```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 1. Create the following types of a files using (i) shell command (ii) system
call
// a. soft link (symlink system call)
// b. hard link (link system call)
// c. FIFO (mkfifo Library Function or mknod system call)

#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    // declaring the names of files
    char *myOriginalFile = "MyOriginalFile.txt";
    char *symLinkFile = "mySymLinkFile";
    char *hardLinkFile = "myHardLinkFile";
    char *fifoFile = "myFifoFile";

    // creating the file with name "MyOriginalFile.txt"  which an original file
for all linker links
    // FILE *fp is file pointer to a file object
    // fopen return the pointer to a FILE structure that represent the open file
 stream on success else null
    //
    FILE *fp = fopen(myOriginalFile, "w");
    if (!fp) {
        perror("Error creating original file");
        return 1;
    }

    // used to write formatted O/P to file stream.
    // fprintf(File *stream, const char *format, ...)
    // 1st arg - file pointer to file
    // 2nd arg - formate string
    // 3rd - additional argument for format
    fprintf(fp, "This is out original file which is used for linking purpose.\n");
    // close the file pointer
    fclose(fp);

    // int symlink(const char *target, const char *linkpath);
    //  it will create new file that points to target file
    // if original file is deleted, moved, renamed the symbolic link may become
"broken"
    // there are there error
    // 1. EEXIST  = sym link already exist
    // 2. ENOENT  = target file not exist
    // 3. EACCES  = parmission denied
    if (symlink(myOriginalFile, symLinkFile) == -1) {
        perror("we have encountered and Error  while creating symbolic link");
        return 1;
    }
    printf("Symbolic link created: %s -> %s\n", symLinkFile, myOriginalFile);

    // hardlink is just a additional entry that points to the same original file
 inode
    // both are same and sharing the same data block changes made to one file wi
ll reflext to another
    // all there error are same but haere one error we can see
    // 4. EXDEV -  target and files are on different file system  as we cannot c
reate hard link on different file system
```

```c
    if (link(myOriginalFile, hardLinkFile) == -1) {
        perror("we have encountered and Error  while creating hard link");
        return 1;
    }

    printf("Hard link created: %s -> %s\n", hardLinkFile, myOriginalFile);

    // Fifo is IPC
    // mkfifo() takes 2 argument path and permissions
    if (mkfifo(fifoFile, 0666) == -1) {
        perror("we have encountered and Error  while creating FIFO");
        return 1;
    }
    printf("Fifo file is : %s\n",fifoFile);

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 2. Write a simple program to execute in an infinite loop at the background. G
o to /proc directory and
// identify all the process related information in the corresponding proc direct
ory.


#include <stdio.h>
#include <unistd.h>

int main() {
    // run this program in backgroud using below command


    // './Question2 &'
    // 'ps aux | grep Question2'

    // now to explore the PID
    // '  cd /proc/<process id of running process> '


    // in this directory we can also get the detailed info about the process
    // cat /proc/<pid>/cmdline - the cmd arg passed to process
    // cat /proc/<PID>/status  -  info about the process status, pid, parent pid
, etc
    // ls -l /proc/<PID>/fd -  list of all opened files
    // cat /proc/<PID>/maps - memory mapping of program
    // ls -l /proc/<PID>/cwd - display process currently working directory
    // cat /proc/<PID>/environ | tr '\0' '\n'  -  file contains the environment
variable passed to process
    // ls -l /proc/<PID>/exe
    // cat /proc/<PID>/stat -   single line space seprated values representing t
he process status and performance
    // cat /proc/<PID>/limits - diplay limit imposed on process (max open files,
 memory usage)
    // cat /proc/<PID>/sched - it display scheduling info  - such as number of v
oluntary and involuntary context switches
    // run this infinitely
    while (1) {
        // sleeping program for every 2-2 sec to utilize the cpu resources
        sleep(1);
    }
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 3. Write a program to create a file and print the file descriptor value. Use
