Class Assignment - 7

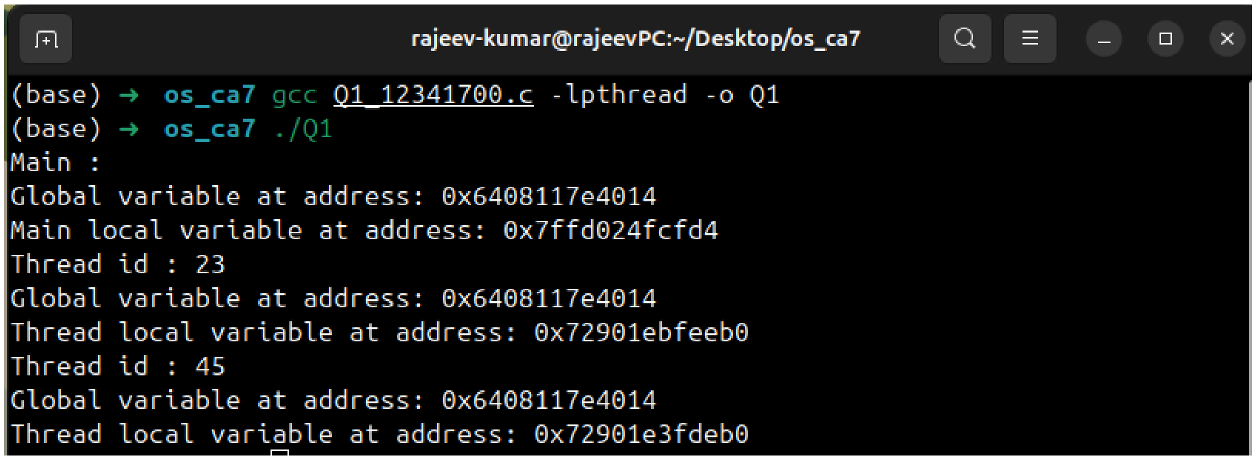
Name - Rajeev Kumar
ID - 12341700

- **Q1**

Code :

```
#include <pthread.h>

#include <stdio.h>

int global_var;

void* var_id(void* arg){

    int thread_local_var;

    int id=*(int*)arg;

    printf("Thread id : %d \nGlobal variable at address: %p \nThread local variable
at address: %p\n",id,(void*)&global_var,(void*)&thread_local_var);

    return NULL;

}

int main(){

    pthread_t tid1;

    pthread_t tid2;

    int id1=23;

    int id2=45;

    int main_local_var;

    printf("Main :\nGlobal variable at address: %p \nMain local variable at address:
%p \n",(void*)&global_var,(void*)&main_local_var);

    pthread_create(&tid1,NULL,var_id,&id1);

    pthread_create(&tid2,NULL,var_id,&id2);

    pthread_join(tid1,NULL);

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

OUTPUT :



```
(base) ➜  os_ca7 gcc Q1_12341700.c -lpthread -o Q1
(base) ➜  os_ca7 ./Q1
Main :
Global variable at address: 0x6408117e4014
Main local variable at address: 0x7ffd024fcfd4
Thread id : 23
Global variable at address: 0x6408117e4014
Thread local variable at address: 0x72901ebfeeb0
Thread id : 45
Global variable at address: 0x6408117e4014
Thread local variable at address: 0x72901e3fdeb0
```

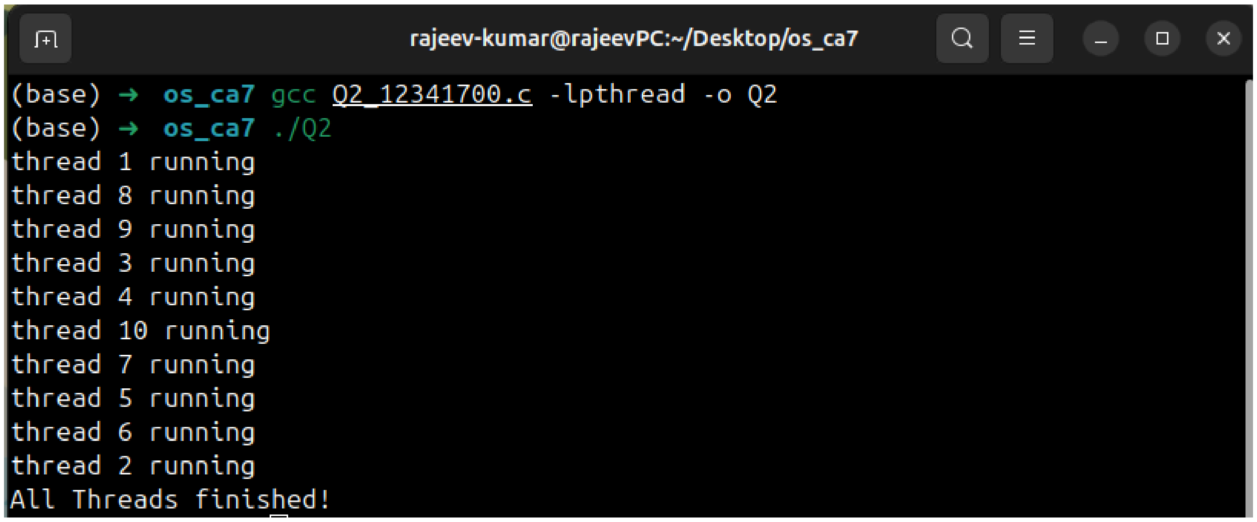
EXPLANATION :
The global variable address is the same in all threads because globals are shared across the process. The local variable addresses are different in each thread (including main) because each thread has its own stack.

- **Q2**

Code :

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

void* print_id(void* arg){
    int id=*(int*)arg;
    printf("thread %d running\n",id);
    return NULL;
}

int main(){
        pthread_t tid[10];
        int mid[10];
        for(int i=0;i<10;i++){
            mid[i]=i+1;
            pthread_create(&tid[i],NULL,print_id,&mid[i]);
        }
        for(int i=0;i<10;i++){
            pthread_join(tid[i],NULL);
        }
        printf("All Threads finished! \n");
        return 0;
    }
```

