Problem Statement – Implementation of Unix commands using C language

a) **ls -** ls [option] [file/directory]

‘ls’ will display the contents of the current directory.

|  |  |
| --- | --- |
| Options | Description |
| -l | known as a long format that displays detailed information about files and directories. |
| -a | Represent all files Include hidden files and directories in the listing. |
| -R | List files and directories recursively, including subdirectories. |
| ls <dirname> | List files of directory named <dirname> |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <pwd.h>

#include <grp.h>

#include <time.h>

void list\_directory(const char \*path, int show\_hidden, int long\_format, int recursive) {

DIR \*dir;

struct dirent \*entry;

struct stat file\_info;

if ((dir = opendir(path)) == NULL) {

perror("opendir");

exit(1);

}

if (!recursive)

printf("Directory: %s\n", path);

while ((entry = readdir(dir)) != NULL) {

if (!show\_hidden && entry->d\_name[0] == '.')

continue;

char full\_path[1024];

snprintf(full\_path, sizeof(full\_path), "%s/%s", path, entry->d\_name);

if (lstat(full\_path, &file\_info) == -1) {

perror("lstat");

exit(1);

}

if (long\_format) {

struct passwd \*pw = getpwuid(file\_info.st\_uid);

struct group \*gr = getgrgid(file\_info.st\_gid);

struct tm \*tm\_info = localtime(&file\_info.st\_mtime);

printf("%s %6ld %s %s %5ld %4d-%02d-%02d %02d:%02d %s\n",

(S\_ISDIR(file\_info.st\_mode) ? "d" : "-"),

(long)file\_info.st\_size,

(pw ? pw->pw\_name : "?"),

(gr ? gr->gr\_name : "?"),

(long)file\_info.st\_size,

1900 + tm\_info->tm\_year,

tm\_info->tm\_mon + 1,

tm\_info->tm\_mday,

tm\_info->tm\_hour,

tm\_info->tm\_min,

entry->d\_name);

}

else {

printf("%s ", entry->d\_name);

}

if (recursive && S\_ISDIR(file\_info.st\_mode) && strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, "..") != 0) {

printf("\n\n%s:\n",entry->d\_name);

list\_directory(full\_path, show\_hidden, long\_format, recursive);

}

}

closedir(dir);

}

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

int show\_hidden = 0;

int long\_format = 0;

int recursive = 0;

char \*directory = ".";

for (int i = 1; i < argc; i++) {

if (strcmp(argv[i], "-a") == 0) {

show\_hidden = 1;

} else if (strcmp(argv[i], "-l") == 0) {

long\_format = 1;

} else if (strcmp(argv[i], "-R") == 0) {

recursive = 1;

} else if (argv[i][0] != '-') {

// If not an option, assume it's a directory

directory = argv[i];

} else {

fprintf(stderr, "Unknown option: %s\n", argv[i]);

exit(1);

}

}

list\_directory(directory, show\_hidden, long\_format, recursive);

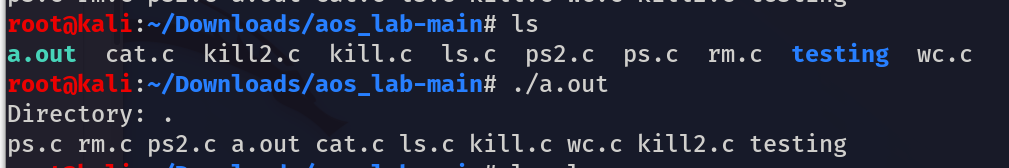
printf("\n");

return 0;

