



[Workshop Report-2]

Student Id : NP03A190051
Student Name : Rishab Sharma
Section : L5CG7
Lecturer : Jnaneshwor Bohara
Module Leader : Hiran Patel
Submitted on : 22nd September, 2021

1. Write a multithreaded C program to print out all the prime numbers between 1 to 10000. Use exactly 3 threads.

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

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

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

```
void *printPrimeNumbers() {
```

```
    int flag, i, j;
```

```
    for(i=1; i<=10000; i++) {
```

```
        flag = 1;
```

```
        if(i==1 || i==0) {
```

```
            continue;
```

```
        }
```

```
        for (j = 2; j <= i/2; j++) {
```

```
            if(i % j == 0) {
```

```
                flag = 0;
```

```
                break;
```

```
            }
```

```
        }
```

```
        if(flag == 1) {
```

```
            printf("%d\n", i);
```

```
        }
```

```
    }
```

```
}
```

```
int main() {
```

```
    int n;
```

```

pthread_t thread_id;

printf("Enter the number of threads u want to use: ");

scanf("%d", &n);

int i;

for(i = 0; i < n; i++) {
    pthread_create(&thread_id, NULL, printPrimeNumbers, &thread_id);
    pthread_join(thread_id, NULL);
}

exit(0);
}

```



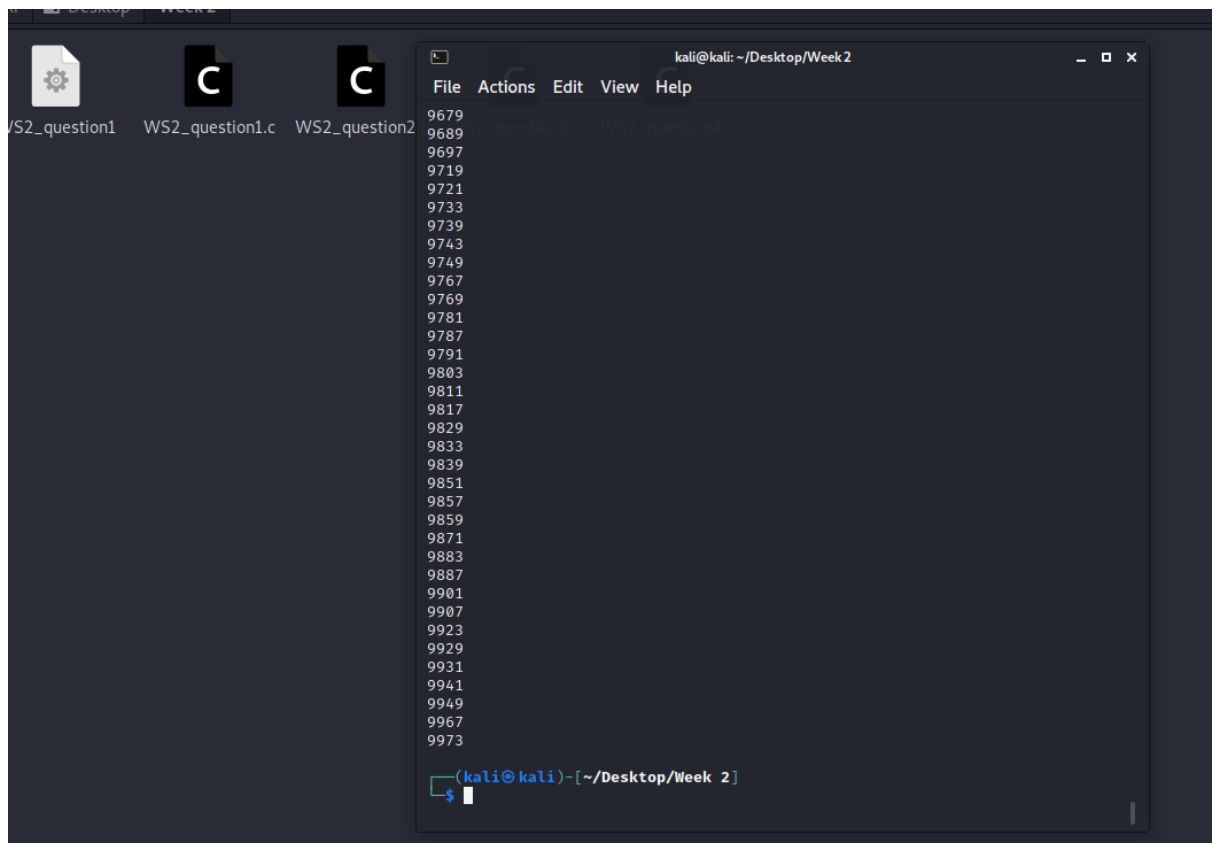
The screenshot shows a Kali Linux desktop environment. On the left, a file manager window displays the contents of the 'Week 2' directory, which includes files named 'WS2_question1', 'WS2_question1.c', and 'WS2_question2'. On the right, a terminal window is open, showing the compilation and execution of a C program. The terminal output is as follows:

```

kali@kali: ~/Desktop/Week 2
$ gcc WS2_question1.c -o WS2_question1 -lpthread

kali@kali: ~/Desktop/Week 2
$ ./WS2_question1
Enter the number of threads u want to use: 3
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
107
109
113
127

