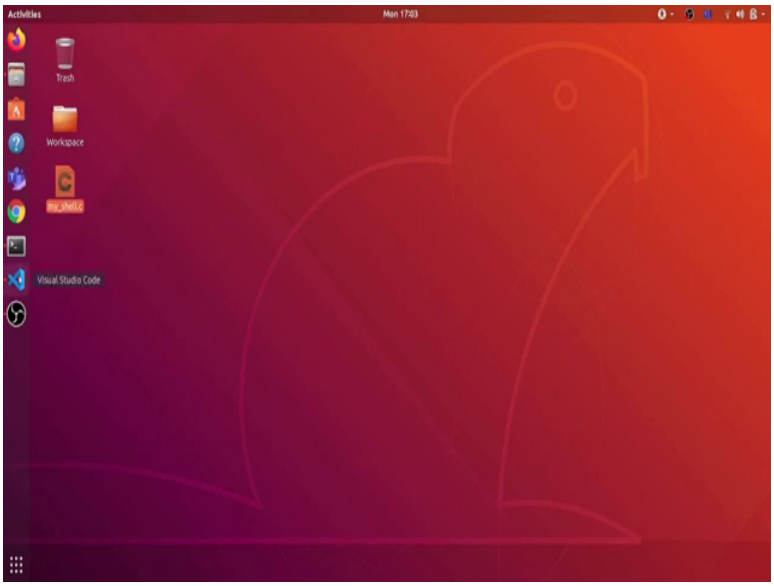
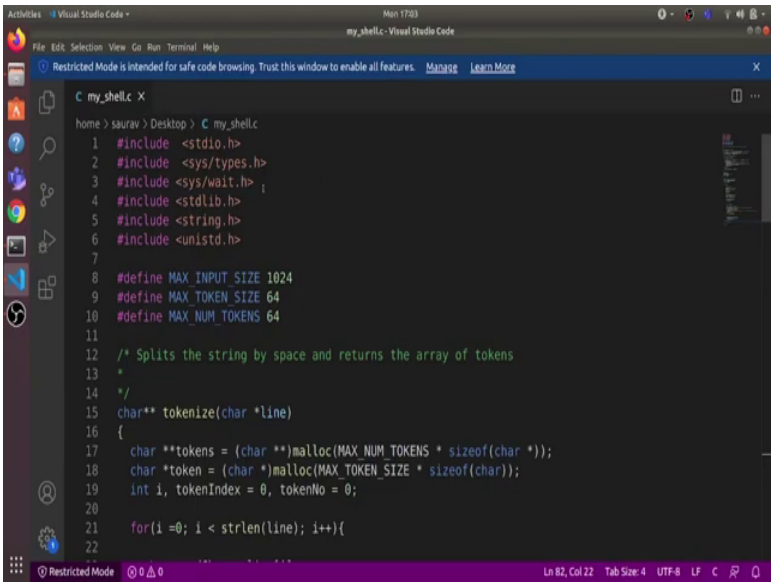
Design and Engineering of Computer Systems
Professor Mythilli Vutukuru
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
A Simple Shell

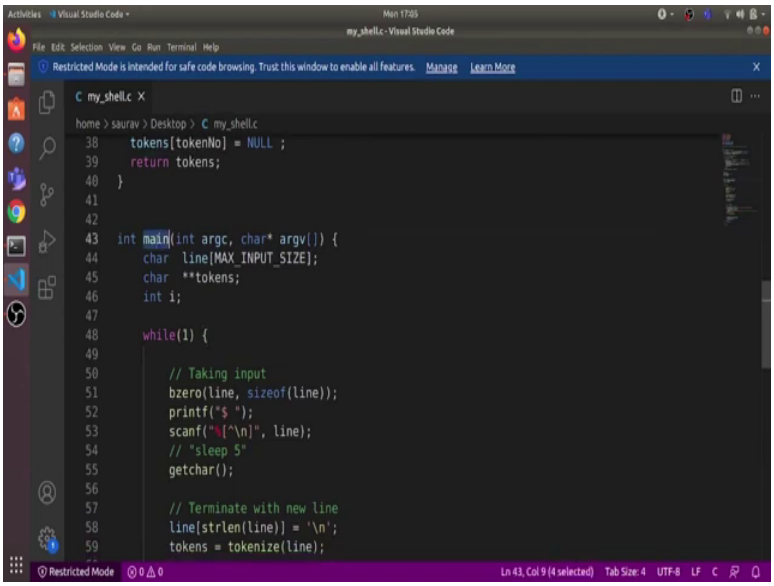
(Refer Slide Time: 00:15)