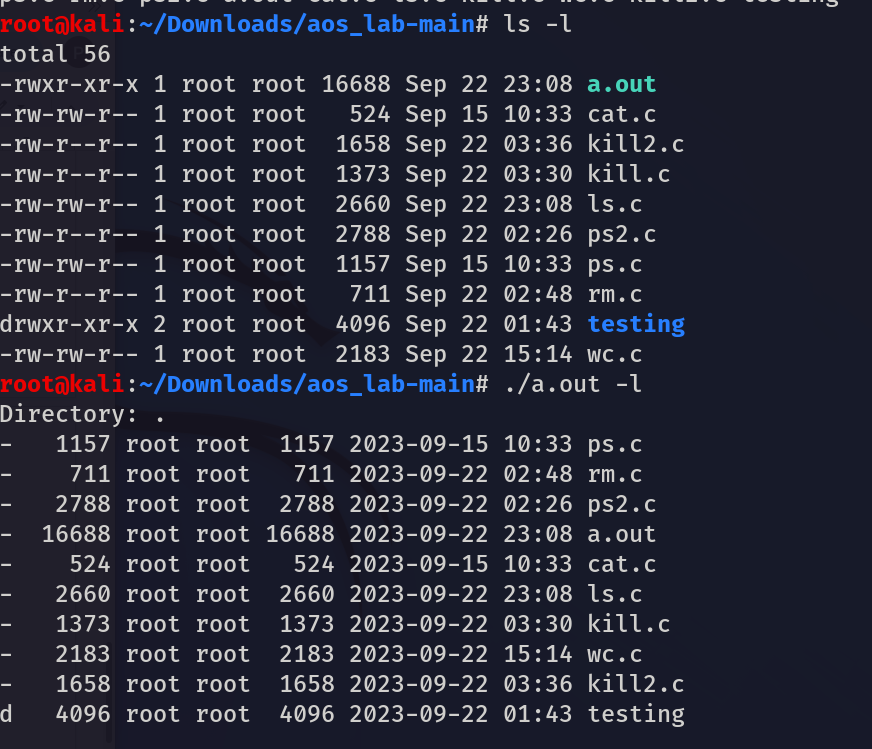
}

**Output:**

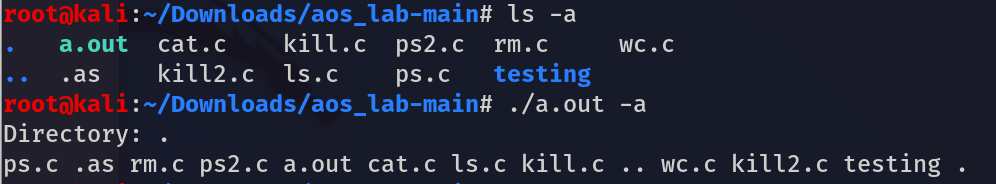
**ls:**

****

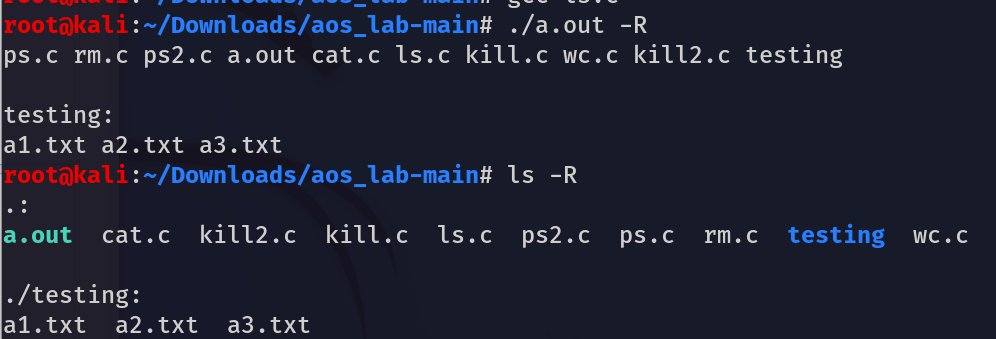
**ls -l:**

****

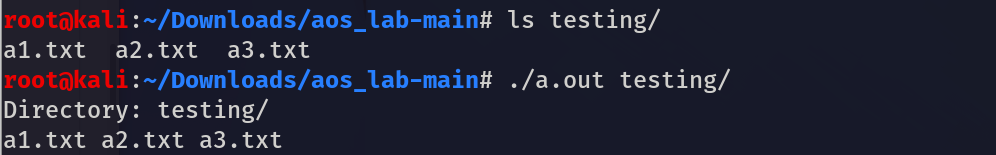
**ls -a:**

****

**ls -R:**

****

**ls <dirname>:**

****

b) **rm - rm [OPTION]... FILE...**

The 'rm' means remove. This command is used to remove a file.