OUTPUT :



```
rajeev-kumar@rajeevPC:~/Desktop/os_ca7
(base) ➜  os_ca7 gcc Q2_12341700.c -lpthread -o Q2
(base) ➜  os_ca7 ./Q2
thread 1 running
thread 8 running
thread 9 running
thread 3 running
thread 4 running
thread 10 running
thread 7 running
thread 5 running
thread 6 running
thread 2 running
All Threads finished!
```

EXPLANATION :
The order varies because thread scheduling is non-deterministic; the OS decides which thread runs first, so thread execution order can change each run.

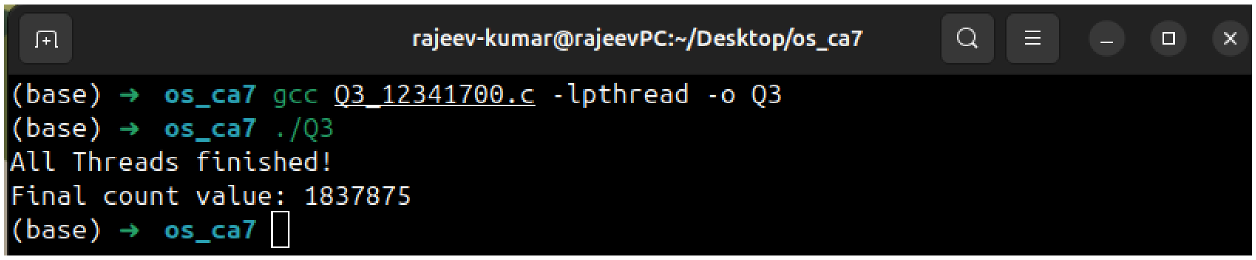
- **Q3**

Code :

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

int count=0;
```

```
void* increment_count(void* arg){
    int id=*(int*)arg;
    for(int i=0;i<1000000;i++){
        count=count+1;
    }
    return NULL;
}

int main(){
    pthread_t tid[10];
    int mid[10];
    for(int i=0;i<10;i++){
        mid[i]=i+1;
        pthread_create(&tid[i],NULL,increment_count,&mid[i]);
    }
    for(int i=0;i<10;i++){
        pthread_join(tid[i],NULL);
    }
    printf("All Threads finished! \n");
    printf("Final count value: %d\n",count);
    return 0;
}
```

OUTPUT :



