1. 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.

```
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t1.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t1.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out
Child process: PID 3162 - Listing directory contents
total 88
drwxrwxr-x 2 k200281kainat k200281kainat 4096 چرام  2 19:39 backup
drwxr-xr-x 3 k200281kainat k200281kainat 4096 يروف 9 17:41 Desktop
drwxrwxr-x 2 k200281kainat k200281kainat 4096 چرام  2 19:23 dir
drwxr-xr-x 2 k200281kainat k200281kainat 4096 يروف 9 14:54 Documents
drwxr-xr-x 3 k200281kainat k200281kainat 4096 ليپ  4  2022 Downloads
-rw-rw-r-- 1 k200281kainat k200281kainat   18 چرام  2 19:56 file1_2025-03-02.tx
t
-rw-rw-r-- 1 k200281kainat k200281kainat   18 چرام  2 19:55 file1.txt
-rw-rw-r-- 1 k200281kainat k200281kainat  622 يروف 23 13:09 idscode.c
drwxr-xr-x 2 k200281kainat k200281kainat 4096 ليپ  5  2022 Music
-rw-rw-r-- 1 k200281kainat k200281kainat  365 چرام  4 13:37 osl4t1.c
drwxrwxr-x 2 k200281kainat k200281kainat 4096 يروف 9 17:26 OSlab
-rwxrwxr-x 1 k200281kainat k200281kainat  610 چرام  2 18:06 OSlab3task2.sh
-rwxrwxr-x 1 k200281kainat k200281kainat  460 چرام  2 19:52 OSlab3task3.sh
-rwxrwxr-x 1 k200281kainat k200281kainat  359 چرام  2 19:42 OSlab3task5.sh
-rwxrwxr-x 1 k200281kainat k200281kainat  687 چرام  2 19:22 OSlab3task8.sh
-rwxrwxr-x 1 k200281kainat k200281kainat 8912 چرام  4 13:37 out
drwxr-xr-x 2 k200281kainat k200281kainat 4096 يروف 9 17:23 Pictures
drwxr-xr-x 2 k200281kainat k200281kainat 4096 ليپ  5  2022 Public
drwxr-xr-x 2 k200281kainat k200281kainat 4096 ليپ  5  2022 Templates
drwxr-xr-x 2 k200281kainat k200281kainat 4096 ليپ  5  2022 Videos
```

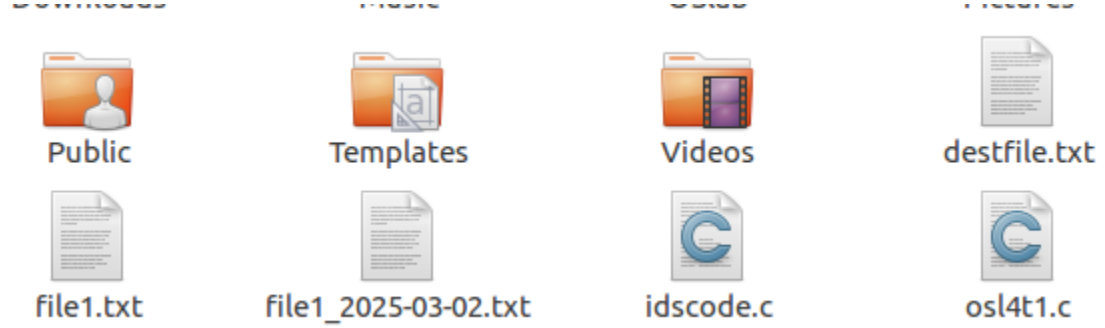
```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>

int main()
{
pid_t f;
f= fork();
if(f<0)
{
printf("error");
exit(1);
}
else if(f==0)
{
printf("Child process: PID %d - Listing directory contents\n",getpid());
execlp("ls", "ls", "-l", NULL);
perror("execlp failed");
exit(1);
}
else
{
wait(NULL);
}
return 0;
}
```

2. 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.

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t2.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t2.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t2.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out
Original process (Before Fork) - PID: 3323
Parent process: PID 3323, Child process: PID 3324
Child process: PID 3324 , Parent process: PID 3323
```

```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>

int main()
{
printf("Original process (Before Fork) - PID: %d\n",getpid());
pid_t f;
f= fork();
if(f<0)
{
printf("error");
exit(1);
}
else if(f==0)
{
printf("Child process: PID %d , Parent process: PID %d\n",getpid(),getppid());
}
else
{
printf("Parent process: PID %d, Child process: PID %d\n",getpid(),f);
wait(NULL);
}
return 0;
}
```

3. 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.




