REPORT

- **Exercise2**

  The main program takes a command line parameter ***n*** and creates two threads to which it is passed, as parameter, a structure in which there are:

  1. a counter that keeps track at which level of the tree we are;
  2. a vector that contains the sequence of thread identifiers from the root to the specific leaf;
  3. a counter of the not-empty cells and it is used as index when it is necessary to add a new element inside the vector.

```
typedef struct Arg{
    int nLevel;
    pthread_t* v;
    int index;
}Argument;

int main(int argc, char** argv){

    if(argc != 2){
     fprintf(stderr, "Error! -> Usage:%s n\n", argv[0]);
     return 1;
    }

    ...

    //Initialization
    n = atoi(argv[1]);
    arg.v = (pthread_t*)malloc(sizeof(pthread_t)*n+2);
    sem = (sem_t*)malloc(sizeof(sem_t));

    sem_init(sem, 0, 1);

    arg.index = 0; //How many thread ids have been stored
    arg.v[0] = pthread_self();

    arg.nLevel = 1;

    pthread_create(&t1, NULL, C, (void*) &arg);
    pthread_create(&t2, NULL, C, (void*) &arg);

    sleep(1); //Waste a bit of time before to free.

    free(arg.v);
    pthread_exit(0);
}
```

Each child, increments the counter of the not-empty cells and adds its own thread identifier, then generates other two threads until have been generated 2^n leaves. In order to meet this last constraint, in the child body is checked the number of the level where we are and if we are at the last level, the child will print its own entire tree. Furthermore, in this solution it is added a semaphore in order to be sure to print before the left subtree and then the right one.

```
void* C(void* a){

    ...

    arg.index++;
    arg.v[arg.index] = pthread_self();

    ...

    sem_wait(sem);//The semaphore needs in order to print before
                  //the left SubTree and then the right one.
    if(arg.nLevel < n){
     arg.nLevel++;

     pthread_create(&t1, NULL, C, (void*) &arg);
     pthread_create(&t2, NULL, C, (void*) &arg);

     sleep(1);   //It needs to give time to copy the memory
                //address of the vector before to free it.
    }else{
     int i = 0;
     printf("Thread Tree: ");
     for(i = 0; i < n+1; i++){
          printf("%ld ", arg.v[i]);
     }
     printf("\n");
    }
    free(arg.v);
    sem_post(sem);
    pthread_exit(0);
}
```