|  |  |
| --- | --- |
| Options | Description |
| -rm <filename> | Remove the file named “filename” |
| rm -i <filename> | Remove the file named – “filename” interactively |
| rm <file1> <file2> ... | Remove multiple files name – “file1” , “file2” , and so on |

**Code:**

#include <stdio.h>

#include <stdlib.h>

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

if (argc < 2) {

fprintf(stderr, "Usage: %s <file1> [<file2> ...]\n", argv[0]);

exit(EXIT\_FAILURE);

}

int c;

if (argv[1][0] == '-') {

printf("Do you want to remove file? (1=yes/0=no): ");

scanf("%d", &c);

if (c == 1) {

remove(argv[2]);

} else {

printf("File not removed\n");

}

} else {

for (int i = 1; i < argc; i++) {

if (remove(argv[i]) == 0) {

printf("Removed file: %s\n", argv[i]);

} else {

perror("Error deleting file");

}

}

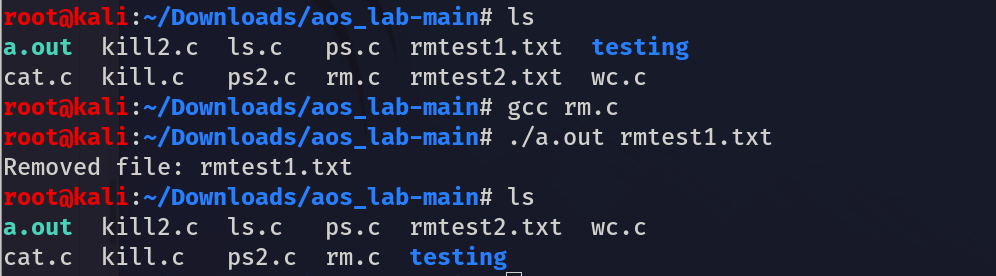
}

return 0;

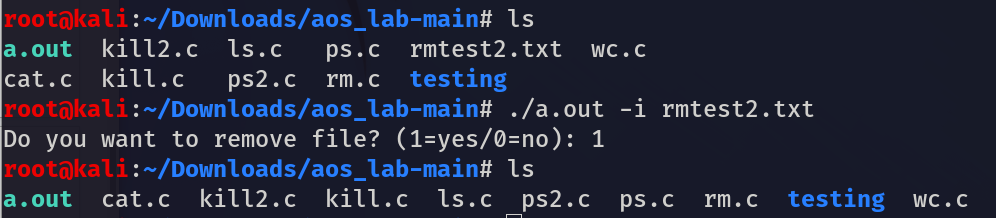
}

**Output:**

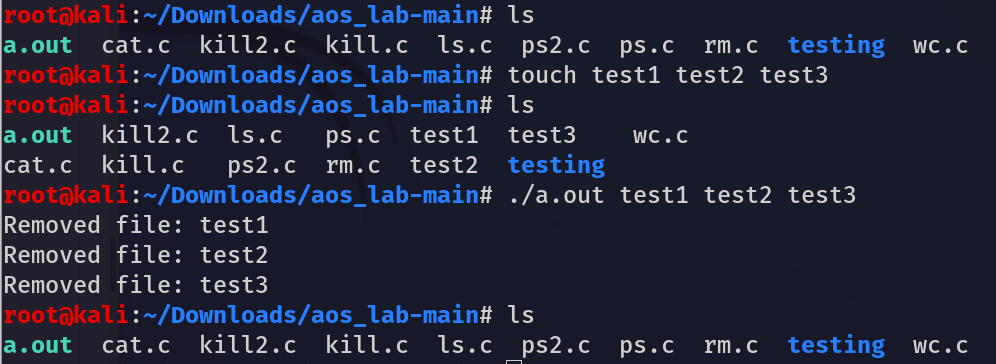
**rm <filename>:**

****

**rm -i <filename>:**

****

**rm <file1> <file2> …:**

****

**c) cat** - cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives its content as output. It helps us to create, view, and concatenate files.

|  |  |
| --- | --- |
| Option | Description |
| -cat <file> | To view a single file named – “file” |
| cat <file1> <file2> ... | To view multiple files named – “file1”, “file2”, and so on. |