Hi students. In this video, we will see how to write a very simple Linux shell in C programming language. So, here, I have written this my_underscore shell dot C programme. Let us open it in visual code.

(Refer Slide Time: 00:29)



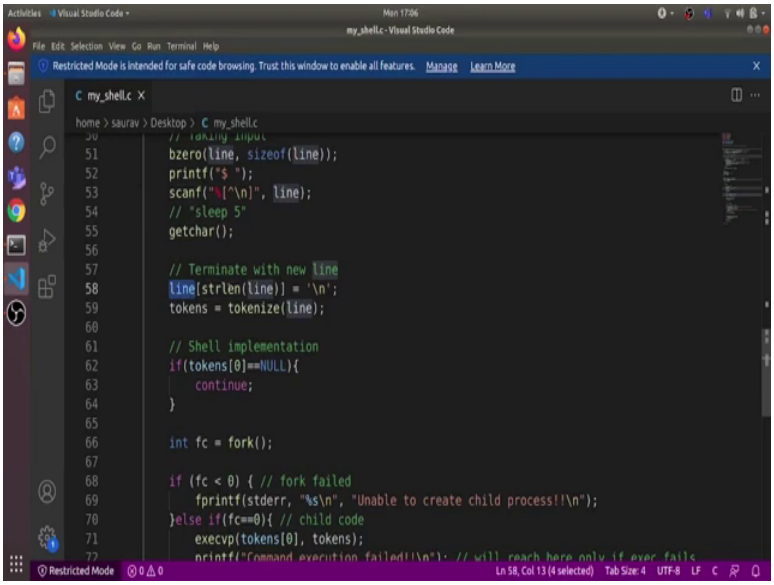


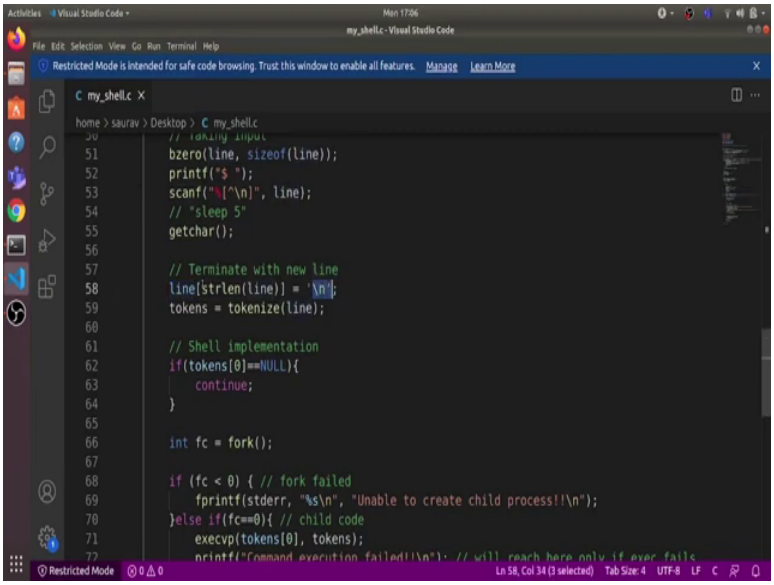
So, this is the my_underscore shell dot C. And let us go over the code line by line. So, here is the main programme. First of all, what do we want our my shell programme to do? We want our my shell programme to first display some prompt to the user and then take command from the user as an input and then it should execute that command and again, show another prompt the user to take in another command and repeat.

So, we have this while loop in the code. And what it does is first it prints a prompt. So, here it is $, followed by a space, then it takes in input. So, it will take all the characters up to the newline

character and store it in the line variable. And then we are using a getchar. So, that it waits for the user to enter this newline character.

