

Operating

Systems

LAB - 05

* OS lab (lab 1-4)
* Faculty name: Sir Usama Bin Umer
* Student ID: K22-4779
* Student name: Muhammad Bilal

**LAB-01**

**Q1. Provide details about the following commands?**

1. apt-get: Command-line package management tool for Debian-based Linux distributions.

2. yum: Command-line package management tool for Red Hat-based Linux distributions.

3. wget: Command-line utility for downloading files from the web.

4. gzip tar: Combined use of `tar` to create and manipulate archives and `gzip` to compress them.

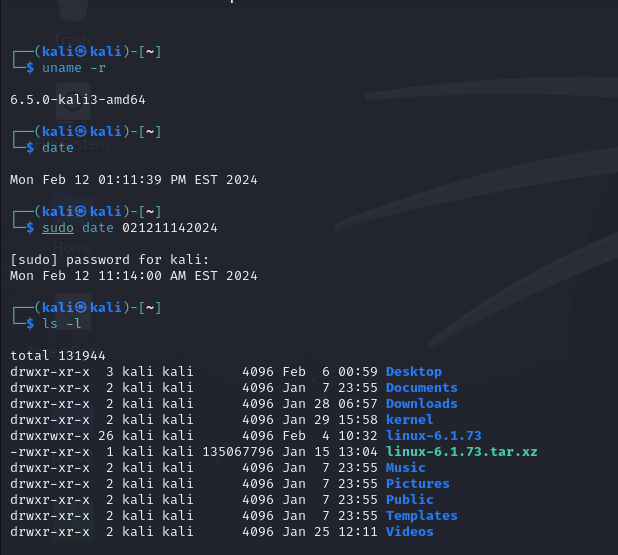
5. rar Command-line utility for creating, extracting, and managing RAR archive files.

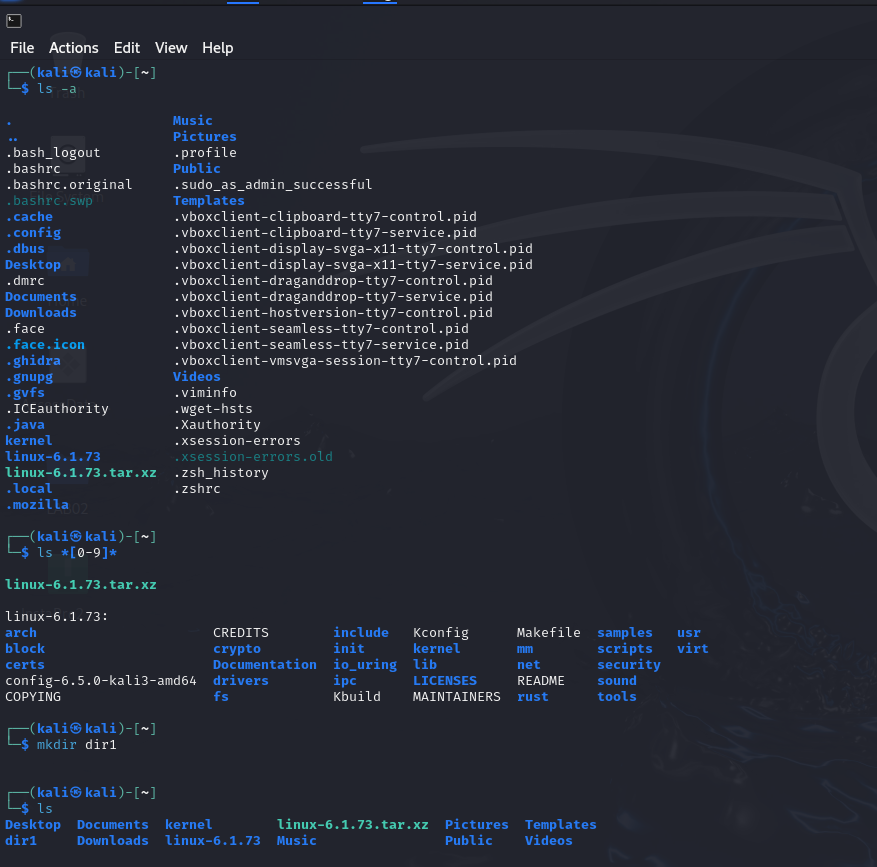
**Q2. Find and Execute following commands in Linux Shell?**

