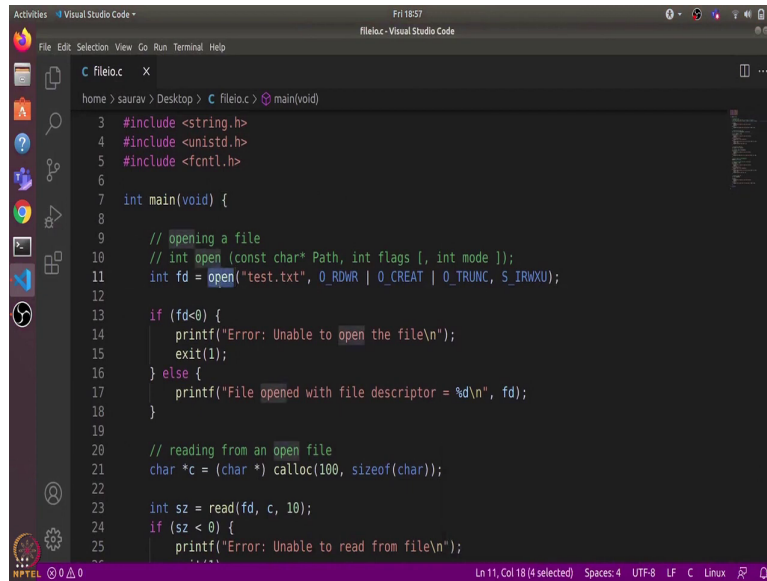


Design and Engineering of Computer Systems
Professor Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 29 (Week - 4, Tutorial - 1)
File I/O in C

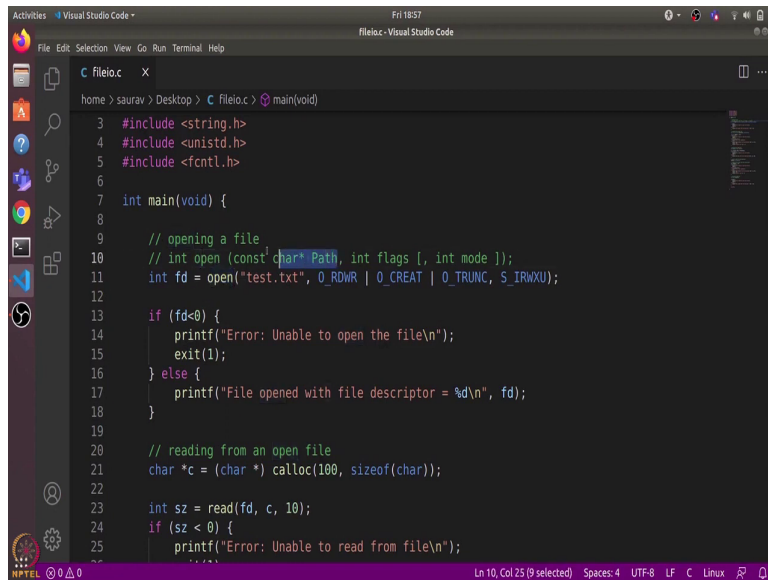
(Refer Slide Time: 0:15)

A screenshot of the Visual Studio Code editor interface. The editor window shows a C program named fileio.c. The code includes headers for string, unistd, and fcntl. The main function opens a file named 'test.txt' with flags O_RDWR, O_CREAT, O_TRUNC, and S_IRWXU. It checks if the file was opened successfully and prints an error message if not. Then, it reads 10 bytes from the file into a buffer 'c' and prints an error message if the read failed. The status bar at the bottom indicates 'Ln 11, Col 18 (4 selected)', 'Spaces: 4', 'UTF-8', 'LF', 'C', 'Linux', and a search icon.

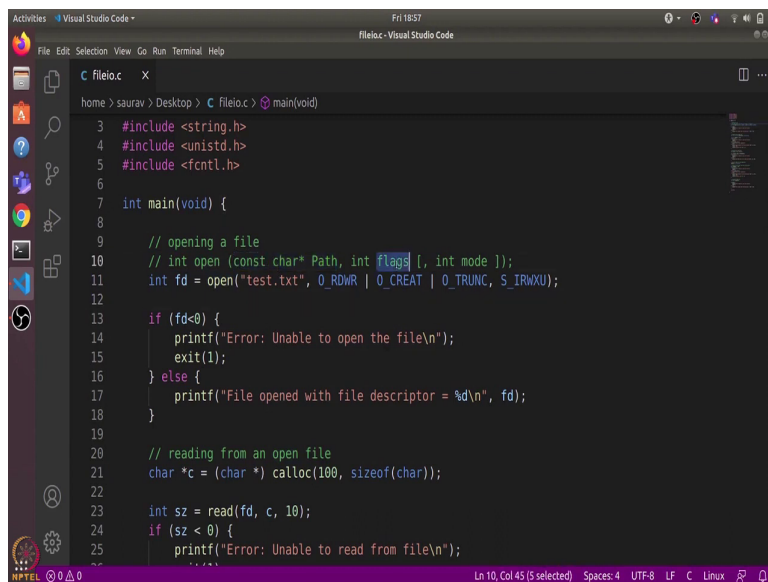
```
1 // fileio.c
2
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8
9     // opening a file
10    // int open (const char* Path, int flags [, int mode]);
11    int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13    if (fd < 0) {
14        printf("Error: Unable to open the file\n");
15        exit(1);
16    } else {
17        printf("File opened with file descriptor = %d\n", fd);
18    }
19
20    // reading from an open file
21    char *c = (char *) calloc(100, sizeof(char));
22
23    int sz = read(fd, c, 10);
24    if (sz < 0) {
25        printf("Error: Unable to read from file\n");
26    }
27 }
```

Hi everyone. In this video, we will learn about file I/O in C programming language. So, I have written this fileio.c program. Let us open it with visual code. So, this is the overall code. Let us go over the code. So, here is the main program. The very first system call that we use is the open system call, which is used to open a new file. So, what all argument does it take?

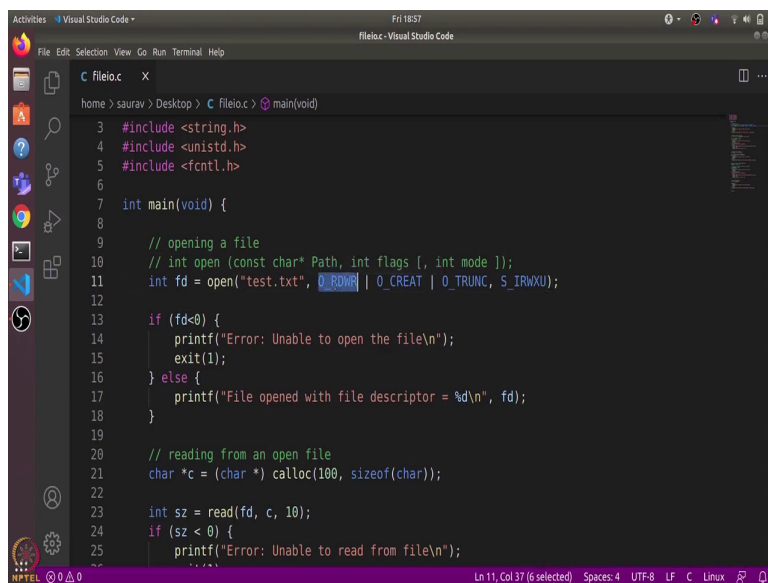
(Refer Slide Time: 0:46)



