

Operating Systems

LAB#04



23K-2001

BCS-4J

Q1:

The screenshot displays an Ubuntu 24.04.1 virtual machine interface. On the left, a sidebar shows the file manager with options like Recent, Starred, Home, Documents, Downloads, Music, Pictures, Videos, and Trash. The main window shows the Desktop directory with files 'a.out' and 'Q1_k232001.c'. A terminal window in the background shows the following commands and output:

```
muzammil@muzammil: ~/Desktop/Lab04_23K2001
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gedit Q1_k232001
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q1_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ ./a.out
a.out  Q1_k232001.c

Child process terminated. Exiting program.
muzammil@muzammil:~/Desktop/Lab04_23K2001$
```

In the foreground, a code editor window titled 'Q1_k232001.c' shows the following C code:

```
1 // 23K2001 - Muzammil
2 /*Write a program which uses fork () system-call to create a child process. The child process prints
   the contents of the current
3 directory, and the parent process waits for the child process to terminate.*/
4
5 #include<stdio.h>
6 #include<unistd.h>
7 #include<sys/types.h>
8 #include<sys/wait.h>
9
10 int main(){
11     size_t p = fork();
12     if(p == 0){
13         execlp("/bin/ls", "ls", NULL);
14     }
15     else{
16         wait(NULL);
17         printf("\nChild process terminated. Exiting program.\n");
18     }
19     return 0;
20 }
```

// 23K2001 - Muzammil

/*Write a program which uses fork () system-call to create a child process. The child process prints the contents of the current directory, and the parent process waits for the child process to terminate.*/

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

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
#include<sys/wait.h>
```

```
int main(){
```

```
    size_t p = fork();
```

```
    if(p == 0){
```

```
        execlp("/bin/ls","ls",NULL);
```

```
    }
```

```
    else{
```

```
        wait(NULL);
```

```
        printf("\nChild process terminated. Exiting program.\n");
```

```
    }
```

```
    return 0;
```

```
}
```

Q2:

Ubuntu 24.04.1 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Mar 1 18:06

Files

Home / Desktop / Lab04_23K2001

Recent

Starred

Home

Documents

Downloads

Music

Pictures

Videos

Trash

VirtualBox

Other Locations

Lab04_23K2001

Q1_k232001.a.out

Q1_k232001.c

Q2_k232001.c

Q2_k232001.c

Ln 18, Col 58

```
1 // 23K2001 - Muzammil
2 /*Write a program which prints its PID and uses fork () system call to create a child process. After
   fork () system call, both
3 parent and child processes print what kind of process they are and their PID. Also, the parent process
   prints its child's PID,
4 and the child process prints its parent's PID.*/
5
6 #include<stdio.h>
7 #include<unistd.h>
8 #include<sys/types.h>
9 #include<sys/wait.h>
10
11 int main(){
12     size_t p = fork();
13     if(p == 0){
14         printf("\n(Inside child):\nCurrent Process: p_id = %d\nParent Process: p_id =
           %d\n",getpid(),getppid());
15     }
16     else{
17         wait(NULL);
18         printf("\n(Inside parent):\nChild Process: p_id = %lu\nCurrent Process: p_id = %d\nParent Process:
           p_id = %d\n",p,getpid(),getppid());
19     }
20     return 0;
21 }
```

```
muzammil@muzammil: ~/Desktop/Lab04_23K2001
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gedit Q2_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q2_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ ./a.out

(Inside child):
Current Process: p_id = 4306
Parent Process: p_id = 4305

(Inside parent):
Child Process: p_id = 4306
Current Process: p_id = 4305
Parent Process: p_id = 3358
muzammil@muzammil:~/Desktop/Lab04_23K2001$
```

// 23K2001 - Muzammil

/*Write a program which prints its PID and uses fork () system call to create a child process.

After fork () system call, both

parent and child processes print what kind of process they are and their PID. Also, the parent process prints its child's PID,

and the child process prints its parent's PID.*/

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

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
#include<sys/wait.h>
```

```
int main(){
```

```
    size_t p = fork();
```

```
    if(p == 0){
```

```
        printf("\n(Inside child):\nCurrent Process: p_id = %d\nParent Process: p_id = %d\n",getpid(),getppid());
```

```
    }
```

```
    else{
```

```
        wait(NULL);
```

