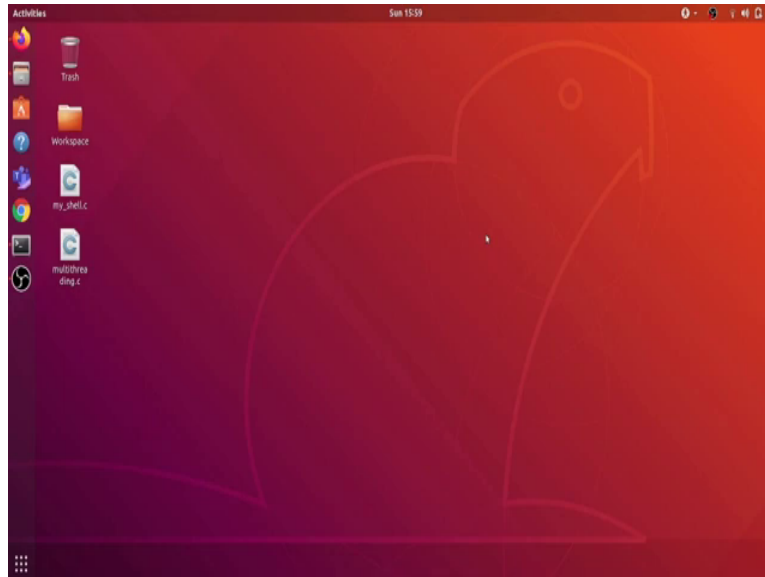


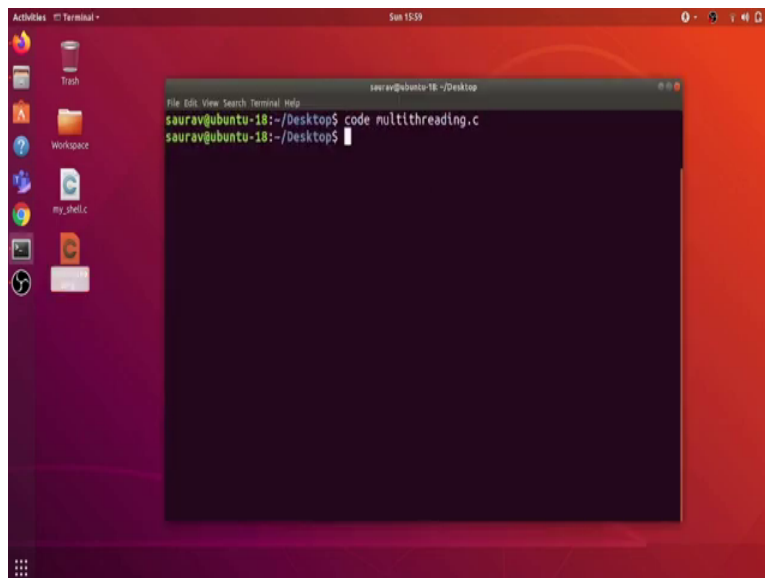
Design and Engineering of Computer Systems
Professor Mythilli Vutukuru
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Multithreading

(Refer Slide Time: 00:15)



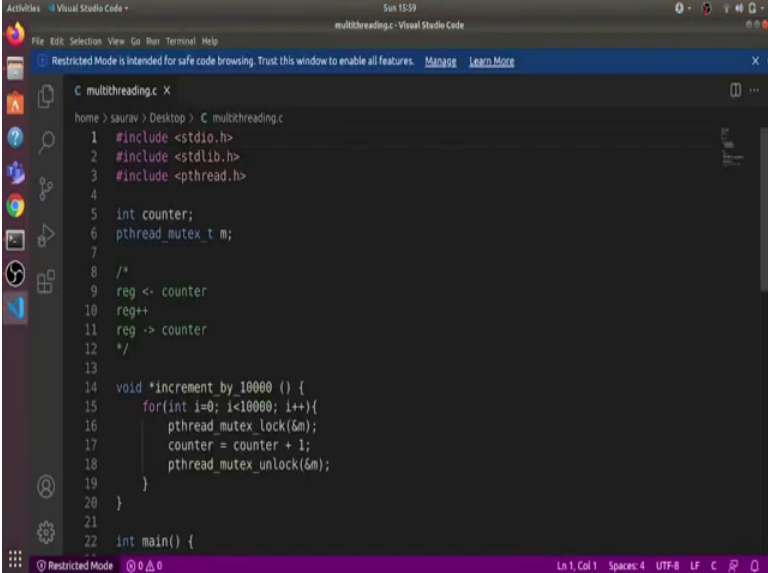
Hi everyone, in this video, we will learn about multi-threading, we will write the C programme, which spawns multiple threads and which access is a global counter in both the threads.

(Refer Slide Time: 00:33)



So, I have written this multi-threading in dot C programme. Let us, open it in visual code.

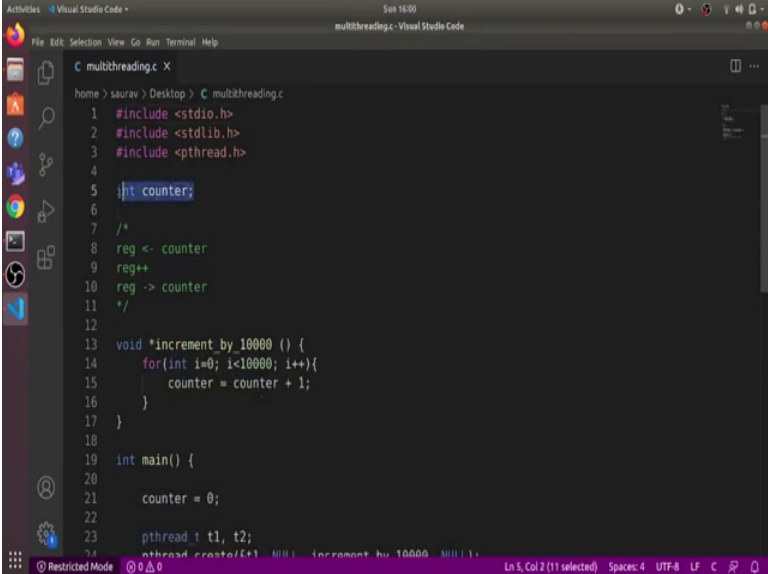
(Refer Slide Time: 00:37)



```
home > saurav > Desktop > C multithreading.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9 reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
```

So, this is the multi-threading dot C programme.

(Refer Slide Time: 00:42)



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27     pthread_join(t1, NULL);
28     pthread_join(t2, NULL);
29
30     printf("Counter: %d\n", counter);
31     return 0;
32 }
```

```
Activities Visual Studio Code Sun 16:00
multithreading.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

C multithreading.c X
home > saurav > Desktop > C multithreading.c

7  /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30 }
```

Ln 21, Col 17 (12 selected) Spaces: 4 UTF-8 LF C

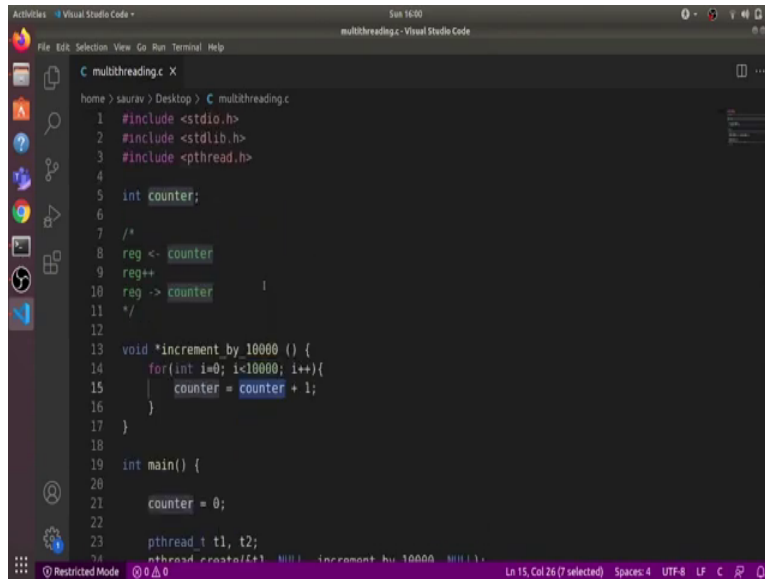
```
Activities Visual Studio Code Sun 16:00
multithreading.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