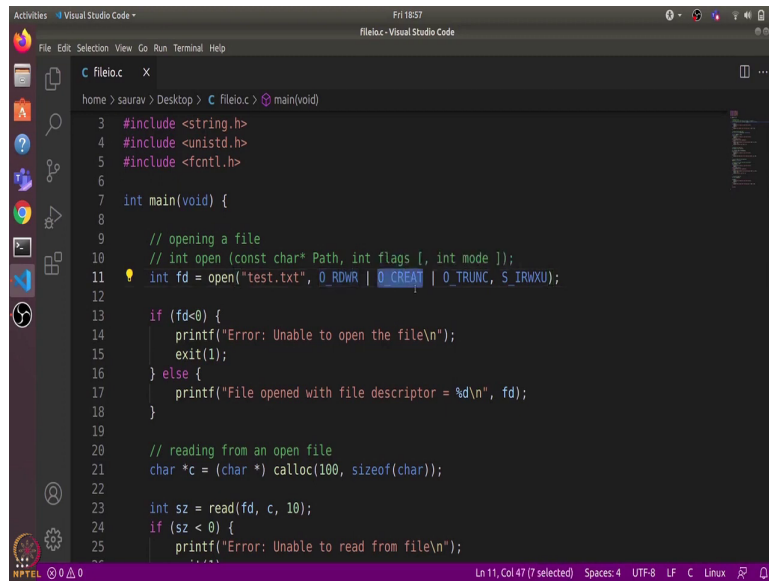
```
home > saurav > Desktop > C fileio.c > main(void)
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8
9     // opening a file
10    // int open (const char* Path, int flags [, int mode]);
11    int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13    if (fd<0) {
14        printf("Error: Unable to open the file\n");
15        exit(1);
16    } else {
17        printf("File opened with file descriptor = %d\n", fd);
18    }
19
20    // reading from an open file
21    char *c = (char *) calloc(100, sizeof(char));
22
23    int sz = read(fd, c, 10);
24    if (sz < 0) {
25        printf("Error: Unable to read from file\n");
26    }
```



```
home > saurav > Desktop > C fileio.c > main(void)
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8
9     // opening a file
10    // int open (const char* Path, int flags [, int mode]);
11    int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13    if (fd<0) {
14        printf("Error: Unable to open the file\n");
15        exit(1);
16    } else {
17        printf("File opened with file descriptor = %d\n", fd);
18    }
19
20    // reading from an open file
21    char *c = (char *) calloc(100, sizeof(char));
22
23    int sz = read(fd, c, 10);
24    if (sz < 0) {
25        printf("Error: Unable to read from file\n");
26    }
```



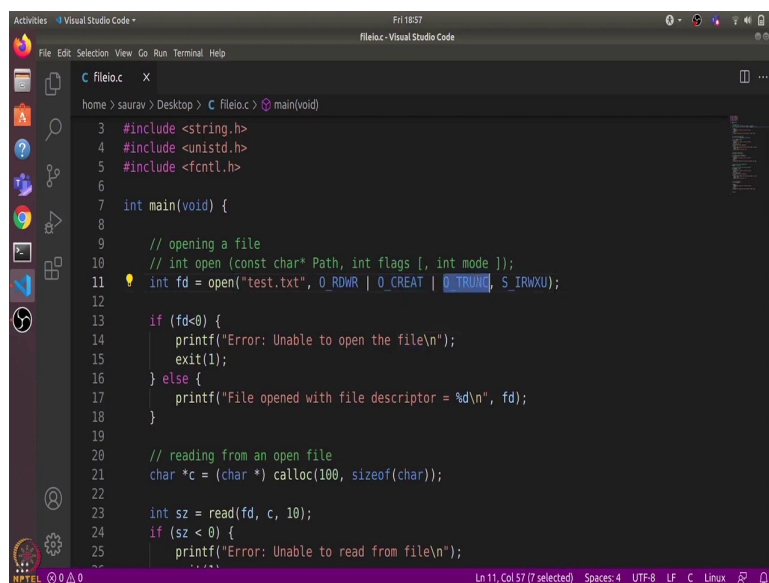
```
home > saurav > Desktop > C fileio.c > main(void)
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8
9     // opening a file
10    // int open (const char* Path, int flags [, int mode]);
11    int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13    if (fd<0) {
14        printf("Error: Unable to open the file\n");
15        exit(1);
16    } else {
17        printf("File opened with file descriptor = %d\n", fd);
18    }
19
20    // reading from an open file
21    char *c = (char *) calloc(100, sizeof(char));
22
23    int sz = read(fd, c, 10);
24    if (sz < 0) {
25        printf("Error: Unable to read from file\n");
26    }
```



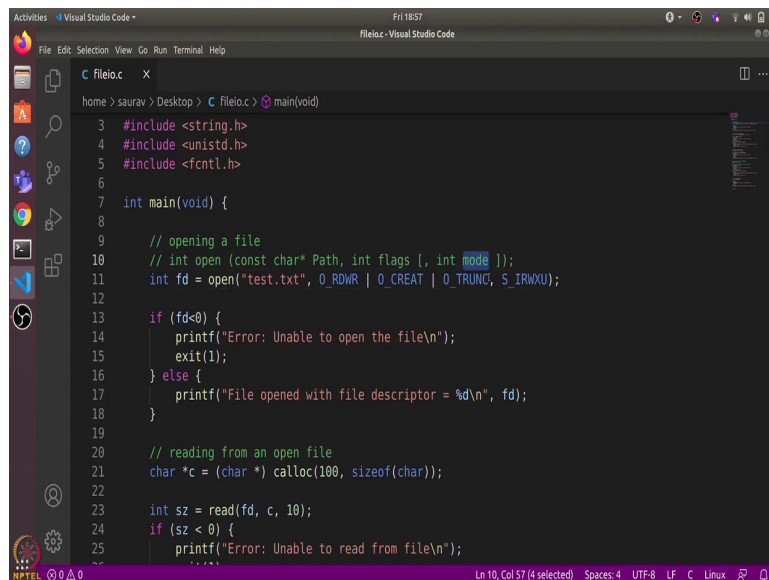
```
1  home > saurav > Desktop > C fileio.c > main(void)
2
3  #include <string.h>
4  #include <unistd.h>
5  #include <fcntl.h>
6
7  int main(void) {
8
9      // opening a file
10     // int open (const char* Path, int flags [, int mode]);
11     int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13     if (fd<0) {
14         printf("Error: Unable to open the file\n");
15         exit(1);
16     } else {
17         printf("File opened with file descriptor = %d\n", fd);
18     }
19
20     // reading from an open file
21     char *c = (char *) calloc(100, sizeof(char));
22
23     int sz = read(fd, c, 10);
24     if (sz < 0) {
25         printf("Error: Unable to read from file\n");
26     }
```

The first argument that it takes is the path to the file, then it takes certain flags. For instance, here we have used this RDWR flag, which means we want to both read and write using this file descriptor and second flag that we use is O_CREATE, which means if the file does not already exist, then we want to create the file.

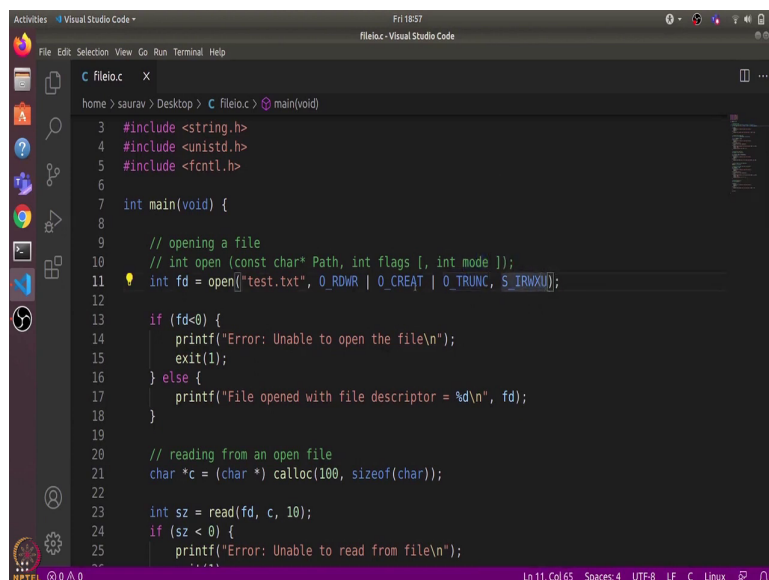
(Refer Slide Time: 1:05)



```
1  home > saurav > Desktop > C fileio.c > main(void)
2
3  #include <string.h>
4  #include <unistd.h>
5  #include <fcntl.h>
6
7  int main(void) {
8
9      // opening a file
10     // int open (const char* Path, int flags [, int mode]);
11     int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
12
13     if (fd<0) {
14         printf("Error: Unable to open the file\n");
15         exit(1);
16     } else {
17         printf("File opened with file descriptor = %d\n", fd);
18     }
19
20     // reading from an open file
21     char *c = (char *) calloc(100, sizeof(char));
22
23     int sz = read(fd, c, 10);
24     if (sz < 0) {
25         printf("Error: Unable to read from file\n");
26     }
```