```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t3.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t3.c
k200281kainat@k200281kainat-VirtualBox:~$ touch destfile.txt
k200281kainat@k200281kainat-VirtualBox:~$ cat destfile.txt
k200281kainat@k200281kainat-VirtualBox:~$ cat file1.txt
This is me KINZA
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t3.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out 3 file.txt destfile.txt
Usage: ./out <source_file> <destination_file>
k200281kainat@k200281kainat-VirtualBox:~$ ./out file1.txt destfile.txt
File copied successfully from 'file1.txt' to 'destfile.txt'
k200281kainat@k200281kainat-VirtualBox:~$ cat destfile.txt
This is me KINZA
k200281kainat@k200281kainat-VirtualBox:~$ cat file1.txt
This is me KINZA
```

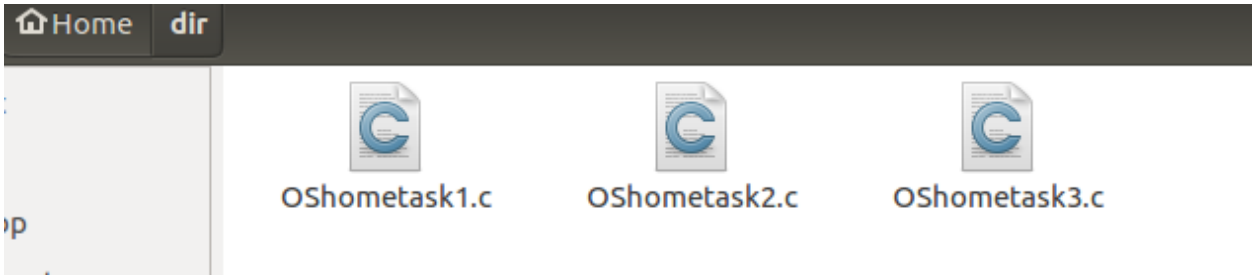
file1.txt is the source file here.
And destfile.txt is the destination file here.

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 1024
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source_file> <destination_file>\n", argv[0]);
        exit(1);
    }
    int src_fd, dest_fd;
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read, bytes_written;

    src_fd = open(argv[1], O_RDONLY);
    if (src_fd == -1) {
        perror("Error opening source file");
        exit(1);
    }

    dest_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(src_fd);
        exit(1);
    }
    while ((bytes_read = read(src_fd, buffer, BUFFER_SIZE)) > 0) {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written == -1) {
            perror("Error writing to destination file");
            close(src_fd);
            close(dest_fd);
            exit(1);
        }
    }

    if (bytes_read == -1) {
        perror("Error reading source file");
    }

    close(src_fd);
    close(dest_fd);

    printf("File copied successfully from '%s' to '%s'\n", argv[1], argv[2]);
    return 0;
}
```

4. 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).

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t4.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t4.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t4.c -o out
osl4t4.c: In function ‘list_directory’:
osl4t4.c:36:51: error: expected statement before ‘]’ token
             list_directory(full_path, depth + 1); ]
                                                   ^
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t4.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t4.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out /home/k200281kainat/dir
Listing contents of directory: /home/k200281kainat/dir
[FILE] OShometask2.c
[FILE] OShometask3.c
[FILE] OShometask1.c
```

I have choosen this directory named dir
Which contains these 3 files.



```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
void list_directory(const char *dir_path, int depth) {
    DIR *dir;
    struct dirent *entry;
    struct stat file_stat;
    char full_path[1024];
    if ((dir = opendir(dir_path)) == NULL) {
        perror("Error opening directory");
        return;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;
       snprintf(full_path, sizeof(full_path), "%s/%s", dir_path, entry->d_name);

        if (stat(full_path, &file_stat) == -1) {
            perror("Error getting file stats");
            continue;
        }

        for (int i = 0; i < depth; i++)
            printf("  ");

        if (S_ISDIR(file_stat.st_mode)) {
            printf("[DIR]  %s\n", entry->d_name);
            list_directory(full_path, depth + 1);
        } else {
            printf("[FILE] %s\n", entry->d_name);
        }
    }
    closedir(dir);
}
int main(int argc, char *argv[]) {
    const char *start_dir = (argc == 2) ? argv[1] : ".";
    printf("Listing contents of directory: %s\n", start_dir);
    list_directory(start_dir, 0);
    return 0;
}
```

5. 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.

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t5.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t5.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t5.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ cat file1.txt
This is me KINZA
k200281kainat@k200281kainat-VirtualBox:~$ cat destfile.txt
This is me KINZA
k200281kainat@k200281kainat-VirtualBox:~$ ./out
Usage: ./out <source_file> <backup_file>
k200281kainat@k200281kainat-VirtualBox:~$ ./out file1.txt destfile.txt
Parent Process (PID: 3870) - Waiting for backup to complete...
Child Process (PID: 3871) - Copying file...
Child Process - Backup completed successfully.
Parent Process - Backup process completed.
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define BUFFER_SIZE 1024
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source_file> <backup_file>\n", argv[0]);
        exit(1);
    }
    int src_fd, dest_fd;
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read, bytes_written;
    src_fd = open(argv[1], O_RDONLY);
    if (src_fd == -1) {
        perror("Error opening source file");
        exit(1);
    }
    dest_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (dest_fd == -1) {
        perror("Error opening backup file");
        close(src_fd);
        exit(1);
    }
    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        close(src_fd);
        close(dest_fd);
        exit(1);
    }
    else if (pid == 0) {
        printf("Child Process (PID: %d) - Copying file...\n", getpid());
```

```
        while ((bytes_read = read(src_fd, buffer, BUFFER_SIZE)) > 0) {
            bytes_written = write(dest_fd, buffer, bytes_read);
            if (bytes_written == -1) {
                perror("Error writing to backup file");
                close(src_fd);
                close(dest_fd);
                exit(1);
            }
        }
        if (bytes_read == -1) {
            perror("Error reading source file");
        }
        printf("Child Process - Backup completed successfully.\n");
        close(src_fd);
        close(dest_fd);
        exit(0);
    }
    else {
        printf("Parent Process (PID: %d) - Waiting for backup to complete...\n", getpid
());
        wait(NULL);
        printf("Parent Process - Backup process completed.\n");
        close(src_fd);
        close(dest_fd);
    }

    return 0;
}
```