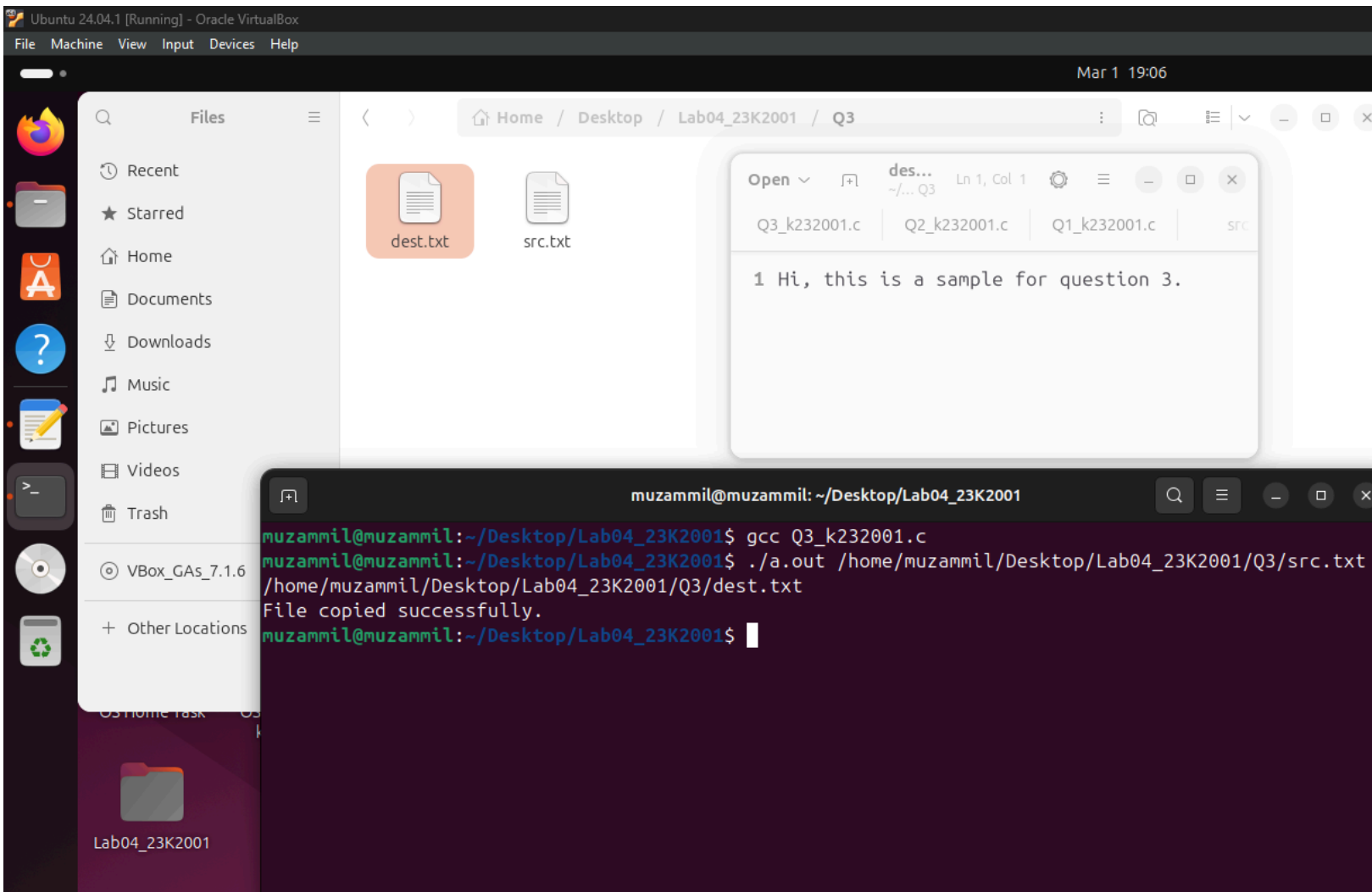
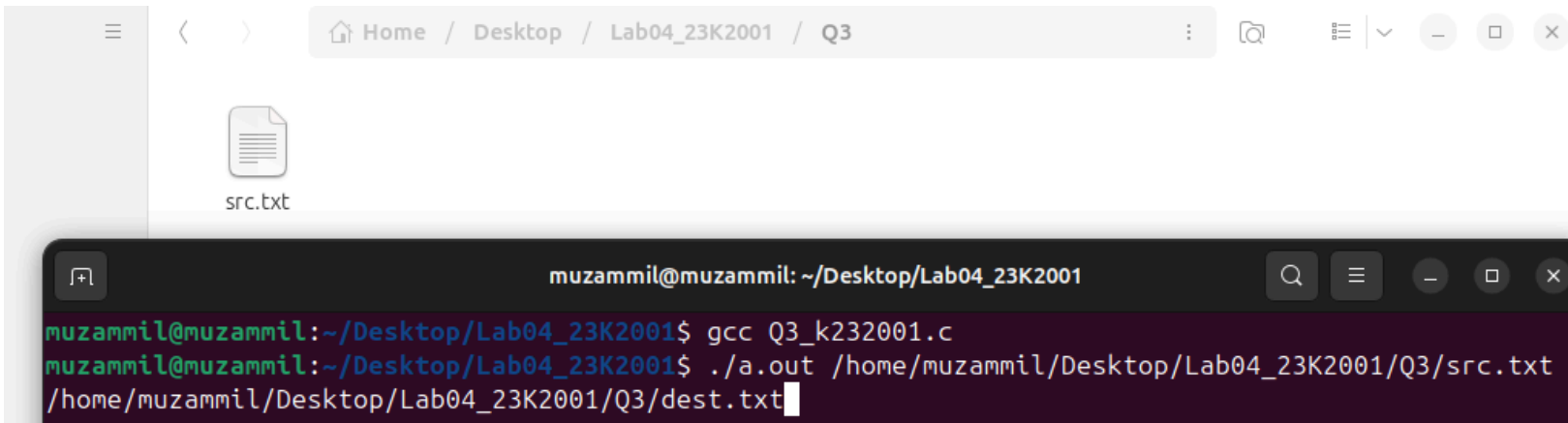
```
        printf("\n(Inside parent):\nChild Process: p_id = %lu\nCurrent Process: p_id = %d\nParent Process: p_id = %d\n",p,getpid(),getppid());
```