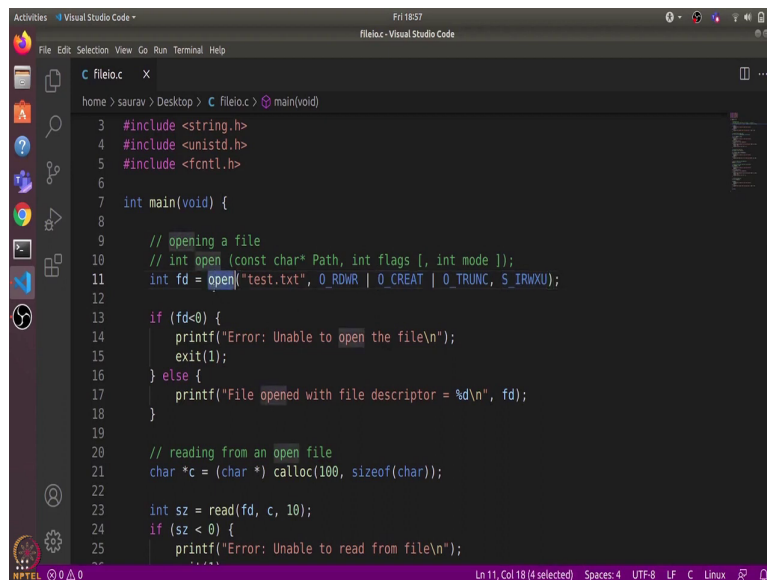
```
1  #include <string.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main(void) {
6
7      // opening a file
8      // int open (const char* Path, int flags [, int mode]);
9      int fd = open("test.txt", O_RDONLY | O_CREAT | O_TRUNC, S_IRWXU);
10
11     if (fd < 0) {
12         printf("Error: Unable to open the file\n");
13         exit(1);
14     } else {
15         printf("File opened with file descriptor = %d\n", fd);
16     }
17
18     // reading from an open file
19     char *c = (char *) calloc(100, sizeof(char));
20
21     int sz = read(fd, c, 10);
22     if (sz < 0) {
23         printf("Error: Unable to read from file\n");
24     }
25 }
```



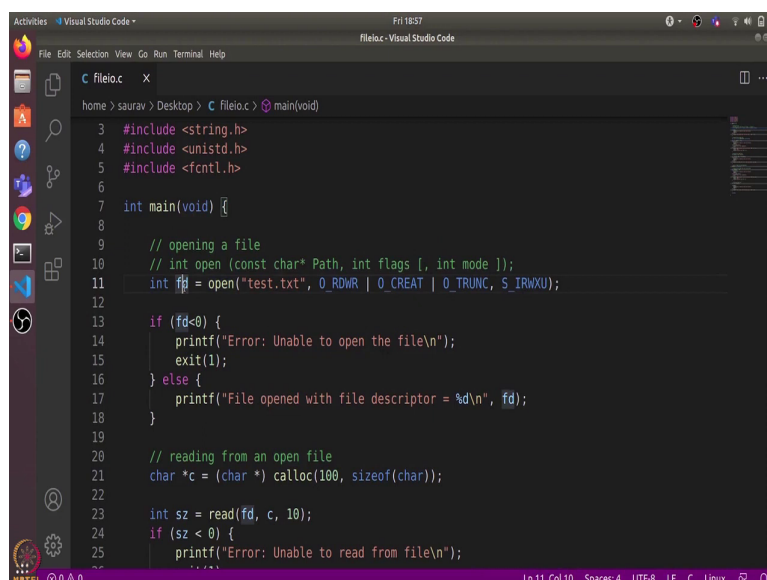
```
1  #include <string.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main(void) {
6
7      // opening a file
8      // int open (const char* Path, int flags [, int mode]);
9      int fd = open("test.txt", O_RDONLY | O_CREAT | O_TRUNC, S_IRWXU);
10
11     if (fd < 0) {
12         printf("Error: Unable to open the file\n");
13         exit(1);
14     } else {
15         printf("File opened with file descriptor = %d\n", fd);
16     }
17
18     // reading from an open file
19     char *c = (char *) calloc(100, sizeof(char));
20
21     int sz = read(fd, c, 10);
22     if (sz < 0) {
23         printf("Error: Unable to read from file\n");
24     }
25 }
```

And the last flag is the truncate flag, which means if the file already exists, then we want to clear all its contents. And finally, the last argument is an optional argument. And it is used only when we create a file, it is used to set the permissions of the file. So, here we have given read, write and execute permission to the user.

(Refer Slide Time: 1:24)



```
1  #include <string.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main(void) {
6
7      // opening a file
8      // int open (const char* Path, int flags [, int mode]);
9      int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
10
11     if (fd < 0) {
12         printf("Error: Unable to open the file\n");
13         exit(1);
14     } else {
15         printf("File opened with file descriptor = %d\n", fd);
16     }
17
18     // reading from an open file
19     char *c = (char *) calloc(100, sizeof(char));
20
21     int sz = read(fd, c, 10);
22     if (sz < 0) {
23         printf("Error: Unable to read from file\n");
24     }
25 }
```

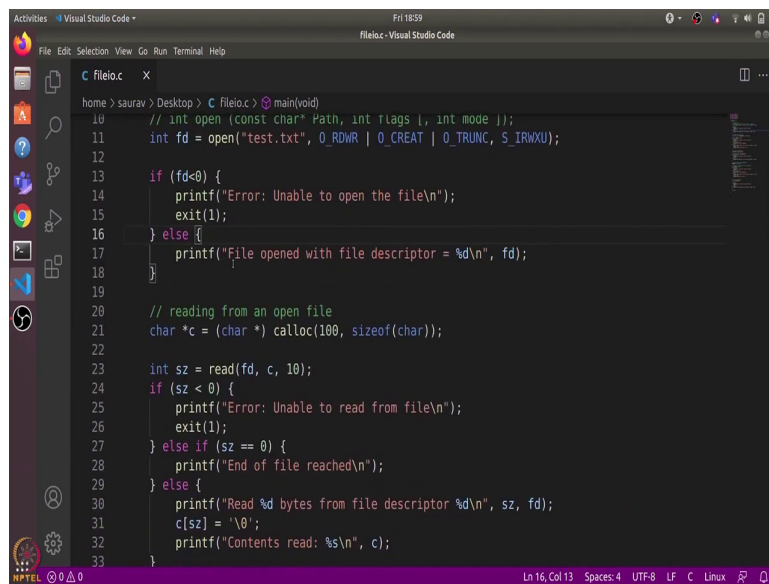


```
1  #include <string.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main(void) {
6
7      // opening a file
8      // int open (const char* Path, int flags [, int mode]);
9      int fd = open("test.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRWXU);
10
11     if (fd < 0) {
12         printf("Error: Unable to open the file\n");
13         exit(1);
14     } else {
15         printf("File opened with file descriptor = %d\n", fd);
16     }
17
18     // reading from an open file
19     char *c = (char *) calloc(100, sizeof(char));
20
21     int sz = read(fd, c, 10);
22     if (sz < 0) {
23         printf("Error: Unable to read from file\n");
24     }
25 }
```

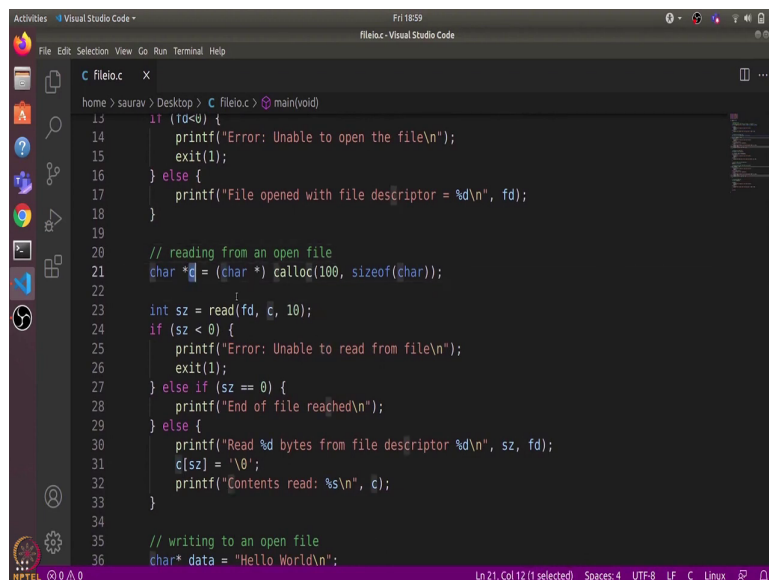
And this open system call returns a file descriptor, which we store in this fd variable. So, what happens when we make this open system call, the process will create the new file, it will allocate a new inode and add a mapping from the file name to this new inode number in the parent directory. And then it will copy this inode into memory from the disk. So, now that we have inode in the memory, it will create a new entry in the open file table, which will point to this in memory inode.