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

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t6.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t6.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t6.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out file1.txt
Parent Process (PID: 4134) - Waiting for child process...
Child Process (PID: 4135) - Counting lines...
Child Process - Total lines: 1
Parent Process - Line counting completed.
k200281kainat@k200281kainat-VirtualBox:~$ ./out osl4t5.c
Parent Process (PID: 4140) - Waiting for child process...
Child Process (PID: 4141) - Counting lines...
Child Process - Total lines: 65
Parent Process - Line counting completed.
k200281kainat@k200281kainat-VirtualBox:~$ ./out idscode.c
Parent Process (PID: 4172) - Waiting for child process...
Child Process (PID: 4173) - Counting lines...
Child Process - Total lines: 45
Parent Process - Line counting completed.
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(1);
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("Error opening file");
        exit(1);
    }
    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        close(fd);
        exit(1);
    }
    else if (pid == 0) {
        char ch;
        int line_count = 0;
        ssize_t bytes_read;

        printf("Child Process (PID: %d) - Counting lines...\n", getpid());
        while ((bytes_read = read(fd, &ch, 1)) > 0) {
            if (ch == '\n')
                line_count++;
        }

        if (bytes_read == -1) {
            perror("Error reading file");
            close(fd);
            exit(1);
        }
        printf("Child Process - Total lines: %d\n", line_count);
        close(fd);
        exit(0);
    }
    else {
        printf("Parent Process (PID: %d) - Waiting for child process...\n", getpid());
        wait(NULL);
        printf("Parent Process - Line counting completed.\n");
        close(fd);
    }
        return 0;
}
```

7. 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.

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t7.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t7.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t7.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ cat file1.txt
This is me KINZA
k200281kainat@k200281kainat-VirtualBox:~$ ./out file1.txt
Parent Process (PID: 2346) - Waiting for child to finish...
Child Process (PID: 2347) - Displaying file contents:

This is me KINZA


Child Process - File display completed.
Parent Process - File reading completed.
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define BUFFER_SIZE 1024
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        exit(1);
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("Error opening file");
        exit(1);
    }
    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        close(fd);
        exit(1);
    }
    else if (pid == 0) {
        char buffer[BUFFER_SIZE];
        ssize_t bytes_read;
        printf("Child Process (PID: %d) - Displaying file contents:\n\n", getpid());

        while ((bytes_read = read(fd, buffer, BUFFER_SIZE)) > 0) {
            write(STDOUT_FILENO, buffer, bytes_read);
        }
```

```
        if (bytes_read == -1) {
            perror("Error reading file");
            close(fd);
            exit(1);
        }
        printf("\n\nChild Process - File display completed.\n");
        close(fd);
        exit(0);
    }
    else {
        printf("Parent Process (PID: %d) - Waiting for child to finish...\n", getpid());
        wait(NULL);
        printf("Parent Process - File reading completed.\n");
        close(fd);
    }
    return 0;
}
```

8. 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.

```
k200281kainat@k200281kainat-VirtualBox:~$ touch osl4t8.c
k200281kainat@k200281kainat-VirtualBox:~$ gedit osl4t8.c
k200281kainat@k200281kainat-VirtualBox:~$ gcc osl4t8.c -o out
k200281kainat@k200281kainat-VirtualBox:~$ ./out
Parent Process (PID: 2543) - Waiting for child to finish...
Parent Process - Output has been successfully stored in 'who_output.txt'.
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main() {
    pid_t pid;
    char *output_file = "who_output.txt";
    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }
    else if (pid == 0) {
        int fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if (fd == -1) {
            perror("Error opening file");
            exit(1);
        }
        dup2(fd, STDOUT_FILENO);
        close(fd);

        printf("Child Process (PID: %d) - Executing 'who' command...\n", getpid());

        execlp("who", "who", NULL);
        perror("execlp failed");
        exit(1);
    }
    else {
     printf("Parent Process (PID: %d) - Waiting for child to finish...\n", getpid());
     wait(NULL);
     printf("Parent Process - Output has been successfully stored in '%s'.\n", output_file);
    }

    return 0;
}
```