enter_the_bridge(char colour){
    if(colour =='r'){
        pthread_mutex_lock(&mtx);
        if(  (blue_on_bridge > 0) || (red_on_bridge > (N-1)) ||
            ( (blue_waiting > 0) && (total_counter > ((2*N) - 1)) )  ) {

            red_waiting++;
            pthread_cond_wait(&rq, &mtx);
            if (red_waiting > 0 && total_counter < ((2*N) - 1) &&
                (red_on_bridge < N)) {

                red_waiting--;
                red_on_bridge++;
                total_counter++;
                pthread_cond_signal(&rq);
            }
        }
        else{
            red_on_bridge++;
            total_counter++;
        }
        pthread_mutex_unlock(&mtx);
    }
    else if(colour =='b'){


        ...

    }
}
```

```
void exit_the_bridge(char colour){
    if(colour =='r'){
        pthread_mutex_lock(&mtx);
        red_on_bridge--;
        if(  (red_waiting > 0) && (red_on_bridge < N) &&
            ((total_counter < (2*N))||(blue_waiting == 0))  ) {

            red_waiting--;
            red_on_bridge++;
            total_counter++;
            pthread_cond_signal(&rq);
        }
        else if ((red_on_bridge == 0) && (blue_waiting > 0)) {
            blue_waiting--;
            total_counter = 0;
            blue_on_bridge++;
            total_counter++;
            pthread_cond_signal(&bq);
        }
        pthread_mutex_unlock(&mtx);
    }
    else if(colour =='b'){
        ...
    }
}
void *foo(void *arg){

    enter_the_bridge(colour);

    sleep();  // Κρίσιμο Τμήμα

    exit_the_bridge(colour);

}
```

3.3 Τρενάκι με Condition Variables

```
int enter_the_train(train_info *passenger_info){
   pthread_mutex_lock(&mtx);

   passenger_info->counter++;

   if( (passenger_info->counter % passenger_info->N) == 0) {
      passengers_ready++;
      if(train_waiting_to_start_ride == 1){
         pthread_cond_signal(&train_cond);
      }
   }

   pthread_cond_wait(&passengers_cond1, &mtx);

   pthread_mutex_unlock(&mtx);
   return 0;
}
```

```
void exit_the_train(train_info *passenger_info){
   pthread_mutex_lock(&mtx);
   passengers_ready_to_exit++;
   if(train_waiting_to_end == 1 &&
         passengers_ready_to_exit == passenger_info->N){
      pthread_cond_signal(&train_cond);
   }
   pthread_cond_wait(&passengers_cond2, &mtx);
   pthread_mutex_unlock(&mtx);
}
```

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

   exit_the_train(pas_info);
}
```

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

```
void *train_foo(void *arg){
   while(1){
      pthread_mutex_lock(&mtx);
      if(passengers_ready == 0){
         train_waiting_to_start_ride = 1;
         pthread_cond_wait(&train_cond, &mtx);
      }
      train_waiting_to_start_ride = 0;
      passengers_ready--;
      for(i=0; i<ride_info->N; i++){
         pthread_cond_signal(&passengers_cond1);
      }
      pthread_mutex_unlock(&mtx);
      // Start of CS
      sleep(T);

      pthread_mutex_lock(&mtx);
      if(passengers_ready_to_exit < ride_info->N){
         train_waiting_to_end = 1;
         pthread_cond_wait(&train_cond, &mtx);
      }
      passengers_ready_to_exit = 0;
      train_waiting_to_end = 0;
      for(i=0; i<ride_info->N; i++){
         pthread_cond_signal(&passengers_cond2);
      }
      pthread_mutex_unlock(&mtx);
      sleep(2);
   }
   return NULL;
}
```