And it will also add a new entry in the file descriptor array for this process. And that file descriptor array entry will point to this open file table entry. And then it will return the index in that file descriptor array as the file descriptor. So, if it returns a negative value, that means that there was some error and we print out unable to open the file.

(Refer Slide Time: 2:16)



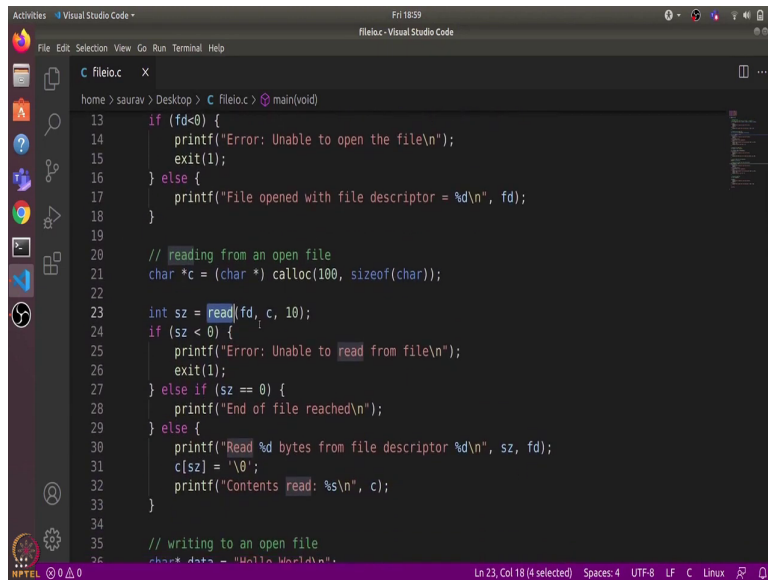
```
10 // int open (const char* Path, int flags [, int mode ]);
11 int fd = open("test.txt", O_RDONLY | O_CREAT | O_TRUNC, S_IRWXU);
12
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
```



```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

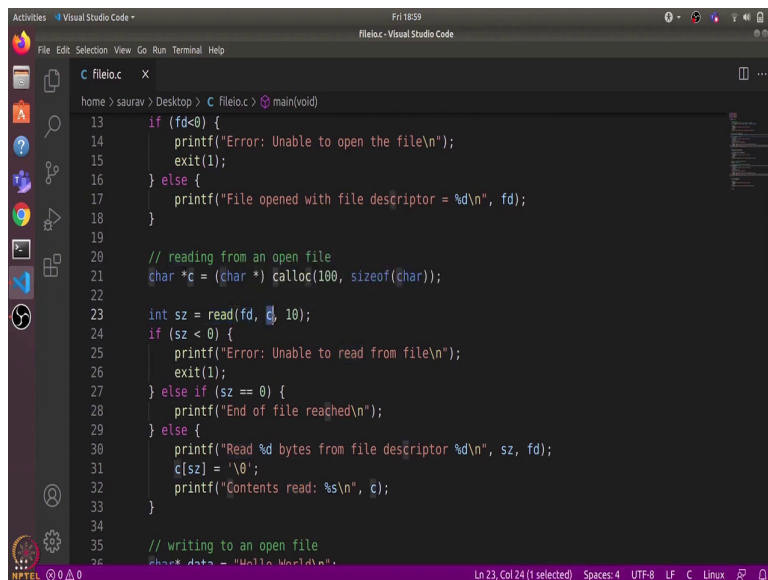
Otherwise, we print out file open with this as file descriptor. Now, we use the read system call to read from an open file. Firstly, we allocate some region where we can read the contents.

(Refer Slide Time: 2:28)



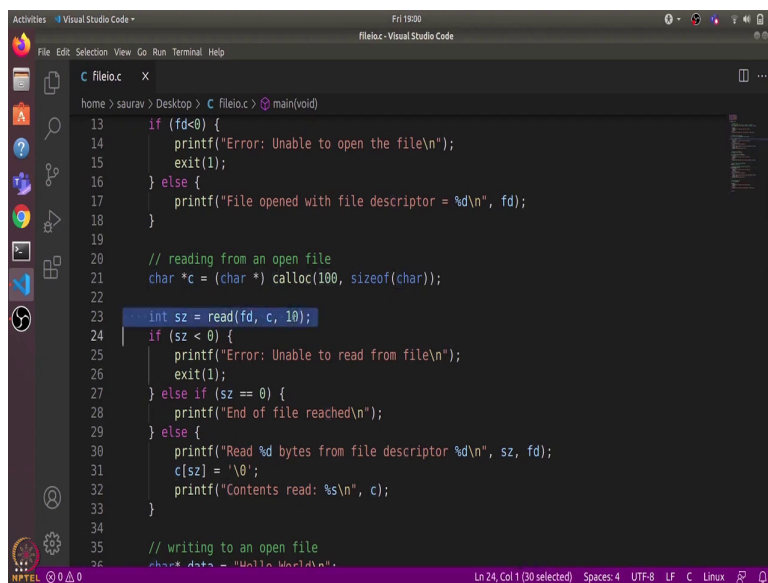
```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

Ln 23, Col 18 (4 selected) Spaces: 4 UTF-8 LF C Linux



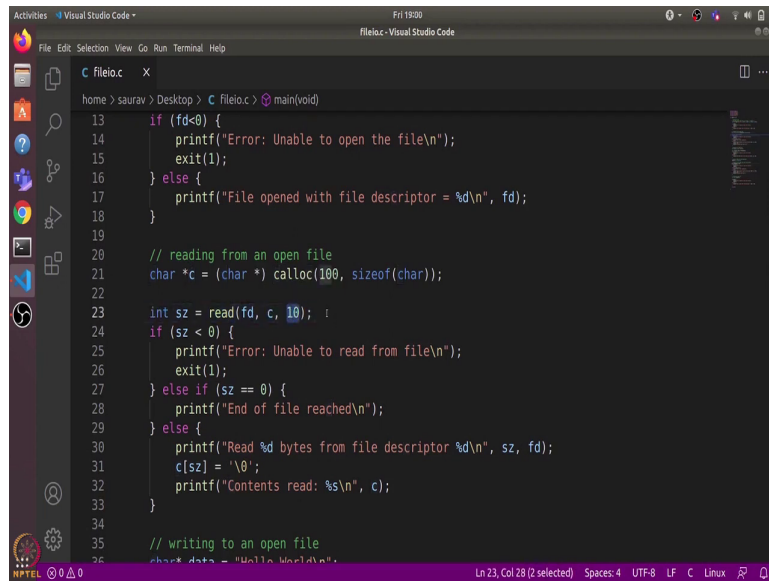
```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

Ln 23, Col 24 (1 selected) Spaces: 4 UTF-8 LF C Linux



```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

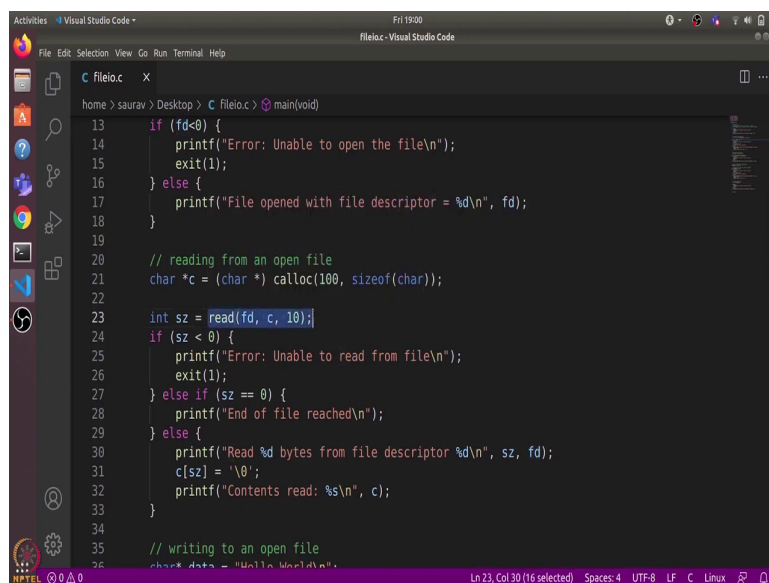
Ln 24, Col 1 (30 selected) Spaces: 4 UTF-8 LF C Linux



```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

And then we use this read system call to read the content from the file descriptor. So, read system call takes three arguments. First is the file descriptor itself. Second is a pointer to the region where it can store the data. So, we have used this c variable here. And finally, the number of bytes which we want to read.

(Refer Slide Time: 2:49)

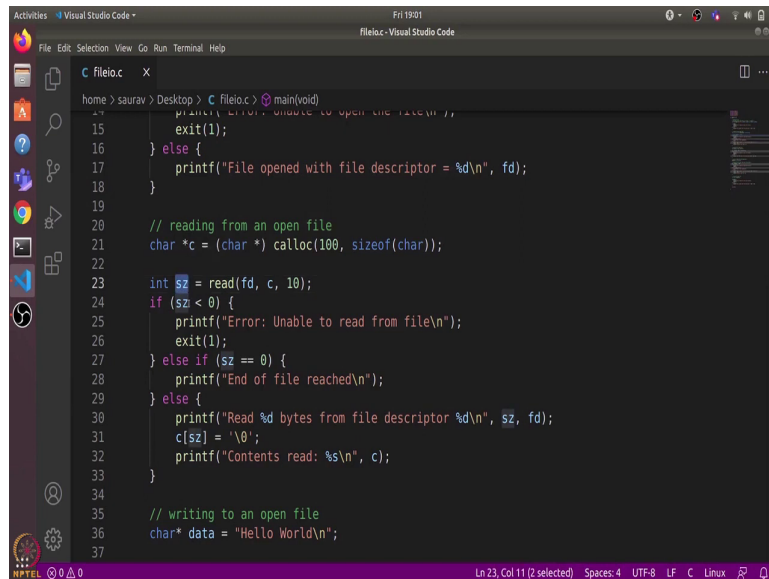