creat ( ) system call


#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int main() {
    char fileName[250];  // File name buffer
    int fd; // declaring the FD
    printf("Enter the file name - \n");
    if(fgets(fileName,sizeof(fileName),stdin)==NULL){
        perror("Error while reading filename");
        return 1;
    }

    // Create System call is used to create a new file,truncate an existing file
 and open file in write only mode
    // creat(pathname,mode_t mode) - mode -> it is file permission to be set whe
n file is created. uses
    // S_IRUSER -  read permission for owner
    // S_IWUSER - write permission for owner
    // S_IXUSER - execute/search permission for owner
    // and same stand for  - S_IRGRP,S_IWGRP,S_IXGRP for group
    // and same stand for  - S_IROTH,S_IWOTH,S_IXOTH for other
    // its equivalent open is open(filename,O_CREAT | O_WRONLY | O_TRUNC, mode)
    // creat return FD
    fd = creat(fileName, S_IRUSR | S_IWUSR);
    if (fd < 0) {
        perror("There is an Error while creating the file.");
        return 1;
    }

    // Print the file descriptor value and name
    printf("Filename is : %s \nDescriptor  value is : %d\n",fileName ,fd);

    // Close the file descriptor
    close(fd);
    // return success code

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 4. Write a program to open an existing file with read write mode. Try O_EXCL
flag also.

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

int main() {
    char fileName[250];  // Replace with your file name
    int fd;
    printf("Enter the filename \n");
    if(fgets(fileName,sizeof(fileName),stdin)== NULL) {
        perror("Error  while reading input");
        return 1;
    }
    // O_CREAT Create the file if does not exist, always use O_RDWR to ensure
    // that a file can be created if it does not exist
    // O_EXCL - prevent from overwritng the file
    fd = open(fileName, O_RDWR | O_CREAT | O_EXCL, 0666);
    // now check the fd if it able to open or not
    if (fd < 0) {

        // here we compare the error
        // by compare errno with  EEXIST flag
        if (errno == EEXIST) {
            printf("File is already exists.\nO_EXCL flag prevented opening the file.\n.Please enter different
name\n");
        } else {
            perror("Error while opening the file");
        }
        // return error code
        return 1;
    }

    printf("Congratulations: Your File is opened successfully with O_EXCL flag.\n");
    // Close the file
    close(fd);
    // return success code
    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 5. Write a program to create five new files with infinite loop. Execute the p
rogram in the background
// and check the file descriptor table at /proc/pid/fd.
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
// defining basefile name
#define NAME_OF_THE_BASE_FILE "testfile_Q5_"
#define EXTENSION ".txt"
// as explained in question setted the file count to 5
#define FILE_COUNT 5

int main() {
    // declaring the array of int for file descriptor of size 5
    int file_descriptors_array[FILE_COUNT];
    // declaring the array of char for filename
    char filename[256];

    int i;

    while (1) {
        for (i = 0; i < FILE_COUNT; ++i) {
            // snprintf() used to format and store a string in a buffer with spe
cified size
            // 1st param filename is bufffer where formatted data is stored
            // 2nd param the size of the buffer
            // 3rd param %s%d%s are replaces by the filename + number + extensio
n
            // 4th param base name of the file
            // 5th param number of the file
            // 6th param extension
            snprintf(filename, sizeof(filename), "%s%d%s", NAME_OF_THE_BASE_FIL
E, i,EXTENSION);
            // create a file with specified name and 0666 permission and open it
 with Read write permission
            file_descriptors_array[i] = open(filename, O_CREAT | O_RDWR | O_APPE
ND, 0666);

            if (file_descriptors_array[i] < 0) {
                perror("There is an Error  while opening file");
                exit(EXIT_FAILURE);
            }

            char *tempdata = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnop
qrstuvwxyz";
            if (write(file_descriptors_array[i], tempdata, strlen(tempdata)) < 0