C multithreading.c X
home > saurav > Desktop > C multithreading.c

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  int counter;
6
7  /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30 }
```

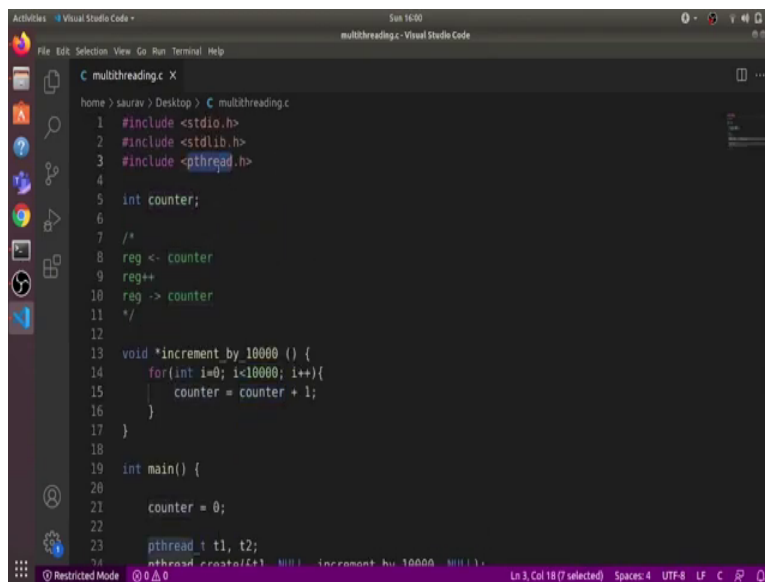
Ln 13, Col 25 (18 selected) Spaces: 4 UTF-8 LF C



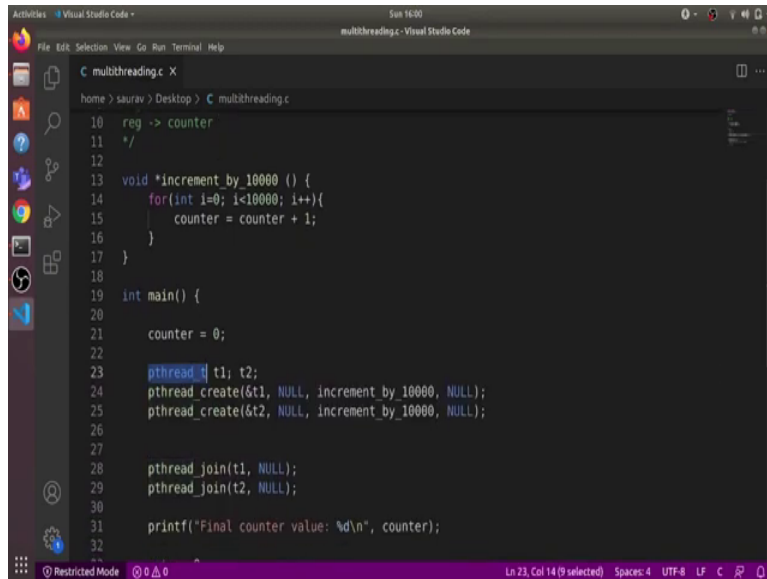
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

Let us, go over the code. We have this global counter variable and in the main function we initialise this counter variable to 0, then there is this increment by 10,000 function, which increments this counter variable by 10,000. So, it runs a loop for 10,000 times and each time it adds one to the counter variable.

(Refer Slide Time: 01:03)

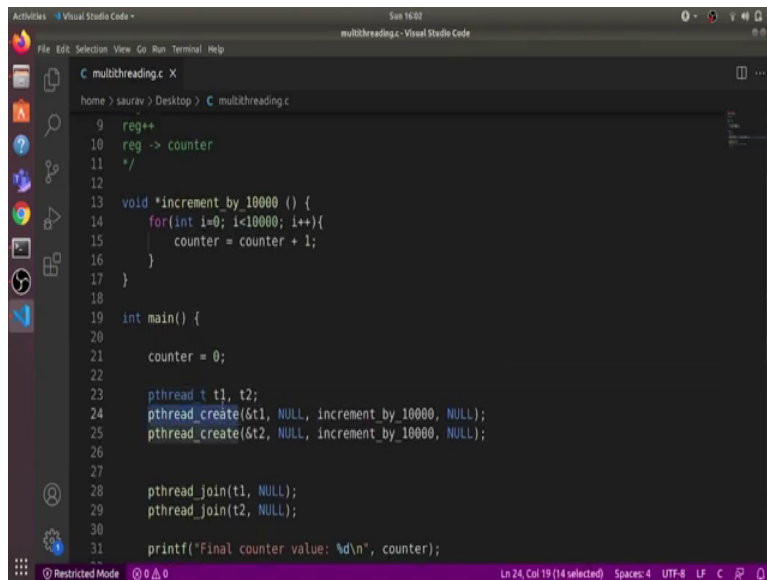


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```



```
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32 }
```

Ln 23, Col 14 (9 selected) Spaces: 4 UTF-8 LF C



```
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32 }
```

Ln 24, Col 19 (14 selected) Spaces: 4 UTF-8 LF C

```
Activities Visual Studio Code Sun 16/02
multithreading.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

C multithreading.c X
home > saurav > Desktop > C multithreading.c

9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32 }
```

Ln 24, Col 29 (4 selected) Spaces: 4 UTF-8 LF C

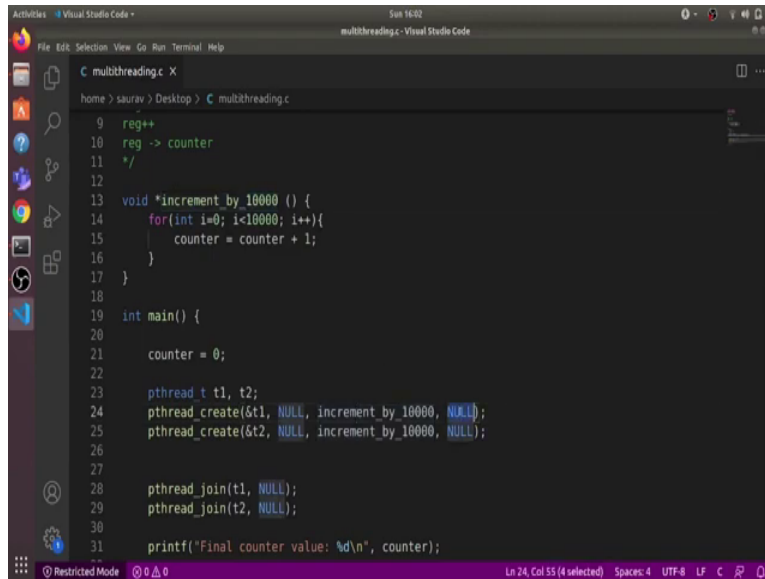
```
Activities Visual Studio Code Sun 16/02
multithreading.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

C multithreading.c X
home > saurav > Desktop > C multithreading.c

9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32 }
```

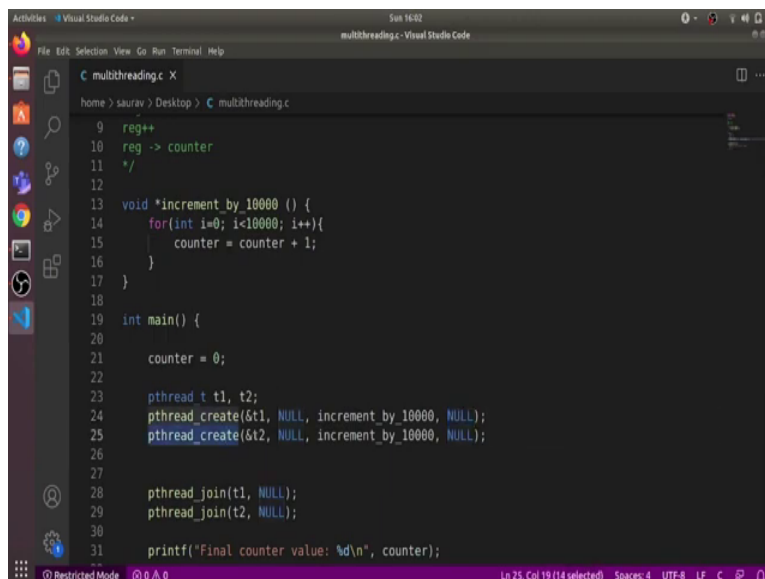
Ln 24, Col 49 (18 selected) Spaces: 4 UTF-8 LF C



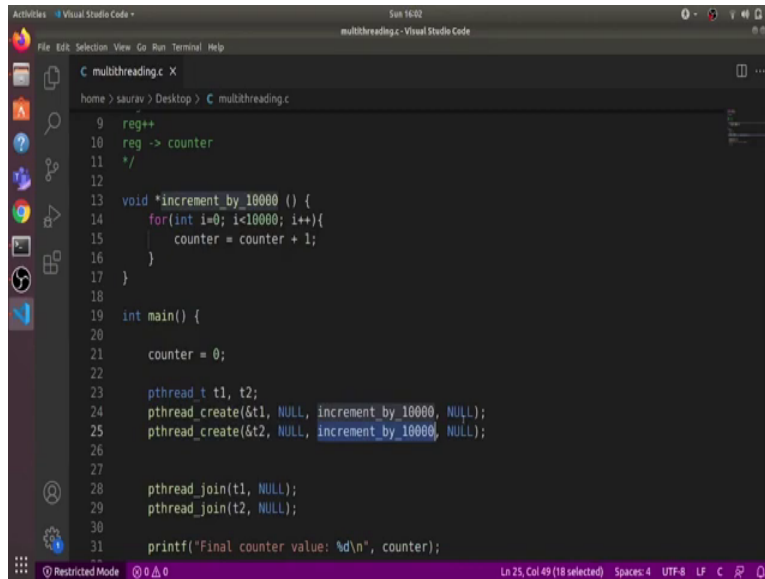
```
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25
26
27     pthread_join(t1, NULL);
28     pthread_join(t2, NULL);
29
30     printf("Final counter value: %d\n", counter);
31 }
```

So here we use pthread library, which is the POSIX threads library, which allows us to create multiple threads, we first declare two threads, t1 and t2. So, we use the pthread_create function to create the threads. So, first you create t1 thread and we do not specify any attribute. And we give it increment by 10,000 function so that this thread runs this increment by 10,000 function. And because increment by 10,000, does not take any argument, so we specified null here.

(Refer Slide Time: 01:34)

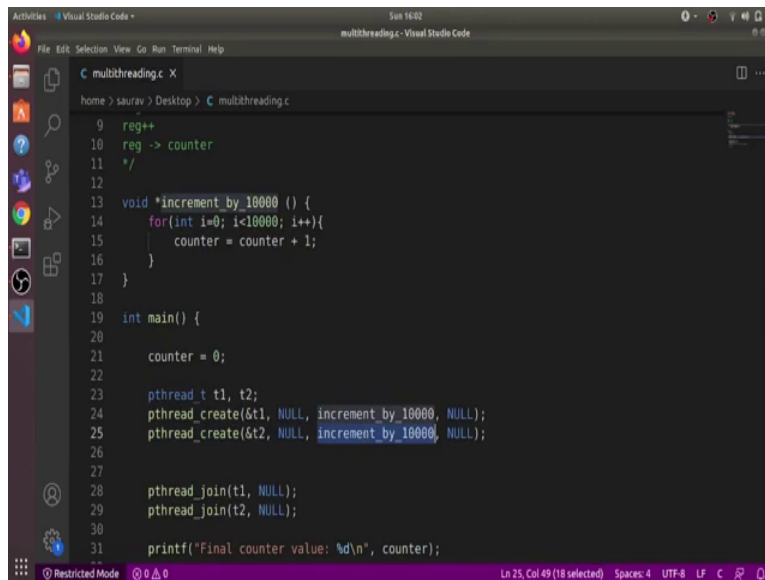


