Esercitazione Posix

1. Si vuole realizzare una funzionalità “barrier” utlizzando la libreria posix thread (senza utilizzare la specifica funzionalità “barrier” offerta della libreria) che permetta ai thread di sincronizzarsi in uno specifico punto. In particolare, si consideri il seguente codice:

```
int nThread = 5;

.... dichiarazione variabili


void initBarrier(){
    ....
}

void barrier(){
    ....
}

void destroyBarrier(){
    ....
}

void* threadFunc(void* arg) {
    printf("inizio\n");
    barrier();
    printf("fine\n");
    return NULL;
}



int main(int argc, char* argv[]) {
    pthread_t th[nThread];
    initBarrier();
    for (int i = 0; i < nThread; i++) {
        pthread_create(&th[i], NULL, &threadFunc, NULL);

    }

    for (int i = 0; i < nThread; i++) {
        pthread_join(th[i], NULL);
    }
    destroyBarrier();
    return 0;
}
```

Implementare le funzioni `initBarrier()`,`barrier()` e `destroyBarrier()` (ed eventuali dichiarazioni di variabili) in modo tale che i threads si sincronizzino alla

chiamata di barrier. In particolare, l'output atteso del codice di esempio prevederebbe quindi prima 5 stampe "inizio" seguite da 5 stampe "fine".

2.) L'esecuzione del seguente programma:

```
void* run(void* arg) {
      int* p = (int*)arg;
      sleep(1);

      sleep(4-(*p));

}

int main(int argc, char* argv[]) {
      pthread_t thid;
      int i = 1;
      pthread_create(&thid, NULL, &run, &i);
      sleep(1);
      i++;
      sleep(i);
      pthread_join(thid, NULL);

   return 0;
}
```

Su una architettura quad-core, durerà all'incirca:

a. Poco più di 3 secondi
b. Poco più di 4 secondi
c. La durata può cambiare ad ogni run
d. Poco più di 7 secondi

Signature Posix

```
//creazione thread
int pthread_create(pthread_t * thread,
                        const pthread_attr_t * attr,
                        void * (*start_routine)(void *),
                        void *arg);


// join
int pthread_join( pthread_t thread,void** value_ptr );
```

```
//mutex
int pthread_mutex_init(pthread_mutex_t *mutex,
     pthread_mutex_attr *attr);
int pthread_mutex_lock(pthread_mutex_t* mutex );
int pthread_mutex_unlock(pthread_mutex_t* mutex );
int pthread_mutex_destroy(pthread_mutex_t *mutex);

//condition
int pthread_cond_init( pthread_cond_t *cond,
     pthread_condattr_t *cond_attr )
int pthread_cond_destroy( pthread_cond_t *cond )
pthread_cond_wait(&a_c_v,&a_mutex);
pthread_cond_signal (pthread_cond_t *cond)
pthread_cond_broadcast (pthread_cond_t *cond)
```