) {
                perror("Error writing to file");
                close(file_descriptors_array[i]);
                exit(EXIT_FAILURE);
            }
        }

        // Sleeping for 5 sec so program do not use extra time
        sleep(5);

        // Closing all the opened files
        for (i = 0; i < FILE_COUNT; ++i) {
```

```
            close(file_descriptors_array[i]);
        }
    }

    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 6. Write a program to take input from STDIN and display on STDOUT. Use only r
ead/write system calls

#include <unistd.h>
#include <stdio.h>

int main() {
    char maxBufferSpace[500];
    // Diff between stdin vs STDIN_FILENO
    // 1. stdin - is just a file pointer to ' FIle * ' defined in <stdio.h>
    //    it uses higher level I/O function like fscanf and fgets and fread
    // 2. STDIN_FILENO is an integer constant representing the FD for standard i
nput in <unistd.h>
    //    typically has the value '0'
    //    it uses low level system calls like read and write
    //    STDIN_FILE - interact with OS directly bypassing standard library buff
ering
    //    so we have direct control over I/O

    // abstraction
    // std -> highlevel abstraction offers buffered I/O - slow
    // stdin_fileno -> unbuffered I/O , direct access to system I/O,LL abstracti
on - faster
    // Buffered I/O -> instead of transferring data immediately to or from file
or device
    // data is first stored in a buffer, once it is full then data is transferre
d all at once
    // by doing this system calls are redunce but performance of program is decr
eased
    // and for reading 'fgets' and 'fread' instead of making system call to feta
ch data immediatley
    // it reads the block of data and requested data return to the program and p
rocess so on


    // there are 3 FD
    //     1. STDIN_FILENO (0){stdin} - take input from keyboard or pipe etc
    //     2. STDOUT_FILENO (1){stdout} -  write o/p
    //     3. STDERR_FILENO (2){stderr} - for error messages writing

    // read is LL function used read from FD such as file,pipe. it is unbuffered
    // on success return the no of bytes it reads.
    printf("Write Something \n");
    ssize_t bytesReadSize = read(STDIN_FILENO, maxBufferSpace, sizeof(maxBufferS
pace));

    // Writing with
    // write(fd,buff[],size);
    // this method is used to write in a file using FD
    // 3rd arg is provided to specify the length of data we are going to write
    // on success return the no of bytes it actually returned
    // it is block I/O - it block the program untill it the data is written to t
he file or device
    // handle partial writes  - if it block by system then check if no bytes  ar
e fewer then requested then just retry
    write(STDOUT_FILENO, maxBufferSpace, bytesReadSize);


    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 7. Write a program to copy file1 into file2 ($cp file1 file2).
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_BUF_SIZE 10024

int main() {
    char sourcefileName[250];
    printf("Enter Source File name \n");
    if(fgets(sourcefileName,sizeof(sourcefileName),stdin) == NULL) {
        perror("Error reading input");
        return 1;
    }
    sourcefileName[strcspn(sourcefileName,"\n")] = '\0';
    char destfileName[250];
    printf("Enter Destination File name \n");
    fgets(destfileName,sizeof(destfileName),stdin);
    destfileName[strcspn(destfileName,"\n")]='\0';
    int sourceFileDesp = open(sourcefileName, O_RDONLY);
    if (sourceFileDesp< 0) {
        perror("There is an error while openning a Source File:");
        return 1;
    }

    int destFileDesp = open(destfileName, O_WRONLY | O_CREAT | O_TRUNC, 0777);
    if (destFileDesp < 0) {
        perror("There is an error while openning a File");
        return 1;
    }


    char data[MAX_BUF_SIZE];
    ssize_t bytesReaded = read(sourceFileDesp, data, MAX_BUF_SIZE);
    if (write(destFileDesp, data, bytesReaded) != bytesReaded) {
            perror("There is an error while Writing the data to Destination file");
            return 1;
        }

    close(sourceFileDesp);
    close(destFileDesp);
    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 8. Write a program to open a file in read only mode, read line by line and di
splay each line as it is read.