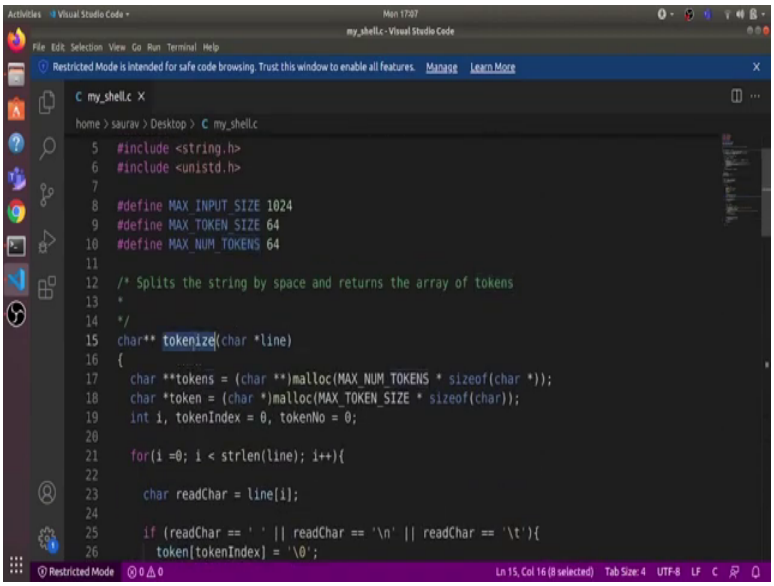
(Refer Slide Time: 01:27)

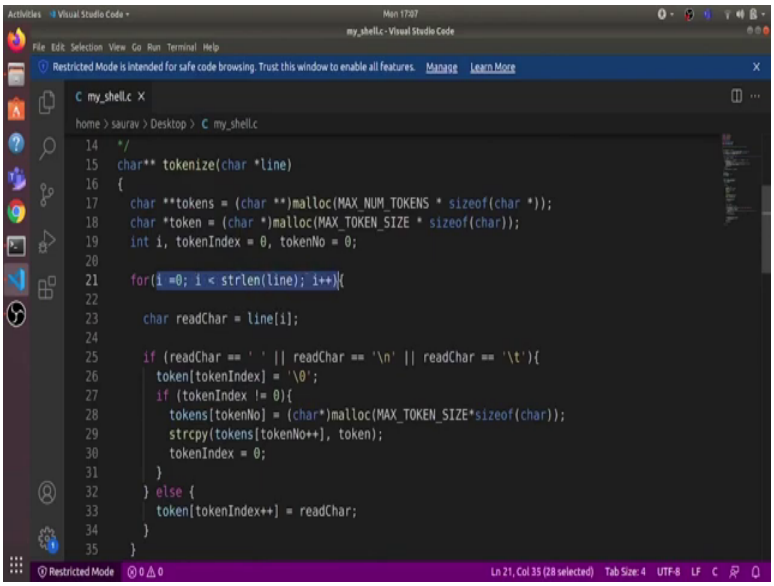




We want this line variable to have that string which ends with a newline character. So, we just add this newline character at the end of the line string. And then we tokenize this line, and what does this tokenize do? This tokenize will separate this line string into different words. So, let us say user enter sleep 5, then tokenize will separate of the sleep and 5 as separate strings in a string array. So, tokens is a char **.