```
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25
26
27     pthread_join(t1, NULL);
28     pthread_join(t2, NULL);
29
30     printf("Final counter value: %d\n", counter);
31 }
```



```
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);

```

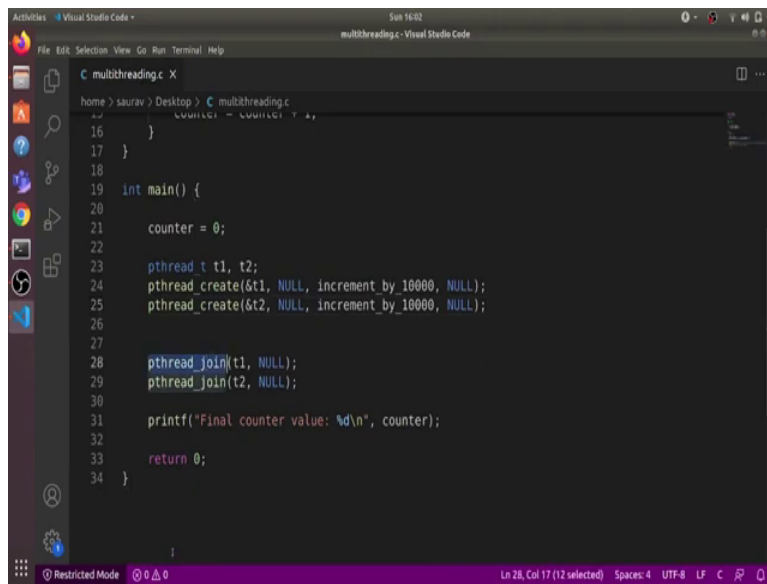


```
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);

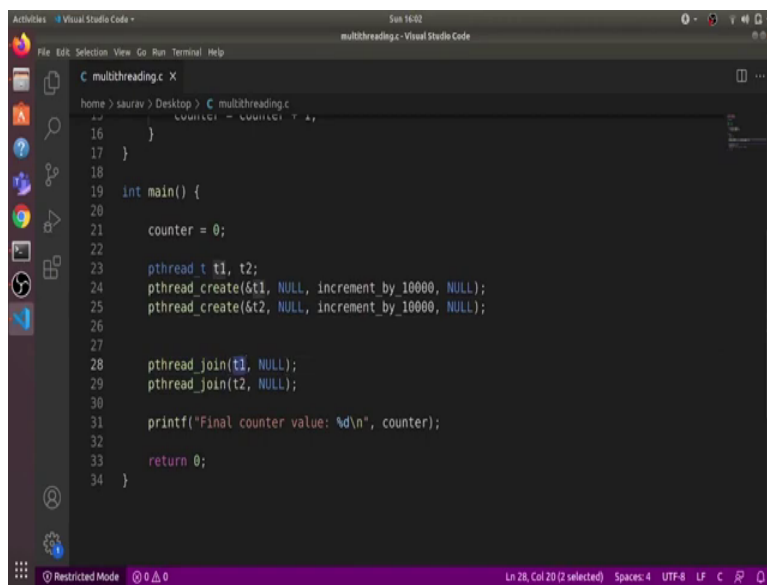
```

Then we created t2 thread again using the same increment were 10,000 function and again, we do not specify any arguments.

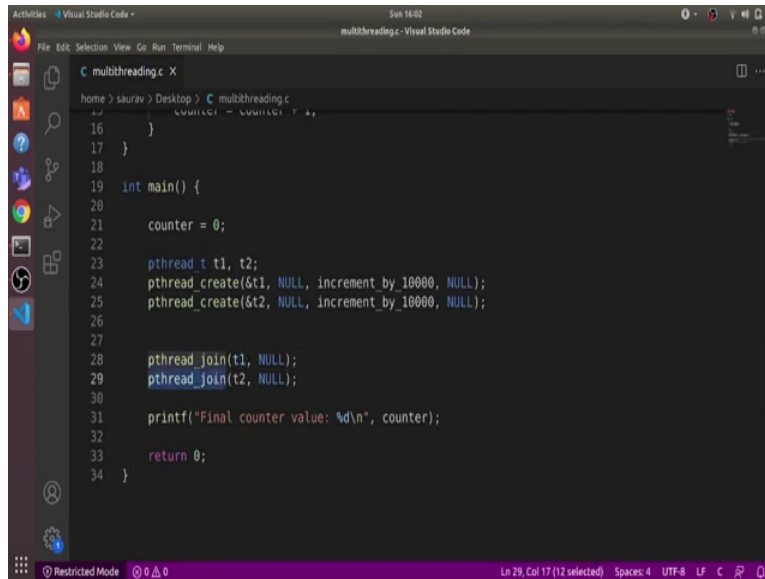
(Refer Slide Time: 01:43)



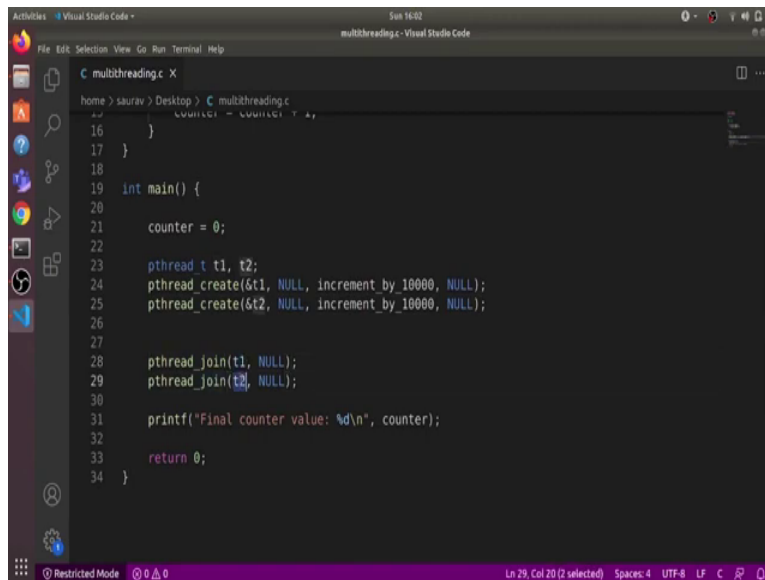
```
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32
33     return 0;
34 }
```



```
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32
33     return 0;
34 }
```



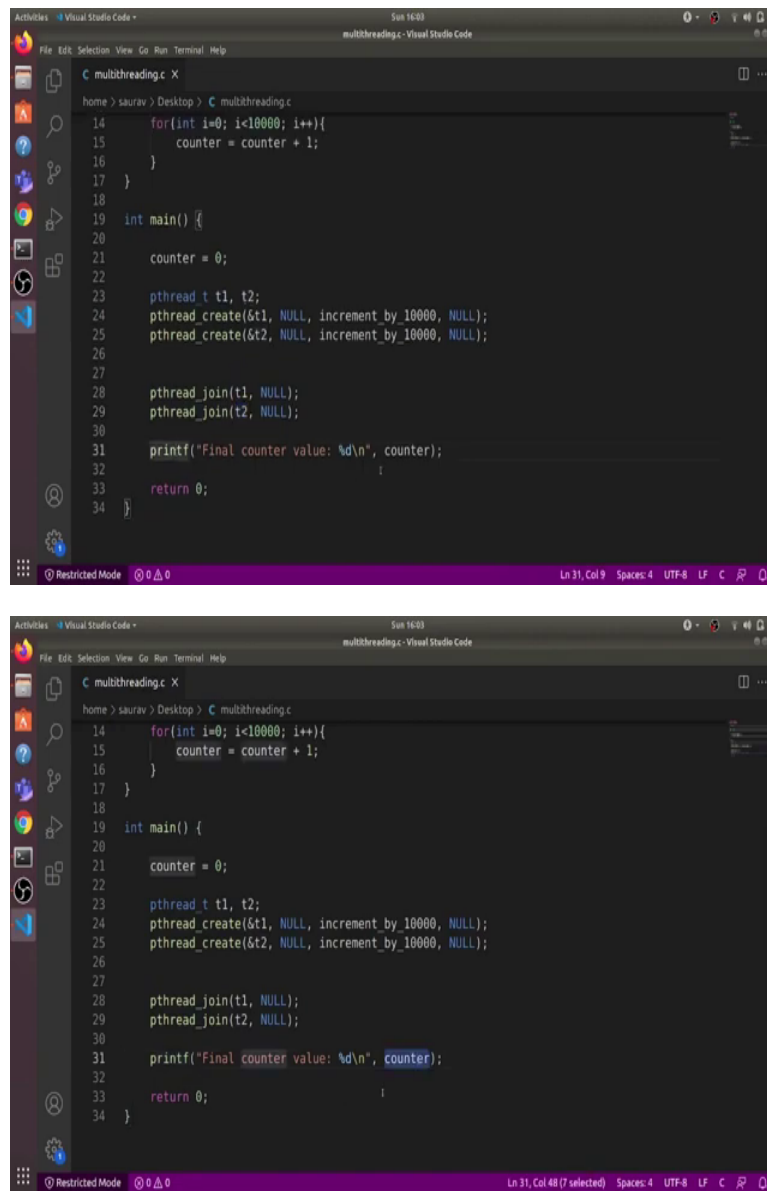
```
16 }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25
26
27     pthread_join(t1, NULL);
28     pthread_join(t2, NULL);
29
30     printf("Final counter value: %d\n", counter);
31
32     return 0;
33 }
34 }
```