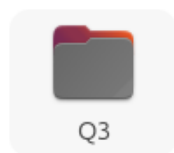
```
    }
```

```
    return 0;
```

```
}
```

Q3:





a.out



Q1_k232001
.c



Q2_k232001
.c



Q3_k232001
.c

// 23K2001 - Muzammil

/*Develop a program that copies the contents of one file into another file using system calls.

The program should accept two file paths as command-line arguments: the source file to be copied

from and the destination file to be copied to. Ensure proper error handling for file opening, reading, and writing operations.*/

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

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

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int main(int argc, char *argv[]) {
```

```
    int f1, f2;
```

```
    ssize_t bytes_read, bytes_written;
```

```
    char buffer[1024];
```

```
    // Open input file (1st argument) for reading
```

```
    f1 = open(argv[1], O_RDONLY);
```

```
    if (f1 == -1) {
```

```
        perror("Failed to open input file");
```

```
        exit(EXIT_FAILURE);
```

```
    }
```

```
    // Open output file (2nd argument) for writing (create if not exists, truncate if exists)
```

```
    f2 = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
```

```
    if (f2 == -1) {
```

```
        perror("Failed to open output file");
```

```
        close(f1);
```

```
        exit(EXIT_FAILURE);
```

```
    }
```

```
    // Read from input file and write to output file
```

```
    while ((bytes_read = read(f1, buffer, sizeof(buffer))) > 0) {
```

```
        bytes_written = write(f2, buffer, bytes_read);
```

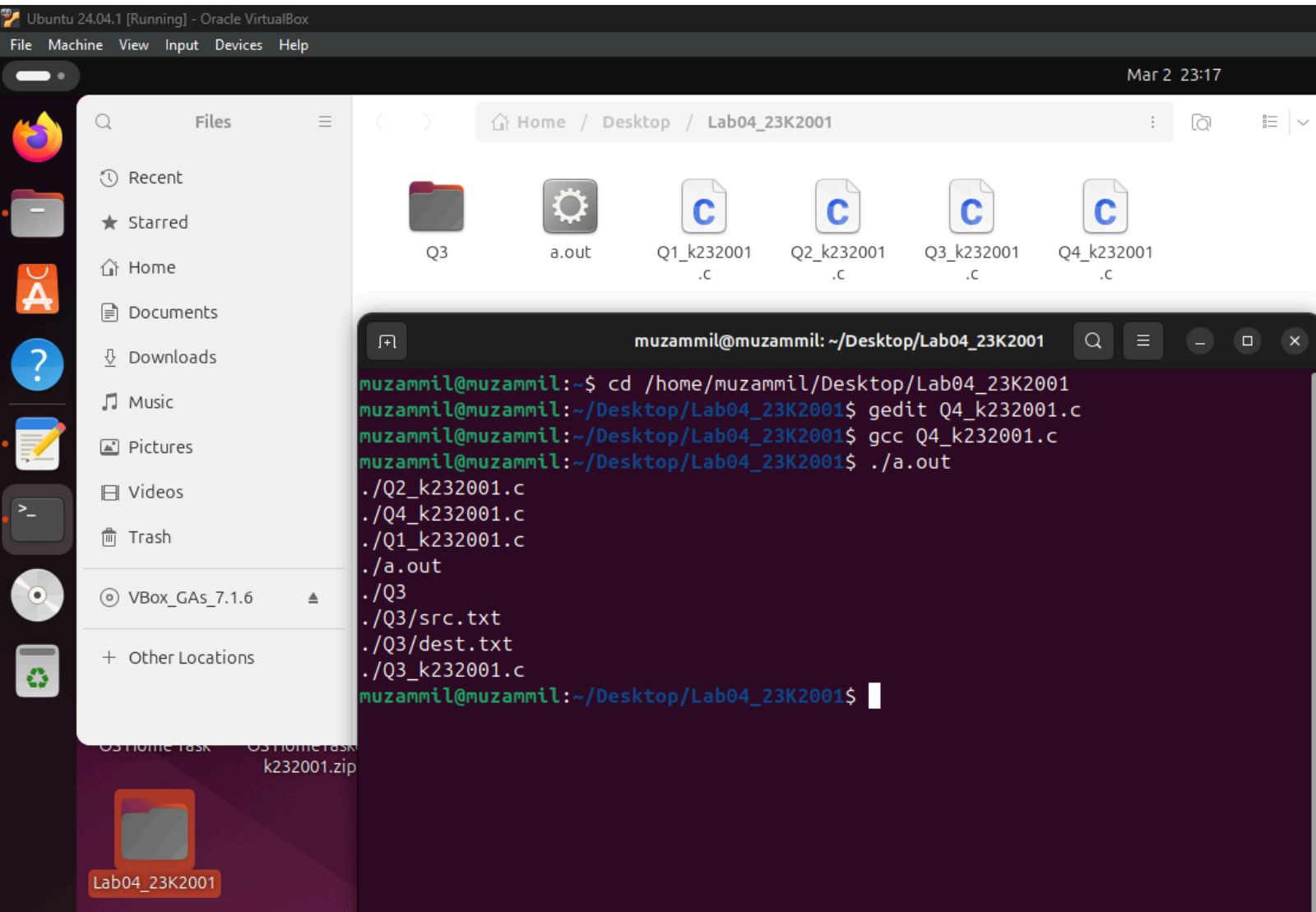
```
        if (bytes_written != bytes_read) {
```