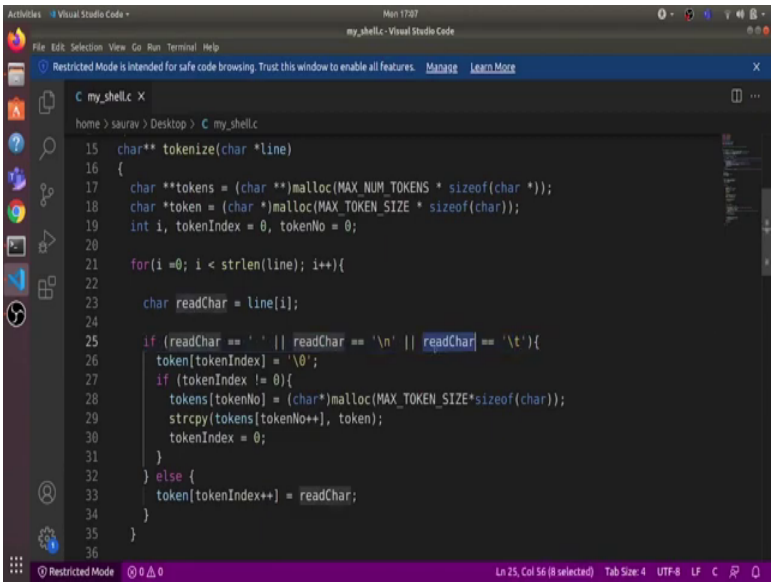
(Refer Slide Time: 02:00)





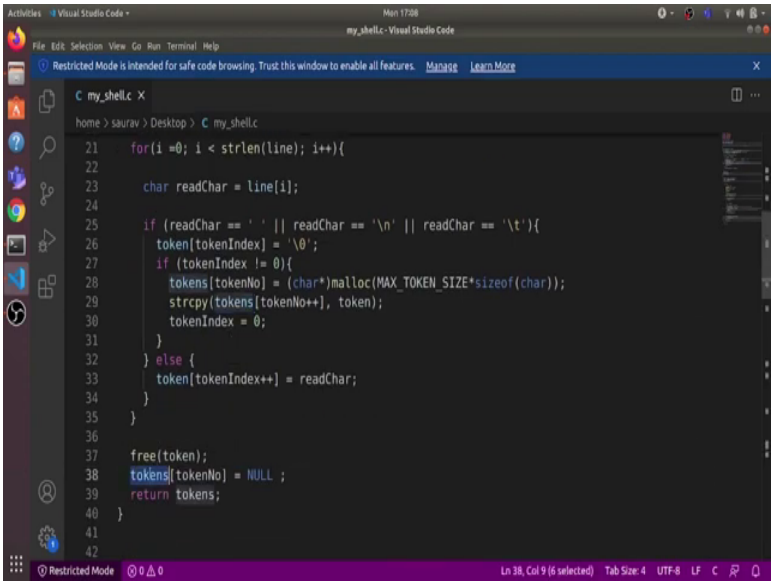
And let us see this tokenize implementation. So, it takes this character as an input, and it goes over every character of the line. And it checks if the character is a whitespace character.

(Refer Slide Time: 02:14)



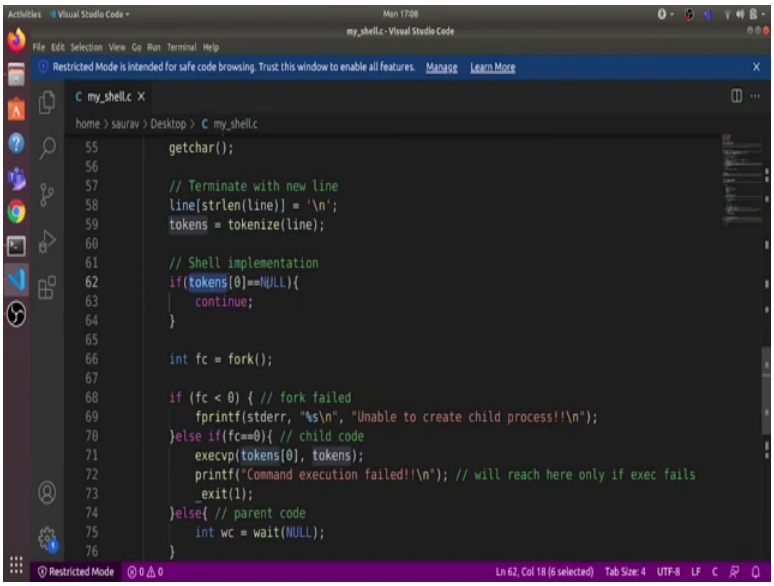
If it is not a whitespace character, then it adds that character to the token. And if it is a whitespace character, then whatever token it has stored till now it stores it in the token number position in the tokens array and increases the token number by 1.