```
13 if (fd<0) {
14     printf("Error: Unable to open the file\n");
15     exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
```

```
Activities Visual Studio Code Fri 19:00
File Edit Selection View Go Run Terminal Help
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
14 printf("Error: Unable to open the file\n");
15 exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
```

```
Activities Visual Studio Code Fri 19:01
File Edit Selection View Go Run Terminal Help
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
14 printf("Error: Unable to open the file\n");
15 exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
```

```
Activities Visual Studio Code Fri 19:01
File Edit Selection View Go Run Terminal Help
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
14 printf("Error: Unable to open the file\n");
15 exit(1);
16 } else {
17     printf("File opened with file descriptor = %d\n", fd);
18 }
19
20 // reading from an open file
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
```

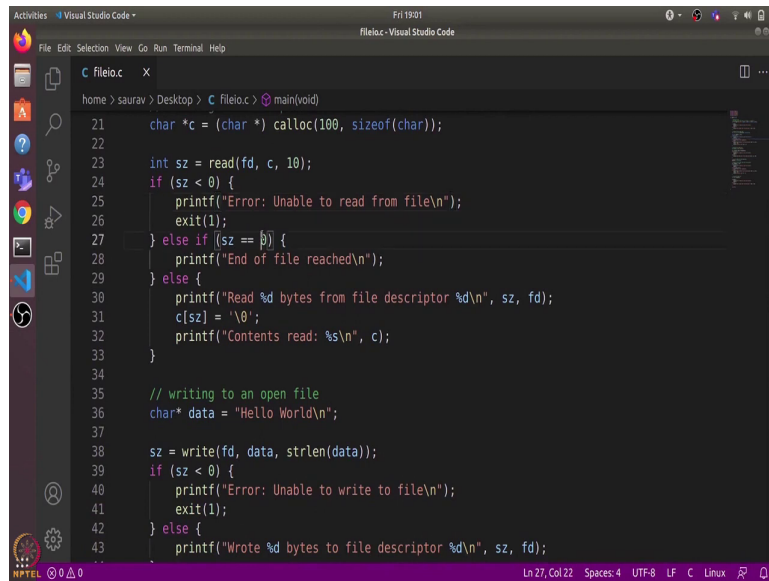


```
14 // If it fails, unable to open the file\n15 exit(1);\n16 } else {\n17     printf(\"File opened with file descriptor = %d\\n\", fd);\n18 }\n19\n20 // reading from an open file\n21 char *c = (char *) calloc(100, sizeof(char));\n22\n23 int sz = read(fd, c, 10);\n24 if (sz < 0) {\n25     printf(\"Error: Unable to read from file\\n\");\n26     exit(1);\n27 } else if (sz == 0) {\n28     printf(\"End of file reached\\n\");\n29 } else {\n30     printf(\"Read %d bytes from file descriptor %d\\n\", sz, fd);\n31     c[sz] = '\\0';\n32     printf(\"Contents read: %s\\n\", c);\n33 }\n34\n35 // writing to an open file\n36 char* data = \"Hello World\\n\";\n37
```

And this read system call returns the number of bytes that were actually read. So, for instance, if a file has just 5 bytes, then it will return 5 instead of 10. So, what happens when we make a read system call? On making this read system call, the process will use the file descriptor array index and access the open file table entry. And based on that open file table entry, it will access the inode. From that inode, it will see what all are the data blocks that stores the corresponding data that you want.

And it will check if that data block is already there in disk buffer cache. If it is not there in this buffer cache, then it will issue commands to the hard disk to pull the relevant data blocks into the disk buffer cache. And then it will copy the relevant contents from the data block to the region pointed by this address. And finally, it will return the number of bytes which were read.

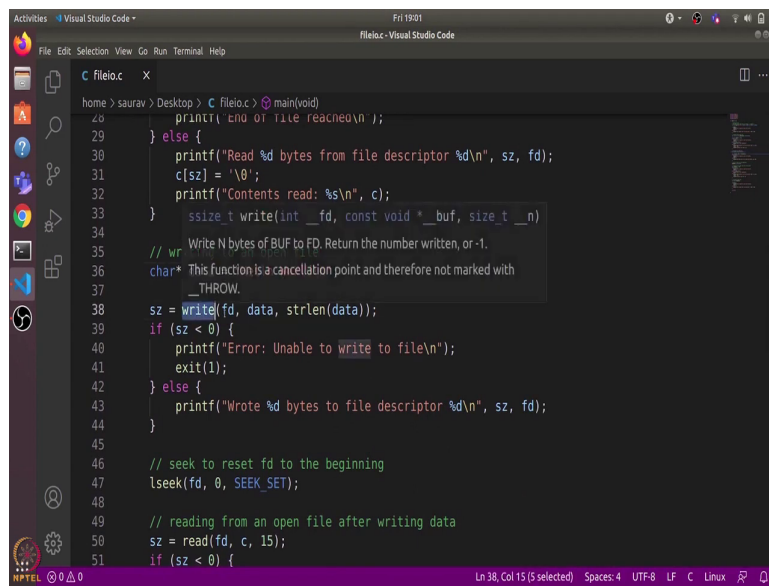
(Refer Slide Time: 3:41)



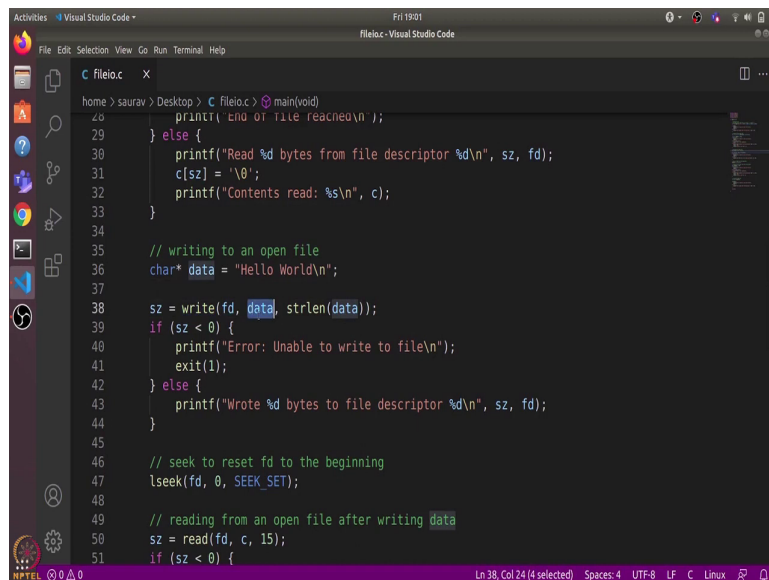
```
21 char *c = (char *) calloc(100, sizeof(char));
22
23 int sz = read(fd, c, 10);
24 if (sz < 0) {
25     printf("Error: Unable to read from file\n");
26     exit(1);
27 } else if (sz == 0) {
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
```

So, if it returns a negative value, that means there was some error. And if it returns 0, that means that we have reached the end of file, so we print out end of file reached. Otherwise, we print out these many bytes were read, and we print out the contents which are actually read.

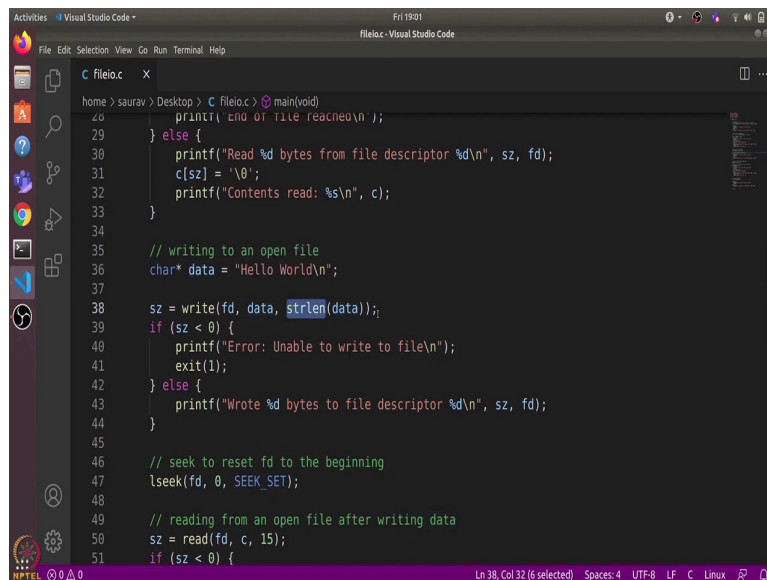
(Refer Slide Time: 3:55)



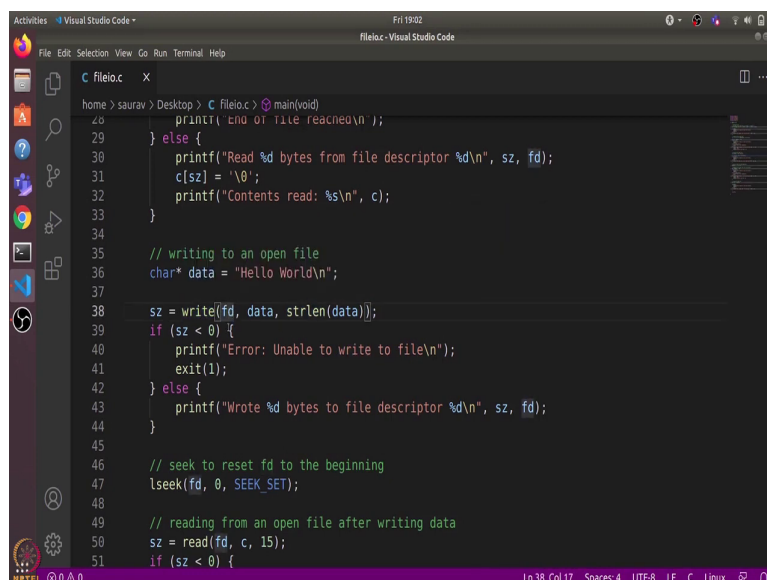
```
home > saurav > Desktop > C fileio.c > main(void)
28 printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34 // ssize_t write(int __fd, const void * __buf, size_t __n)
35 // Write N bytes of BUF to FD. Return the number written, or -1.
36 char* This function is a cancellation point and therefore not marked with
37 _THROW.
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
```



