6. [2,0 pontos] Agora relembre do seguinte exercício apresentado em aula/laboratório:

"- A partir da thread principal criar N threads;
- Cada thread executa, basicamente, a mesma tarefa que consiste em incrementar uma variável global inicializada com valor zero (0); no entanto, a cada rodada envolvendo todas as threads, cada thread incrementa a variável global uma única vez. Além disso, a alternância entre as threads dá-se sempre em ordem crescente de identificadores. Assumir identificadores das threads incrementais iniciando-se a primeira thread com ID=0. Exemplo: assumindo que existem 3 threads (ids 0, 1 e 2), ter-se-á a seguinte apresentação de incremento da variável global:
- thread 0: global = 1;
- thread 1: global = 2;
- thread 2: global = 3;
- thread 0: global = 4;
- thread 1: global =5;..."

Pois bem, ***SO Kid*** codificou na linguagem C (plataforma Linux) mas, apesar de compilar corretamente, a execução do seu programa não apresenta o resultado esperado. **Utilizando-se do fragmento principal do código abaixo, identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões), mas sem remover ou acrescentar novas estruturas de dados. Caso deseje, realize a(s) correção(ões) diretamente no código abaixo.**

```
...
void *mythread(void *data);

#define N 3 // number of threads
#define MAX 10
// vetor de semáforos (uma entrada por thread)
// utilizado para controlar o rodizio
sem_t turn[N];

int x = 0;

int main(void) {
  pthread_t tids[N];
  int i=0;

  // inicializa vetor de semáforos
  for(i=0; i<N; i++) {
    sem_init(&turn[i], 0, 0);
  }

  for(i=0; i<N; i++) {
    int *j = malloc(sizeof(int));
    *j = i;
    pthread_create(&tids[i], NULL, mythread, (void *)j);
  }

  for(i=0; i<N; i++) {
    pthread_join(tids[i], NULL);
    printf("Thread id %ld retornou \n", tids[i]);
  }

return(1);


}
```

```
void *mythread(void *data) {

  int id;

  id = *((int *) data);

  while(x < MAX) {
    sem_wait(&turn[id]);
    x++;
    printf("\n thread %d: global = %d", id, x);
    sem_post(&turn[(id+1)%N]);
    sleep(2);
  }

  pthread_exit(NULL);

}
```

(2,0)

7. [2,0 pontos] ***SO Kid*** também se dedicou a outro exercício apresentado em aula/laboratório. Semelhante ao exercício anterior, agora há um conjunto de *threads* tentando manipular uma variável global mas sem nenhuma ordem preestabelecida. No entanto, **apenas exige-se que se garanta a exclusão mútua ao se atualizar a variável global**. Após inicializar a execução do programa de *SO Kid*, observa-se que o mesmo não produz o resultado esperado e sequer finaliza. **Utilizando-se do fragmento principal do código de SO Kid abaixo, identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões).**
→ **Caso deseje, realize a(s) correção(ões) diretamente no código abaixo.**

```
void *mythread(void *data);
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;

#define N 3 // number of threads
#define MAX 10
int x = 0;

int main(void) {
   pthread_t tids[N];
   int i;

    for(i=0; i<N; i++) {
        pthread_create(&tids[i], NULL, mythread, NULL);
    }

    for(i=0; i<N; i++) {
        pthread_join(tids[i], NULL);
        printf("Thread id %ld returned\n", tids[i]);
    }
    return(1);
}

void *mythread(void *data) {


 while(x < MAX) {
   pthread_mutex_lock(&count_mutex);
   x++;
   printf("Thread ID%ld: x is now %d.\n", pthread_self(), x);
   sleep(2);
}

pthread_exit(NULL);
```

→ pthread_mutex_unlock(&cont_mutex); ✓