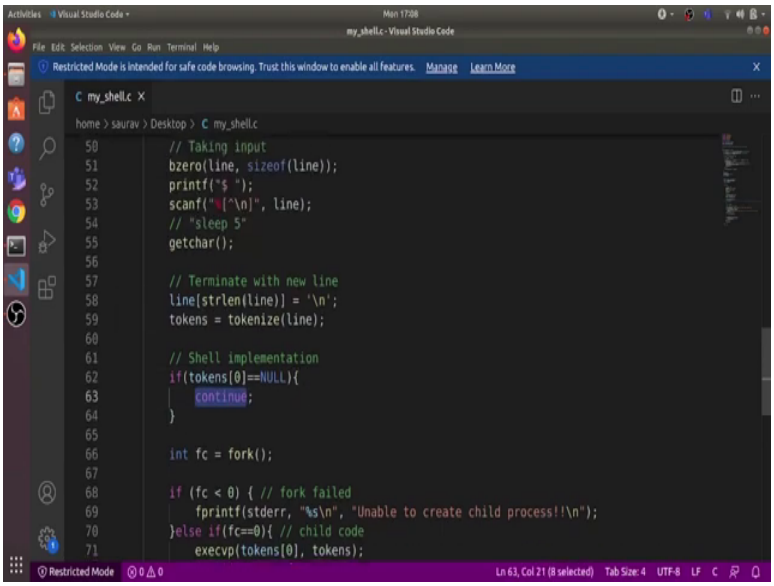
(Refer Slide Time: 02:34)

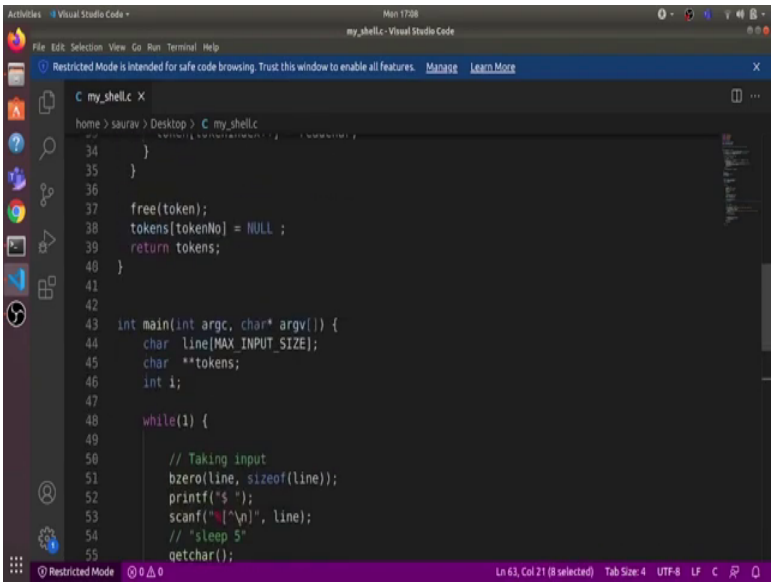


And then resets the token index 0, so that we can start taking the new token again. So this is how it separates out word by words and create these tokens array and finally returns this tokens array.

(Refer Slide Time: 02:54)