```
home > saurav > Desktop > C fileio.c > main(void)
28 printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
```

```
home > saurav > Desktop > C fileio.c > main(void)
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
```

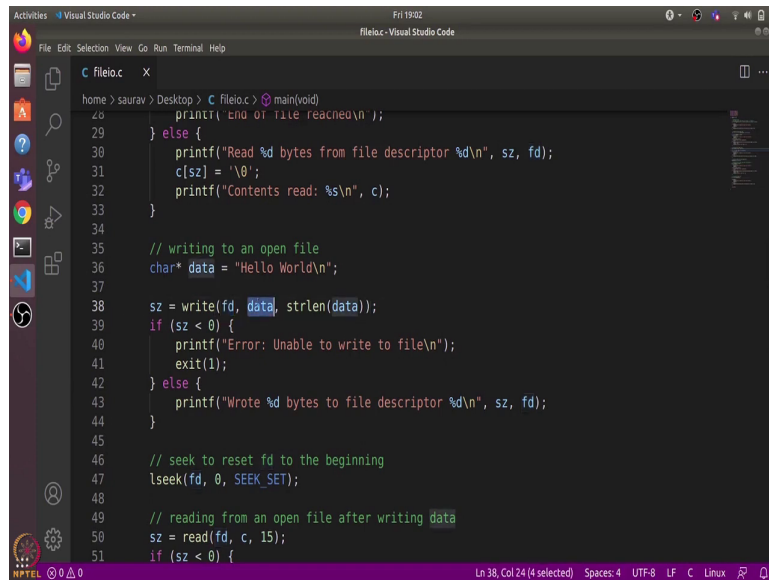


```
home > saurav > Desktop > C fileio.c > main(void)
28     printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
```

Now we use the write system call to write this Hello World in this file. So, write system call also takes first argument as the file descriptor. Second is the pointer to memory location from where we want to copy the data and write to the file. And the last argument says what is the size of data that we want to write.

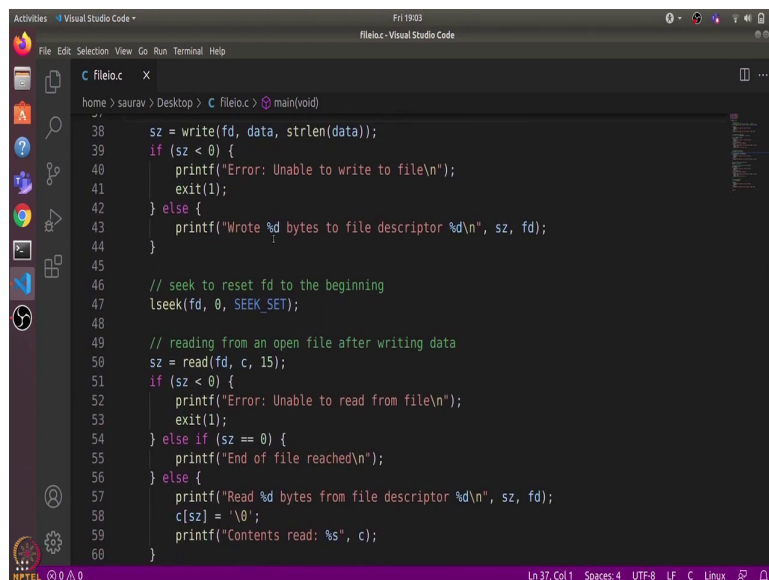
So, this write system call also returns the number of bytes which were actually return to the file. And what happens when we make this write system call based on the file descriptor, the system will access the relevant data block where we want to write the data. And in case, we want to write beyond the end of file, it will allocate a new data block and add its number to the file inode. And then it will pull this data block into the disk buffer cache.

(Refer Slide Time: 4:44)



```
home > saurav > Desktop > C fileio.c > main(void)
28 printf("End of file reached\n");
29 } else {
30     printf("Read %d bytes from file descriptor %d\n", sz, fd);
31     c[sz] = '\0';
32     printf("Contents read: %s\n", c);
33 }
34
35 // writing to an open file
36 char* data = "Hello World\n";
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
```

Ln 38, Col 24 (4 selected) Spaces: 4 UTF-8 LF C Linux

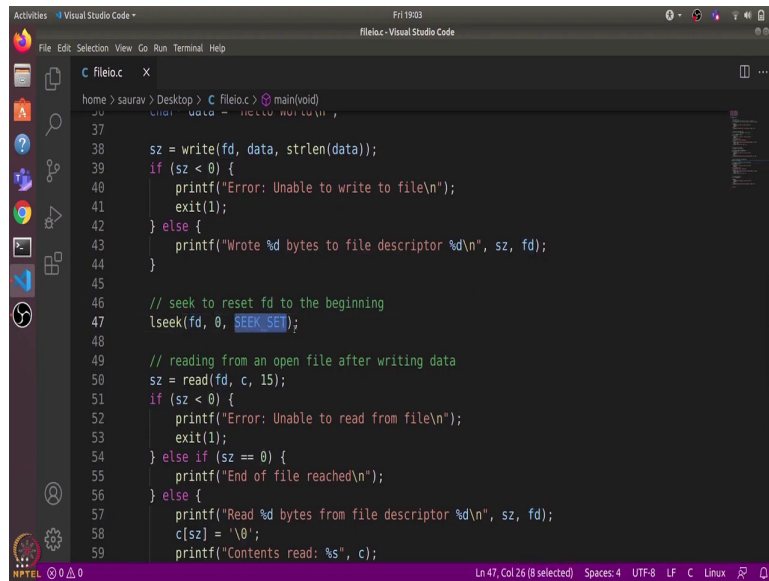