```



2. Convert this program to prompt the user for a number and then to create the number of threads the user has specified to find the prime numbers.

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

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

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

```
void *printPrimeNumbers() {
```

```
    int flag, i, j;
```

```
    for(i=1; i<=10000; i++) {
```

```
        flag = 1;
```

```
        if(i==1 || i==0) {
```

```
            continue;
```

```
        }
```

```

    for (j = 2; j <= i/2; j++) {
        if(i % j == 0) {
            flag = 0;
            break;
        }
    }

    if(flag == 1) {
        printf("%d\n", i);
    }
}

int main() {
    int n;
    pthread_t thread_id;
    printf("Enter the number of threads u want to use: ");
    scanf("%d", &n);
    int i;
    for(i = 0; i < n; i++) {
        pthread_create(&thread_id, NULL, printPrimeNumbers, &thread_id);
        pthread_join(thread_id, NULL);
    }
    exit(0);
}

```

```
kali@kali: ~/Desktop/Week 2
File Actions Edit View Help
9967
9973
(kali@kali)~[~/Desktop/Week 2]
$ gcc WS2_question2.c -o WS2_question2 -lpthread
(kali@kali)~[~/Desktop/Week 2]
$ ./WS2_question2
Enter the number of threads u want to use: 5
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
107
109
```

3. Convert the program in (2) so that each thread returns the number of prime numbers that it has found using `pthread_exit()` and for main program to print out the number of prime number that each thread has found.

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

#include<stdlib.h>

#include<pthread.h>
```

```
typedef struct{
    int start;
    int end;
}
```

```
range;
```

```
void *prime(void *ptr)
{
```

```

int i,c;
int *counter;
counter = malloc(sizeof(int));
int counts = 0;

range *p = ptr;
int nstart=p->start, nend=p->end;

for(i=nstart; i<=nend; i++){
    for(c=2; c<=i-1; c++) {
        if ( i%c==0 ) {
            break;
        }

        if ( c==i ) {
            counts = counts + 1;
        }
    }

    *counter = counts;
    pthread_exit(counter);
}

```

```

void main(){
    void *pointer;
    int count = 1000;
    int thread;
    printf("Number of threads:");

```

```

scanf("%d", &thread);
int sliceList[thread]; //3 slice boxes

int rem = count % thread;

for (int i =0; i<thread; i++){
    sliceList[i] = count / thread;
}

// equally distribute the remainders in each thread

for(int j = 0; j<rem; j++){
    sliceList[j] = sliceList[j] + 1;
}

int startList[thread];
int endList[thread];
// start = 0
// end = 3333

for(int l = 0 ;l < thread; l++){
    // if it is the start
    if(l == 0){
        startList[l] = 0;
    }
    // endList[l-1] = 0 which is starting index gets the endlist value + 1
    else{
        startList[l] = endList[l-1] + 1;
    }
}

```



```

        endList[l] = startList[l] + sliceList[l] - 1; //3334+3333-1 -> 6666
    }

    range nums[thread];