// Close the file when end of file is reached
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

#define MAX_BUFFER_SIZE 10240

int main() {
    // give a name for a file to point
    // and open it with O_RDONLY access
    // #####################
    // a note for TAs please change the file name in your pc if your are checkin
g
    int fd = open("Q7Sfile.txt", O_RDONLY);
    // if fd is less than 0 that means there is an error
    if (fd < 0) {
        perror("There is an Error  while opening  the file");
        return 1;
    }
    // declaring the array of char to store tthe char byte by byte
    char buffer[MAX_BUFFER_SIZE];
    // declaring ssize_t varibale to track record the no of with to read or read
ed from file
    ssize_t bytesRead, i = 0;
    // declaring the line valriable and initilizing to 1
    int line =1;
    // now reading the while using read fucntion
    // here we  passed the FD and buffer and buffer size
    while((bytesRead = read(fd, buffer, MAX_BUFFER_SIZE))>0){
    // itareting through byte by byte or char by char to check a new line char "
\n"
        for (ssize_t j = 0; j < bytesRead; ++j) {
            // if we encounter the new line char just enter in it
            if (buffer[j]== '\n') {
                // calculate the lenght from which index to which we have to rea
d
                int len = j-i+1;
                // now print the line with line no
                // and print the string using char[] by defining the line size
                // for 1st it will run from 0 to the first new line character' \
n'
                // then it increament the i index from 0 to j+1 for next line an
d so on
                printf("%d : %.*s\n",line++,len,buffer+i);
                char ch;
                ch = getchar();
                // incrementing the i index to start of new linel
                i = j+1;
            }
            // because after some time i become greater than j
            // i=0;
        }
        }
    // of this varible is less then it means there is an error while reading
        if (bytesRead < 0) {
        perror("Please check again there is an Error while reading this file");
```

```
        // closing the FD and exiting with code 0
        close(fd);
        // return error code
        return 1;
    }
    // close fd
    close(fd);
    // return success code
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 9. Write a program to print the following information about a given file.
// a. inode
// b. number of hard links
// c. uid
// d. gid
// e. size
// f. block size
// g. number of blocks
// h. time of last access
// i. time of last modification
// j. time of last change

#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char fileNameAndPath[250];
    printf("Enter the file name \n");
    if (fgets(fileNameAndPath, sizeof(fileNameAndPath), stdin) == NULL) {
        perror("Error reading input");
        return 1;
    }
        // Removeing the newline character

    fileNameAndPath[strcspn(fileNameAndPath, "\n")] = '\0';
    // file metadata are stored in a structure of type  below defined 'struct st
at'
    struct stat fileStat;

    // Get file status
    if (stat(fileNameAndPath, &fileStat) < 0) {
        perror("There is an Error while getting the file status");
        return 1;
    }

    // Print file information
    printf("1 .File Name is : %s\n", fileNameAndPath);
    printf("2. Inode number : %ld\n", (long)fileStat.st_ino);
    printf("3. Number of hard links of giver file is : %ld\n", (long)fileStat.st_nlink);
    printf("4. User ID (UID) : %d\n", fileStat.st_uid);
    printf("5. Group ID (GID) : %d\n", fileStat.st_gid);
    printf("6. Size of the File in Bytes: %ld bytes\n", (long)fileStat.st_size);
    printf("7. Total Block size: %ld bytes\n", (long)fileStat.st_blksize);
    printf("8. Total Number of blocks: %ld\n", (long)fileStat.st_blocks);
    // ctime() function is used to convert the time to a human readable format
    printf("9. Last accessed at: %s", ctime(&fileStat.st_atime));
    printf("10. Last modified at : %s", ctime(&fileStat.st_mtime));
    printf("11. Last changed at: %s", ctime(&fileStat.st_ctime));

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 10. Write a program to open a file with read write mode, write 10 bytes, move
 the file pointer by 10
// bytes (use lseek) and write again 10 bytes.