```
16 }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25
26
27     pthread_join(t1, NULL);
28     pthread_join(t2, NULL);
29
30     printf("Final counter value: %d\n", counter);
31
32     return 0;
33 }
34 }
```

Then we use this pthread_join function so that the parent thread waits for thread t1 to complete. And similarly, it waits for the thread t2 complete.

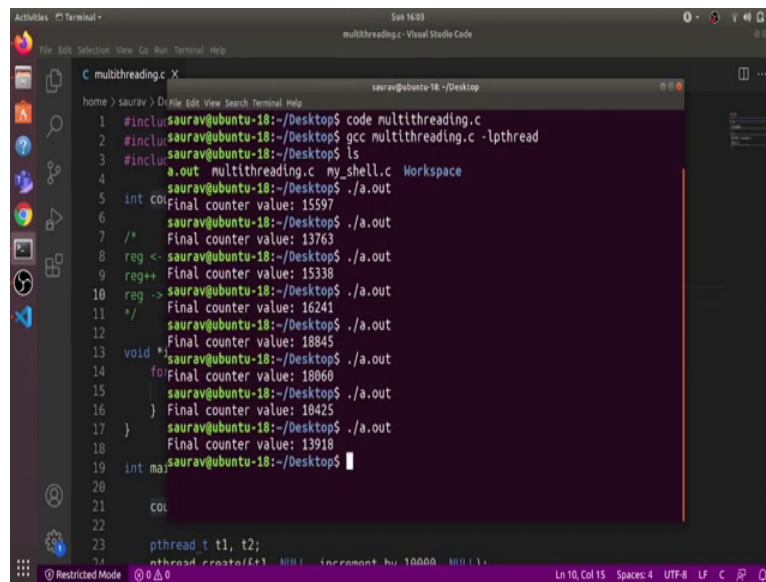
(Refer Slide Time: 01:57)



```
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26
27
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30
31     printf("Final counter value: %d\n", counter);
32
33     return 0;
34 }
```

So, once both threads have finished execution, then the parent thread prints the final counter value. So, we expect that because we are incrementing, this counter by 10000 two times. So, the final counter value should be 20,000.

(Refer Slide Time: 02:12)



```
home > saurav > Desktop > C multithreading.c
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 int counter = 0;
6 /*
7  * reg <- counter
8  * reg++
9  * reg -> counter
10  */
11 void *increment_by_10000 () {
12     for(int i=0; i<10000; i++){
13         counter = counter + 1;
14     }
15 }
16 int main() {
17     counter = 0;
18     pthread_t t1, t2;
19     pthread_create(&t1, NULL, increment_by_10000, NULL);
20     pthread_create(&t2, NULL, increment_by_10000, NULL);
21     pthread_join(t1, NULL);
22     pthread_join(t2, NULL);
23     printf("Final counter value: %d\n", counter);
24 }
```

Final counter value: 15597

Final counter value: 13763

Final counter value: 15338

Final counter value: 16241

Final counter value: 18845

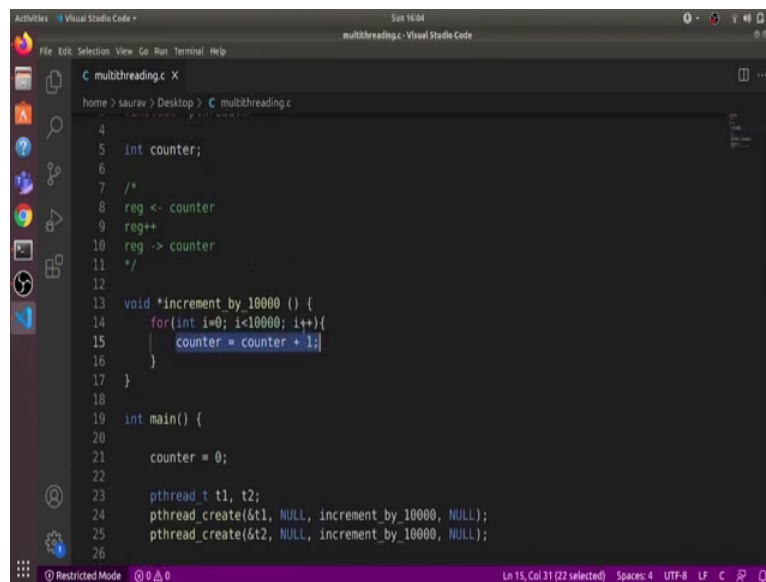
Final counter value: 18060

Final counter value: 18425

Final counter value: 13918

So, let us compile this programme and run it. So, I will open that terminal and compile it using gcc. Because we are using the pthread library, so we need to specify that while compiling. So this has created this a dot out file. Let us, run this a dot out. So, you can see that the counter value is less than 20,000. Let us, run it a couple of more times. So, every time we get some different value and which is less than 20,000. So, why is that so? This is because there is a race condition in the code.

(Refer Slide Time: 02:44)



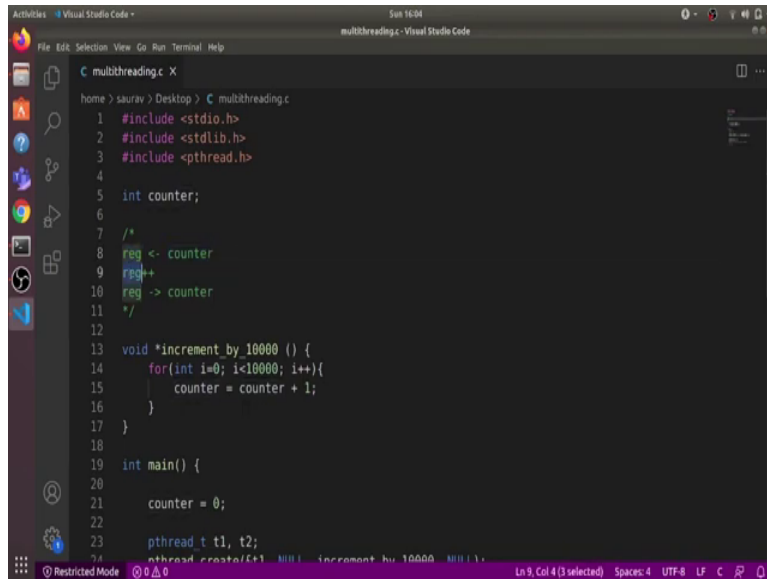
```
4
5 int counter;
6
7 /*
8  * reg <- counter
9  * reg++
10  * reg -> counter
11  */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

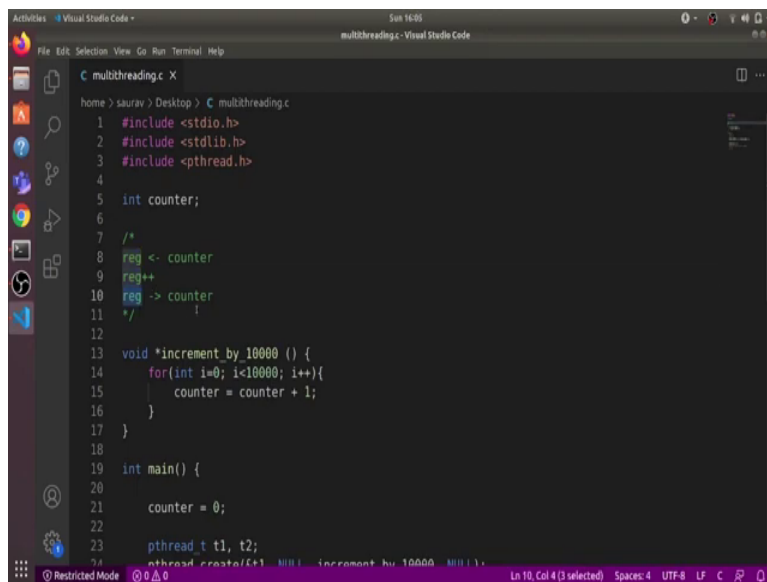
Ln 8, Col 4 (3 selected) Spaces: 4 UTF-8 LF C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counted
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