**Code:**

#include <stdio.h>

#include <stdlib.h>

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

if (argc < 2) {

fprintf(stderr, "Usage: %s <file1> [<file2> ...]\n", argv[0]);

exit(EXIT\_FAILURE);

}

for (int i = 1; i < argc; i++) {

FILE \*file = fopen(argv[i], "r");

if (file == NULL) {

perror("Error opening file");

continue;

}

int ch;

while ((ch = fgetc(file)) != EOF) {

putchar(ch);

}

fclose(file);

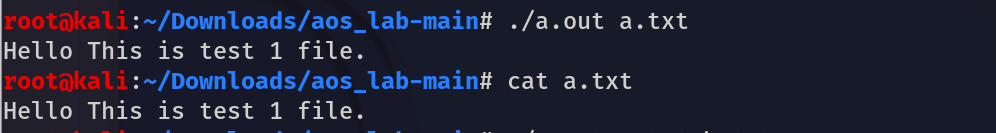
}

return 0;

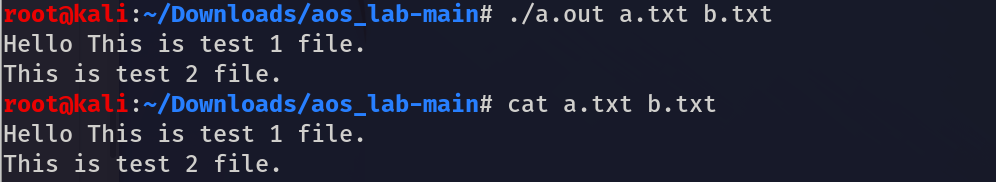
}

**Output:**

**cat <file>:**

****

**cat <file1> <file2> …:**

****

**d) kill** :- *kill* command in Linux is a built-in command which is used to terminate processes manually. *kill* command sends a signal to a process that terminates the process. If the user doesn’t specify any signal which is to be sent along with the kill command, then a default *TERM*signal is sent that terminates the process.

|  |  |
| --- | --- |
| **Option** | **Description** |
| -kill pid | This option kills the process of the process id – “pid” |
| kill -<signo> <pid>, | This option specifies the signal (“signo”) to be sent to the process with process id – “pid” |
| kill <pid1> <pid2> ..., | This option kills multiple processes with process id – “pid1”, “pid2” and so on |
| kill -<signo> <pid1> <pid2> ... | This option specifies the signal (“signo”) to be sent to multiple processes with process id – “pid1” , “pid2”, and so on. |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

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

if (argc < 2) {

fprintf(stderr, "Usage:\n");

fprintf(stderr, " %s <pid> (kill <pid>)\n", argv[0]);

fprintf(stderr, " %s -<signo> <pid> (kill -<signo> <pid>)\n", argv[0]);

fprintf(stderr, " %s <pid1> <pid2> ... (kill <pid1> <pid2> ...)\n", argv[0]);

fprintf(stderr, " %s -<signo> <pid1> <pid2> ... (kill -<signo> <pid1> <pid2> ...)\n", argv[0]);

return 1;

}

// Determine the signal number and PIDs based on command-line arguments

int signo = SIGTERM; // Default signal is SIGTERM

int arg\_start = 1; // Index of the first argument

if (argv[1] == '-k') {

// If the first argument starts with a '-', it's a signal option

printf(argv[2]);

kill(argv[2],SIGSEGV);

signo = atoi(argv[1] + 1); // Extract the signal number

arg\_start = 2; // Start from the next argument

}

else{

// Iterate through the PIDs and send the signal to each process

for (int i = arg\_start; i < argc; i++) {

printf("in for lop");

int pid = atoi(argv[i]);

if (kill(pid, signo) == 0) {

printf("Sent signal %d to process %d\n", signo, pid);

} else {

perror("Error sending signal");