// a. check the return value of lseek
// b. open the file with od and check the empty spaces in between the data.

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

int main() {
    int fd;
    //ssize_t is signed data type used to track record
    //number of bytes written and readed by system call "read()" or "write()"
    ssize_t bytes_written;
    // off_t is data type used for file sie and offset
    off_t offset;

    const char *data1 = "TEMPDATA01";  // 10 bytes of data we will add first
    const char *data2 = "TEMPDATA02";  // 10 bytes of data we will add later
    char fileName[250];
    printf("Enter the file name \n");
    if(fgets(fileName,sizeof(fileName),stdin)== NULL) {
        perror("Error  while reading input");
        return 1;
    }
    // removing the newline character
    fileName[strcspn(fileName ,"\n")] = '\0';
    // Open the file in read and write mode
    // or  create  new file if it does not exist
    // and truncate to 0 length
    fd = open(fileName, O_RDWR | O_CREAT | O_TRUNC, 0777);
    if (fd < 0) {
        perror("Error while opening the file");
        return 1;
    }

    const char *tempData = "----------------------------------------------------
-------";
    // writing temp data in so while using lseek no error occur in file
    bytes_written = write(fd, tempData, strlen(tempData));
    if (bytes_written < 0) {
        perror("Error while writing Temp data to the file");
        close(fd);
        return 1;
    }

    // now just setting the cuurent point to middle of data
    // otherwise if i directly use lseek on empty position
    // it will print unknown char in between that causes file error
    offset = lseek(fd, -30, SEEK_CUR);
    if (offset == (off_t)-1) {
        perror("There is an Error while using lseek");
        // close FD and exit with failure code
        close(fd);
        return 1;
    }
```

```c
    // Now write the first 10 bytes to the new file
    // fd - targeted file
    // data1 - buffer data we are going to write
    // 10 - size of bytes count
    bytes_written = write(fd, data1, 10);
    if (bytes_written < 0) {
        perror("Error writing to the file");
        close(fd);
        return 1;
    }

    // Move the file pointer by 10 bytes using lseek()
    // lseek() uses whence arg to indicate how the offset argu should be interpr
eted
    //  when changing the file pointer position
    // 1. SEEK_SET - offset is set relative to starting of file
    //    lseek(fd,10,SEEK_SET) - move file pointer to 10 bytes from start of th
e file
    // 2. SEEK_CUR - offset is applied to current file position
    //    offset can be mode forward or backward from current position
    //    lseek(fd, -5, SEEK_CUR) // move file pointer 5 bytes backward from cur
rent position
    // 3. SEEK_END - offset is applied to end of the file. file pointer is moved
 to a position
    //    'offset' bytes from the end of file . it often used for appendinf data
 to a file
    //    lseek(fd,0,SEEK_END) // file pointer is moved to end of the file


    // here we are just moving the file pointer to 10th position from start
    offset = lseek(fd, 10, SEEK_CUR);
    if (offset == (off_t)-1) {
        perror("There is an Error while using lseek");
        // close FD and exit with failure code
        close(fd);
        return 1;
    }
    // it will print the new position of offset
    printf("lseek() returned the new position  : %ld\n", (long)offset);

    // now again write the new 10 bytes in files
    // on success it will return the no bytes it writes
    // failure it will return -1
    bytes_written = write(fd, data2, 10);
    if (bytes_written < 0) {
        perror("Error writing to the file");

        // error caused so just close the FD and
        //  close the program with exit code 1(error)
        close(fd);
        return 1;
    }

    // Close the file
    close(fd);

    // Run the 'od' command to check the file contents
    char cmd[300];
    snprintf(cmd, sizeof(cmd), "od -c %s", fileName);
    printf("File Data:\n");

    return 0;
```

```
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 11. Write a program to open a file, duplicate the file descriptor and append
the file with both the
// descriptors and check whether the file is updated properly or not.