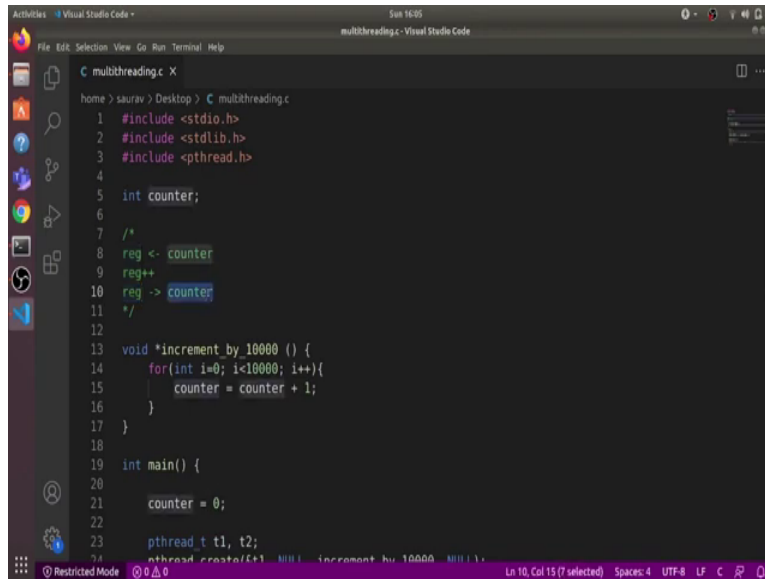
Ln 8, Col 15 (7 selected) Spaces: 4 UTF-8 LF C



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```



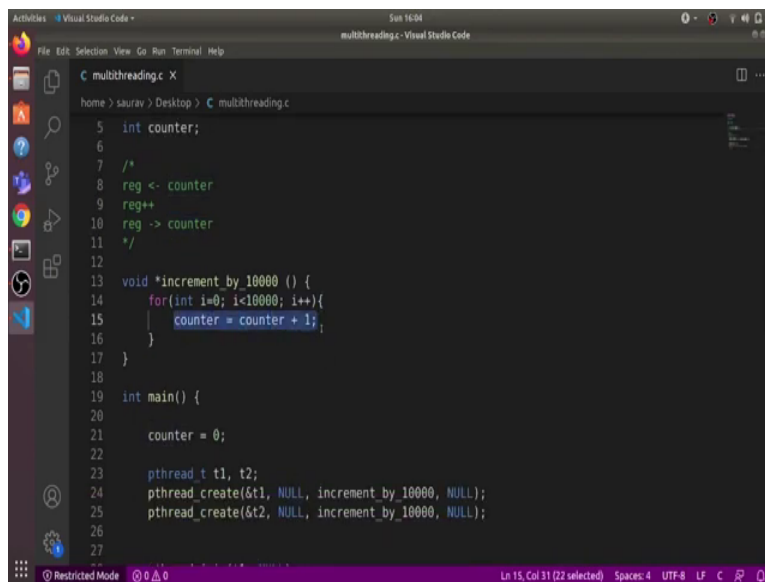
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

And what do I mean by race condition. So, there is this line in the code which mentions `counter = counter + 1`. So, when it is translated to assembly, then there are three instructions, it first loads this counter value in a register, then it increments the value in the register by 1. And finally it again stores back the register value to the counter variable.

(Refer Slide Time: 03:12)



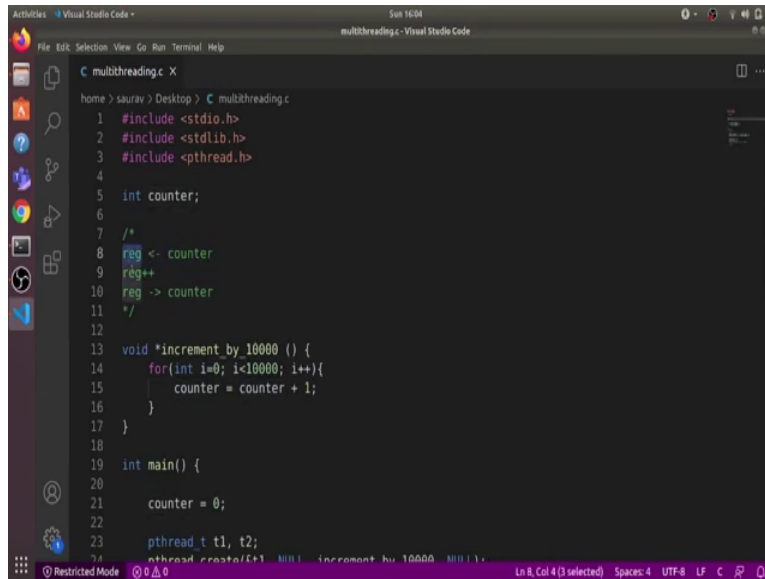
```
5 int counter;
6
7 /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26     pthread_join(t1, NULL);
27     pthread_join(t2, NULL);
28     printf("Counter: %d\n", counter);
29 }
```

Ln 8, Col 14 (13 selected) Spaces: 4 UTF-8 LF C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26     pthread_join(t1, NULL);
27     pthread_join(t2, NULL);
28     printf("Counter: %d\n", counter);
29 }
```

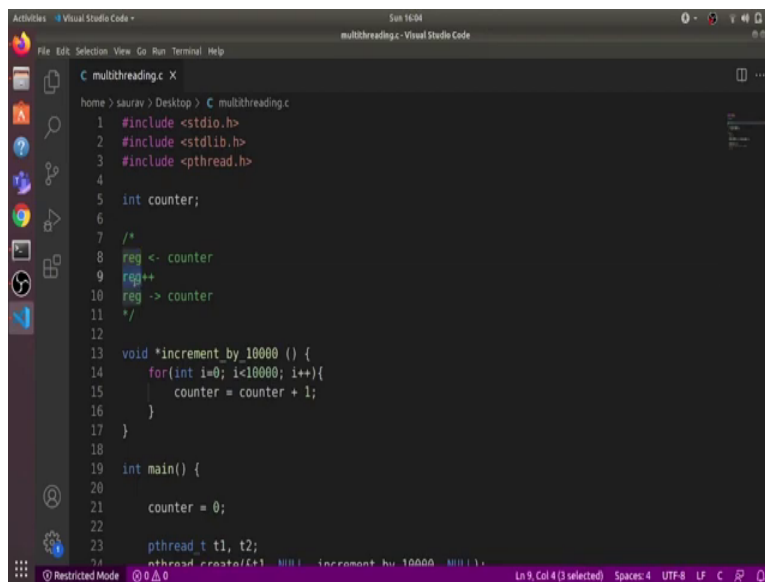
Ln 8, Col 15 (7 selected) Spaces: 4 UTF-8 LF C



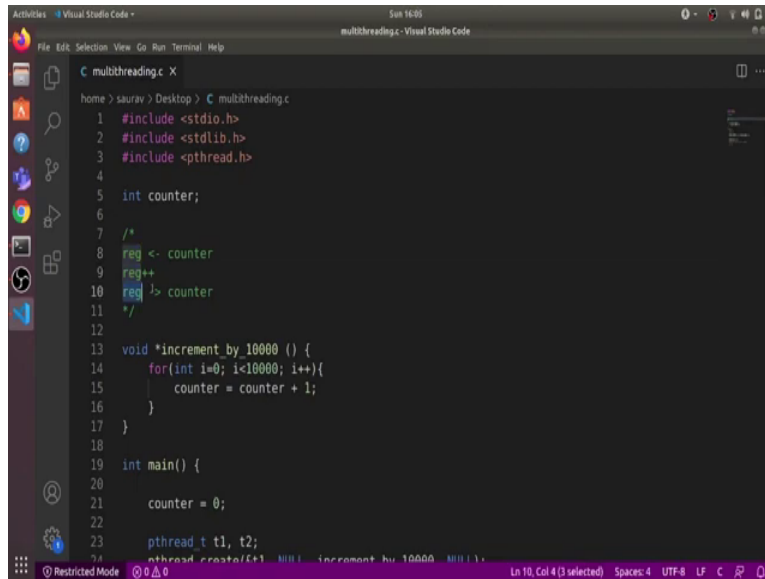
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```

Because there are two threads which are accessing this part of code concurrently. So, it might happen that first thread execute this instruction, it loads the counter value in the register.

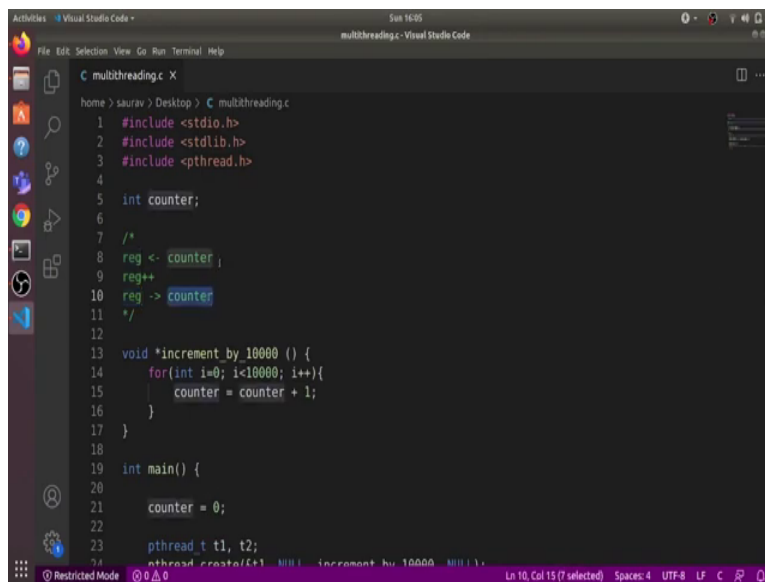
(Refer Slide Time: 03:19)



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```



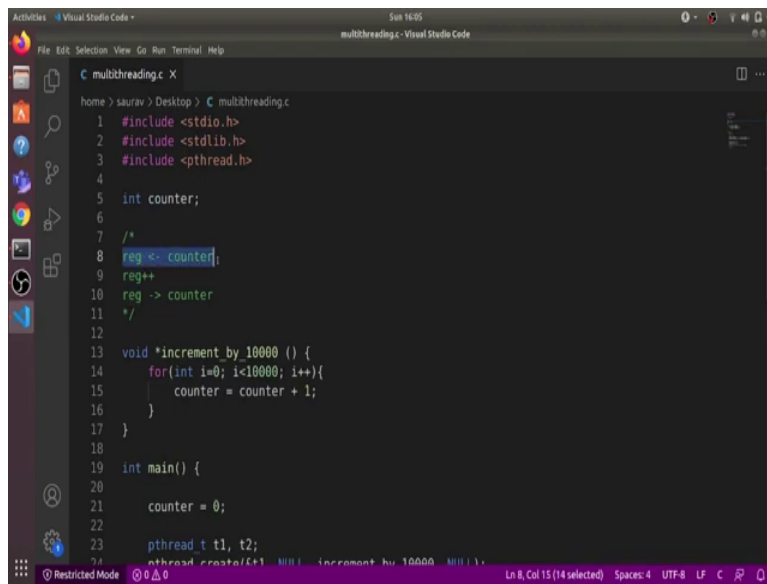
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25 }
```

