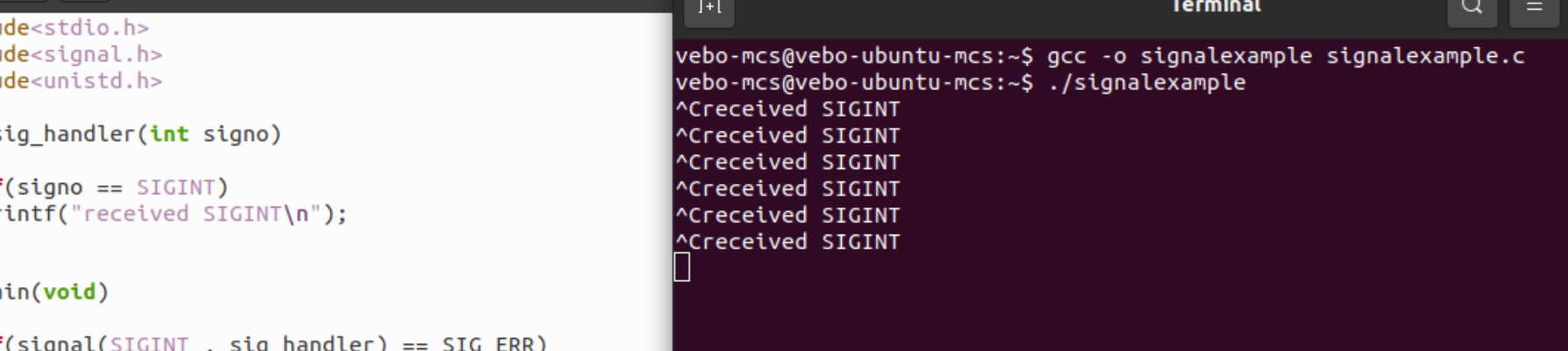


# PROGRAM 1



The screenshot shows a Linux desktop environment. On the left is a vertical dock with icons for Activities, Files, Terminal, Firefox, VS Code, a folder, PC, and a shopping bag. The main window is a code editor titled 'Terminal' showing a C program named 'sigexample.c'. The code defines a signal handler for SIGINT that prints 'received SIGINT\n' and then enters an infinite loop with a 1-second sleep. The terminal window shows the compilation and execution of the program, which successfully catches five SIGINT signals (Ctrl-C) and prints the message each time.

```
1 #include<stdio.h>
2 #include<signal.h>
3 #include<unistd.h>
4
5 void sig_handler(int signo)
6 {
7     if(signo == SIGINT)
8         printf("received SIGINT\n");
9 }
10
11 int main(void)
12 {
13     if(signal(SIGINT , sig_handler) == SIG_ERR)
14         printf("\n can't catch SIGINT\n");
15
16     while(1)
17         sleep(1);
18     return 0;
19 }
```

Terminal output:

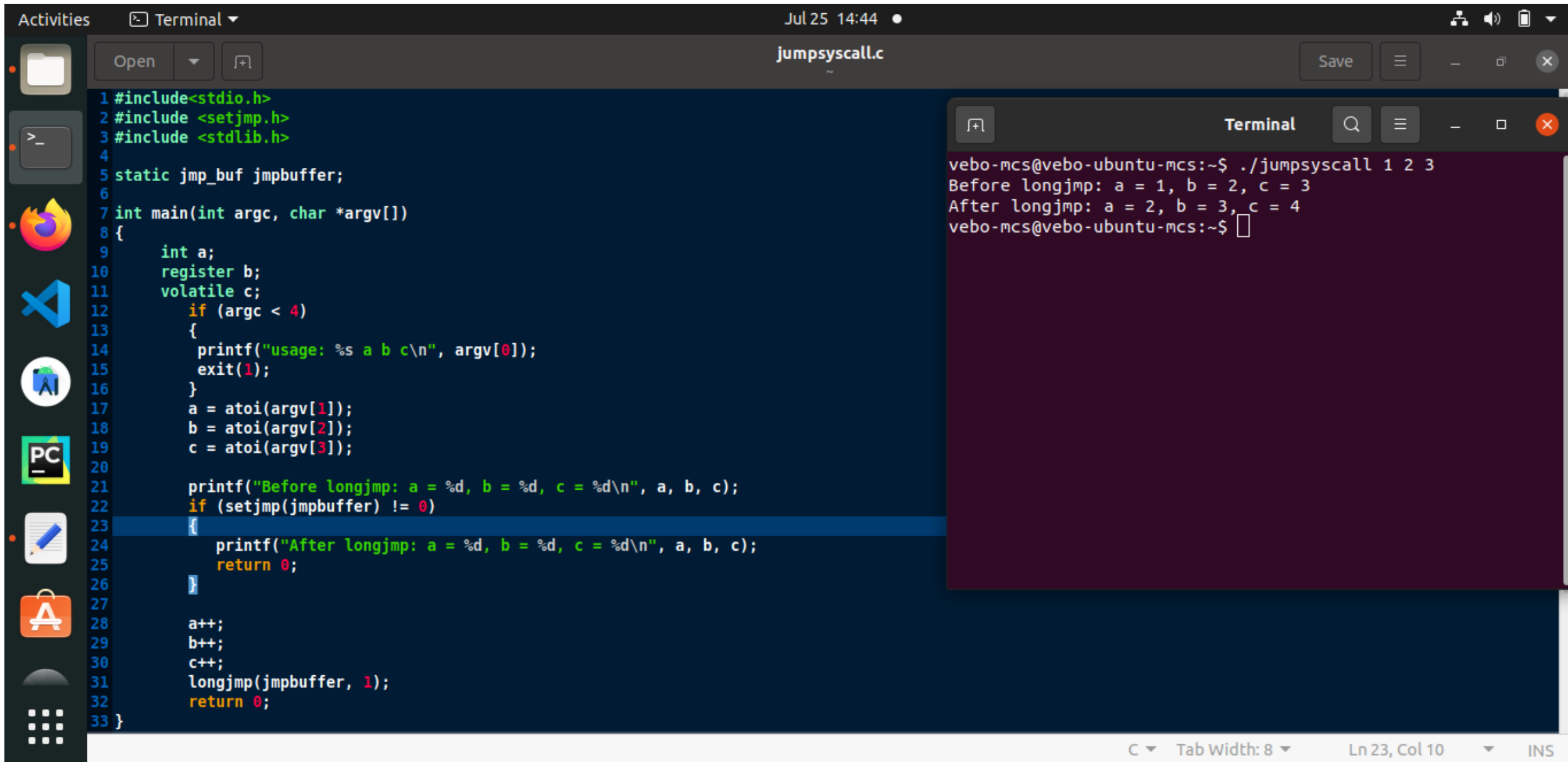
```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o sigexample sigexample.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./sigexample
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
```

## PROGRAM EXPLANATION 1

`sig_handler()` This function is registered to the kernel by passing it as the second argument of the system call 'signal' in the `main()` function.