```
perror("Write error");
close(f1);
close(f2);
exit(EXIT_FAILURE);
}
}
if (bytes_read == -1) {
perror("Read error");
close(f1);
close(f2);
exit(EXIT_FAILURE);
}

// Close input and output files
if (close(f1) == -1) {
perror("Failed to close input file");
exit(EXIT_FAILURE);
}
if (close(f2) == -1) {
perror("Failed to close output file");
exit(EXIT_FAILURE);
}

printf("File copied successfully.\n");
return 0;
}
```


Q4:



Open ▾

Q4_k232001.c

~/Desktop/Lab04_23K2001

Ln 31, Col 1



Q4_k232001.c



Q3_k232001.c

Q2_k232001.c

Q1_

```
6 #include <stdio.h>
7 #include <dirent.h>
8 #include <sys/stat.h>
9 #include <string.h>
10
11 void listFiles(const char *path) {
12     DIR *dir = opendir(path);
13     if (!dir) return;
14
15     struct dirent *entry;
16     char newPath[1024];
17
18     while ((entry = readdir(dir))) {
19         if (entry->d_name[0] == '.')
20             continue;
21
22         printf("%s/%s\n", path, entry->d_name);
23         struct stat st;
24         snprintf(newPath, sizeof(newPath), "%s/%s", path, entry->d_name);
25         if (stat(newPath, &st) == 0 && S_ISDIR(st.st_mode))
26             listFiles(newPath);
27     }
28
29     closedir(dir);
30 }
31
32 int main() {
33     listFiles("."); // Current directory
34     return 0;
35 }
```

```
// 23K2001 - Muzammil
```

```
/*Write a program that lists all files and directories in the current directory using system calls.
```

```
The program should traverse the
```

```
directory structure recursively and print the names of all files and directories found, along with
```

```
their respective types (file or
```

```
directory).*/
```

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

```
#include <dirent.h>
```

```
#include <sys/stat.h>
```

```
#include <string.h>
```

```
void listFiles(const char *path) {
```

```
    DIR *dir = opendir(path);
```

```
    if (!dir) return;
```

```
    struct dirent *entry;
```

```
    char newPath[1024];
```

```
    while ((entry = readdir(dir))) {
```

```
        if (entry->d_name[0] == '.')
```

```
            continue;
```

```
        printf("%s/%s\n", path, entry->d_name);
```

```
        struct stat st;
```

```
        snprintf(newPath, sizeof(newPath), "%s/%s", path, entry->d_name);
```

```
        if (stat(newPath, &st) == 0 && S_ISDIR(st.st_mode))
```

```
            listFiles(newPath);
```

```
    }
```

```
    closedir(dir);
```

```
}
```

```
int main() {
```

```
    listFiles("."); // Current directory
```

```
    return 0;
```

```
}
```

Q5:

The screenshot displays a Linux desktop environment with a file manager and a terminal window. The file manager shows a directory structure with files like `src.txt`, `Q5_k232001.c`, `Q4_k232001.c`, `Q3_k232001.c`, and `Q2_k232001.c`. The terminal window shows the execution of several commands to create a directory, edit a file, and compile a program. A warning message is displayed during compilation, and the final output shows the successful backup of the source file.

Home / Desktop / Lab04_23K2001 / Q5

src.txt

Open ▾ [icon] **src.txt** ~/Desktop/Lab04_23K2001/Q5 Ln 1, Col 1 [icon] [icon] [icon] [icon]

src.txt × Q5_k232001.c Q4_k232001.c Q3_k232001.c Q2_k232001.c

1 Hi, this is a sample for question 05.

Ubuntu 24.04.1 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Files [icon] [icon] Home / Desktop / Lab04_23K2001 / Q5

Recent

Starred

Home

dest.txt src.txt

muzammil@muzammil: ~/Desktop/Lab04_23K2001 [icon] [icon] [icon] [icon]

