UNIVERSITY OF
THESSALY

Ταυτόχρονος Προγραμματισμός

Εργασία 3,Ομάδα 15

Αγγέλης Μάριος-Κασιδάκης Θεόδωρος

ΑΕΜ:2406-2258

1.Prime numbers recognition

```
//master thread main
//create "size" threads
while(1){
    //reading a number from a file
    if (EOF){close_flag=1;break;}
    pthread_mutex_lock(&mtx);  //Check if there are blocked threads
    if(num_threads_blocked!=0){  pthread_cond_signal(&thread_q);  } //Wake up a thread
    else{ main_is_blocked=1;}
    pthread_cond_wait(&main_q,&mtx);   //Main is waiting fo thread's reaction
    for(i=0;i<size;i++)                //Main is not spinning
        if(info[i].job_flag==0){       //Main found a free worker
            info[i].prime=number;
            info[i].job_flag=1;        //Worker has job now
            pthread_cond_signal(&info[i].pcod);
            break;
        }
    }
    pthread_mutex_unlock(&mtx);
}
```

```
//Threads function
void *thread_func(void  *arg){
    struct T *thread_struct=(struct  T*)arg;
    while(1){
        //Waiting main to increase thread semaphore
        pthread_mutex_lock(&mtx);
        if(main_is_blocked==0){
            num_threads_blocked++;    //Thread is waiting for main
            pthread_cond_wait(&thread_q,&mtx);
            num_threads_blocked--;
        }
        else{main_is_blocked=0;}
        info[thread_struct->position].job_flag=0;
        pthread_cond_signal(&main_q);   //Wake up main
        //Waiting main to give a prime number
        pthread_cond_wait(&info[thread_struct->position].pcod,&mtx);
        pthread_mutex_unlock(&mtx);
        //Execute primetest function
        primetest(info[thread_struct->position].prime);
    }
}
```

2.Narrow bridge

```
//master thread main
while(1){
    //read 2 numbers from a file
    //first number:1 for blue,0 for red car
    //second number : sleeping time
    if (EOF){
      pthread_mutex_lock(&mtx);
      close_flag=1;
    }
    //create a thread
    if(car[i].type==1){ blue_waiting++}
    else if(car[i].type==0) red_waiting++}
}
pthread_cond_wait(&main_q,&mtx);
//main is waiting all threads to close
struct T {
    pthread_t id;
    int iret;
    int type;           //0 for red ,1 for blue
    int sleep_time;
};
```

```
//threads function , code is symmetric for blue and red cars , so we present only the code for blue cars
void *thread_func(void *arg){
    struct T *thread_struct=(struct T*)arg;
    pthread_mutex_lock(&mtx);
    num_blue_blocked++;
    if(first==1){pthread_cond_wait(&blue_q,&mtx);}      //Not first car
    else{first=1; }                                      //First car
    num_blue_blocked--;
    blue_remain++;  blue_waiting--; change_counter++;
    pthread_mutex_unlock(&mtx);
    sleep(thread_struct->sleep_time);                    //A blue car is passing the bridge
    pthread_mutex_lock(&mtx);
    blue_remain--;                                       //A blue car just passed the bridge
    if((change_counter==bridge_size && blue_remain==0) || (blue_remain==0 && blue_waiting==0)){
        //Change side from blue to blue because counter=max or there are not red cars
        if(change_counter==bridge_size && red_waiting==0 && blue_waiting!=0){change_counter=0,pthread_cond_signal(&blue_q);}
        //Change side from blue to red because counter=max
        else if(change_counter==bridge_size && red_waiting!=0){change_counter=0,pthread_cond_signal(&red_q);}
        //Change side from blue to red because all blue cars passed the bridge
        else if(change_counter!=bridge_size && blue_remain==0 && blue_waiting==0 &&  red_waiting!=0){
            change_counter=0; pthread_cond_signal(&red_q);
        }
    }
    //Last car wakes up main and return
    if(blue_waiting==0 && red_waiting==0 && blue_remain==0 && red_remain==0 && close_flag==1){
       pthread_cond_signal(&main_q);
       pthread_mutex_unlock(&mtx);
    }
}
```

4.Train with semaphores

```
while(1){                           //main function
    if (EOF){                       //Reading numbers from a file,if EOF->break
        end_file=1;
        //create a last extra thread
        break;
    }
    else{pthread_mutex_lock(&mtx) ; passengers_num++; }
    //create a thread
    pthread_mutex_unlock(&mtx);
}
pthread_cond_wait(&main_q,&mtx);
void *train_func(void *arg){        //train function
      while(1){
           pthread_mutex_lock(&mtx);
           pthread_cond_wait(&train_q,&mtx);
           if(close_flag==1){  //Wake up main and return
                pthread_cond_signal(&main_q);
                pthread_mutex_unlock(&mtx);
                break;
           }
           sleep(3);
           //Train is back,so wake up the first passenger
           pthread_cond_signal(&passengers_q);
           pthread_mutex_unlock(&mtx);
      }
}
```

```
void *passengers_func(void *arg){                          //passenger's function
    pthread_mutex_lock(&mtx);
    num_blocked++;                                          //Passenger is waiting
    if(first==1){pthread_cond_wait(&passengers_q,&mtx);}
    else{first=1;}
    num_blocked--;
    waiting_num++;
    if(train_capacity<train_size):                          //Train is not full
        train_capacity++;
        if(end_file==0):                                    //Main is still reading numbers from file
            //Train is not full,wake up another passenger
            if((waiting_num <= passengers_num && train_capacity < train_size)){pthread_cond_signal(&passengers_q);}
            //Train is full,so train starts the ride,wake up a passenger and wake up train
             else if(waiting_num <= passengers_num && train_capacity == train_size){pthread_cond_signal(&train_q);}
        else:                                               //Main stopped reading numbers from file
            //Train is not full,wake up another passenger
            if(waiting_num <= passengers_num && train_capacity < train_size){pthread_cond_signal(&passengers_q);}
            //Train is full,so train starts the ride,wake up a passenger and wake up train
            else if(waiting_num <= (passengers_num) && train_capacity == train_size){pthread_cond_signal(&train_q);}
            //Last extra thread wakes up train and return
            else if(waiting_num== passengers_num+1){close_flag=1;pthread_cond_signal(&train_q);}
            //The train is not full,so  the train does not start,wake up the last extra thread
    pthread_mutex_unlock(&mtx);
}
```