`sleep(1)` This function has been used in the while loop so that while loop executes after some time.

## PROGRAM 2



The screenshot shows a code editor window titled 'jumpsyscall.c' with a dark theme. The code is a C program that demonstrates the use of `setjmp` and `longjmp`. It includes headers for `stdio.h`, `setjmp.h`, and `stdlib.h`. A static jump buffer `jmp_buf jmpbuffer;` is declared. The `main` function takes command-line arguments. It checks if there are at least four arguments; if not, it prints a usage message and exits. Otherwise, it converts the arguments to integers `a`, `b`, and `c`. It prints the values before the jump, sets the jump buffer with `setjmp`, prints the values after the jump, and then increments `a`, `b`, and `c` before jumping back to the start of the `if` block with `longjmp`. The terminal window on the right shows the execution of the program with arguments `1 2 3`, displaying the state before and after the long jump.

```
1 #include<stdio.h>
2 #include <setjmp.h>
3 #include <stdlib.h>
4
5 static jmp_buf jmpbuffer;
6
7 int main(int argc, char *argv[])
8 {
9     int a;
10    register b;
11    volatile c;
12    if (argc < 4)
13    {
14        printf("usage: %s a b c\n", argv[0]);
15        exit(1);
16    }
17    a = atoi(argv[1]);
18    b = atoi(argv[2]);
19    c = atoi(argv[3]);
20
21    printf("Before longjmp: a = %d, b = %d, c = %d\n", a, b, c);
22    if (setjmp(jmpbuffer) != 0)
23    {
24        printf("After longjmp: a = %d, b = %d, c = %d\n", a, b, c);
25        return 0;
26    }
27
28    a++;
29    b++;
30    c++;
31    longjmp(jmpbuffer, 1);
32    return 0;
33 }
```

Terminal Output:

```
vebo-mcs@vebo-ubuntu-mcs:~$ ./jumpsyscall 1 2 3
Before longjmp: a = 1, b = 2, c = 3
After longjmp: a = 2, b = 3, c = 4
vebo-mcs@vebo-ubuntu-mcs:~$
```

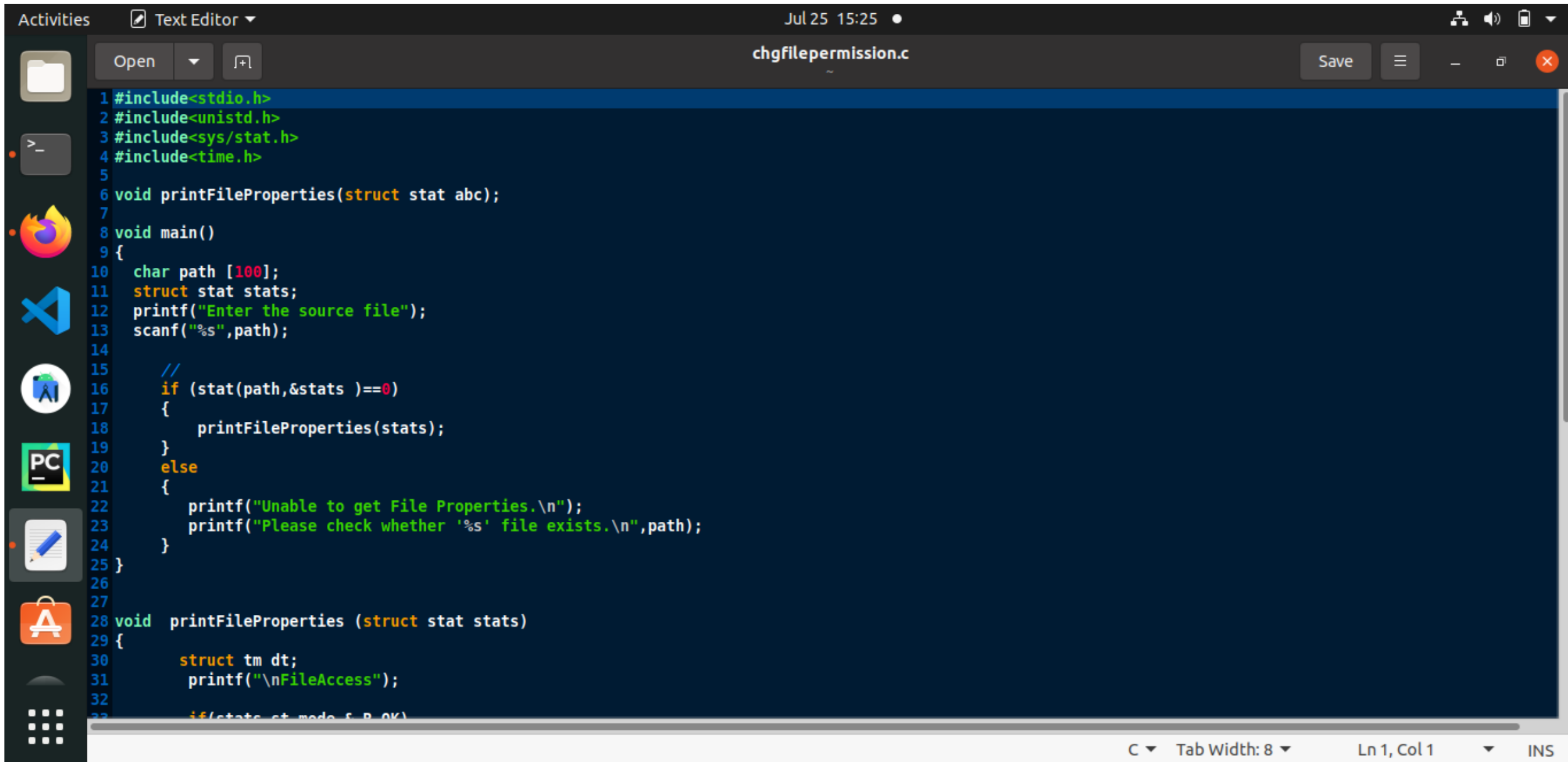
Footer: C Tab Width: 8 Ln 23, Col 10 INS

## PROGRAM EXPLANATION 2

Using `atoi()` we convert these argument into integers and set these values to variables `a,b,c`.after that we print the value of `a,b` and `c` we get `a=1,b=2` and `c=3`.

initially `setjmp()` false hence control comes out from if condition and increase the value of variables `a,b` and `c`. `longjmp()` transfers control to the point set by `setjmp()`and it prints the value of `a,b` and `c`. hence `a=2,b=3` and `c=4` prints.