```
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
```

Ln 37, Col 1 Spaces: 4 UTF-8 LF C Linux

And then copy the data from this location to the data block. And finally return the number of bytes which are written, so if it returns a negative value that means that there was some error otherwise we print out that it has written these many bytes to the file descriptor.

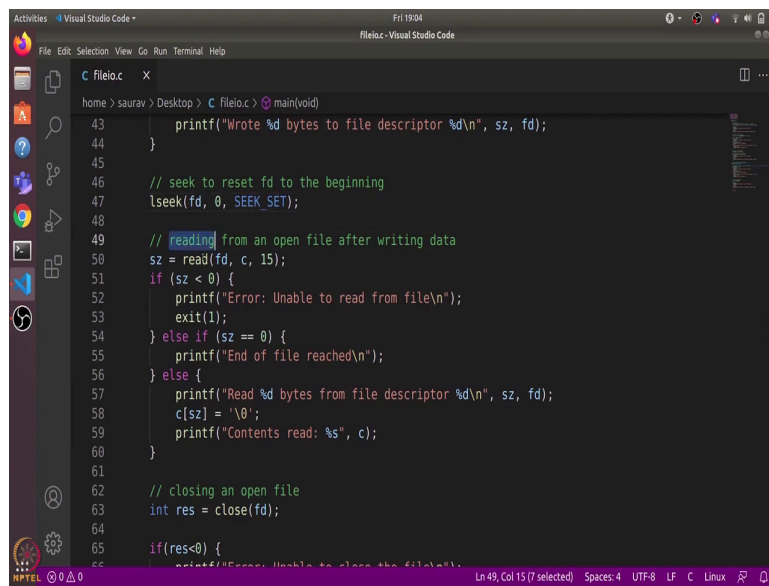
(Refer Slide Time: 4:58)



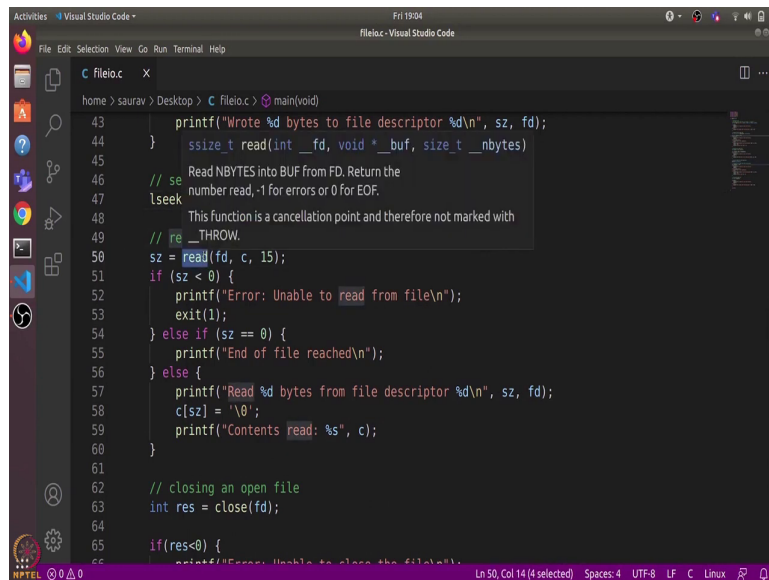
```
37
38 sz = write(fd, data, strlen(data));
39 if (sz < 0) {
40     printf("Error: Unable to write to file\n");
41     exit(1);
42 } else {
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
```

Now after writing to the file descriptor, the offset inside the inode will point to the end of file. And if you want to read the data again, we need to use the seek system call to again reset the offset to 0. So, we use the lseek function which takes in the file descriptor, the offset that we want to set. And if you want to set this offset in absolute way, or we want to set it related to the current offset position.

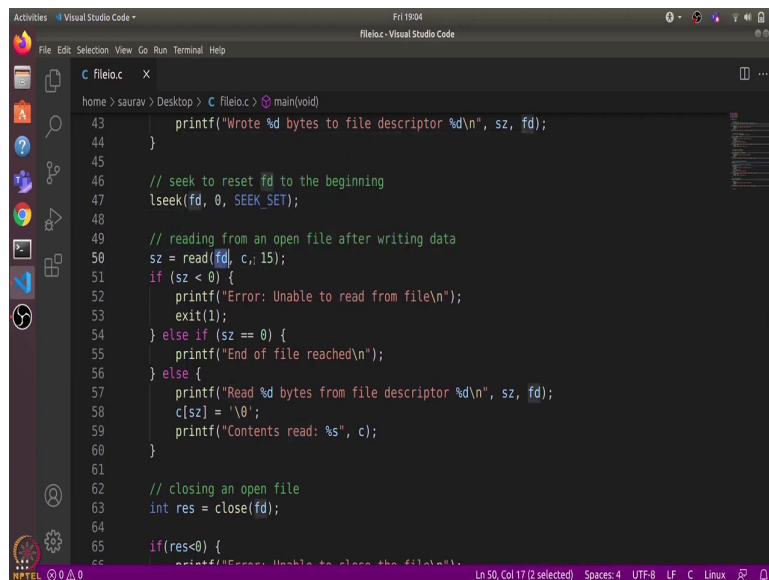
(Refer Slide Time: 5:26)



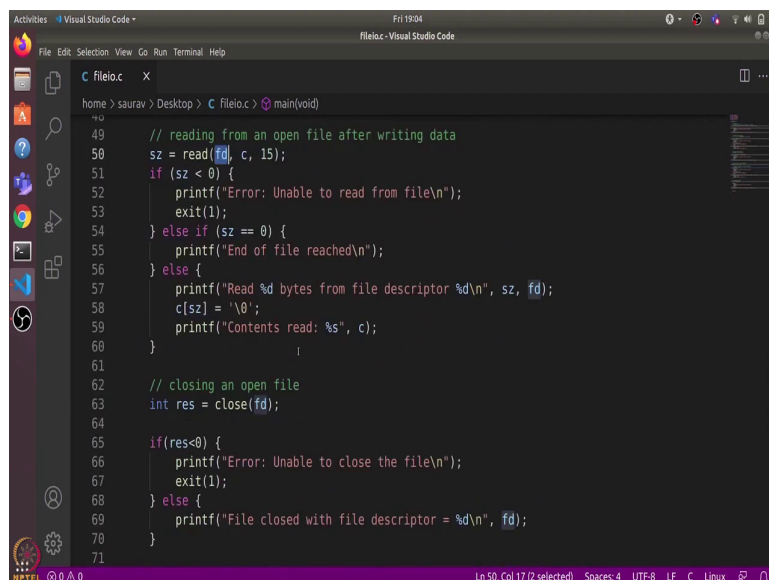
```
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if (res < 0) {
66     printf("Error: Unable to close the file\n");
67 }
```



