**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);

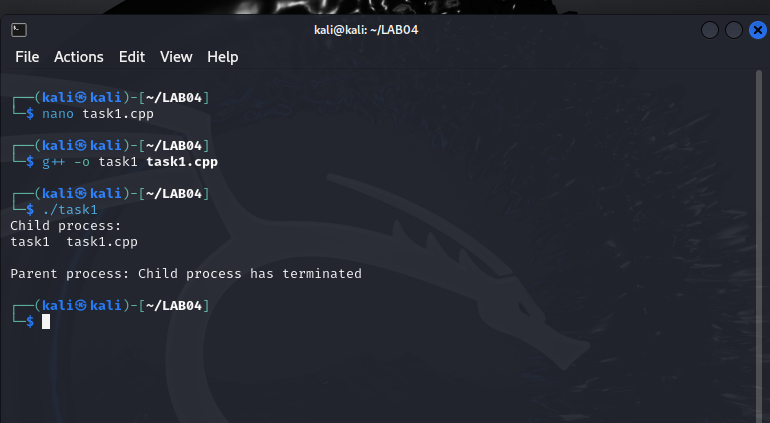
cout << endl << "Parent process: Child process has terminated" << endl;

}

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 {

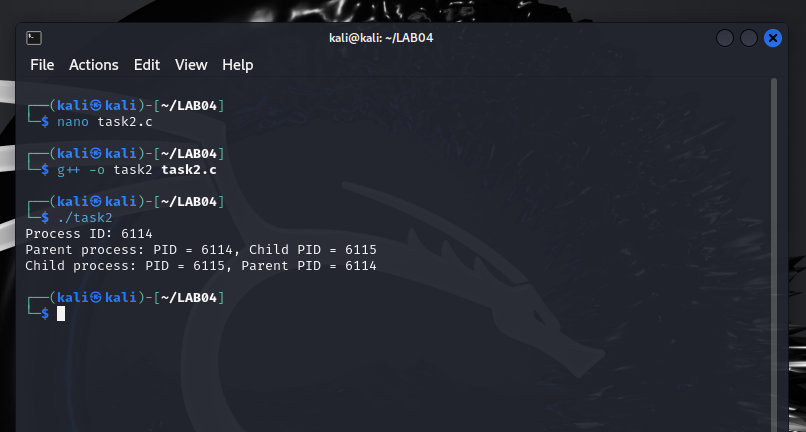
printf("Parent process: PID = %d, Child PID = %d\n", getpid(), child\_pid);

}

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;

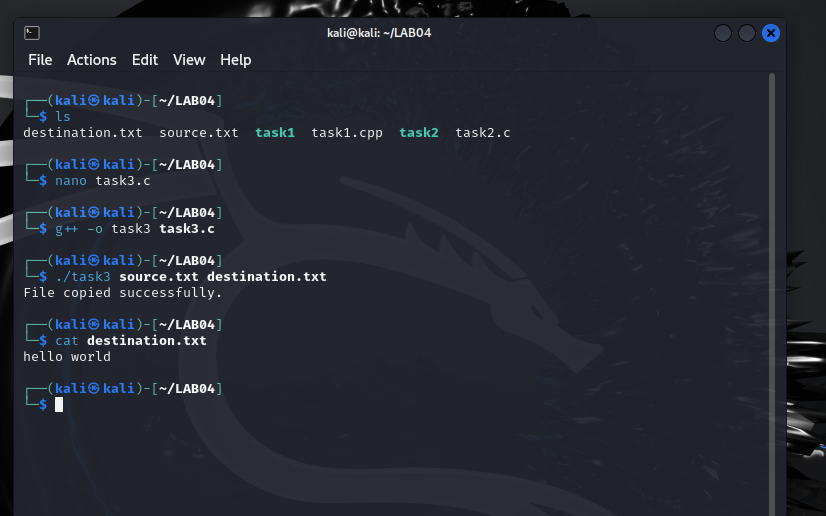
}

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:**