## PROGRAM 3



```
1#include<stdio.h>
2#include<unistd.h>
3#include<sys/stat.h>
4#include<time.h>
5
6void printFileProperties(struct stat abc);
7
8void main()
9{
10    char path [100];
11    struct stat stats;
12    printf("Enter the source file");
13    scanf("%s",path);
14
15    //
16    if (stat(path,&stats )==0)
17    {
18        printFileProperties(stats);
19    }
20    else
21    {
22        printf("Unable to get File Properties.\n");
23        printf("Please check whether '%s' file exists.\n",path);
24    }
25 }
26
27
28void printFileProperties (struct stat stats)
29{
30    struct tm dt;
31    printf("\nFileAccess");
32
33    if(stats.st_mode & S_IRWXU)
```

Activities Text Editor Jul 25 15:25 chgfilepermission.c Save

C Tab Width: 8 Ln 1, Col 1 INS

ActivitiesTerminal

Jul 25 15:26

chgfilepermission.c

Save

24 }  
25 }  
26  
27  
28 void printFileProperties (struct stat stats)  
29 {  
30 struct tm dt;  
31 printf("\nFileAccess");  
32  
33 if(stats.st\_mode & R\_OK)  
34 printf("\t\tread");  
35  
36 if(stats.st\_mode & R\_OK)  
37 printf("\t\twrite");  
38  
39  
40 if(stats.st\_mode & R\_OK)  
41 printf("\t\texecute");  
42  
43 //file size  
44 printf("\n File Size: %ld",stats.st\_size);  
45  
46  
47 // Get File creation time in seconds and  
48 // Convert seconds to date and time format  
49  
50 dt=(gmtime(&stats.st\_ctime));  
51 printf("\nCreated on: %d-%d-%d %d:%d:%d",  
52 dt.tm\_mday , dt.tm\_mon , dt.tm\_year +1900, dt.tm\_hour , dt.tm\_min , dt.tm\_sec);  
53  
54  
55  
56 // File modification time  
57 dt =(gmtime(&stats.st\_mtime));  
58 printf("\nModified on : %d-%d-%d %d:%d:%d" ,  
59 dt.tm\_mday, dt.tm\_mon , dt.tm\_year + 1900 ,dt.tm\_hour , dt.tm\_min , dt.tm\_sec);  
60  
61 }

Terminal

vebo-mcs@vebo-ubuntu-mcs:~\$ gcc -o chgfilepermission chgfilepermission.c  
vebo-mcs@vebo-ubuntu-mcs:~\$ ./chgfilepermission  
Enter the source file name.txt  
  
FileAccess read write execute  
File Size: 25  
Created on: 25-6-2021 9:51:39  
vebo-mcs@vebo-ubuntu-mcs:~\$

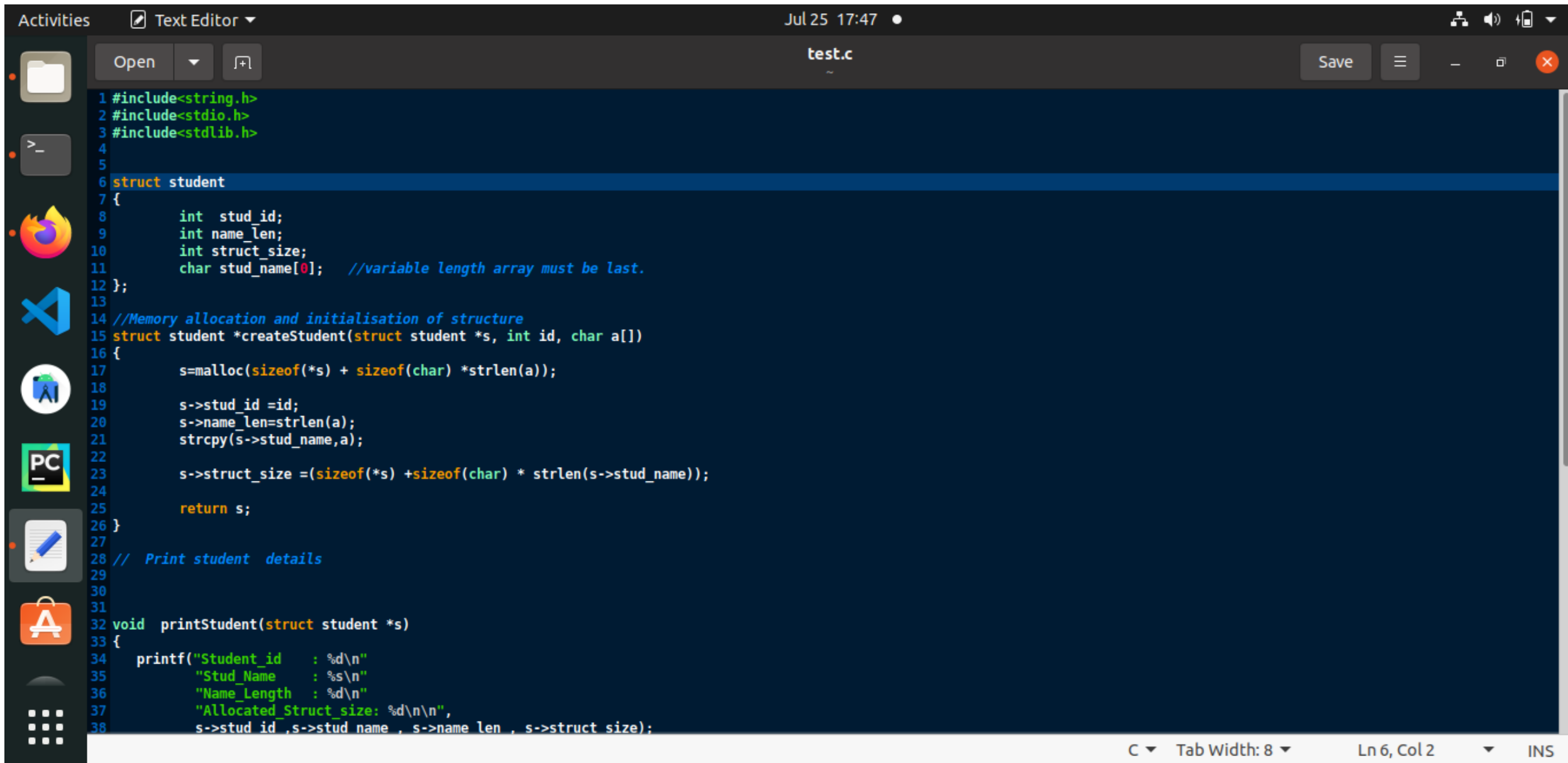
C Tab Width: 8 Ln 37, Col 35 INS

## PROGRAM EXPLANATION 4

stat() function is used to list properties of a file identified by path  
void printFileProperties(struct stat stats) function is used to print file properties

i.e File access, File size, file creation date and time, File modification date time