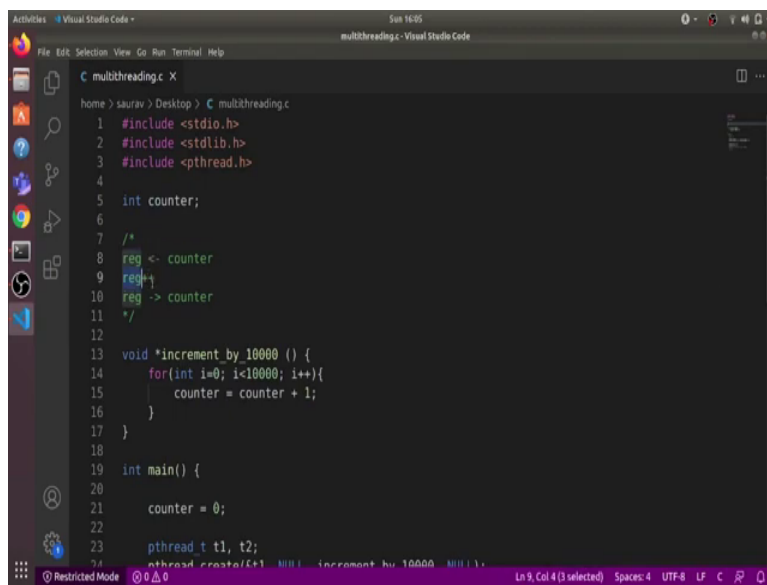
It increments the register value by 1. But before it stores back the counter value back to the counter variable.

(Refer Slide Time: 03:30)



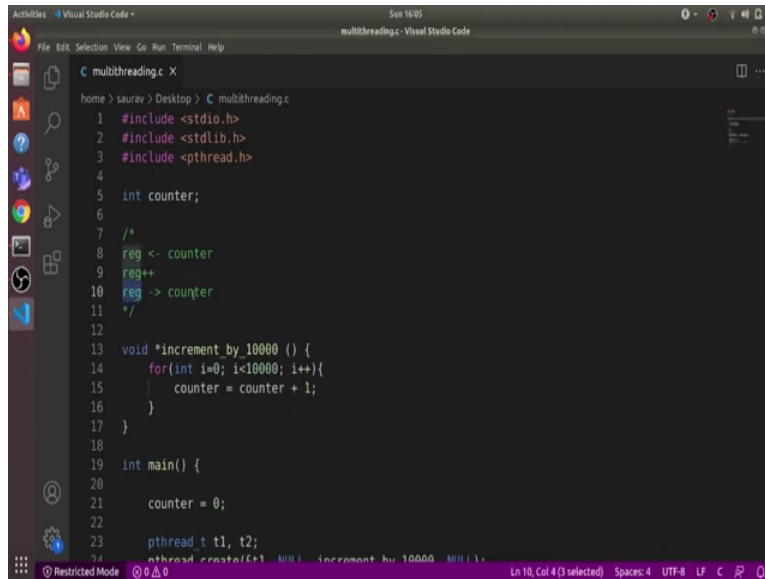
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counter;
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

Ln 8, Col 15 (14 selected) Spaces: 4 UTF-8 LF C

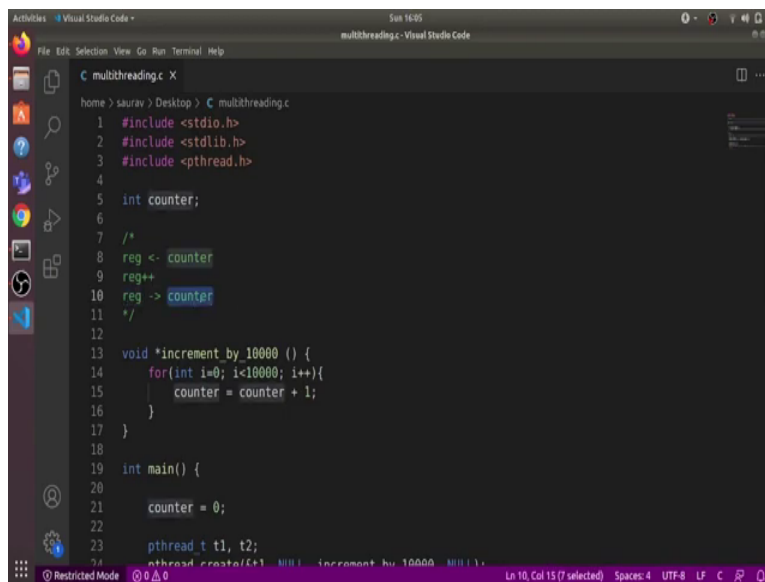


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

Ln 9, Col 4 (3 selected) Spaces: 4 UTF-8 LF C



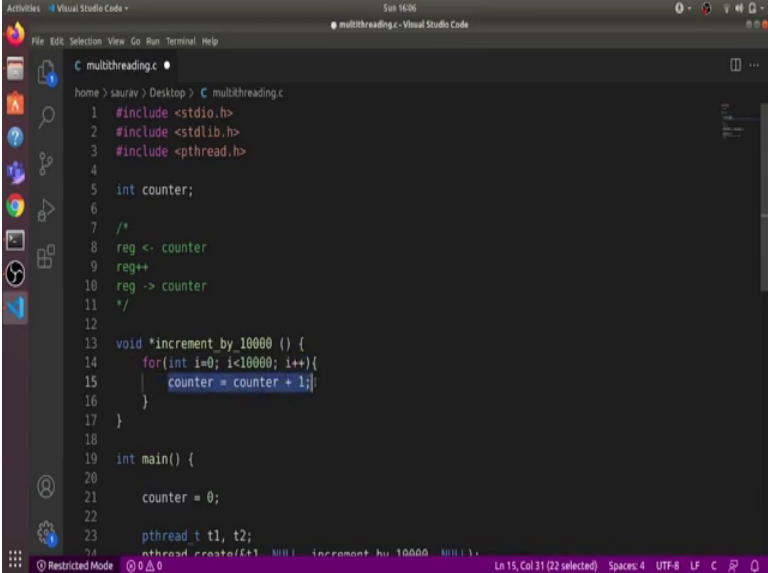
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8 reg <- counter
9 reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20
21     counter = 0;
22
23     pthread_t t1, t2;
24     pthread_create(&t1, NULL, increment_by_10000, NULL);
25     pthread_create(&t2, NULL, increment_by_10000, NULL);
26 }
```

The another thread executes this instruction. So, it takes in the previous value, it also incremented by 1. And finally, both the threads store the same value to the counter variable. In essence, instead of increasing the counter value by two the execution just in visit the counter value by 1. So, how do we avoid this reg condition?

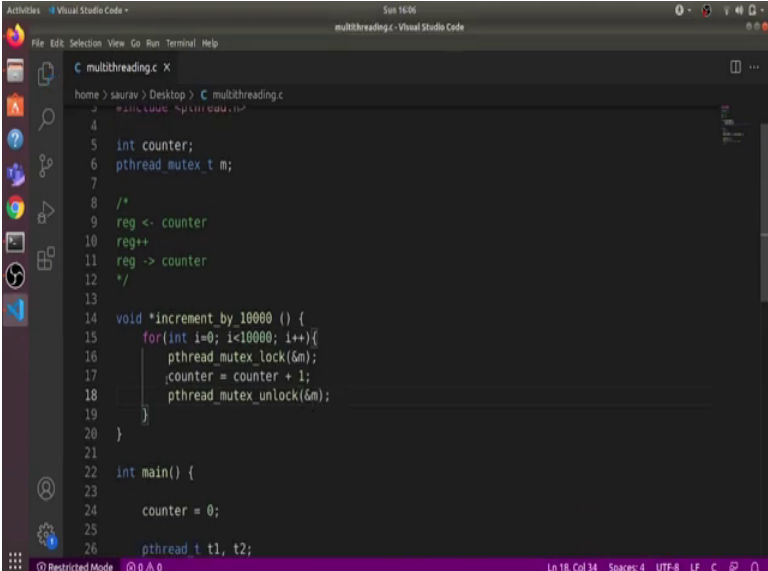
(Refer Slide Time: 03:55)



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 int counter;
6
7 /*
8  reg <- counter
9  reg++
10 reg -> counter
11 */
12
13 void *increment_by_10000 () {
14     for(int i=0; i<10000; i++){
15         counter = counter + 1;
16     }
17 }
18
19 int main() {
20     counter = 0;
21
22     pthread_t t1, t2;
23     pthread_create(&t1, NULL, increment_by_10000, NULL);
24     pthread_create(&t2, NULL, increment_by_10000, NULL);
25     pthread_join(t1, NULL);
26     pthread_join(t2, NULL);
27     printf("Counter: %d\n", counter);
28 }
```

So, you can get rid of this race condition using locks, we will lock the critical section of the code which is this line so that only one thread can execute it at a time. So first, we will declare a lock `pthread_mutex_t` and we will call this lock `m`. So, this mutex stands for mutual exclusion, because we want mutual exclusion for this part of code.