**1 & 2**

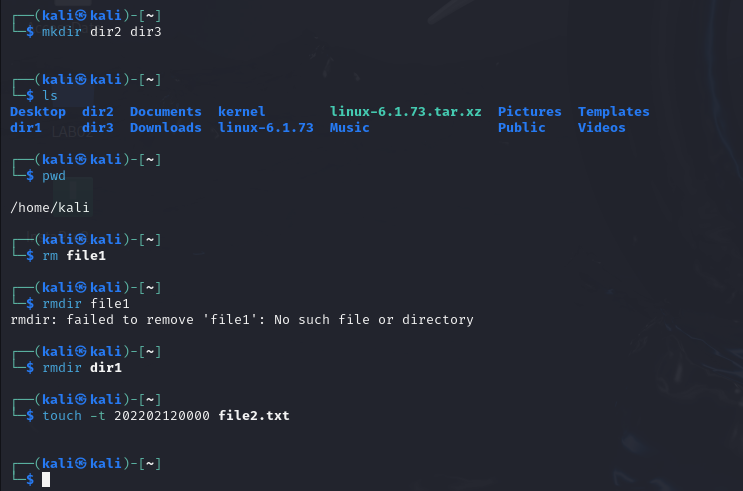
****

**3, 4, 5 & 6**

****

**7, 8, 9**

**10, 11,12,14,15**

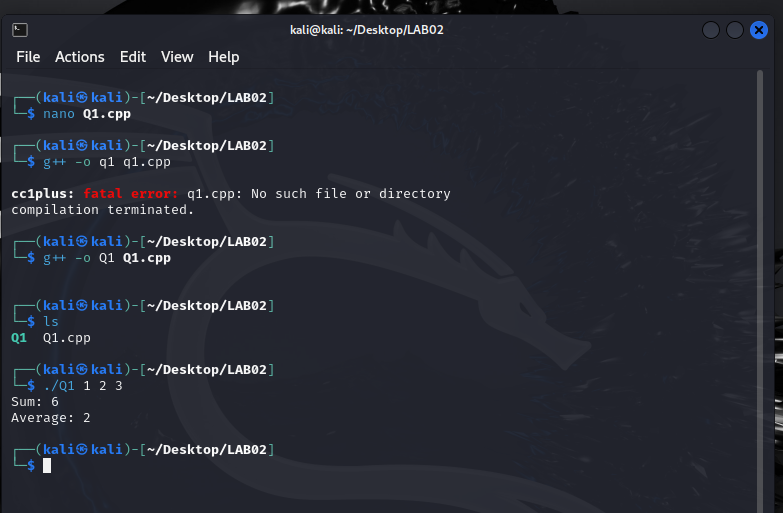
****

**13**

****

**Lab 02**

**TASK # 1**



**Code:**

#include <iostream>

#include <cstdlib>

using namespace std;

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

if (argc <= 1) {

cout << "Error: No integers provided.\n";

return 1;

}

int sum = 0;

int count = 0;

int \*arr = new int[argc - 1];

// Loop through command line arguments starting from index 1

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

arr[i - 1] = atoi(argv[i]);

sum += arr[i - 1];

count++;

}

double average = static\_cast<double>(sum) / count;

cout << "Sum: " << sum << endl;

cout << "Average: " << average << endl;