# PROGRAM 5



```
1 #include<string.h>
2 #include<stdio.h>
3 #include<stdlib.h>
4
5
6 struct student
7 {
8     int stud_id;
9     int name_len;
10    int struct_size;
11    char stud_name[0]; //variable length array must be last.
12 };
13
14 //Memory allocation and initialisation of structure
15 struct student *createStudent(struct student *s, int id, char a[])
16 {
17     s=malloc(sizeof(*s) + sizeof(char) *strlen(a));
18
19     s->stud_id =id;
20     s->name_len=strlen(a);
21     strcpy(s->stud_name,a);
22
23     s->struct_size =(sizeof(*s) +sizeof(char) * strlen(s->stud_name));
24
25     return s;
26 }
27
28 // Print student details
29
30
31
32 void printStudent(struct student *s)
33 {
34     printf("Student_id   : %d\n"
35           "Stud_Name    : %s\n"
36           "Name_Length  : %d\n"
37           "Allocated_Struct_size: %d\n\n",
38           s->stud_id,s->stud_name , s->name_len , s->struct_size);
```



Open



test.c

Save



Terminal



```
27
28 // Print student details
29
30
31
32 void printStudent(struct student *s)
33 {
34     printf("Student_id   : %d\n"
35           "Stud_Name     : %s\n"
36           "Name_Length   : %d\n"
37           "Allocated_Struct_size: %d\n\n",
38           s->stud_id ,s->stud_name , s->name_len , s->struct_size);
39
40     //Value of Allocated_Struct_size here is in bytes.
41 }
42
43 //Driver code
44
45 int main()
46 {
47     struct student *s1 , *s2;
48     s1=createStudent(s1 , 523,"Bablu");
49     s2=createStudent(s2 , 535,"Guddu");
50
51     printStudent(s1);
52     printStudent(s2);
53
54     //size in bytes
55     printf("Size of struct student %lu\n", sizeof(struct student));
56     //size in bytes
57     printf("Size of struct student %lu\n", sizeof(s1));
58
59     return 0;
60 }
61
62
63
64
```

```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o test test.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./test
Student_id       : 523
Stud_Name        : Bablu
Name_Length      : 5
Allocated_Struct_size: 17

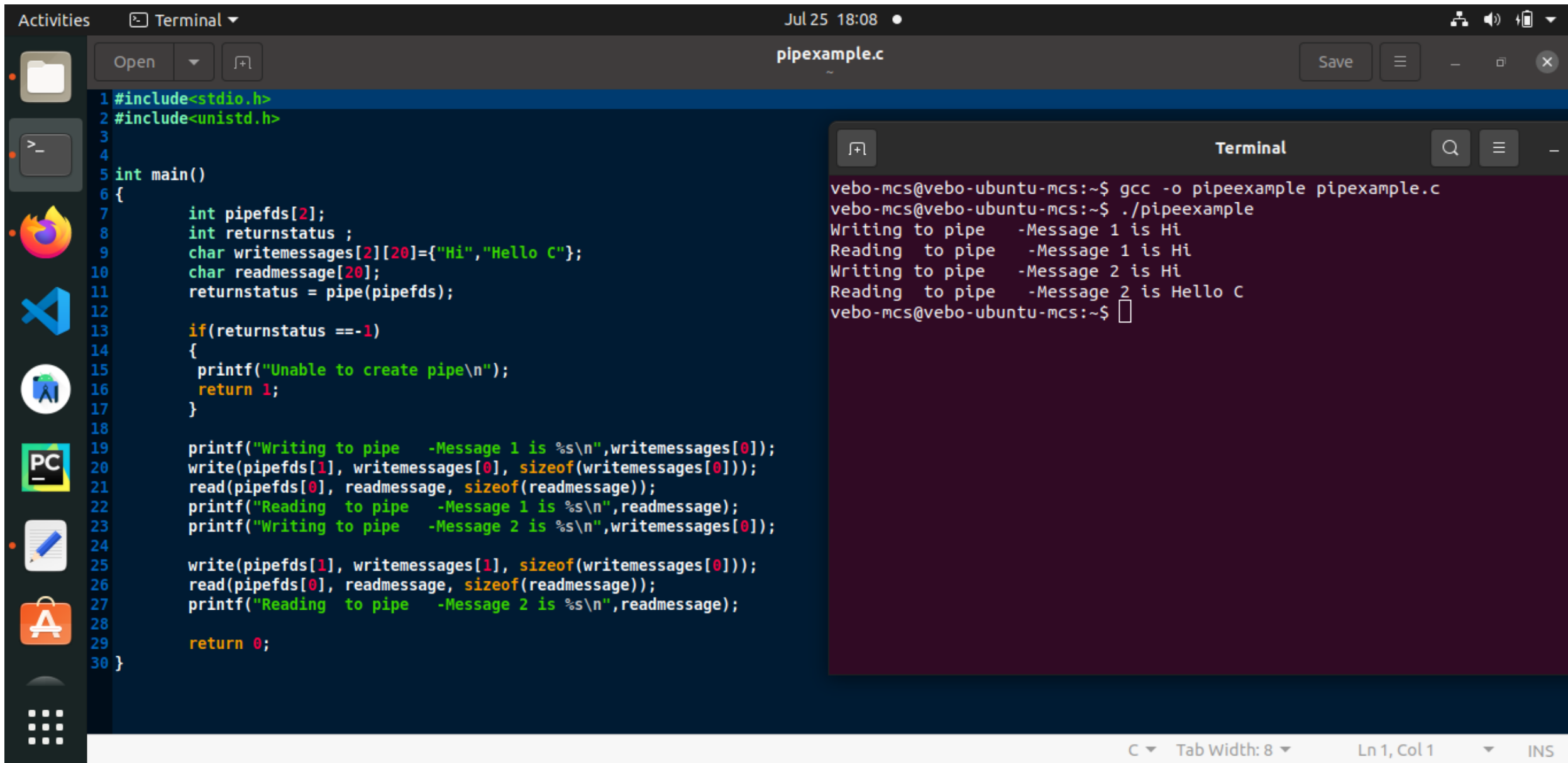
Student_id       : 535
Stud_Name        : Guddu
Name_Length      : 5
Allocated_Struct_size: 17

Size of struct student 12
Size of struct student 8
vebo-mcs@vebo-ubuntu-mcs:~$
```