```
(base) ➜  os_ca7 gcc Q3_12341700.c -lpthread -o Q3
(base) ➜  os_ca7 ./Q3
All Threads finished!
Final count value: 1837875
(base) ➜  os_ca7
```

EXPLANATION :

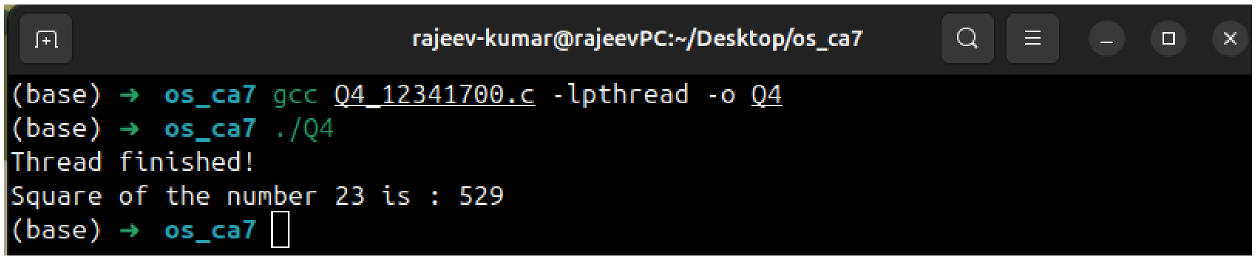
The final value differs because threads access and modify global_counter at the same time without synchronization, causing lost updates.
A race condition happens when multiple threads access shared data concurrently and the result depends on the timing of their execution, leading to unpredictable or incorrect results.

- **Q4**

Code :

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

void* compute_square(void* arg){
   int num=*(int*)arg;
   int * result=malloc(sizeof(int));
   *result=num*num;
   return result;
}


int main(){
   pthread_t tid;
   int n=23;
   pthread_create(&tid,NULL,compute_square,&n);
   int* result;
   pthread_join(tid,(void**)&result);
   printf("Thread finished! \n");
   printf("Square of the number %d is : %d\n",n,*result);
   free(result);
   return 0;
}
```

OUTPUT :



```
(base) ➜ os_ca7 gcc Q4_12341700.c -lpthread -o Q4
(base) ➜ os_ca7 ./Q4
Thread finished!
Square of the number 23 is : 529
(base) ➜ os_ca7
```

EXPLANATION :

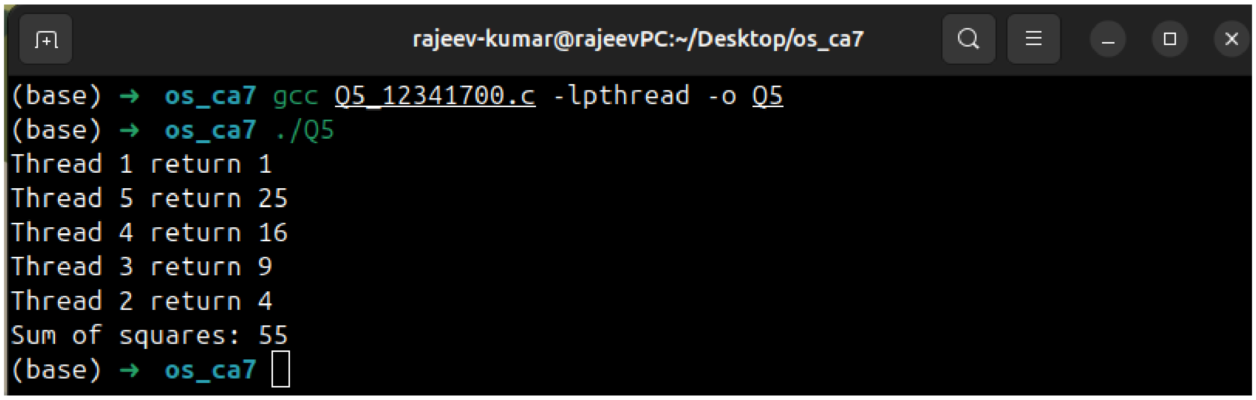
The thread computes the square of num and returns it via dynamically allocated memory. pthread_join retrieves the result, which is printed and then freed to avoid memory leaks.

- **Q5**

Code :

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
void* square(void* arg){
    int num = *(int*)arg;
    int* result = malloc(sizeof(int));
    *result = num * num;
    printf("Thread %d return %d\n", num, *result);
    return (void*)result;
}
int main(){
    pthread_t tids[5];
    int args[5];
    for(int i = 0; i < 5; i++){
        args[i] = i + 1;
        pthread_create(&tids[i], NULL, square, (void*)&args[i]);
    }
    int* results[5];
    int sum = 0;
    for(int i = 0; i < 5; i++){
        pthread_join(tids[i], (void**)&results[i]);
        sum += *results[i];
        free(results[i]);
    }
    printf("Sum of squares: %d\n", sum);
    return 0;
}
```

OUTPUT :



```
(base) ➜  os_ca7 gcc Q5_12341700.c -lpthread -o Q5
(base) ➜  os_ca7 ./Q5
Thread 1 return 1
Thread 5 return 25
Thread 4 return 16
Thread 3 return 9
Thread 2 return 4
Sum of squares: 55
(base) ➜  os_ca7 
```

EXPLANATION :

Each thread computes the square of a number and returns it via dynamically allocated memory. pthread_join retrieves the result, which is added to sum. Memory is freed afterward. This allows threads to safely return values to main.