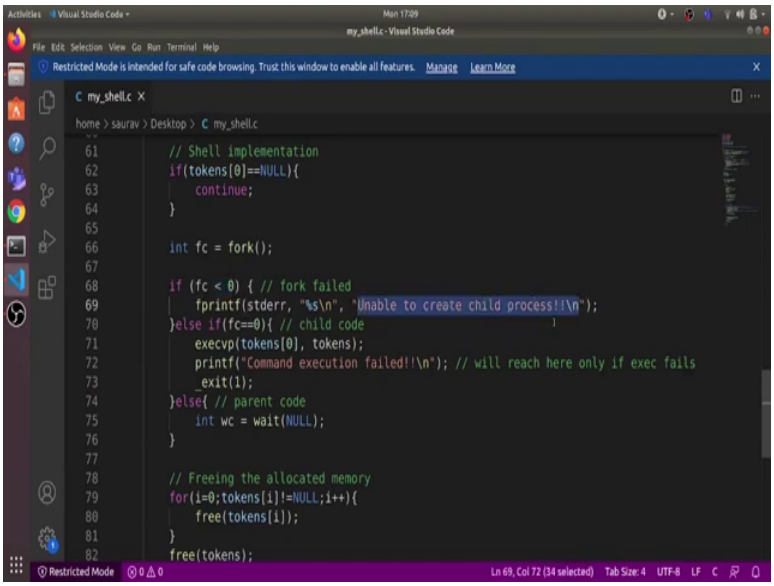




Let us, continue. Now, here is where the shell implementation begins. If token 0 is null, which means that user has directly pressed enter without entering anything, then we want this programme to continue and just again display a new prompt to the user. So, if tokens 0 is not null, that means user has entered some command.

So, we first use the fork system call to create a new child process. So, what does the fork do? Fork will create a new child process, this new child process will have the exact same memory image as the parent child process. But what will be different is the return value of fork in parent and child. In parent the fork will return the PID of the child as return value and in child, fork will return 0 as return value.

(Refer Slide Time: 03:39)

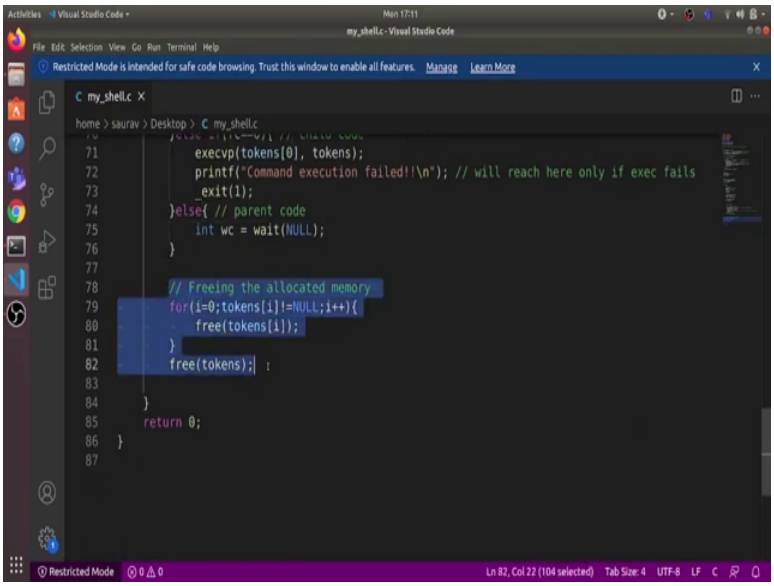


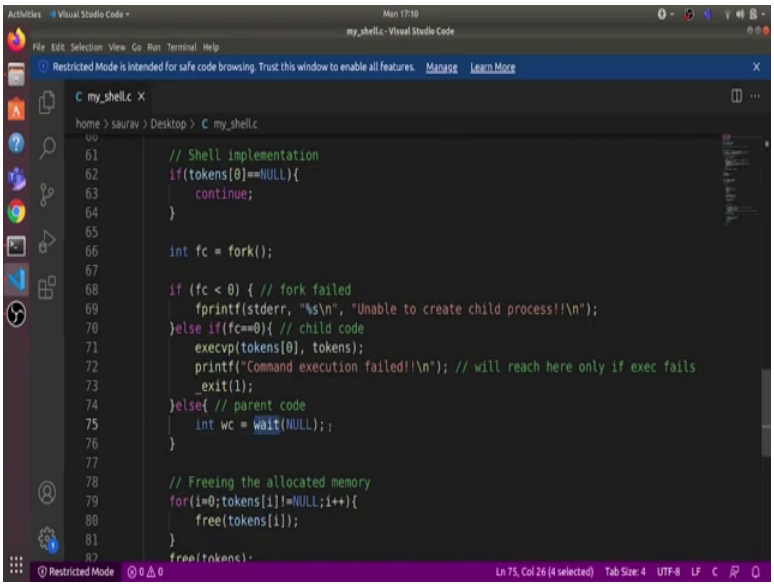
So, here we have different conditions based on fc. So, if fc is negative, which means that fork has fail due to some reason, then we just print in the std error unable to create child process. Otherwise, if fc 0, which means child will execute this part of code, we are using this execvp system call. Now what does this execvp system call do?