## PROGRAM EXPLANATION 5

Structure – Structure is a user defined datatype which can hold different types of elements. In the above program we create a studentstructure which holds student data. We create two variables of structre s1 and s2. Inside main mehod we create a method createstudent() and pass the student values. Now control comes in createstudent(). Inside this function we allocate the memory for each structure variable using malloc(). Createstudent() returns student details and using printstudent() we display the student details

# PROGRAM 6



The image shows a code editor window titled "pipeexample.c" with a dark theme. The code is a C program that demonstrates pipe communication. It includes `<stdio.h>` and `<unistd.h>`. The `main` function creates a pipe, writes two messages ("Hi" and "Hello C") to it, and reads them back. It includes error handling for pipe creation. The terminal window on the right shows the compilation and execution of the program, displaying the output of the pipe operations.

```
1#include<stdio.h>
2#include<unistd.h>
3
4
5int main()
6{
7    int pipefds[2];
8    int returnstatus ;
9    char writemessages[2][20]={"Hi","Hello C"};
10   char readmessage[20];
11   returnstatus = pipe(pipefds);
12
13   if(returnstatus ==-1)
14   {
15       printf("Unable to create pipe\n");
16       return 1;
17   }
18
19   printf("Writing to pipe   -Message 1 is %s\n",writemessages[0]);
20   write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
21   read(pipefds[0], readmessage, sizeof(readmessage));
22   printf("Reading  to pipe   -Message 1 is %s\n",readmessage);
23   printf("Writing to pipe   -Message 2 is %s\n",writemessages[0]);
24
25   write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
26   read(pipefds[0], readmessage, sizeof(readmessage));
27   printf("Reading  to pipe   -Message 2 is %s\n",readmessage);
28
29   return 0;
30 }
```

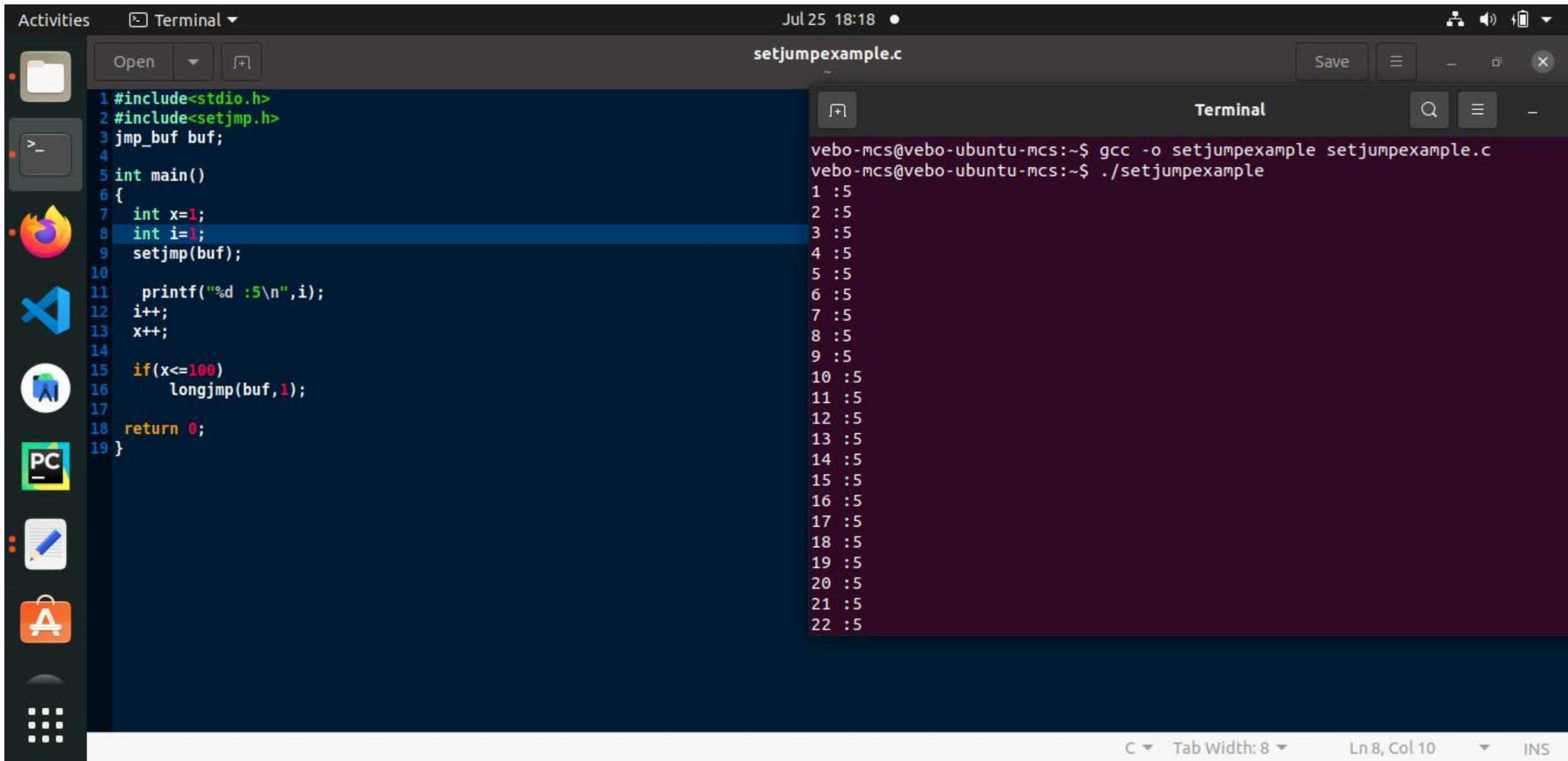
Terminal Output:

```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o pipeexample pipeexample.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./pipeexample
Writing to pipe   -Message 1 is Hi
Reading  to pipe   -Message 1 is Hi
Writing to pipe   -Message 2 is Hi
Reading  to pipe   -Message 2 is Hello C
vebo-mcs@vebo-ubuntu-mcs:~$
```

## PROGRAM EXPLANATION 6

`pipe(pipefds)`: This system call would create a pipe for one-way communication i.e., it creates two descriptors, first one is connected to read from the pipe and other one is connected to write into the pipe.