}

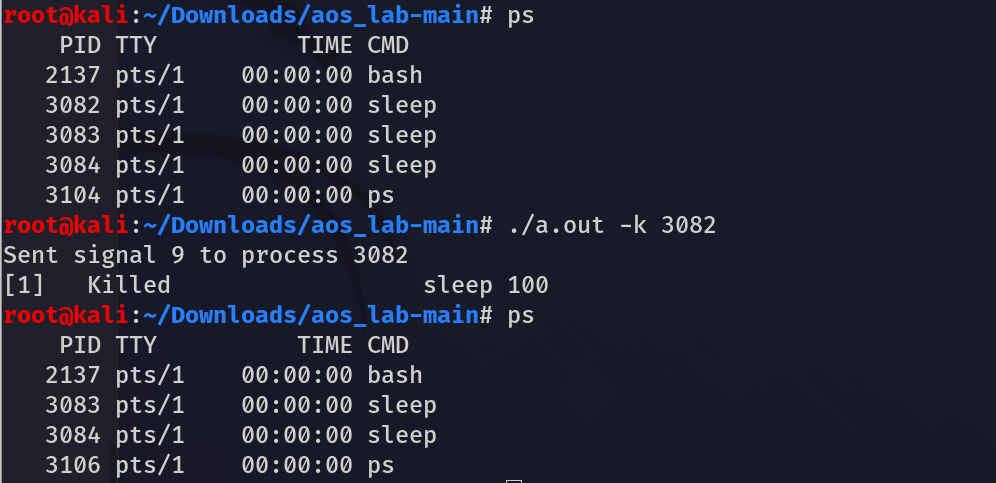
}}

return 0;

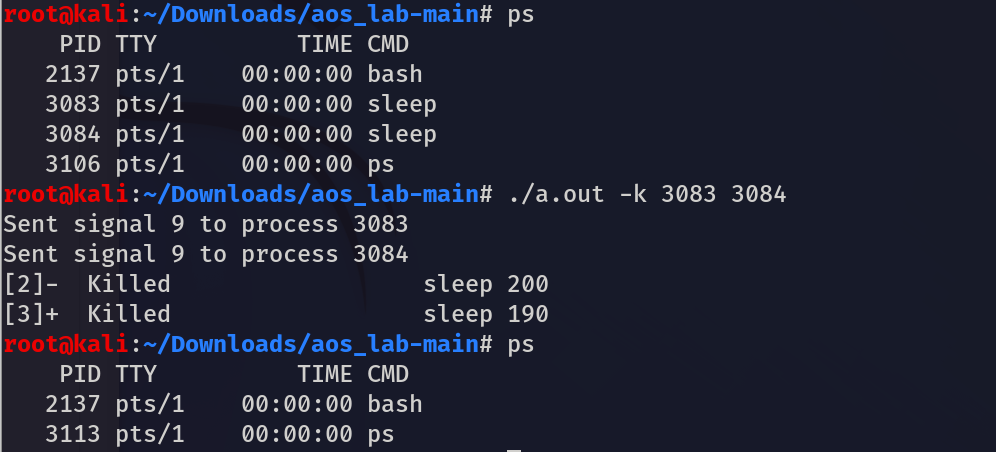
}

**Output:**

**-k pid:**

****

**-k <pid1> <pid2>:**

****

e) **ps** :- ps command is used to list the currently running processes and their PIDs along with some other information depends on different options.

|  |  |
| --- | --- |
| **Option** | **Description** |
| -ps, | Shows the processes for the current shell |
| ps -a, | View all processes except both session leaders and processes not associated with a terminal. |
| ps -ae, | The **ps -ae** command is commonly used to display a list of all processes running on the system, including those associated with other users. |
| ps -u <username> | This command will display a list of processes that are owned by the specified user. |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void execute\_ps(const char \*options) {

char command[256];

snprintf(command, sizeof(command), "ps %s", options);

FILE \*fp = popen(command, "r");

if (fp == NULL) {

perror("Error executing ps command");

exit(EXIT\_FAILURE);

}

char buffer[1024];

while (fgets(buffer, sizeof(buffer), fp) != NULL) {

printf("%s", buffer);

}

pclose(fp);

}

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

if (argc < 2) {

fprintf(stderr, "Usage: %s [-a] [-ae] [-u <username>]\n", argv[0]);

exit(EXIT\_FAILURE);