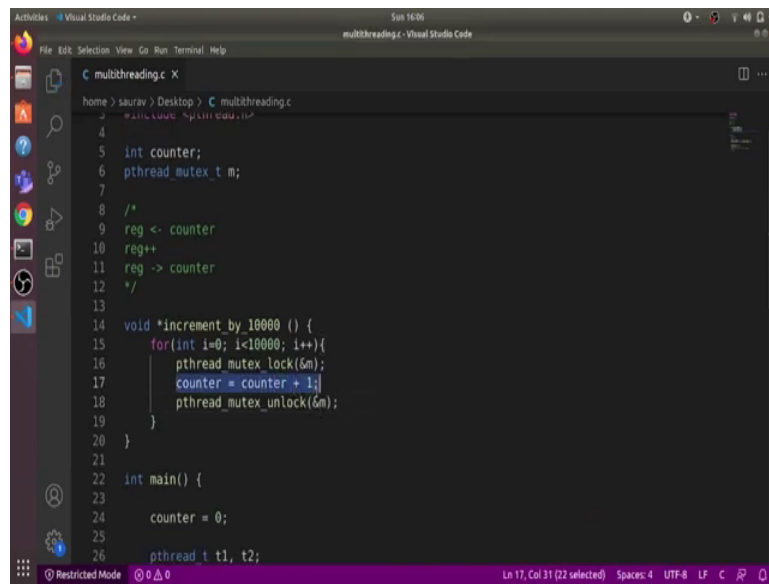
(Refer Slide Time: 04:16)



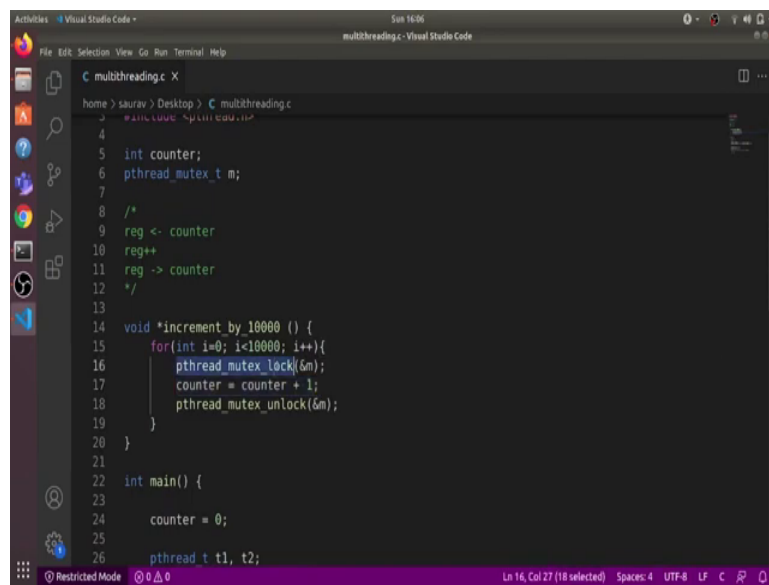
```
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23     counter = 0;
24
25     pthread_t t1, t2;
26     pthread_create(&t1, NULL, increment_by_10000, NULL);
27     pthread_create(&t2, NULL, increment_by_10000, NULL);
28     pthread_join(t1, NULL);
29     pthread_join(t2, NULL);
30     printf("Counter: %d\n", counter);
31 }
```

So, we acquire lock before entering the critical section, use `pthread_mutex_lock` and we give it a pointer to `m`. Then, after executing the `counter = counter + 1`, we unlock it so that another thread can access the lock and run this part of code. So, we use `pthread_mutex_unlock(&m)`.

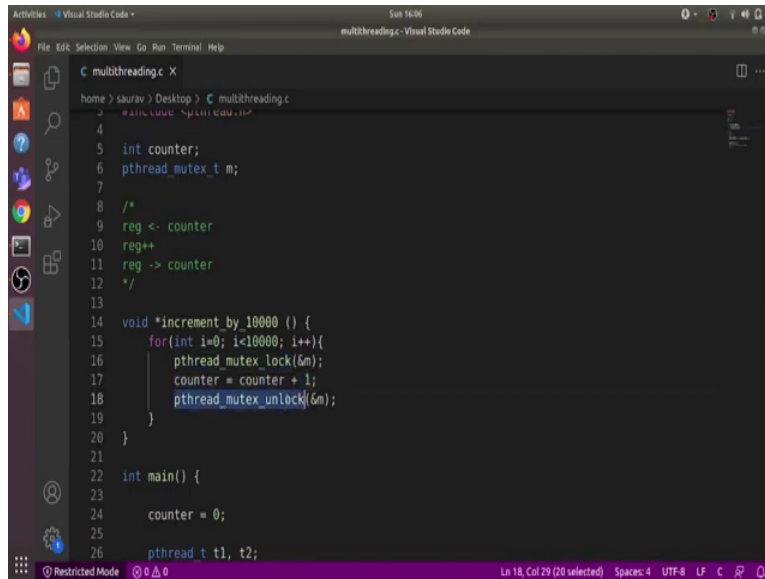
(Refer Slide Time: 04:46)



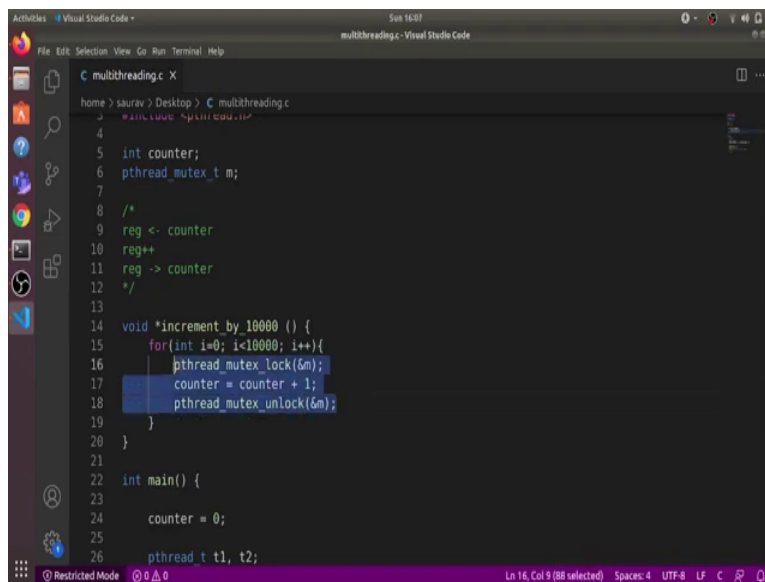
```
1 #include <pthread.h>
2
3
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23
24     counter = 0;
25
26     pthread_t t1, t2;
```



```
1 #include <pthread.h>
2
3
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23
24     counter = 0;
25
26     pthread_t t1, t2;
```



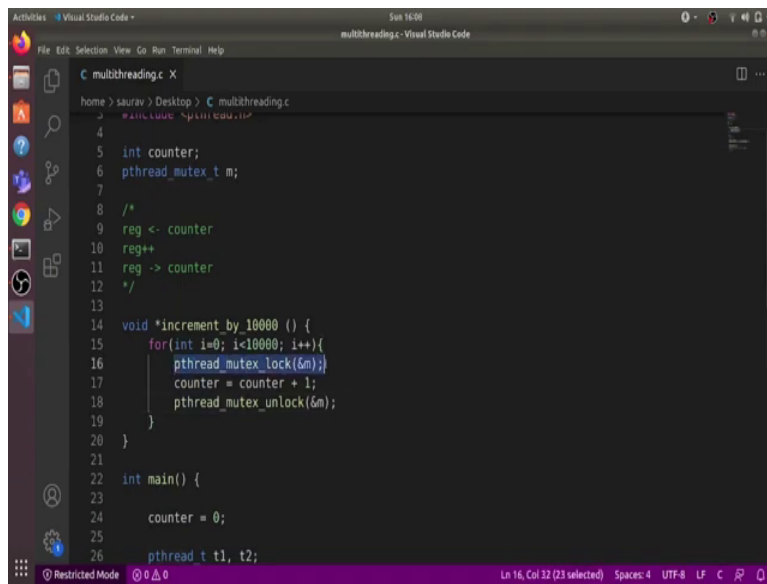
```
1  #include <pthread.h>
2
3
4
5  int counter;
6  pthread_mutex_t m;
7
8  /*
9   reg <- counter
10  reg++
11  reg -> counter
12  */
13
14  void *increment_by_10000 () {
15      for(int i=0; i<10000; i++){
16          pthread_mutex_lock(&m);
17          counter = counter + 1;
18          pthread_mutex_unlock(&m);
19      }
20  }
21
22  int main() {
23
24      counter = 0;
25
26      pthread_t t1, t2;
```