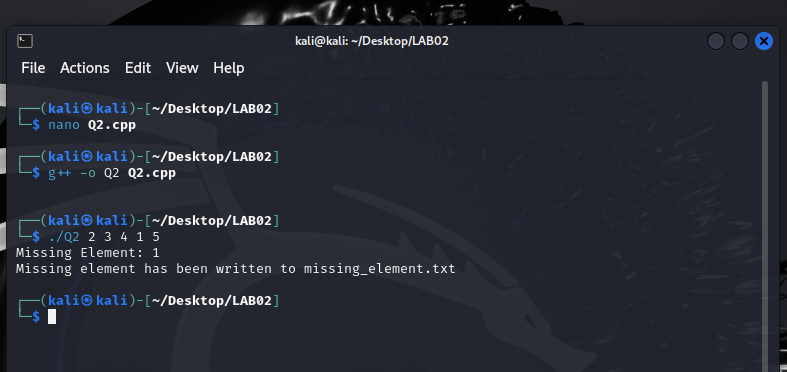
// Free dynamically allocated memory

delete[] arr;

return 0;

}

**TASK # 2**



**Code:**

#include <iostream>

#include <fstream>

#include <vector>

#include <cstdlib>

using namespace std;

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

if (argc <= 2) {

cout << "Error: Insufficient integers provided.\n";

return 1;

}

vector<int> series;

// Loop through command line arguments starting from index 2

for (int i = 2; i < argc; ++i) {

series.push\_back(atoi(argv[i]);

}

int missingElement = atoi(argv[1]);

for (int num : series) {

missingElement ^= num;

}

ofstream outFile("missing\_element.txt");

if (!outFile) {

cerr << "Error opening file for writing." << endl;

return 1;

}

outFile << "Missing Element: " << missingElement << endl;

cout << "Missing Element: " << missingElement << endl;

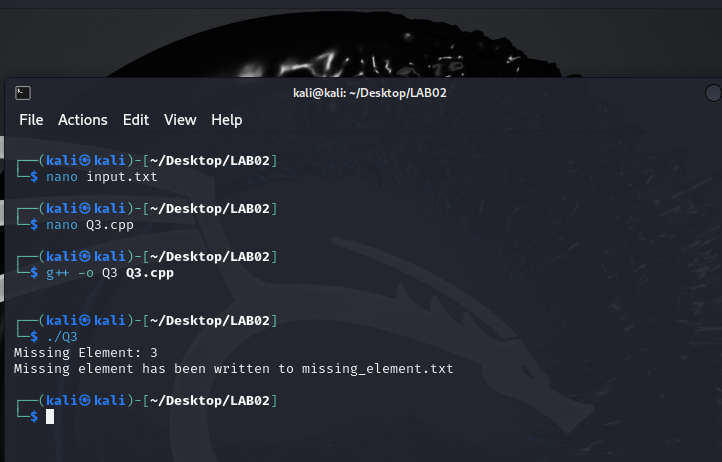
cout << "Missing element has been written to missing\_element.txt" << endl;

outFile.close();

return 0;

}

**TASK # 3**

****

**Code:**

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

int findMissing(const vector<int>& series) {

int n = series.size() + 1;

int total = (n \* (n + 1)) / 2;

int sum = 0;

for (int num : series) {

sum += num;

}

return total - sum;

}

int main() {

string filename = "input.txt";

ifstream inFile(filename);

if (!inFile.is\_open()) {

cerr << "Error: Unable to open file " << filename << endl;

return 1;

}

vector<int> series;

int num;

while (inFile >> num) {

series.push\_back(num);

}

inFile.close();

int missingElement = findMissing(series);

string outputFilename = "missing\_element.txt";

ofstream outFile(outputFilename);

if (!outFile.is\_open()) {

cerr << "Error: Unable to open file " << outputFilename << " for writing." << endl;

return 1;

}

outFile << "Missing Element: " << missingElement << endl;

