Q1. Write a C program to create two threads to show that they are running in quasi-parallel mode with separate thread ID under single core CPU and check the PID of the process and the the thread IDs of the two threads.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<pthread.h>

void* tf(void* arg)
{
      printf("\n Thread %lu is running.", pthread_self());
      sleep(1);
}

int main()
{

      pthread_t tid1, tid2;

      pthread_create(&tid1,NULL,tf,NULL);
      pthread_create(&tid2,NULL,tf,NULL);

      printf("\n Process %d is running.", getpid());
      sleep(1);

      pthread_join(tid1,NULL);
      pthread_join(tid2,NULL);
      return 0;
}
```

Q2. Write a C Program to create multiple threads in a single process and terminate the threads in reverse order.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<pthread.h>
#define THREAD_COUNT 5

void* tf(void* arg)
{
      printf("\n Thread %lu is running.", pthread_self());
      sleep(1);
}

int main()
{
      int i;
      pthread_t tid[THREAD_COUNT];

      for(i=0;i<THREAD_COUNT;i++)
      {
            pthread_create(&tid[i],NULL,tf,NULL);
      }

      printf("\n Process %d is running.", getpid());
      sleep(1);
```

```
      for(i=THREAD_COUNT-1;i>=0;i--)
      {
            pthread_join(tid[i],NULL);
      }
      return 0;
}
```

Q3. Write a C program to show that a thread executes separately from the main process and can acess the global variable of the process if any.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<string.h>
#include<pthread.h>

int run_now=1;

void* tf(void* arg)
{
      int print_count2 = 0; // local to thread funtion
      while(print_count2 < 20)
      {
            if (run_now == 2)
            {
                  printf("\t Thread %lu is running, run_now=%d,
print_count2=%d.", pthread_self(), run_now, print_count2);
                  run_now = 1; // run_now is the global variable accessible
to both threads
            }
            else
            {
                  sleep(1);
            }
            print_count2++;
      }
}

int main()
{
      pthread_t tid;
      //int ret=0;
      //void* result;

      pthread_create(&tid,NULL,tf,NULL);

      int print_count1 = 0; //local to main thread
      while(print_count1 < 20)
      {
            if (run_now == 1)
            {
                  printf("\n Process %d is running, run_now=%d,
print_count1=%d", getpid(), run_now, print_count1);
                  run_now = 2; // run_now is the global variable accessible
to both threads
            }
            else
```

```
            {
                  sleep(1);
            }
            print_count1++;
      }

      pthread_join(tid,NULL);

      return 0;
}
```