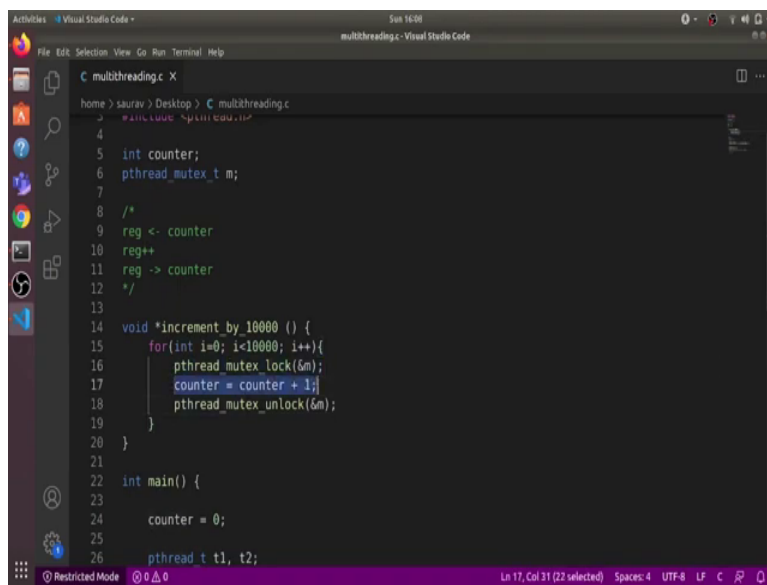
```
1  #include <pthread.h>
2
3
4
5  int counter;
6  pthread_mutex_t m;
7
8  /*
9   reg <- counter
10  reg++
11  reg -> counter
12  */
13
14  void *increment_by_10000 () {
15      for(int i=0; i<10000; i++){
16          pthread_mutex_lock(&m);
17          counter = counter + 1;
18          pthread_mutex_unlock(&m);
19      }
20  }
21
22  int main() {
23
24      counter = 0;
25
26      pthread_t t1, t2;
```

So, now before entering the critical section, each of the thread will have to acquire the lock and then they will release it after exiting from the critical section. So, only one thread will be able to execute this part of code.

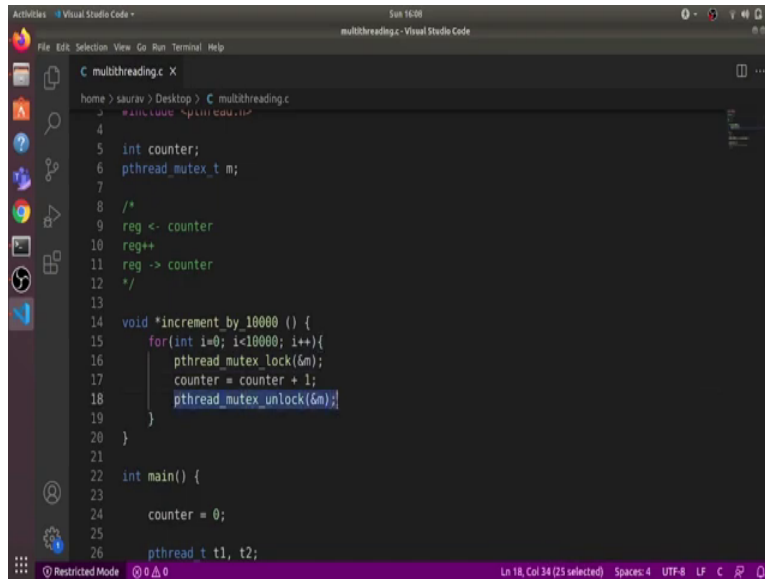
(Refer Slide Time: 04:58)



```
1 #include <pthread.h>
2
3
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23
24     counter = 0;
25
26     pthread_t t1, t2;
```



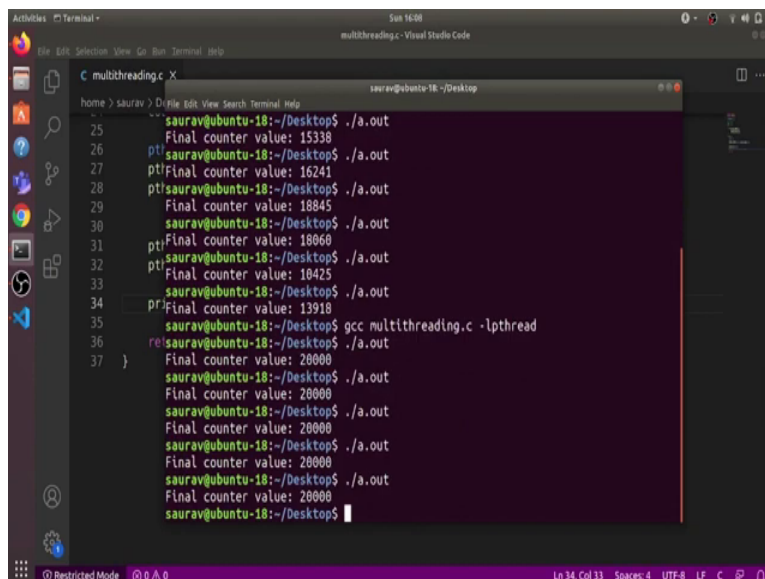
```
1 #include <pthread.h>
2
3
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23
24     counter = 0;
25
26     pthread_t t1, t2;
```

```
1 // multithreading.c
2
3 #include <stdio.h>
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10  reg++
11  reg -> counter
12 */
13
14 void *increment_by_10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23     counter = 0;
24
25     pthread_t t1, t2;
```

Now, what will happen is first thread will acquire the lock then when it is executing `counter = counter + 1`. Even if the second thread reaches this particular part of code, it will be unable to acquire the lock because first thread already has the lock and only when first thread releases the lock after this particular code, will the second thread be able to acquire it and execute `counter = counter + 1`. So, this is how we can get rid of the race condition.

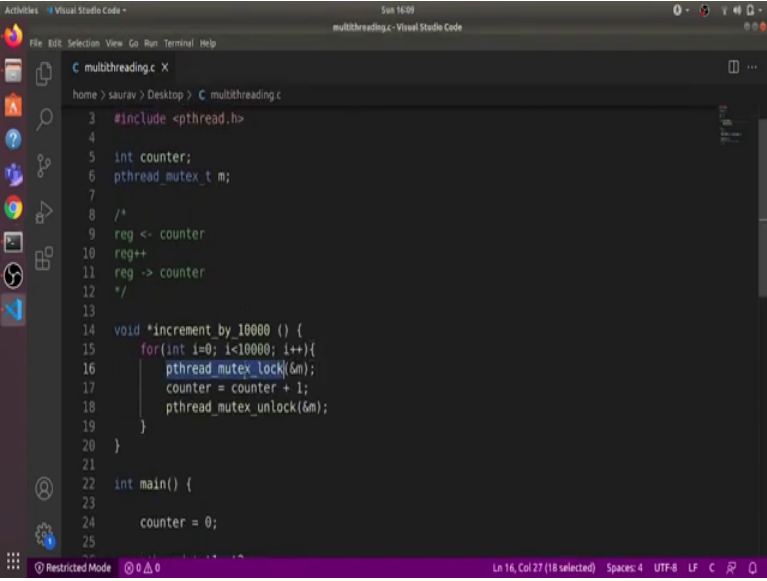
(Refer Slide Time: 05:25)



```
25 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 15338
26 pt1 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 16241
27 pt1 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 18845
28 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 18060
29 pt1 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 18425
30 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 13918
31 pt1 saurav@ubuntu-18:~/Desktop$ gcc multithreading.c -lpthread
32 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 20000
33 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 20000
34 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 20000
35 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 20000
36 saurav@ubuntu-18:~/Desktop$ ./a.out
   Final counter value: 20000
37 saurav@ubuntu-18:~/Desktop$
```

So, let us compile and run it again. So, I will open a terminal and compile it again with the thread library. Alright, so now you see we are getting 20,000 every time we run the code.

(Refer Slide Time: 05:41)

A screenshot of the Visual Studio Code editor interface. The title bar shows 'Visual Studio Code' and 'multithreading.c - Visual Studio Code'. The file explorer on the left shows a file named 'multithreading.c'. The editor window displays the following C code:

```
3 #include <pthread.h>
4
5 int counter;
6 pthread_mutex_t m;
7
8 /*
9  reg <- counter
10 reg++
11 reg -> counter
12 */
13
14 void *increment by 10000 () {
15     for(int i=0; i<10000; i++){
16         pthread_mutex_lock(&m);
17         counter = counter + 1;
18         pthread_mutex_unlock(&m);
19     }
20 }
21
22 int main() {
23
24     counter = 0;
25 }
```

The status bar at the bottom indicates 'Ln 16, Col 27 (18 selected)', 'Spaces: 4', 'UTF-8', 'LF', and 'C'.

So, it is very important to use this mutual exclusion locks whenever you are accessing a global variable in case of multi threading programme. So, that is it for this video. Thanks and have a nice day.