# PROGRAM 7



The screenshot shows a code editor window titled "setjumpexample.c" with a dark theme. The code is a C program that demonstrates the use of `setjmp` and `longjmp`. The code is as follows:

```
1 #include<stdio.h>
2 #include<setjmp.h>
3 jmp_buf buf;
4
5 int main()
6 {
7     int x=1;
8     int i=1;
9     setjmp(buf);
10
11     printf("%d :5\n",i);
12     i++;
13     x++;
14
15     if(x<=100)
16         longjmp(buf,1);
17
18     return 0;
19 }
```

The code is running in a terminal window titled "Terminal". The terminal output shows the program's execution, which prints the number 5 repeatedly, corresponding to the value of `i` at the point of the `longjmp` call. The output is as follows:

```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o setjumpexample setjumpexample.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./setjumpexample
1 :5
2 :5
3 :5
4 :5
5 :5
6 :5
7 :5
8 :5
9 :5
10 :5
11 :5
12 :5
13 :5
14 :5
15 :5
16 :5
17 :5
18 :5
19 :5
20 :5
21 :5
22 :5
```

The status bar at the bottom of the editor shows "C", "Tab Width: 8", "Ln 8, Col 10", and "INS".

Open



setjumpexample.c

Save



Terminal



```
1 #include<stdio.h>
2 #include<setjmp.h>
3 jmp_buf buf;
4
5 int main()
6 {
7     int x=1;
8     int i=1;
9     setjmp(buf);
10
11     printf("%d :5\n",i);
12     i++;
13     x++;
14
15     if(x<=100)
16         longjmp(buf,1);
17
18     return 0;
19 }
```

```
78 :5
79 :5
80 :5
81 :5
82 :5
83 :5
84 :5
85 :5
86 :5
87 :5
88 :5
89 :5
90 :5
91 :5
92 :5
93 :5
94 :5
95 :5
96 :5
97 :5
98 :5
99 :5
100 :5
vebo-mcs@vebo-ubuntu-mcs:~$
```

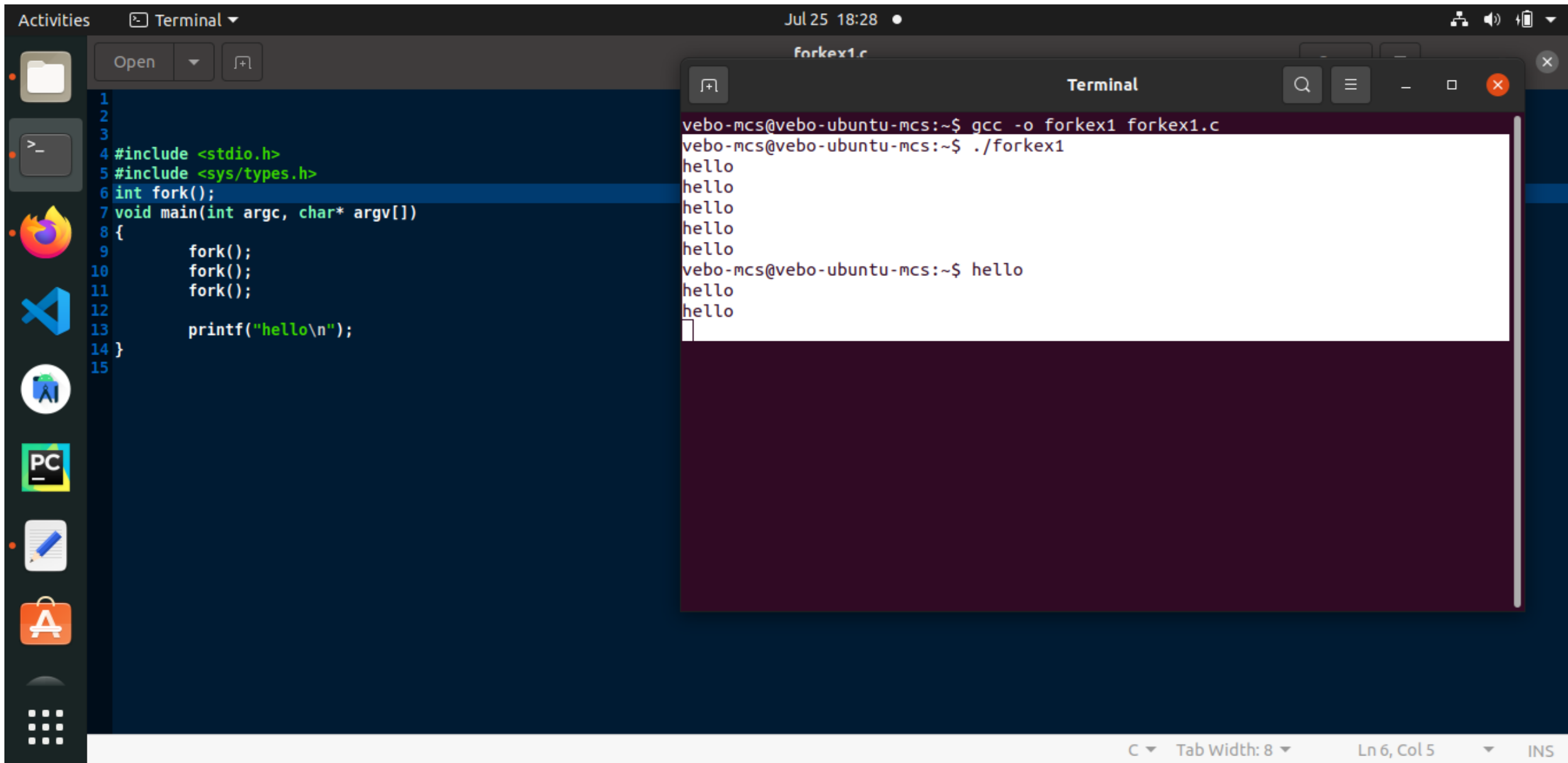
## PROGRAM EXPLANATION 7

`setjmp(buf)` This system call is used to set the jump position using `buf`.

`longjmp()` This system call is used to Jump to the point located by `setjmp`.

The `longjump()` transfers control the pointe which is pointed by `setjump()`.this process will be terminate after the terminating loop.