Execvp system call is just a variant of the exec system call, it takes in an executable name as the first argument and all the arguments that are to be supplied to that executable as a second argument in an array form. And then it searches for this executable in the Linux system.

And if it finds the executable, then it will reinitialize the memory native child process with the code of this executable. And in case this execvp fails, and it is unable to find this executable in the Linux system, then the control will reach here and the child will print command execution failed and then it will exit with the return value of 1.

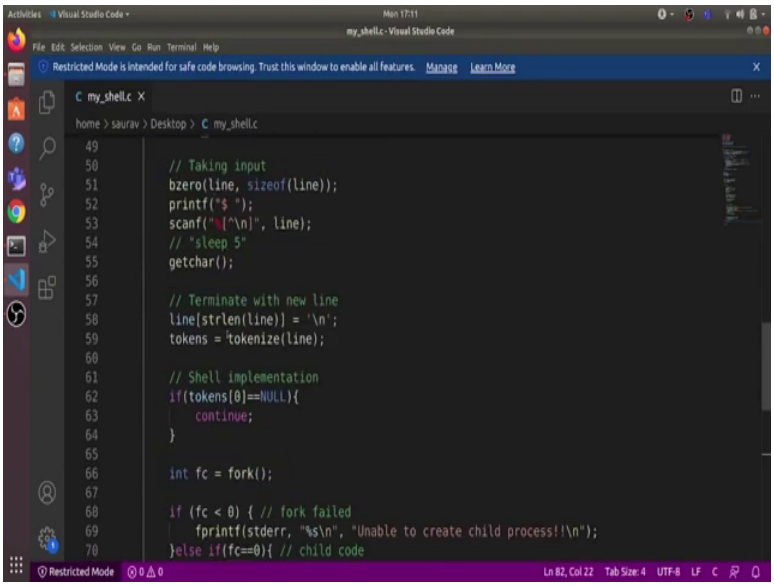
(Refer Slide Time: 04:43)

