Ταυτόχρονος Προγραμματισμός

Εργασία 2,Ομάδα 15

Αγγέλης Μάριος-Κασιδάκης Θεόδωρος

ΑΕΜ:2406-2258

1.Semaphores

```
//Initialize semaphores
int mysem_create(int semid,int initial_value) {
        semid = semget(IPC_PRIVATE,1,S_IRWXU);
        semctl(semid,0,SETVAL,initial_value); // initial semaphore to initial_value value
        return semid;
}
//Semaphore down
void mysem_down(int semid) {
        struct sembuf op;
        op.sem_num=0;
        op.sem_flg=0;
        op.sem_op=-1;
        semop(semid,&op,1);
}
```

```
//Semaphore up
int mysem_up(int semid) {
        int ret;
        struct sembuf op;

        op.sem_num=0;
        op.sem_flg=0;
        op.sem_op=1;
        semop(semid,&op,1);
        ret=semctl(semid,0,GETVAL);
        if(ret>1){//Too many ups }
        return(ret);
}
//Semaphore destroy
void mysem_destroy(int semid) {
        semctl(semid,0,IPC_RMID);
}
```

2.Prime numbers recognition

```
//master thread main
//create "size" threads
while(1){
        //reading a number from a file
        if (EOF){break;}
        mysem_up(thread_sem.semid);          //Wake up a thread
        mysem_down(main_sem.semid);          //Waiting thread to react
        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
                        mysem_up(info[i].sem.semid);  //Wake up the worker
                        break;
                }
        }
}
//Notify threads that main is clossing
mysem_up(thread_sem.semid);          //Wake up a thread
mysem_down(main_sem.semid);          //Waiting last thread to close
```

```
//Threads function
void *thread_func(void  *arg){
        struct T *thread_struct=(struct  T*)arg;
        while(1){
                //Waiting main to increase thread semaphore
                 mysem_down(thread_sem.semid);
                if(close_master==1  && counter<size-1){
                    //This is not the last thread,so increase counter and return
                }
                else if(close_master==1  && counter==size-1){
                     //This is the last thread ,so wake up main and return
                }
                //Thread is available now
                info[thread_struct->position].job_flag=0;
                //Wake up main
                mysem_up(main_sem.semid);
                //Waiting main to give a prime number
                mysem_down(info[thread_struct->position].sem.semid);
                //Execute primetest function
                primetest(info[thread_struct->position].prime);
    }
}
```

3.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){//notify threads that main is closing}
    //create a thread
    if(car[i].type==1){ //if car is blue
      blue_waiting++;
      //if the first car from file is blue,then blue cars start passing the bridge
    }
    else if(car[i].type==0){ //if car is red
      red_waiting++;
      //if the first car from file is red, then red cars start passing the bridge
    }
  }
//main is waiting all threads to close
mysem_down(main_sem.semid);
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; int up_flag=0;
      mysem_down(blue_sem.semid);
      blue_remain++;                                    //a blue car is over the bridge
      blue_waiting--;
      change_counter++;
      if(change_counter!=bridge_size && blue_waiting!=0){      //If bridge is not full ,wake up another blue car
            up_flag=1;
            mysem_up(blue_sem.semid);
       }
       sleep(thread_struct->sleep_time);                //A blue car is passing the bridge
       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,up(blue_sem)}
          //Change side from blue to red because counter=max
          else if(change_counter==bridge_size && red_waiting!=0){change_counter=0,up(red_sem)}
          //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,up(red_sem)}
    }
      //Do not do extra up's .Only one time a car can make an up
      else if(change_counter!=bridge_size && blue_waiting!=0 && up_flag==0){ mysem_up(blue_sem.semid);}
       //Last car wakes up main and return
       if(blue_waiting==0 && red_waiting==0 && blue_remain==0 && red_remain==0 && close_flag==1){up(main_sem)}
       //Some blue cars passed the bridge and this thread does not see neither blue nor red cars
       else if(blue_waiting==0 && red_waiting==0 && blue_remain==0 && red_remain==0 && close_flag==0){...}
}
```

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{passengers_num++;}
    //create a thread
}
void *train_func(void *arg){       //train function
    while(1){
        mysem_down(train_sem.semid);
        if(close_flag==1){
            //Wake up main and return
            mysem_up(main_sem.semid);
            break;
        }
        sleep(3);
        //Train is back,so wake up the first passenger
        mysem_up(empty_sem.semid);
    }
    return(NULL);
}
```

```
void *passengers_func(void *arg){                    //passenger's function
    mysem_down(passengers_sem);                      //Passenger is waiting
    waiting_passengers++;
    if(train_capacity==train_size){                  //Train is full,block until finishing the ride
        mysem_down(empty_sem);    close_flag=1;
    }
    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)){up(passengers_sem)}
            //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){up(passengers_sem),up(train_sem)}
        else:                                        //Main stopped reading numbers from file
            //Train is not full,wake up another passenger
            if(waiting_passengers < passengers_num && train_capacity < train_size){up(passengers_sem)}
            //Train is full,so train starts the ride,wake up a passenger and wake up train
            else if(waiting_passengers < (passengers_num) && train_capacity == train_size){up(passengers_sem,up(train_sem))}
            //Last extra thread wakes up train and return
            else if(waiting_passengers == passengers_num+1){close_flag=1;up(train_sem)}
            //The train is not full,so the train does not start,wake up the last extra thread
            else if(waiting_passengers == passengers_num && train_capacity < train_size){up(passengers_sem)}
            //Train starts the ride and terminates,wake up the last passenger and wake up train
            else if(waiting_passengers == passengers_num && train_capacity == train_size){up(passengers_sem),up(train_sem)}
}
```