```
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45     ssize_t read(int __fd, void *__buf, size_t __nbytes)
46 // se Read NBYTES into BUF from FD. Return the
47 lseek number read, -1 for errors or 0 for EOF.
48 // re This function is a cancellation point and therefore not marked with
49 // re _THROW.
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if (res < 0) {
66     printf("Error: Unable to close the file\n");
67 }
```



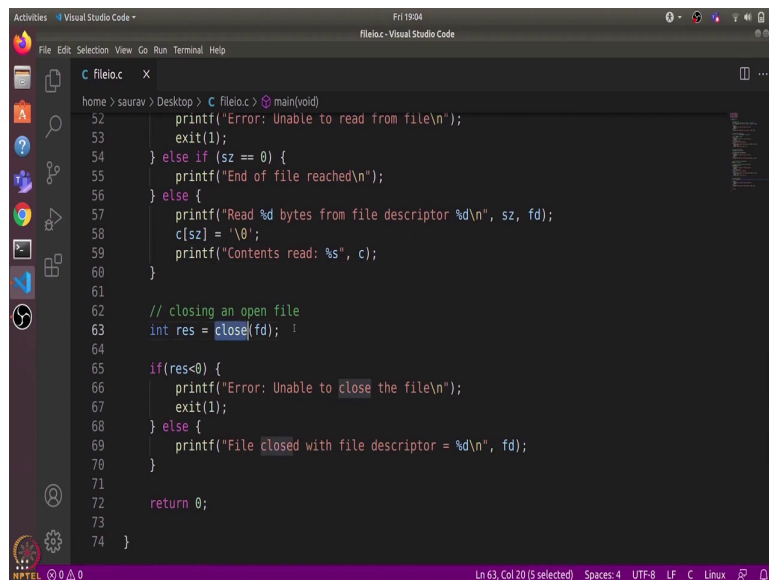
```
File Edit Selection View Go Run Terminal Help
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
43     printf("Wrote %d bytes to file descriptor %d\n", sz, fd);
44 }
45
46 // seek to reset fd to the beginning
47 lseek(fd, 0, SEEK_SET);
48
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if(res<0) {
66     printf("Error: Unable to close the file\n");
67 }
```



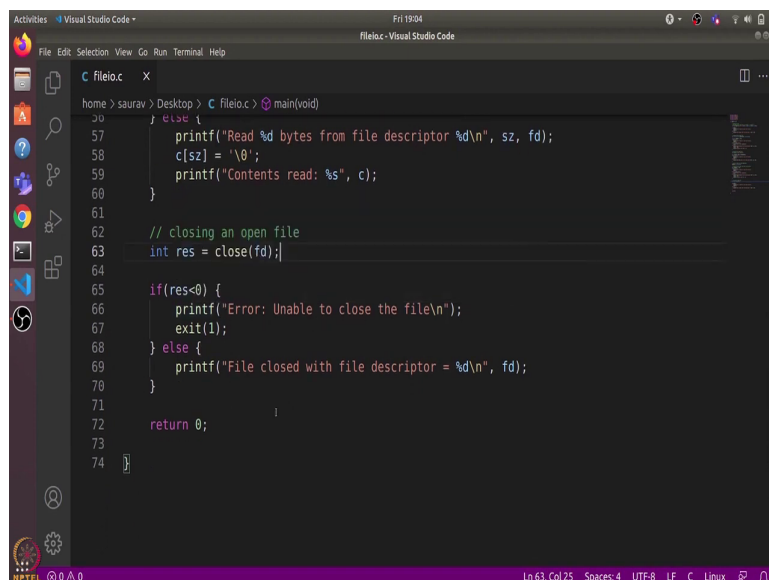
```
File Edit Selection View Go Run Terminal Help
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
49 // reading from an open file after writing data
50 sz = read(fd, c, 15);
51 if (sz < 0) {
52     printf("Error: Unable to read from file\n");
53     exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if(res<0) {
66     printf("Error: Unable to close the file\n");
67     exit(1);
68 } else {
69     printf("File closed with file descriptor = %d\n", fd);
70 }
71 }
```

After setting the offset back to 0, we use the read system call to again read contents from this file descriptor, we try to read 15 bytes, but because we have just written 12 bytes, it will just read these many bytes, and it will print out the contents.

(Refer Slide Time: 5:41)



```
home > saurav > Desktop > C fileio.c > main(void)
52 printf("Error: Unable to read from file\n");
53 exit(1);
54 } else if (sz == 0) {
55     printf("End of file reached\n");
56 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if(res<0) {
66     printf("Error: Unable to close the file\n");
67     exit(1);
68 } else {
69     printf("File closed with file descriptor = %d\n", fd);
70 }
71
72 return 0;
73
74 }
```



```
home > saurav > Desktop > C fileio.c > main(void)
50 } else {
57     printf("Read %d bytes from file descriptor %d\n", sz, fd);
58     c[sz] = '\0';
59     printf("Contents read: %s", c);
60 }
61
62 // closing an open file
63 int res = close(fd);
64
65 if(res<0) {
66     printf("Error: Unable to close the file\n");
67     exit(1);
68 } else {
69     printf("File closed with file descriptor = %d\n", fd);
70 }
71
72 return 0;
73
74 }
```

Finally, we use this closed system call to close the file descriptor. This will remove all the entries which are made in the open file table and the file descriptor array. And it will return 0 in case of success. Otherwise, if it returns a negative value, that means there were some error, and that would end the program.

(Refer Slide Time: 5:59)

```
Activities Terminal Fri 19:05 fileio.c - Visual Studio Code
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8     // opening a file
9     // int open (const char* Path, int flags, mode_t mode);
10    int fd = open("test.txt", O_RDWR | O_CREAT, 0777);
11
12    if (fd < 0) {
13        printf("Error: Unable to open the file\n");
14        exit(1);
15    } else {
16        printf("File opened with file descriptor = %d\n", fd);
17    }
18
19    // writing to a file
20    char *c = (char *) calloc(100, sizeof(char));
21    int sz = read(fd, c, 10);
22    if (sz < 0) {
23        printf("Error: Unable to read from the file\n");
24        exit(1);
25    }
26
27    printf("Contents read: %s\n", c);
28    // closing the file
29    close(fd);
30    printf("File closed with file descriptor = %d\n", fd);
31}
saurav@ubuntu-18:~/Desktop$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc fileio.c
saurav@ubuntu-18:~/Desktop$ ./a.out
File opened with file descriptor = 3
End of file reached
Wrote 12 bytes to file descriptor 3
Read 12 bytes from file descriptor 3
Contents read: Hello World
File closed with file descriptor = 3
saurav@ubuntu-18:~/Desktop$
```

```
Activities Terminal Fri 19:05 fileio.c - Visual Studio Code
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8     // opening a file
9     // int open (const char* Path, int flags, mode_t mode);
10    int fd = open("test.txt", O_RDWR | O_CREAT, 0777);
11
12    if (fd < 0) {
13        printf("Error: Unable to open the file\n");
14        exit(1);
15    } else {
16        printf("File opened with file descriptor = %d\n", fd);
17    }
18
19    // writing to a file
20    char *c = (char *) calloc(100, sizeof(char));
21    int sz = read(fd, c, 10);
22    if (sz < 0) {
23        printf("Error: Unable to read from the file\n");
24        exit(1);
25    }
26
27    printf("Contents read: %s\n", c);
28    // closing the file
29    close(fd);
30    printf("File closed with file descriptor = %d\n", fd);
31}
saurav@ubuntu-18:~/Desktop$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc fileio.c
saurav@ubuntu-18:~/Desktop$ ./a.out
File opened with file descriptor = 3
End of file reached
Wrote 12 bytes to file descriptor 3
Read 12 bytes from file descriptor 3
Contents read: Hello World
File closed with file descriptor = 3
saurav@ubuntu-18:~/Desktop$
```

```
Activities Terminal Fri 19:05 fileio.c - Visual Studio Code
C fileio.c x
home > saurav > Desktop > C fileio.c > main(void)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6
7 int main(void) {
8     // opening a file
9     // int open (const char* Path, int flags, mode_t mode);
10    int fd = open("test.txt", O_RDWR | O_CREAT, 0777);
11
12    if (fd < 0) {
13        printf("Error: Unable to open the file\n");
14        exit(1);
15    } else {
16        printf("File opened with file descriptor = %d\n", fd);
17    }
18
19    // reading from an open file
20    char *c = (char *) calloc(100, sizeof(char));
21    int sz = read(fd, c, 10);
22    if (sz < 0) {
23        printf("Error: Unable to read from the file\n");
24        exit(1);
25    }
26
27    printf("Contents read: %s\n", c);
28    // closing the file
29    close(fd);
30    printf("File closed with file descriptor = %d\n", fd);
31}
saurav@ubuntu-18:~/Desktop$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc fileio.c
saurav@ubuntu-18:~/Desktop$ ./a.out
File opened with file descriptor = 3
End of file reached
Wrote 12 bytes to file descriptor 3
Read 12 bytes from file descriptor 3
Contents read: Hello World
File closed with file descriptor = 3
saurav@ubuntu-18:~/Desktop$
```

So, let us compile and run this program. So, we will open a terminal and compile this program `gcc fileio.c`, and it creates the `a.out` executable let us run that executable. So, we can see that it opened a new file with file descriptor 3. And why is it 3? Because 0, 1, 2 are already used for standard in, standard out and standard error. And as expected, it prints out end of file reached, because there is no data initially in this new file.

Then it writes 12 bytes to the file descriptor. And finally, again reads that 12 bytes which were written there, and the contents are Hello World. And then it closes this file descriptor. So, let us see if we have that `test.txt` file here. So, we have this `test.txt` file. And if we check its contents, we can use `cat text.txt`. So the contents are Hello World. So, that is it for this video. Thanks and have a nice day.