```
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gedit Q5_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ mkdir Q5
muzammil@muzammil:~/Desktop/Lab04_23K2001$ cd Q5
muzammil@muzammil:~/Desktop/Lab04_23K2001/Q5$ gedit src.txt
muzammil@muzammil:~/Desktop/Lab04_23K2001/Q5$ cd ..
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q5_k232001.c
Q5_k232001.c: In function 'main':
Q5_k232001.c:48:9: warning: implicit declaration of function 'wait' [-Wimplicit-
function-declaration]
  48 |         wait(NULL);
     |         ^~~~
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q5_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ ./a.out

Input source file path: /home/muzammil/Desktop/Lab04_23K2001/Q5/src.txt
Input destination file path: /home/muzammil/Desktop/Lab04_23K2001/Q5/dest.txt

Backup successful!
muzammil@muzammil:~/Desktop/Lab04_23K2001$
```



```
// 23K2001 - Muzammil
```

```
/*Write a C program to create a backup of a file. The program takes two filenames as input  
(source and destination), where the parent process opens both files, forks a  
child process to read from the source file and write to the destination file,  
and ensures proper error handling if the file does not exist.*/
```

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

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

```
#include <sys/wait.h>
```

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
int main(){
```

```
    char src[100], dest[100];
```

```
    printf("\nInput source file path: ");
```

```
    scanf("%s", src);
```

```
    printf("Input destination file path: ");
```

```
    scanf("%s", dest);
```

```
    int f1 = open(src, O_RDONLY);
```

```
    if (f1 == -1){
```

```
        perror("\nError opening source file");
```

```
        return 1;
```

```
    }
```

```
    int f2 = open(dest, O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

```
    if (f2 == -1){
```

```
        perror("\nError creating destination file");
```

```
        close(f1);
```

```
        return 1;
```

```
    }
```

```
    pid_t pid = fork();
```

```
    if (pid == -1){
```

```
perror("\nFork failed");
close(f1);
close(f2);
return 1;
}

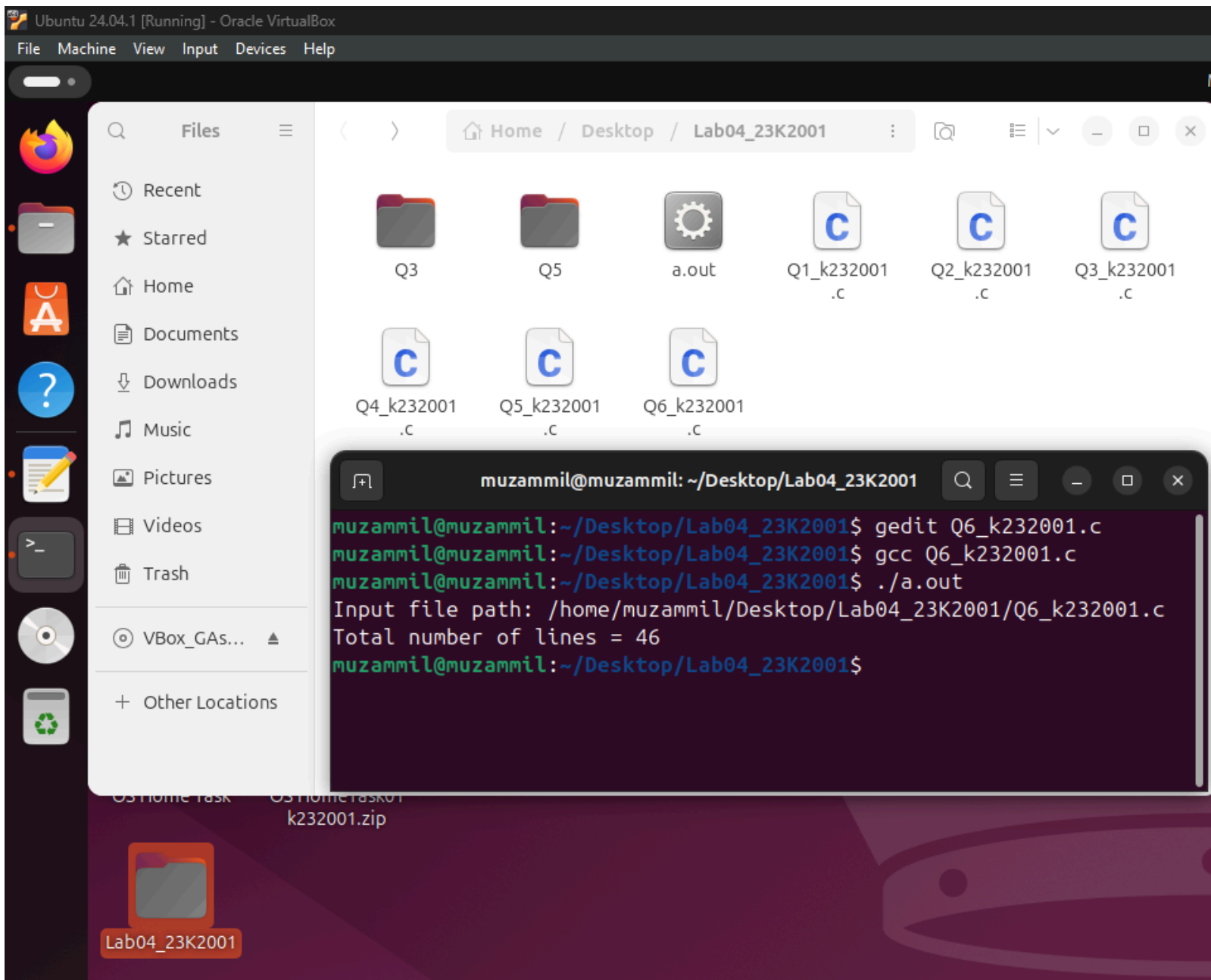
if (pid == 0){
char buffer[1024];
ssize_t bytesRead;