}

if (strcmp(argv[1], "-a") == 0) {

execute\_ps("a");

} else if (strcmp(argv[1], "-ae") == 0) {

execute\_ps("ae");

} else if (strcmp(argv[1], "-u") == 0 && argc >= 3) {

if (strlen(argv[2]) > 0) {

char options[256];

snprintf(options, sizeof(options), "u %s", argv[2]);

execute\_ps(options);

} else {

fprintf(stderr, "Invalid username argument\n");

exit(EXIT\_FAILURE);

}

} else {

execute\_ps("");

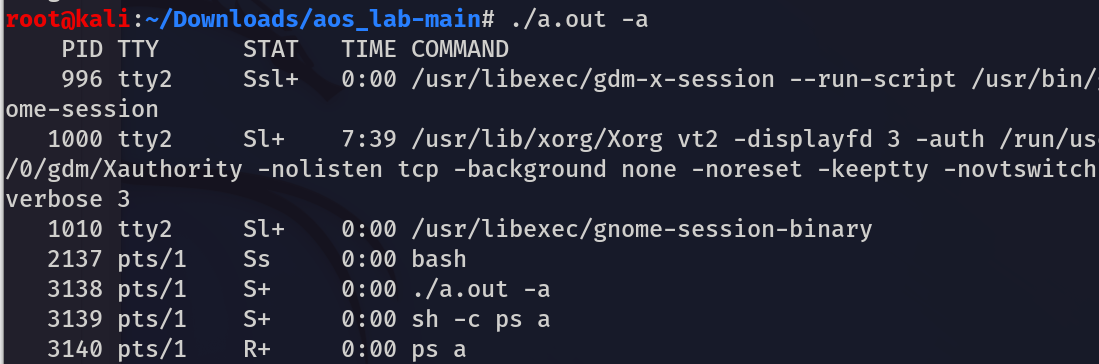
}

return 0;

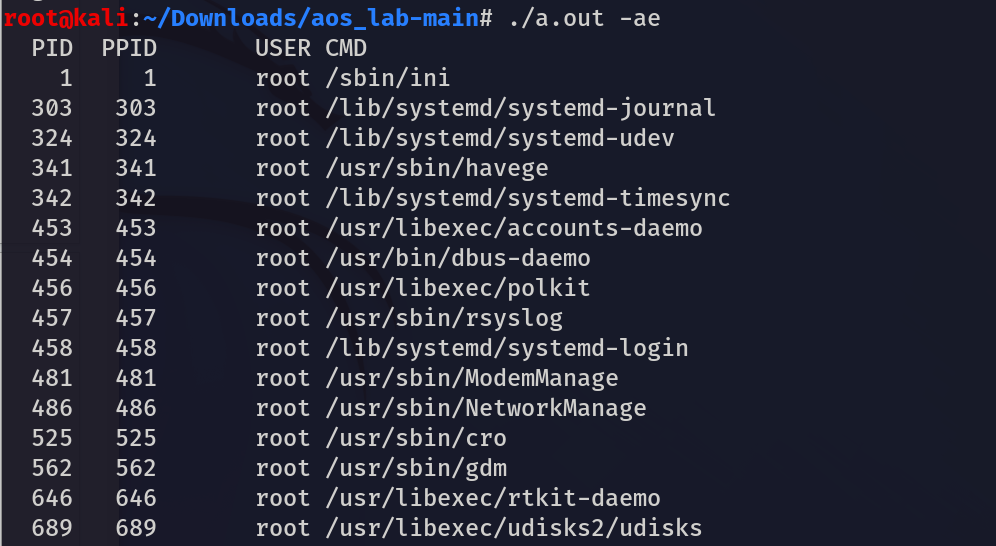
}

**Output:**

**ps -a:**

****

**ps -ae:**

****

f) **wc** :- it is used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments.

