Operating System Assignment

Name- Sanket Kumbhar

PRN No.- 12110235

Roll no.- 14

Write a program demonstrating use of different system calls.(In one program show the implementation of at least 4 system calls such as read(), write() ....)

1) **open()** - To open files for reading and writing.
2) **read()** - To read data from a file.
3) **write()** - To write data to a file.
4) **close()** - To close files after using them.
5) lseek() - To set the file offset, which is used here to seek to the end of the file.
6) dup() - To duplicate a file descriptor. In this program, dup() is used to create a new file descriptor that points to the same file as the original descriptor.
7) fork() - To create a new child process. The parent and child processes have separate memory spaces but share open file descriptors.

Code—

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>


int main() {

   char sourceFile[] = "source.txt";

   char destinationFile[] = "destination.txt";

   int sourceFd, destFd;

   ssize_t bytesRead, bytesWritten;

   char buffer[1024];


   // Open the source file in read-only mode

   sourceFd = open(sourceFile, O_RDONLY);
```

```
if (sourceFd == -1) {
    perror("Error opening source file");
    return EXIT_FAILURE;
}

// Create or truncate the destination file and open it in write-only mode
destFd = open(destinationFile, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if (destFd == -1) {
    perror("Error opening destination file");
    close(sourceFd);
    return EXIT_FAILURE;
}

// Create a child process using fork()
pid_t childPid = fork();

if (childPid < 0) {
    perror("Fork failed");
    close(sourceFd);
    close(destFd);
    return EXIT_FAILURE;
} else if (childPid == 0) {  // Child process
    // Duplicate the file descriptor using dup()
    int newDestFd = dup(destFd);

    // Seek to the end of the file using lseek()
    off_t offset = lseek(newDestFd, 0, SEEK_END);
    if (offset == -1) {
        perror("Error seeking in child process");
        close(sourceFd);
        close(newDestFd);
```

```
        return EXIT_FAILURE;
    }

    // Read data from source file and write it to destination file
    while ((bytesRead = read(sourceFd, buffer, sizeof(buffer))) > 0) {
        bytesWritten = write(newDestFd, buffer, bytesRead);
        if (bytesWritten == -1) {
            perror("Error writing in child process");
            close(sourceFd);
            close(newDestFd);
            return EXIT_FAILURE;
        }
    }

    if (bytesRead == -1) {
        perror("Error reading in child process");
        close(sourceFd);
        close(newDestFd);
        return EXIT_FAILURE;
    }

    close(sourceFd);
    close(newDestFd);

    printf("Child process: File copied successfully.\n");
    return EXIT_SUCCESS;

} else {  // Parent process
    // Wait for the child process to finish
    int status;
    waitpid(childPid, &status, 0);
```

```
    // Close the file descriptors in the parent process
    close(sourceFd);
    close(destFd);

    printf("Parent process: Child process finished.\n");
    return EXIT_SUCCESS;
  }
}
```

Output—