while ((bytesRead = read(f1, buffer, sizeof(buffer))) > 0)
write(f2, buffer, bytesRead);

close(f1);
close(f2);
printf("\nBackup successful!\n");
}
else
wait(NULL);

return 0;
}
```

Q6:



```
5 after the child process finishes execution.*/
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <unistd.h>
9 #include <fcntl.h>
10 #include <sys/wait.h>
11
12 int main(){
13     char path[100];
14     printf("Input file path: ");
15     scanf("%s", path);
16
17     int fd = open(path, O_RDONLY);
18     if (fd == -1){
19         perror("Error opening file");
20         return 1;
21     }
22     pid_t pid = fork();
23     if (pid == -1){
24         perror("Fork failed");
25         close(fd);
26         return 1;
27     }
28
29     if (pid == 0){
30         char ch;
31         int lines = 0;
32
33         while (read(fd, &ch, 1) > 0){
34             if (ch == '\n')
35                 lines++;
36         }
37         close(fd);
38         printf("Total number of lines = %d\n", lines);
39         exit(0);
40     }
41     else
42         wait(NULL);
43
44     return 0;
45 }
46
```


/*Write a C program to lines the number of lines in a file. The parent process opens the file, forks a child process that reads character by character, lines the newline (\n) characters, and prints the total line lines after the child process finishes execution.*/

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

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

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/wait.h>
```

```
int main(){
```

```
    char path[100];
```

```
    printf("Input file path: ");
```

```
    scanf("%s", path);
```

```
    int fd = open(path, O_RDONLY);
```

```
    if (fd == -1){
```

```
        perror("Error opening file");
```

```
        return 1;
```

```
    }
```

```
    pid_t pid = fork();
```

```
    if (pid == -1){
```

```
        perror("Fork failed");
```

```
        close(fd);
```

```
        return 1;
```

```
    }
```

```
    if (pid == 0){
```

```
        char ch;
```

```
        int lines = 0;
```

```
        while (read(fd, &ch, 1) > 0){
```

```
            if (ch == '\n')
```

```
                lines++;
```

```
        }
```

```
        close(fd);
```

```
        printf("Total number of lines = %d\n", lines);
```

```
        exit(0);
```

```
    }
```