5.CCR Library

```
if(R2!=0||signalR1!=0||signalR2!=0||signalR3!=0){
   R3++;
   wait(Rq3);                    // Thread is waiting in q3
   R3--; signalR3--;
   if(signalR3==0 && R1>0){      // Last thread from q3 wakes up first thread of q1
        R1--,signalR1++; signal(Rq1);
   }
   R1++;
   wait(Rq1);                    // Thread is waiting in q1
   signalR1--;
   R2++;
   if(R1>0) {        // If there are threads waiting in q1,wake up one and wait in q2
         R1--; signalR1++; signal(Rq1); wait(Rq2); signalR2--;
   }
   else{
      R2--;
      if(R2>0){      // If there are threads waiting in q2,wake up one and wait in q2
           signalR2++ ; signal(Rq2); wait(Rq2); signalR2--;
      }
   }
}
```

```
while(!cond){                       // if cond=false go to 1st queue
     R1++;
     if(R2>0){                      // If there are threads waiting in q2,wake up one and wait in q1
          R2--; signalR2++; signal(Rq2); wait(Rq1); signalR1--;
     }
     else{
          if(R3!=0 && R2==0 && signalR1==0 && signalR2==0 && signalR3==0) {
               //Wake up all threads waiting in q3
          }
          wait(Rq1); signalR1--;
     }
     R2++;
     if(R1>0) {        // If there are threads waiting in q1,wake up one and wait in q2
          R1--; signalR1++; signal(Rq1); wait(Rq2); signalR2--;
     }
     else{
          R2--;
          if(R2>0){ // If there are threads waiting in q2,wake up one and wait in q2
               signalR2++; signal(Rq2); wait(Rq2); signalR2--;
          }
     }
}
Body-CS
if(R1>0) { // signal to 1st  queue and fix counters }
else if(R2>0){ //signal to 2nd  queue and fix counters  }
else if(R3>0) { // signal  all threads from the 3rd  queue }
```

6.Narrow Bridge , with CCR

```
//master thread main
while(1){
    //read 2 numbers from a file
    //first number:1 for blue,0 for red car
    //second number : sleeping time
    if (EOF){CCR_EXEC(car_synchr,1,close_flag=1;)
        break;
    }
    //create a thread
    CCR_EXEC(car_synchr,1,printf("Main is reading...\n");
        if(car[i].type==1){blue_waiting++;}
        else if(car[i].type==0){red_waiting++;}
        i++; )
}
//main is waiting all threads to close
CCR_EXEC(car_synchr,(main_waiting==1),printf("Main is closing\n");)
struct T {
    pthread_t id;
    int iret;
    int type;            //0 for red ,1 for blue
    int sleep_time;
};
```

```
//threads function , code is symmetric for blue and red cars , so we present only the code for blue cars
void *thread_func(void *arg){
    struct T *thread_struct=(struct T*)arg;
    CCR_EXEC(car_synchr,((change_counter<bridge_size && red_remain==0) | | blue_change_side==1 | | blue_empty_flag==1 | |
    red_empty_flag==1),
        blue_remain++;  blue_waiting--;
        if(blue_change_side==1 && blue_remain==2){ blue_change_side=0;  blue_remain=1; }
        change_counter++;
    )
    sleep(thread_struct->sleep_time);                    //A blue car is passing the bridge
    blue_remain--;                                       //A blue car just passed the bridge
    CCR_EXEC(car_synchr,1,blue_remain--;
      if((change_counter==bridge_size && blue_remain==0) | | (blue_remain==0 && blue_waiting==0)){
        //Change side from blue to blue because counter=max or there are not red cars
        if(change_counter==bridge_size && red_waiting==0 && blue_waiting!=0){change_counter=0;blue_change_side=1;blue_remain++;
        //Change side from blue to red because counter=max
        else if(change_counter==bridge_size && red_waiting!=0){change_counter=0,red_change_side=1;red_remain++;}
        //Change side from blue to red because all blue cars passed the bridge
        else if(change_counter!=bridge_size && blue_remain==0 && blue_waiting==0 &&  red_waiting!=0){
            change_counter=0  ;red_change_side=1;red_remain++;
        }
      }
    )
    //Last car wakes up main and return
    if(blue_waiting==0 && red_waiting==0 && blue_remain==0 && red_remain==0 && close_flag==1){main_waiting=1;}
}
```