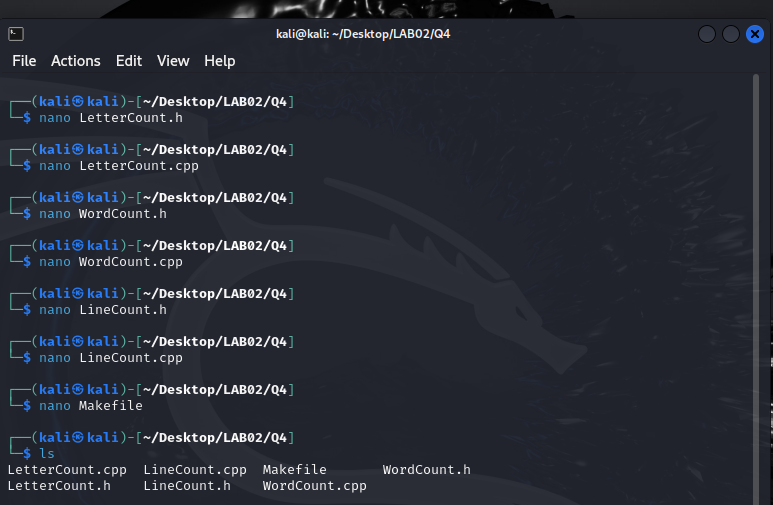
cout << "Missing Element: " << missingElement << endl;

cout << "Missing element has been written to " << outputFilename << endl

outFile.close();

return 0;}

**TASK # 4**



**Codes:**

LetterCount.h:

#ifndef LETTERCOUNT\_H

#define LETTERCOUNT\_H

#include <string>

#include <fstream>

class LetterCount {

private:

int count;

public:

LetterCount();

void countLetters(const std::string& filename);

int getCount() const;

};

#endif // LETTERCOUNT\_H

LetterCount.cpp:

#include "LetterCount.h"

LetterCount::LetterCount() : count(0) {}

void LetterCount::countLetters(const std::string& filename) {

std::ifstream inFile(filename);

if (!inFile.is\_open()) {

std::cerr << "Error: Unable to open file " << filename << std::endl;

return;

}

char ch;

while (inFile.get(ch)) {

if (std::isalpha(ch)) {

count++;

}

}

inFile.close();

}

int LetterCount::getCount() const {

return count;

}

WordCount.h:

#ifndef WORDCOUNT\_H

#define WORDCOUNT\_H

#include <string>

#include <fstream>

class WordCount {

private:

int count;

public:

WordCount();

void countWords(const std::string& filename);

int getCount() const;

};

#endif // WORDCOUNT\_H

WordCount.cpp:

#include "WordCount.h"

WordCount::WordCount() : count(0) {}

void WordCount::countWords(const std::string& filename) {

std::ifstream inFile(filename);

if (!inFile.is\_open()) {

std::cerr << "Error: Unable to open file " << filename << std::endl;

return;

}

std::string word;

while (inFile >> word) {

count++;

}

inFile.close();

}

int WordCount::getCount() const {

return count;

}

LineCount.h:

#ifndef LINECOUNT\_H

#define LINECOUNT\_H

#include <string>

#include <fstream>

class LineCount {

private:

int count;

public:

LineCount();

void countLines(const std::string& filename);

int getCount() const;

};

#endif // LINECOUNT\_H

LineCount.cpp:

#include "LineCount.h"

LineCount::LineCount() : count(0) {}

void LineCount::countLines(const std::string& filename) {

std::ifstream inFile(filename);

if (!inFile.is\_open()) {

std::cerr << "Error: Unable to open file " << filename << std::endl;

return;

}

std::string line;

while (std::getline(inFile, line)) {

count++;

}

inFile.close();

}

int LineCount::getCount() const {

return count;

}

Now, let's create a Makefile to compile these classes:

Makefile:

CC = g++

CFLAGS = -std=c++11 -Wall

all: main

main: main.o LetterCount.o WordCount.o LineCount.o

$(CC) $(CFLAGS) $^ -o $@

main.o: main.cpp

$(CC) $(CFLAGS) -c $<

LetterCount.o: LetterCount.cpp LetterCount.h

$(CC) $(CFLAGS) -c $<

WordCount.o: WordCount.cpp WordCount.h

$(CC) $(CFLAGS) -c $<

LineCount.o: LineCount.cpp LineCount.h

$(CC) $(CFLAGS) -c $<

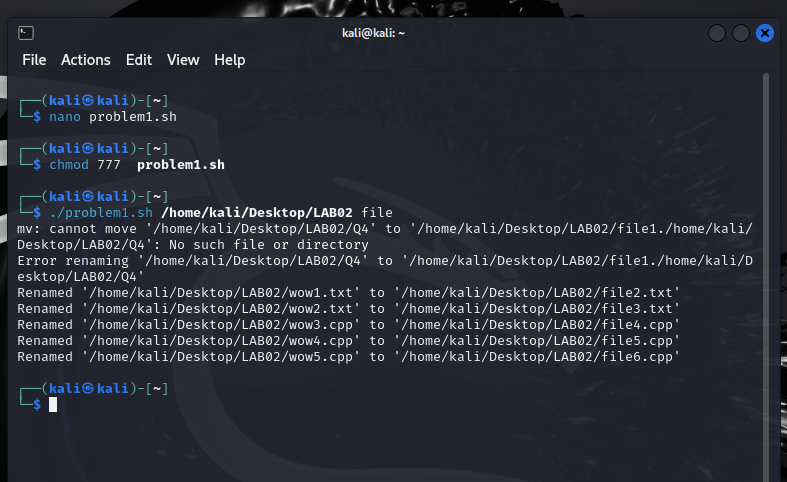
clean:

rm -rf \*.o main

**LAB 03**

**PROBLEM # 1**

**Practical:**



**Code:**

#!/bin/bash

if [ "$#" -ne 2 ]; then

echo "Enter the folder path: $0 <directory\_path>"

exit 1

fi

folder="$1"

name\_pattern="$2"

if [ ! -d "$folder" ]; then

echo "Error: Couldn't find the folder '$folder'."

exit 1

fi

counter=1

for item in "$folder"/\*; do

extension="${item##\*.}"

new\_name="${folder}/${name\_pattern}${counter}.${extension}"

mv "$item" "$new\_name"

if [ $? -eq 0 ]; then

echo "Renamed '$item' to '$new\_name'"

else

echo "Failed to rename '$item' to '$new\_name'"

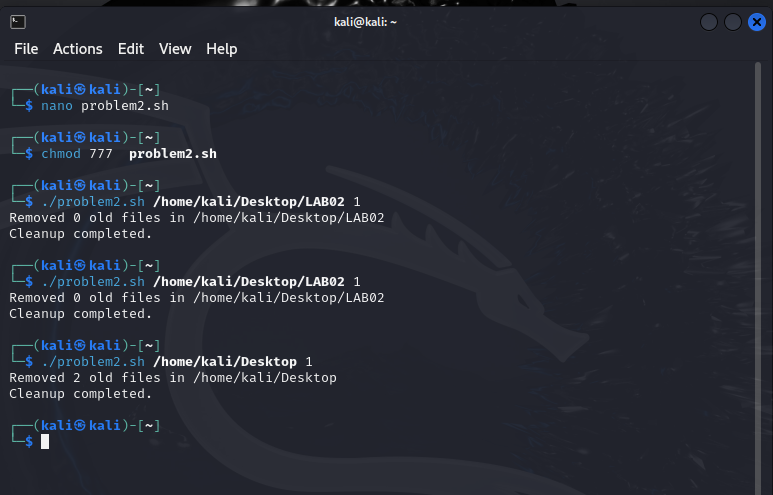
fi

((counter++))

done

**PROBLEM # 2**

**Practical:**

****

**code:**

#!/bin/bash

cleanup\_empty\_directories() {

find "$1" -type d -empty -delete

}

delete\_old\_files() {

local dir="$1"

local days="$2"

local count=0

while IFS= read -r -d '' file; do

rm -f "$file"