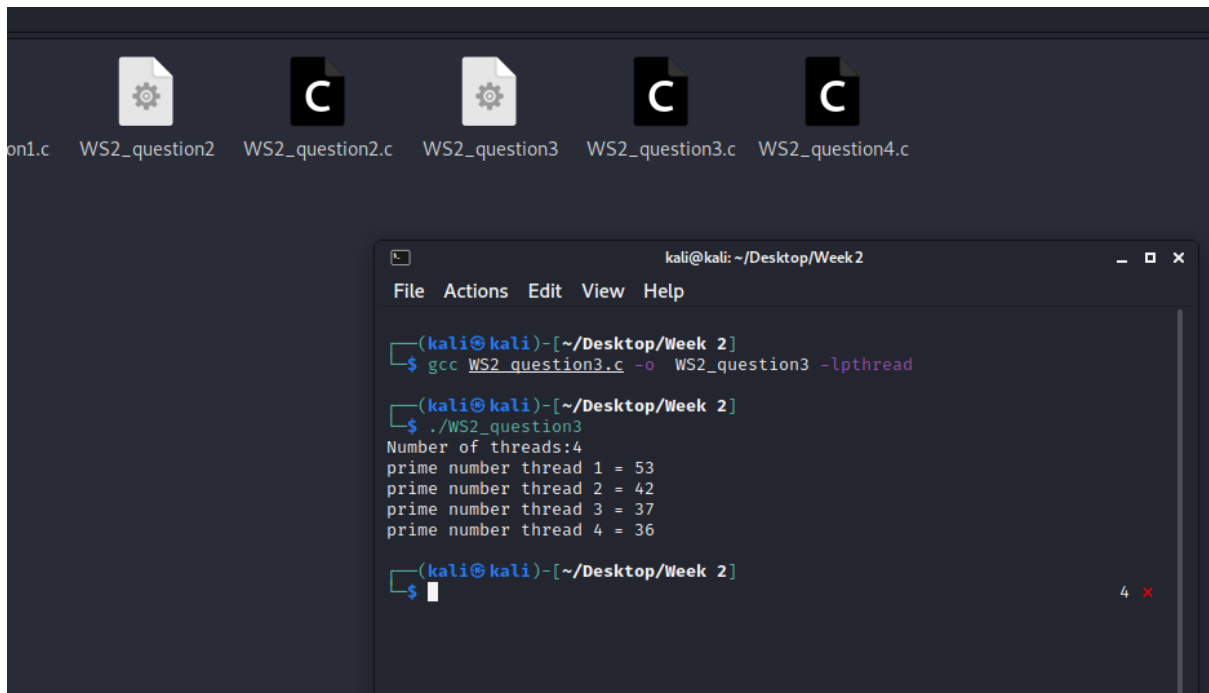
    for(int k = 0; k< thread; k++){
        nums[k].start = startList[k];
        nums[k].end = endList[k];
    }

    // create thread according to nummber of thread by user
    pthread_t threadIDs[thread];

    for(int n = 0; n<thread; n++){
        pthread_create(&threadIDs[n], NULL, prime, &nums[n]);
    }

    for(int n = 0; n<thread; n++){
        pthread_join(threadIDs[n],&pointer);
        printf("prime number thread %d = %d\n",n+1,*(int*)pointer);
    }
}

```



4. Convert the program in (3) to use `pthread_cancel()` to cancel all threads as soon as the 5th prime number has been found.

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

```
int counter=0;
```

```
typedef struct{
    int start;
    int end;
}count;
```

```
void *countPrime(void *p){
    count *parameter;
    parameter = (count *)p;
    int start = parameter->start;
```

```

int end = parameter->end;
int i,c;
for(i=start;i<end;i++){
    for(c=2;c<=i-1;c++){
        if(i%c==0){
            break;
        }
    }
    if (counter >= 5)
    {
        pthread_cancel(pthread_self());
    }
    if(c == i){
        counter+=1;
        printf("%d is prime.\n",i);
    }
}
}

```

```

void main()
{
    int n,i;
    printf("Enter number of threads:");
    scanf("%d",&n);
    pthread_t thread[n];
    count range[n];
    int split = 1000/n;

