Pthreads and Concurrency
(using the POSIX API)

---

1. Consider the following program that creates two threads from the main thread of a process and uses them to increment a counter. Compile the program and run it. Read the code carefully.

```
#include <stdio.h>
#include <pthread.h>

int count=0;

void * inc(void * arg)
{
  for (int i=0; i<1000000; i++) count++;
  return NULL;
}

int main()
{
  printf("Start: %d\n",count);

  pthread_t tid1, tid2;
  pthread_create(&tid1,NULL,inc,NULL);
  pthread_create(&tid2,NULL,inc,NULL);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);


  printf("End: %d\n",count);
}
```

Can you think of a reason why the result may not be what you expect and may even vary between executions?

2. The following example shows how to ensure mutual exclusion in the previous program, when updating the counter, to obtain a correct behavior. Nevertheless, in this solution, each thread is forced to execute the entire loop with exclusive access to the counter. Change the program so that the mutual exclusive operation on the counter is finer grained. Compare the execution times of the original and the improved programs.

```
#include <stdio.h>
#include <pthread.h>

int count=0;
pthread_mutex_t lock;

void * inc(void * arg)
{
  pthread_mutex_lock(&lock);
  for (int i=0; i<1000000; i++) count++;
  pthread_mutex_unlock(&lock);
  return NULL;
}

int main()
{
  printf("Start: %d\n",count);

  pthread_mutex_init(&lock,NULL);

  pthread_t tid1, tid2;
  pthread_create(&tid1,NULL,inc,NULL);
  pthread_create(&tid2,NULL,inc,NULL);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);

  pthread_mutex_destroy(&lock);

  printf("End: %d\n",count);
}
```

Change the program again o that one thread increments the counter and the other decrements it. The final result should be 0 if the same number of increments and decrements are performed. Add some code to print a '.' character every time the counter has a negative value.

3. The following program ensures that the counter never has a negative value. Change it so that it continues to work correctly but with less calls to `pthread_cond_signal`.

```
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

int count=0;
pthread_mutex_t lock;
pthread_cond_t not_zero;

void * inc(void * arg)
{
  for (int i=0; i<1000000; i++)
  {
    pthread_mutex_lock(&lock);
    count++;
    pthread_cond_signal(&not_zero);
    pthread_mutex_unlock(&lock);
  }
  return NULL;
}

void * dec(void * arg)
{
  for (int i=0; i<1000000; i++)
  {
    pthread_mutex_lock(&lock);
    while (count == 0)
      pthread_cond_wait(&not_zero, &lock);
    count--;
    if (count<0) write(1,".",1);
    pthread_mutex_unlock(&lock);
  }
  return NULL;
}


int main()
{
  printf("Start: %d\n",count);

  pthread_mutex_init(&lock,NULL);
  pthread_cond_init(&not_zero,NULL);

  pthread_t tid1, tid2;
```

```
  pthread_create(&tid1,NULL,inc,NULL);
  pthread_create(&tid2,NULL,dec,NULL);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);

  pthread_mutex_destroy(&lock);

  printf("End: %d\n",count);
}
```