# PROGRAM 8



The screenshot shows a Linux desktop environment. On the left is a vertical dock with icons for a file manager, terminal, Firefox, VS Code, a folder icon, a PC icon, a notepad icon, an App Store icon, and a grid icon. The main window is a code editor titled 'forkex1.c' with a dark theme. The code is as follows:

```
1
2
3
4 #include <stdio.h>
5 #include <sys/types.h>
6 int fork();
7 void main(int argc, char* argv[])
8 {
9     fork();
10    fork();
11    fork();
12
13    printf("hello\n");
14 }
15
```

Overlaid on the code editor is a terminal window titled 'Terminal'. It shows the following commands and output:

```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o forkex1 forkex1.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./forkex1
hello
hello
hello
hello
hello
vebo-mcs@vebo-ubuntu-mcs:~$ hello
hello
hello
```

The status bar at the bottom of the code editor shows 'C', 'Tab Width: 8', 'Ln 6, Col 5', and 'INS'.

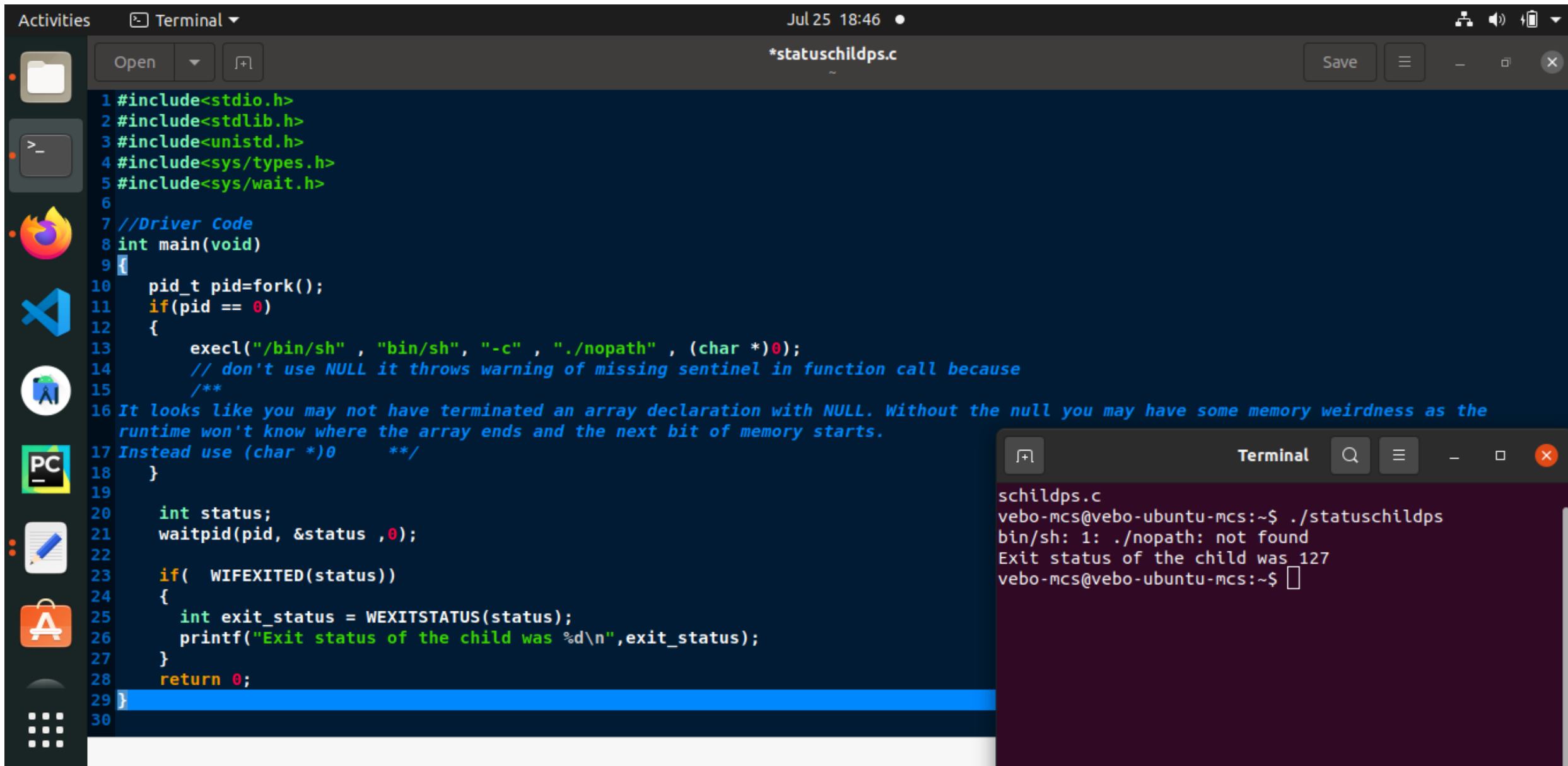


## PROGRAM EXPLANATION 8

The number of times 'hello' is printed is equal to number of process created. Total Number of Processes =  $2^{\text{fork calls}}$ .

So here fork calls = 1,  $2^1 = 2$ . Hence Hello World print 2 times.