**wc [OPTION]... [FILE]...**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -wc <file1> <file2> ..., | This option prints the **number of charcaters, wrods and lines** present in a files named – “file1” , “file2” and so on |
| wc -c <file1> <file2> ..., | This option prints the **number of charcaters** present in a files named – “file1” , “file2” and so on |
| wc -l <file1> <file2> ..., | This option prints the **number of lines** present in a files named – “file1” , “file2” and so on |
| wc -w <file1> <file2> ... | This option prints the **number of words** present in a files named – “file1” , “file2” and so on. |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int total\_char=0;

int total\_words=0;

int total\_lines=0;

void count\_wc(const char \*filename, int count\_chars, int count\_lines, int count\_words) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

perror("Error opening file");

return;

}

int char\_count = 0;

int line\_count = 0;

int word\_count = 0;

int in\_word = 0;

int ch;

while ((ch = fgetc(file)) != EOF) {

char\_count++;

if (count\_lines && ch == '\n') {

line\_count++;

}

if (count\_words) {

if (ch == ' ' || ch == '\t' || ch == '\n') {

if (in\_word) {

word\_count++;

in\_word = 0;

}

} else {

in\_word = 1;

}

}

}

if (count\_words && in\_word) {

word\_count++;

}

fclose(file);

if (count\_chars){

total\_char+=char\_count;

printf("Characters in %s: %d\n", filename, char\_count);

}

if (count\_lines) {

total\_lines+=count\_lines;

printf("Lines in %s: %d\n", filename, line\_count);

}

if (count\_words) {

total\_words+=count\_words;

printf("Words in %s: %d\n", filename, word\_count);

}

}

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

int count\_chars = 0;

int count\_lines = 0;

int count\_words = 0;

if (argc < 2) {

fprintf(stderr, "Usage: %s [-c] [-l] [-w] <file1> [<file2> ...]\n", argv[0]);

exit(EXIT\_FAILURE);

}

for (int i = 1; i < argc; i++) {

if (strcmp(argv[i], "-c") == 0) {

count\_chars = 1;

} else if (strcmp(argv[i], "-l") == 0) {

count\_lines = 1;

} else if (strcmp(argv[i], "-w") == 0) {

count\_words = 1;

} else {

count\_wc(argv[i],1,1,1);

}

}

printf("\ntotal char :- %d\n",total\_char);

printf("total lines :- %d\n",total\_lines);

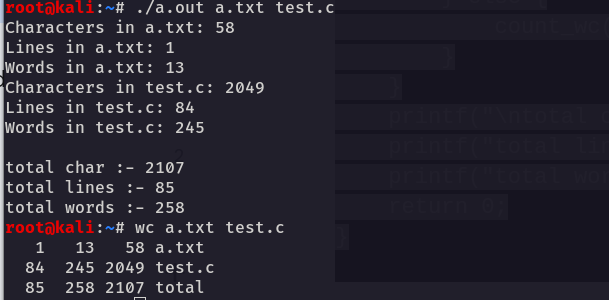
printf("total words :- %d\n",total\_words);

return 0;

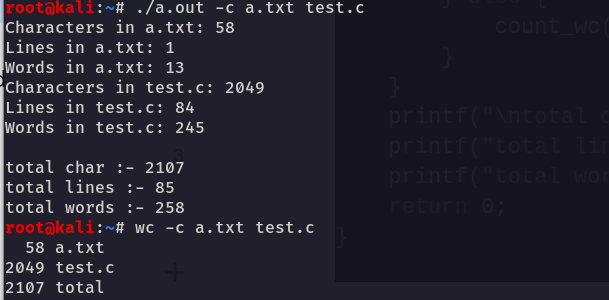
}

**Output:**

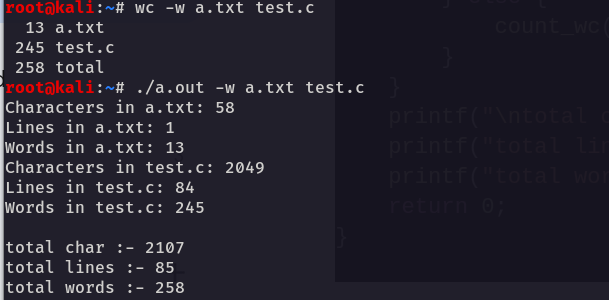
**wc <file1> <file2>...:**

****

**wc -c <file1> <file2> …:**

****

**wc -l <file1> <file2> …:**

****

**wc -w <file1><file2>...:**