3.4 Conditional Critical Regions

```
typedef struct labels{
   int n1;
   int n2;
   pthread_cond_t q1;
   pthread_cond_t q2;
   pthread_mutex_t mtx;
}struct_label;


#define CCR_DECLARE(label) struct_label label;

#define CCR_INIT(label)
   int  mtxtype = PTHREAD_MUTEX_NORMAL;
   pthread_mutexattr_t attr;

   pthread_mutexattr_init(&attr);
   pthread_mutexattr_settype(&attr, mtxtype);
   pthread_mutex_init(&label.mtx, &attr);

   pthread_cond_init(&label.q1, NULL);
   pthread_cond_init(&label.q2, NULL);

   label.n1 = 0;
   label.n2 = 0;
```

```
#define CCR_EXEC(label, cond, body)
   pthread_mutex_lock(&label.mtx);
   while(!(cond)){
      label.n1++;
      if(label.n2>0){
         label.n2--;
         pthread_cond_signal(&label.q2);
      }
      pthread_cond_wait(&label.q1, &label.mtx);
      label.n2++;
      if(label.n1>0){
         label.n1--;
         pthread_cond_signal(&label.q1);
         pthread_cond_wait(&label.q2, &label.mtx);
      }
      else if(label.n2>1){
         label.n2--;
         pthread_cond_signal(&label.q2);
         pthread_cond_wait(&label.q2, &label.mtx);
      }
   }

   body

   if(label.n1>0){
      label.n1--;
      pthread_cond_signal(&label.q1);
   }
   else if(label.n2>0){
      label.n2--;
      pthread_cond_signal(&label.q2);
   }
   pthread_mutex_unlock(&label.mtx);
```

3.4.1 Υπολογισμός Fractals με CCR

```
typedef struct parameters{
   mandel_Pars *pars;
   int maxIters;
   int *res;
   int done;
   CCR_DECLARE(R1)
}worker;

CCR_DECLARE(R2)
int  to_draw_counter = 0;
```

```
void *mandel_foo(void * arg){
   while(1){
      CCR_EXEC(my_parameters->R1, my_parameters->done == 1 ,
         mandel_Calc();
      )
      CCR_EXEC(R2, 1,
         my_parameters->done = 2;
         to_draw_counter++;
      )
   }
   return NULL;
}
```

```
int main(int argc, char *argv[]) {

   CCR_INIT(main_results[i].R1) for i=0,1,...,N
   CCR_INIT(R2)
   create N threads
   while (1) {
      for (i=0; i<nofslices; i++) {
         CCR_EXEC(main_results[i].R1,1,
            Assign values to the workers
            main_results[i].done = 1;
         )
      }
      for (k=0; k<nofslices; k++) {
         CCR_EXEC(R2, to_draw_counter>0,
            to_draw_counter--;
         )
       for (i=0; i<nofslices; i++) {
            if(main_results[i].done == 2){
               Draw ith slice
            }
         }
      }
   }
}
```

3.4.2 Στενή Γέφυρα με CCR

```
void enter_the_bridge(char colour){
   if(colour =='r'){
      CCR_EXEC(R1,1,
         red_waiting++;
      )
      CCR_EXEC(R1, (red_on_bridge < N) && ( (red_counter < (2*N)) || (blue_waiting == 0) ) &&
          (blue_on_bridge == 0) && !( (blue_counter < (2*N)) && (blue_counter > 0) && (blue_waiting > 0) ),

         blue_counter = 0;
         red_on_bridge++;
         red_counter++;
         red_waiting--;
      )
   }
 else if(colour =='b'){
      ...
   }
}

  void exit_the_bridge(char colour){
     if(colour =='r'){
        CCR_EXEC(R1, 1,
            red_on_bridge--;
        )
     }
      else if(colour =='b'){
        CCR_EXEC(R1, 1,
            blue_on_bridge--;
        )
     }
  }
```

3.4.3 Τρενάκι με CCR

```
int enter_the_train(train_info *passenger_info){
    CCR_EXEC(R1, ( (train_waiting_to_start_ride == 1) && (passenger_info->counter < passenger_info->N)),
        passenger_info->counter++;
        if(passenger_info->counter == passenger_info->N) {
            passengers_ready = 1;
            train_waiting_to_start_ride = 0;
        }
    )
    return 0;
}

void exit_the_train(train_info *passenger_info){
    CCR_EXEC(R1, (train_waiting_to_end == 1),
        passenger_info->counter--;
        if(passenger_info->counter == 0) {
            passengers_ready_to_exit = 1;
            train_waiting_to_end = 0;
        }
    )
}
```

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

    exit_the_train(pas_info);
}
```

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

```
void *train_foo(void *arg){
    while(1){
        CCR_EXEC(R1, 1,
            train_waiting_to_start_ride = 1;
        )

        CCR_EXEC(R1, (passengers_ready == 1),
            passengers_ready = 0;
        )

        // Start of CS
        sleep(T);

        CCR_EXEC(R1, 1,
            train_waiting_to_end = 1;
        )
        CCR_EXEC(R1, (passengers_ready_to_exit == 1),
            passengers_ready_to_exit = 0;
        )
        sleep(2);
    }
    return NULL;
}
```