    for(i=0;i<n;i++){
        if(i==0){

```

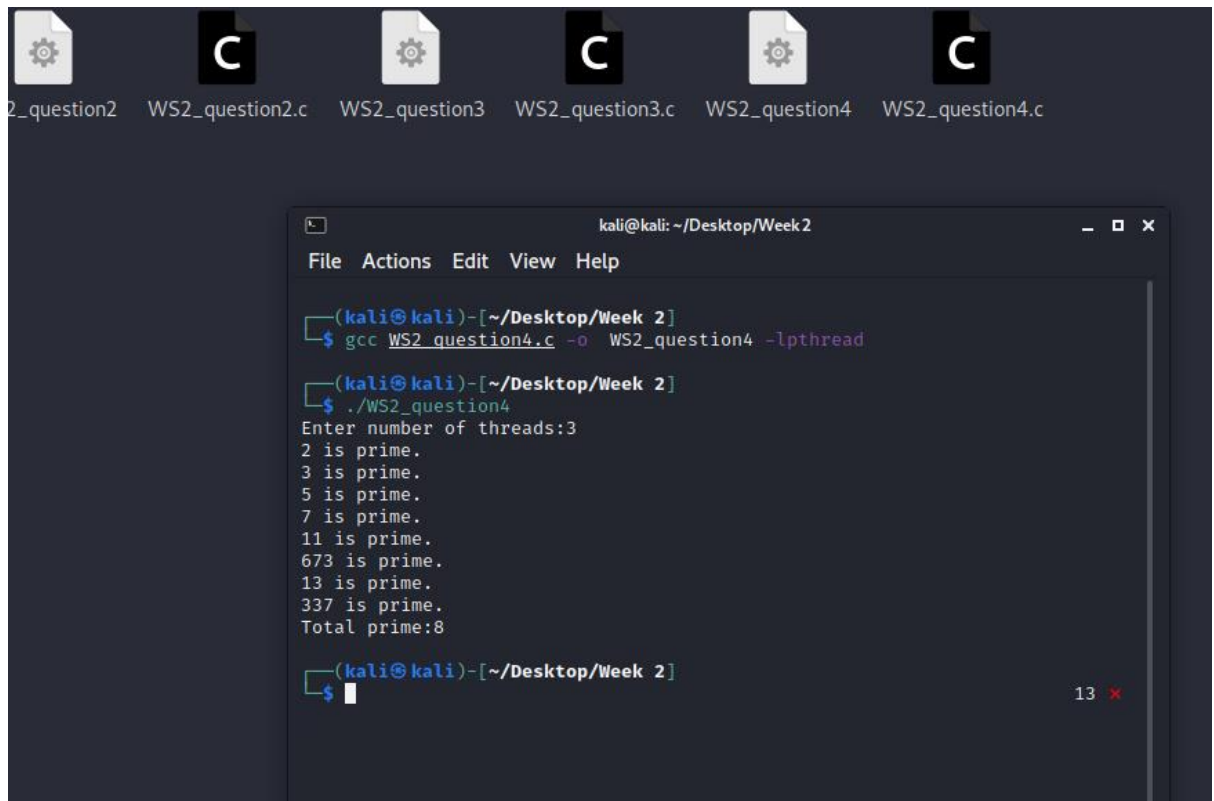
```

        range[i].start = 1;
        range[i].end = split;
    }
    else{
        range[i].start = range[i-1].end + 1;
        range[i].end = range[i-1].end + split;
    }
}

for ( i = 0; i < n; i++)
{
    pthread_create(&thread[i],NULL,countPrime,(void *)&range[i]);
}

for ( i = 0; i < n; i++)
{
    pthread_join(thread[i],NULL);
}
printf("Total prime:%d",counter);
}

```



The screenshot shows a Kali Linux desktop environment. At the top, there is a taskbar with several icons: a gear icon, a black square with a white 'C', a gear icon, a black square with a white 'C', a gear icon, and a black square with a white 'C'. Below the taskbar, there are six files: '2_question2', 'WS2_question2.c', 'WS2_question3', 'WS2_question3.c', 'WS2_question4', and 'WS2_question4.c'. In the center, there is a terminal window titled 'kali@kali: ~/Desktop/Week 2'. The terminal window has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal output shows the following commands and results:

```
(kali@kali)-[~/Desktop/Week 2]
$ gcc WS2_question4.c -o WS2_question4 -lpthread

(kali@kali)-[~/Desktop/Week 2]
$ ./WS2_question4
Enter number of threads:3
2 is prime.
3 is prime.
5 is prime.
7 is prime.
11 is prime.
673 is prime.
13 is prime.
337 is prime.
Total prime:8

(kali@kali)-[~/Desktop/Week 2]
$
```

In the bottom right corner of the terminal window, there is a red '13' and a red 'x' icon.