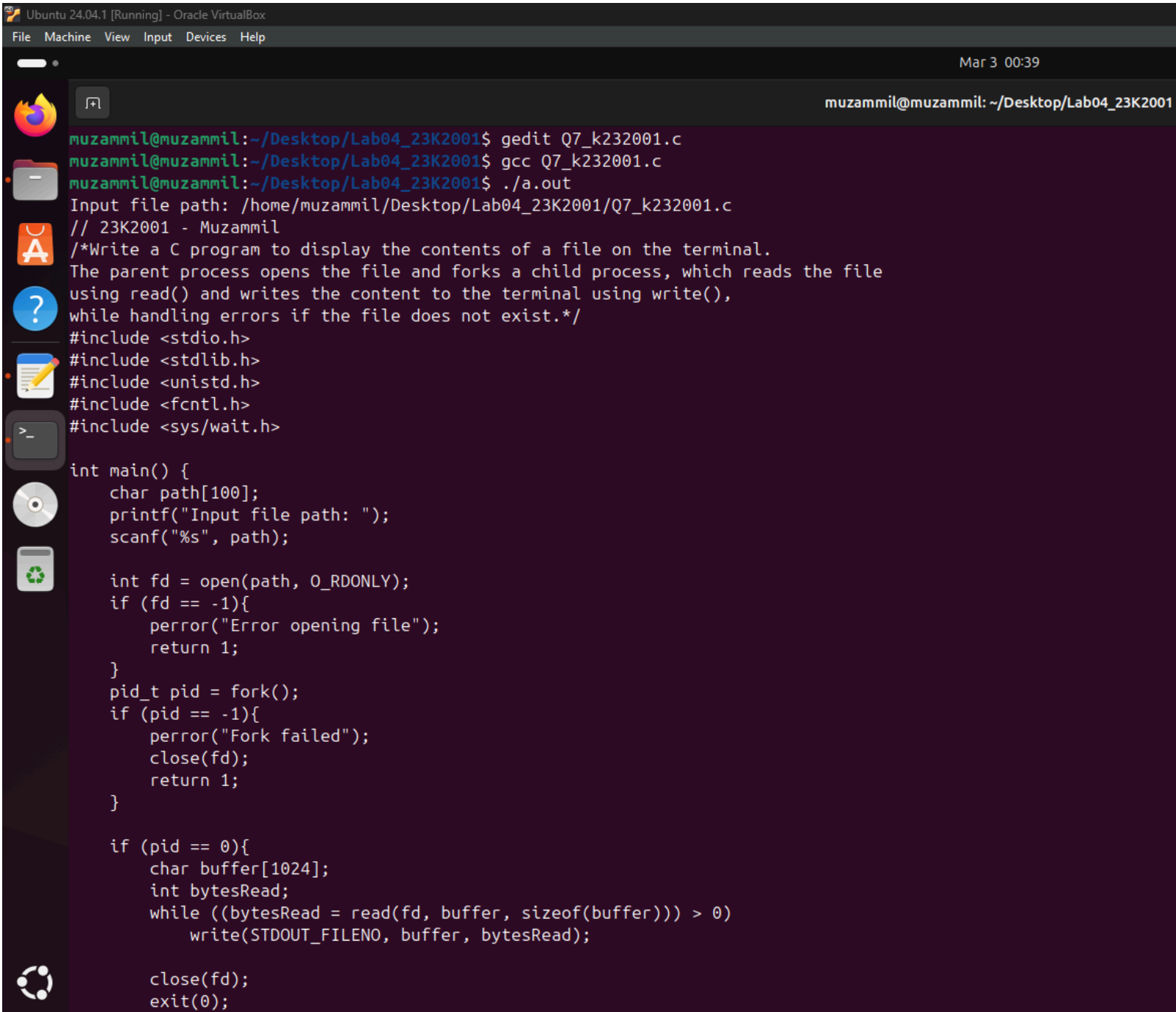
```
    else
```

```
        wait(NULL);
```

```
    return 0;
```

```
}
```

Q7:



```
 muzammil@muzammil: ~/Desktop/Lab04_23K2001
 muzammil@muzammil:~/Desktop/Lab04_23K2001$ gedit Q7_k232001.c
 muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q7_k232001.c
 muzammil@muzammil:~/Desktop/Lab04_23K2001$ ./a.out
 Input file path: /home/muzammil/Desktop/Lab04_23K2001/Q7_k232001.c
 // 23K2001 - Muzammil
 /*Write a C program to display the contents of a file on the terminal.
 The parent process opens the file and forks a child process, which reads the file
 using read() and writes the content to the terminal using write(),
 while handling errors if the file does not exist.*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main() {
    char path[100];
    printf("Input file path: ");
    scanf("%s", path);

    int fd = open(path, O_RDONLY);
    if (fd == -1){
        perror("Error opening file");
        return 1;
    }
    pid_t pid = fork();
    if (pid == -1){
        perror("Fork failed");
        close(fd);
        return 1;
    }

    if (pid == 0){
        char buffer[1024];
        int bytesRead;
        while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0)
            write(STDOUT_FILENO, buffer, bytesRead);

        close(fd);
        exit(0);
    }
}
```

Open ▾

Q7_k232001.c
~/Desktop/Lab04_23K2001

Q7_k232001.c

×

Q6_k232001.c

Q5_k232001.c

```
12 int main() {
13     char path[100];
14     printf("Input file path: ");
15     scanf("%s", path);
16
17     int fd = open(path, O_RDONLY);
18     if (fd == -1){
19         perror("Error opening file");
20         return 1;
21     }
22     pid_t pid = fork();
23     if (pid == -1){
24         perror("Fork failed");
25         close(fd);
26         return 1;
27     }
28
29     if (pid == 0){
30         char buffer[1024];
31         int bytesRead;
32         while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0)
33             write(STDOUT_FILENO, buffer, bytesRead);
34
35         close(fd);
36         exit(0);
37     }
38     else
39         wait(NULL);
40
41     return 0;
42 }
```

// 23K2001 - Muzammil

/*Write a C program to display the contents of a file on the terminal.