((count++))

done < <(find "$dir" -type f -mtime +"$days" -print0)

echo "Removed $count old files in $dir"

}

if [ "$#" -ne 2 ]; then

echo "Usage: $0 <directory\_path> <days>"

exit 1

fi

directory="$1"

days="$2"

if [ ! -d "$directory" ]; then

echo "Error: Directory '$directory' not found."

exit 1

fi

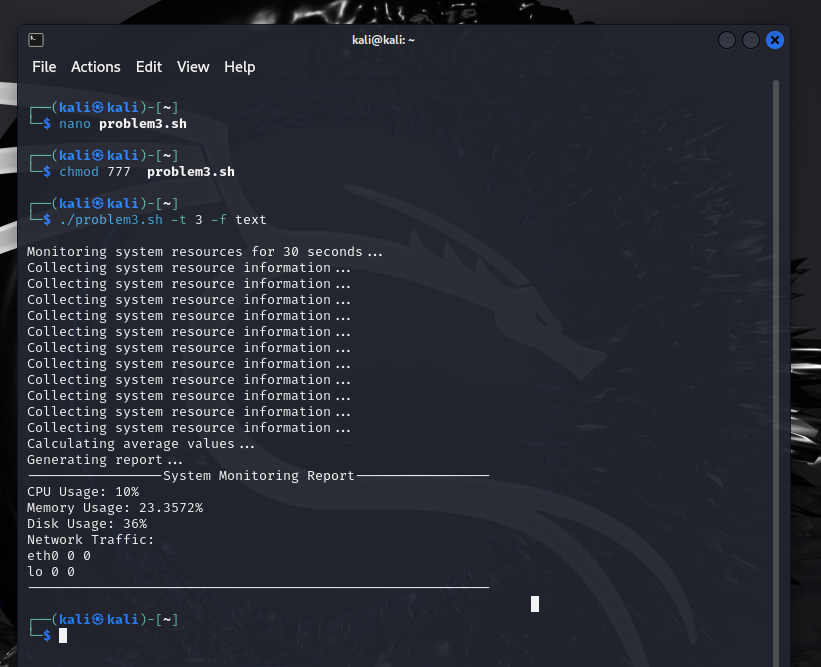
delete\_old\_files "$directory" "$days"

cleanup\_empty\_directories "$directory"

echo "Cleanup completed."

**PROBLEM # 3**

**Practical:**



**Note:** I encountered too many error and didn’t knew many terms so, I have taken some from Google. Kindly consider this.

**Code:**

#!/bin/bash

DEFAULT\_INTERVAL=3

DEFAULT\_DURATION=30

DEFAULT\_FORMAT="text"

display\_usage() {

echo "Usage: $0 [-t <time\_interval>] [-d <monitoring\_duration>] [-f <output\_format>]"

echo "Options:"

echo " -t <time\_interval>: Specify the time interval for monitoring in seconds (default: $DEFAULT\_INTERVAL)"

echo " -d <monitoring\_duration>: Specify the duration for monitoring in seconds (default: $DEFAULT\_DURATION)"

echo " -f <output\_format>: Specify the output format (text or csv, default: $DEFAULT\_FORMAT)"

exit 1

}

collect\_info() {

echo "Collecting system resource information..."

cpu\_usage=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1}')

memory\_usage=$(free | grep Mem | awk '{print $3/$2 \* 100.0}')

disk\_usage=$(df -h | awk '$NF=="/"{printf "%s", $5}')

network\_traffic=$(netstat -i | awk '{if(NR>2) print $1,$4,$8}')

}

calculate\_average() {

echo "Calculating average values..."

avg\_cpu\_usage=$cpu\_usage

avg\_memory\_usage=$memory\_usage

avg\_disk\_usage=$disk\_usage

avg\_network\_traffic=$network\_traffic

}

generate\_report() {

echo "Generating report..."