# PROGRAM EXPLANATION 9



The screenshot shows a code editor window titled `*statuschildps.c` with a dark theme. The code is a C program that forks a child process to run `bin/sh` with the argument `./nopath`. It includes standard headers like `stdio.h`, `stdlib.h`, `unistd.h`, `sys/types.h`, and `sys/wait.h`. The `main` function calls `fork()` and `waitpid()` to manage the child process. A comment on line 16 explains a warning about a missing sentinel in the function call. The terminal output shows the execution of the program, resulting in an error message and an exit status of 127.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<sys/types.h>
5 #include<sys/wait.h>
6
7 //Driver Code
8 int main(void)
9 {
10     pid_t pid=fork();
11     if(pid == 0)
12     {
13         execl("/bin/sh" , "bin/sh", "-c" , "./nopath" , (char *)0);
14         // don't use NULL it throws warning of missing sentinel in function call because
15         /**
16         It looks like you may not have terminated an array declaration with NULL. Without the null you may have some memory weirdness as the
17         runtime won't know where the array ends and the next bit of memory starts.
18         Instead use (char *)0    **/
19     }
20
21     int status;
22     waitpid(pid, &status ,0);
23
24     if( WIFEXITED(status))
25     {
26         int exit_status = WEXITSTATUS(status);
27         printf("Exit status of the child was %d\n",exit_status);
28     }
29     return 0;
30 }
```

Terminal Output:

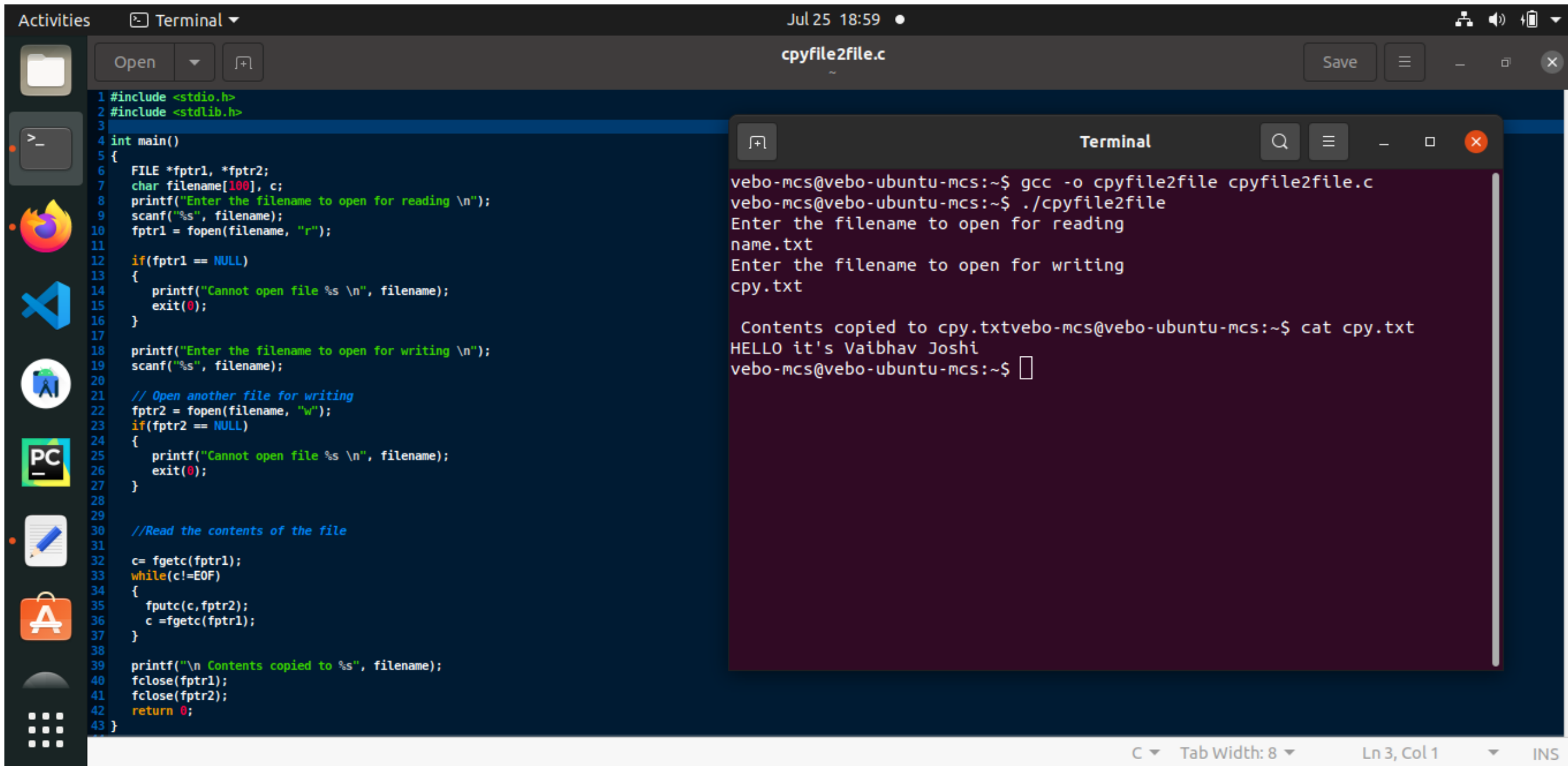
```
schildps.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./statuschildps
bin/sh: 1: ./nopath: not found
Exit status of the child was 127
vebo-mcs@vebo-ubuntu-mcs:~$
```

## PROGRAM EXPLANATION 9

`execl()` system function takes the path of the executable binary file  
`fork()` system call is used to create a new process which becomes child of the caller process.

`WIFEXITED(status)` : returns true if the child terminated normally.  
`waitpid()` system call : It suspends execution of the calling process until a child specified by `pid` argument has changed state.

# PROGRAM 10



The screenshot shows a Linux desktop environment with a sidebar on the left containing icons for Activities, Terminal, Files, and various applications. The main window is a code editor titled 'cpyfile2file.c' with a dark theme. The code is a C program that reads a file and copies its contents to another file. A terminal window is open in the foreground, showing the compilation and execution of the program. The terminal output shows the user entering 'name.txt' for reading and 'cpy.txt' for writing, followed by the contents of 'cpy.txt' being displayed as 'HELLO it's Vaibhav Joshi'.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *fptr1, *fptr2;
7     char filename[100], c;
8     printf("Enter the filename to open for reading \n");
9     scanf("%s", filename);
10    fptr1 = fopen(filename, "r");
11
12    if(fptr1 == NULL)
13    {
14        printf("Cannot open file %s \n", filename);
15        exit(0);
16    }
17
18    printf("Enter the filename to open for writing \n");
19    scanf("%s", filename);
20
21    // Open another file for writing
22    fptr2 = fopen(filename, "w");
23    if(fptr2 == NULL)
24    {
25        printf("Cannot open file %s \n", filename);
26        exit(0);
27    }
28
29    //Read the contents of the file
30
31    c = fgetc(fptr1);
32    while(c != EOF)
33    {
34        fputc(c, fptr2);
35        c = fgetc(fptr1);
36    }
37
38    printf("\n Contents copied to %s", filename);
39    fclose(fptr1);
40    fclose(fptr2);
41    return 0;
42 }
43 }
```

Terminal Output:

```
vebo-mcs@vebo-ubuntu-mcs:~$ gcc -o cpyfile2file cpyfile2file.c
vebo-mcs@vebo-ubuntu-mcs:~$ ./cpyfile2file
Enter the filename to open for reading
name.txt
Enter the filename to open for writing
cpy.txt

Contents copied to cpy.txtvebo-mcs@vebo-ubuntu-mcs:~$ cat cpy.txt
HELLO it's Vaibhav Joshi
vebo-mcs@vebo-ubuntu-mcs:~$
```

Bottom status bar: C Tab Width: 8 Ln 3, Col 1 INS

## PROGRAM EXPLANATION 10

`fopen(filename, "r")` this function is used to open file for reading.

`fopen(filename, "w")` this function is used to open file for writing.

`fgetc(fp1)` read contents from file.

`fputc(c, fp2)` write contents to the file.

`fclose(fp1)` this function is used for close a file.