The parent process opens the file and forks a child process, which reads the file using read() and writes the content to the terminal using write(), while handling errors if the file does not exist.*/

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

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

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/wait.h>
```

```
int main() {
```

```
    char path[100];
```

```
    printf("Input file path: ");
```

```
    scanf("%s", path);
```

```
    int fd = open(path, O_RDONLY);
```

```
    if (fd == -1){
```

```
        perror("Error opening file");
```

```
        return 1;
```

```
    }
```

```
    pid_t pid = fork();
```

```
    if (pid == -1){
```

```
        perror("Fork failed");
```

```
        close(fd);
```

```
        return 1;
```

```
    }
```

```
    if (pid == 0){
```

```
        char buffer[1024];
```

```
        int bytesRead;
```

```
        while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0)
```

```
            write(STDOUT_FILENO, buffer, bytesRead);
```

```
        close(fd);
```

```
        exit(0);
```

```
    }
```

```
    else
```

```
        wait(NULL);
```

```
    return 0;
```

```
}
```

Q8:

Ubuntu 24.04.1 [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Mar 3 00:57

Files

Recent

Starred

Home

Documents

Downloads

Music

Pictures

Videos

Trash

VBox_GAs_7.1

Other Locations

Home / Desktop / Lab04_23K2001

Q3

Q5

a.out

Q1_k232001.c

Q2_k232001.c

Q3_k232001.c

Q4_k232001.c

Q5_k232001.c

Q6_k232001.c

Q7_k232001.c

Q8_k232001.c

Q8Output.txt

muzammil@muzammil: ~/Desktop/Lab04_23K2001

```
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gedit Q8_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ gcc Q8_k232001.c
muzammil@muzammil:~/Desktop/Lab04_23K2001$ ./a.out
Input the file to store the output: /home/muzammil/Desktop/Lab04_23K2001/Q8Output.txt
Output successfully stored in /home/muzammil/Desktop/Lab04_23K2001/Q8Output.txt
muzammil@muzammil:~/Desktop/Lab04_23K2001$
```

Home / Desktop / Lab04_23K2001

Q3

Q5

a.out

Q1_k232001.c

Q2_k232001.c

Q3_k232001.c

Q4_k232001.c

Q5_k232001.c

Q6_k232001.c

Q7_k232001.c

Q8_k232001.c

Q8Output.txt

Open

Q8Output.txt x

Q8_k232001.c

Q7_k232001.c

Q6_k232001.c

Q5_k232001.c

Ln 1, Col 1

```
1 muzammil seat0 2025-03-02 23:11 (login screen)
2 muzammil tty2 2025-03-02 23:11 (tty2)
```

"Q8Output.txt" selected (100 bytes)



Q8_k232001.c



Q7_k232001.c

Q6_k232001.c

```
9  #include <fcntl.h>
10 #include <sys/wait.h>
11
12 int main(){
13     char file[100];
14     printf("Input the file to store the output: ");
15     scanf("%s", file);
16     pid_t pid = fork();
17     if (pid == -1){
18         perror("Fork failed");
19         return 1;
20     }
21
22     if (pid == 0){
23         int fd = open(file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
24         if (fd == -1){
25             perror("Error opening file");
26             exit(1);
27         }
28         dup2(fd, STDOUT_FILENO);
29         close(fd);
30
31         execlp("who", "who", NULL);
32         perror("execlp failed");
33         exit(1);
34     }
35     else{
36         wait(NULL);
37         printf("Output successfully stored in %s\n", file);
38     }
39     return 0;
40 }
```

// 23K2001 - Muzammil

/*Write a C program creates a child process using fork(). The child process redirects its output to a file and executes the who command using execlp(). The parent process waits for the child to finish and informs the user that the output has been successfully stored.*/

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

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

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/wait.h>
```

```
int main(){
```

```
    char file[100];
```

```
    printf("Input the file to store the output: ");
```

```
    scanf("%s", file);
```

```
    pid_t pid = fork();
```

```
    if (pid == -1){
```

```
        perror("Fork failed");
```

```
        return 1;
```

```
    }
```

```
    if (pid == 0){
```

```
        int fd = open(file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

```
        if (fd == -1){
```

```
            perror("Error opening file");
```

```
            exit(1);
```

```
        }
```

```
        dup2(fd, STDOUT_FILENO);
```

```
        close(fd);
```

```
        execlp("who", "who", NULL);
```

```
        perror("execlp failed");
```

```
        exit(1);
```

```
    }
```

```
    else{
```

```
        wait(NULL);
```

```
        printf("Output successfully stored in %s\n", file);
```

```
    }
```

```
    return 0;
```

```
}
```