case $output\_format in

text)

echo "-----------------System Monitoring Report-----------------"

echo "CPU Usage: $cpu\_usage%"

echo "Memory Usage: $memory\_usage%"

echo "Disk Usage: $disk\_usage"

echo "Network Traffic:"

echo "$network\_traffic"

echo "----------------------------------------------------------"

;;

csv)

echo "CPU Usage,Memory Usage,Disk Usage,Network Traffic"

echo "$cpu\_usage%,$memory\_usage%,$disk\_usage,$network\_traffic"

;;

\*)

echo "Invalid output format: $output\_format"

exit 1

;;

esac

}

while getopts ":t:d:f:" opt; do

case $opt in

t)

time\_interval=$OPTARG

;;

d)

duration=$OPTARG

;;

f)

output\_format=$OPTARG

;;

\?)

echo "Invalid option: -$OPTARG" >&2

display\_usage

;;

:)

echo "Option -$OPTARG requires an argument." >&2

display\_usage

;;

esac

done

time\_interval=${time\_interval:-$DEFAULT\_INTERVAL}

duration=${duration:-$DEFAULT\_DURATION}

output\_format=${output\_format:-$DEFAULT\_FORMAT}

echo "Monitoring system resources for $duration seconds..."

while [ $duration -gt 0 ]; do

collect\_info

sleep $time\_interval

((duration -= time\_interval))

done

collect\_info

calculate\_average

generate\_report

**LAB 04**

**Task # 1**

**Code:**

#include <iostream>

#include <cstdlib>

#include <unistd.h>

#include <sys/wait.h>

using namespace std;

int main() {

pid\_t pid = fork();

if (pid < 0) {

cerr << "Fork failed" << endl;

return 1;

} else if (pid == 0) {

cout << "Child process:" << endl;

execlp("/bin/ls", "ls", (char \*)NULL);

exit(0);

} else {

wait(NULL);

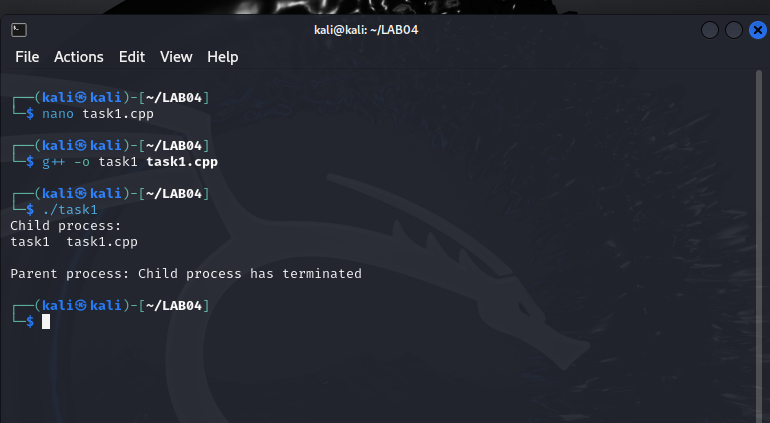
cout << endl << "Parent process: Child process has terminated" << endl;

}

return 0;

}

**Implementation:**



**Task # 2**

**Code:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid = getpid();

printf("Process ID: %d\n", pid);

pid\_t child\_pid = fork();

if (child\_pid < 0) {

fprintf(stderr, "Fork failed\n");

return 1;

} else if (child\_pid == 0) {

printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());