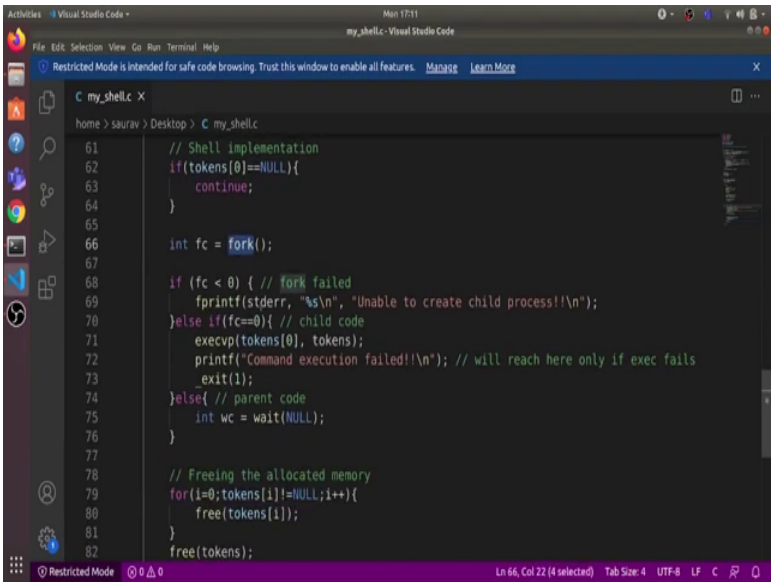


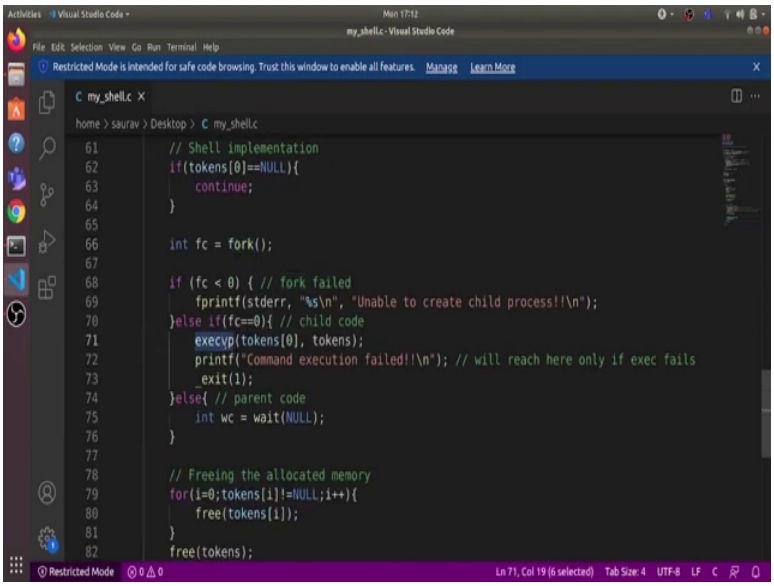


And what is parent do? Parent uses the wait system call to wait for the child to exit and once the child exits, parent reaps the child and clears all the memory trades used by the child. So, here we just free the memory which was allocated to their tokens array. And this is where this loop body ends and then it repeats.

(Refer Slide Time: 05:08)

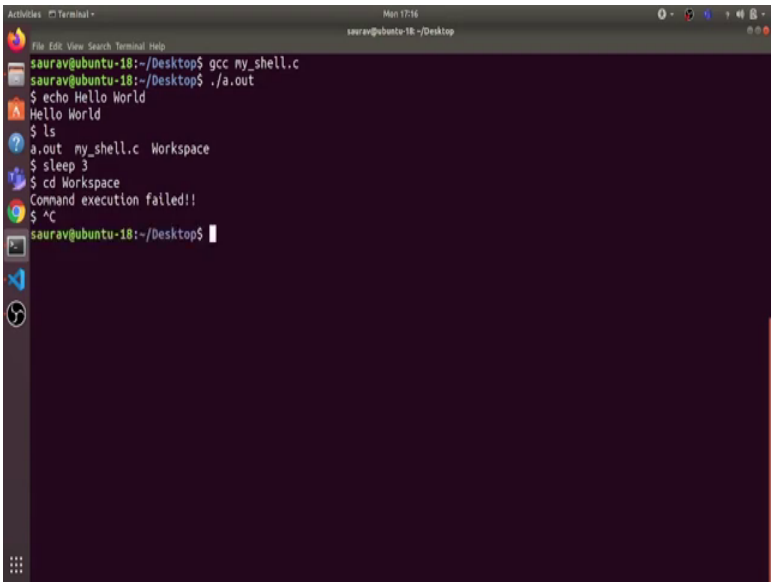






So, first shows the prompt, it takes in some input from the user, it creates a new child process. And in the child, it uses this execvp system call to replace child's code with the executable code. And then the child will execute that executable. And if it fails, due to some reason, then child will print command execution failed. And parent simply waits for the child to exit. And then it reaps the child and it repeats again and again.

(Refer Slide Time: 05:35)



So, let us compile this my_underscore shell dot C, and run the a.out. So, you can see that it is showing us the prompt, and let us enter some command. Let us, say echo Hello World. So, here it

printed Hello World. So, what happened here is parent forked a new child, and that execvp system call replaced the child's code with the code given in the executable of echo command and then the child executed that code to print out Hello World.

Finally, the child exited and the parent who was waiting for child to exit reaps the child. And then it again shows a new prompt so that we can enter other commands. So, let us run a few more commands, ls, so here, it shows us the list of all the files. Let us, see sleep 3, so it will sleep for 3 seconds, and then again, showing you new prompt. Let us, try to execute cd workspace. It says command execution failed. And why is that so?

Because it could not find an executable for cd, because cd is not implemented as an executable in Linux system, rather the shell programme has to implement the cd itself. So, what we can do is we can check if tokens 0 is cd and if that is so then we can use the chdir system call to change the directory, to implement this cd command. And we can exit this shell using ctrl plus c. So, that is it for this video. Thanks and have a nice day.