} else {

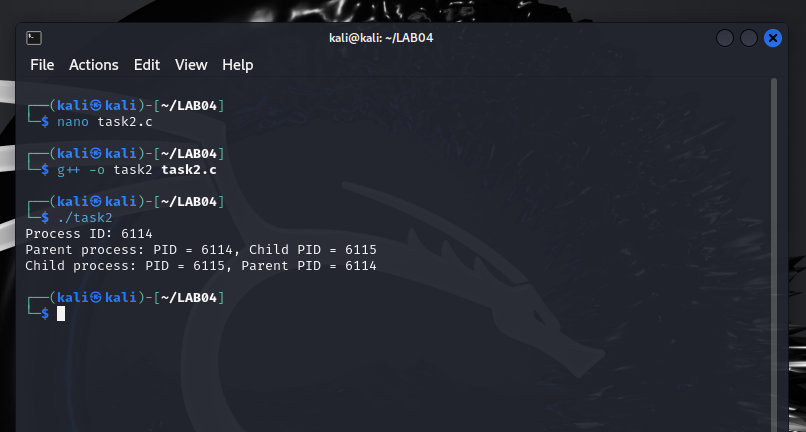
printf("Parent process: PID = %d, Child PID = %d\n", getpid(), child\_pid);

}

return 0;

}

**Implementation:**



**Task # 3**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#define BUFFER\_SIZE 4096

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <source\_file> <destination\_file>\n", argv[0]);

return 1;

}

int source\_fd = open(argv[1], O\_RDONLY);

if (source\_fd == -1) {

perror("Failed to open source file");

return 1;

}

int dest\_fd = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (dest\_fd == -1) {

perror("Failed to open destination file");

close(source\_fd);

return 1;

}

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read, bytes\_written;

while ((bytes\_read = read(source\_fd, buffer, BUFFER\_SIZE)) > 0) {

bytes\_written = write(dest\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read) {

perror("Write error");

close(source\_fd);

close(dest\_fd);

return 1;

}

}

if (bytes\_read == -1) {

perror("Read error");

close(source\_fd);

close(dest\_fd);

return 1;

}

if (close(source\_fd) == -1) {

perror("Failed to close source file");

return 1;

}

if (close(dest\_fd) == -1) {

perror("Failed to close destination file");

return 1;

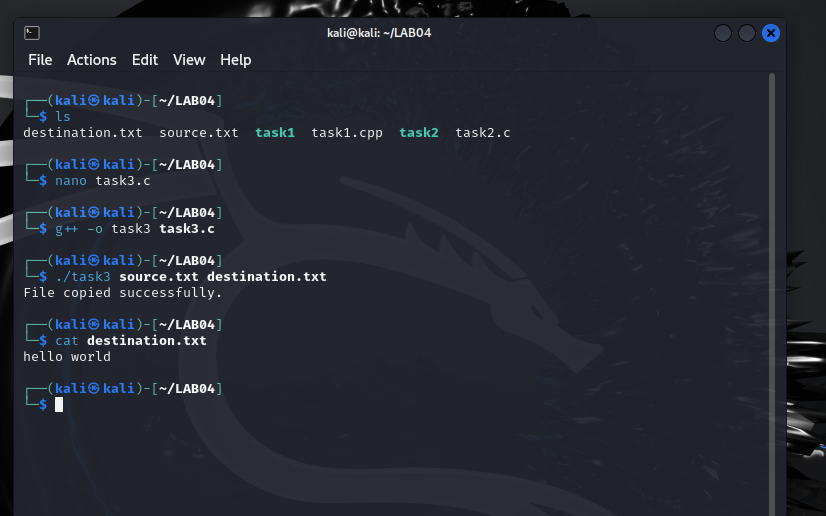
}

printf("File copied successfully.\n");

return 0;

}

**Implementation:**



**Task # 4**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <string.h>

void listFiles(const char \*path) {

DIR \*dir;

struct dirent \*entry;

struct stat statbuf;

dir = opendir(path);

if (dir == NULL) {

perror("opendir");

return;

}

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

char filepath[1024];

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

if (lstat(filepath, &statbuf) == -1) {

perror("lstat");

continue;

}

if (S\_ISDIR(statbuf.st\_mode)) {

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

continue;

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

listFiles(filepath);

} else if (S\_ISREG(statbuf.st\_mode)) {

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

}

}

closedir(dir);

}

int main() {

listFiles(".");

return 0;

}

**Implementation:**