// a. use dup
// b. use dup2
// c. use fcntl
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>


int main() {
    // Defining the name and address of the file
    char fileAdd[]="testfile_Q11.txt";

    // open() function needed file name and address
    // access control flag
    // O_WRONLY = open file for write only access
    // O_APPEND = open in append mode only - data is added at the end of the fil
e
    // O_CREAT =  creat file if not exist -
    // O_EXCL =  fail to create a file if already exist
    // O_RDONLY = open in only read mode
    // O_RDWR =  for reading and writing both
    // O_TRUNC = if file is there then open it and truncate the lenght to 0
    // O_NONBLOCK = open in non-blocking mode - used with pipes and device file.
    // O_SYNC = write to files are syncronised- write phycisally written to disk
 before write call returns
    // O_DSYNC = file data is syncronized not metadata. focus only writing actua
l fata
    // O_NOFOLLOW = do not follow symbolic links
    // O_CLOEXEC = close file descriptor when executing new program - it ensure
FD is not inherited by child process
    // O_TMPFILE = create a temp file that do not linked to any directory - it d
elete automatically
    // O_DIRECTORY = open only if path refer to directory
    // O_NOCTTY = if file is terminal device do not make it the contronlling ter
minal for process
    // , 777,555,456 these are the permissions
    // open() takes address, controll flag and permission
    // and return File descriptor
    // file descriptor is a non negative int value refer to the
    // open file of an running process
    // normally there are below File Descriptors by defaults
    // these three standard file descriptors that are automatically opened for e
ach process
        // 0: Standard Input (stdin) âM-^@M-^S Used for reading input.
        // 1: Standard Output (stdout) âM-^@M-^S Used for writing output.
        // 2: Standard Error (stderr) âM-^@M-^S Used for writing error messages.
    // now after opening new file fd will point to 3 and 4 and so on .....
    int fd = open( fileAdd,O_WRONLY | O_CREAT | O_TRUNC, 0644);
    // if fd is smaller than 0 means file is not able to open
    //perror() print error message related to system call and library func
    //exit() terminate program and provide an exit status
    //   1.EXIT_FAILURE - internally value is 1 (non zero)
    //   2.EXIT_SUCCESS - represent 0 - means success
    if (fd < 0) { perror("open"); exit(EXIT_FAILURE); }
```

```c
    // dup(fd) - LL file management - provide ability to create multiple referen
ce to same file or resource
    // dup2(fd,10) - unlike dup which return smallest available FD and
    // dup2 allow you to specify the exact file descriptor number you want to us
e
    // on faliure return -1 otherwise return new fd which is pointing to same FD
 as oldFD
    int fd_dup = dup(fd);
    int fd_dup2 = dup2(fd, 10);
    // fcntl( fd cmd arg)
    // it is used to provide various control operations on FD, duplicating FD,Ch
anging file status flags, and obtaining info about FD
    // there are some comman cmd
    // 1. F_DUPED - FD ki smallesst available FD number jo equall or greater ho
specified value ka usko duplicate karo
    // 2. F_DUPFD_CLOEXEC - same as above and also FD_CLOEXEC flag set
    // 3. F_GETFD - get FD flags
    // 4. F_SETFD - set FD flag
    // 5. F_GETFL - get file status flag (O_RDONLY,O_WRONLY)
    // 6. F_SETFL - set file status
    // 7. F_GETLK - get info about file lock
    // 8. F_SETLK - set or remove file lock
    // 9. F_SETLKW - set or remove file lock or wait if necessary
    // 10. F_GETOWN - get procrss id or procress group ID that will receice SIGU
RG signals
    // 11. F_SETOWN -  set process or Process group ID
    int fd_fcntl = fcntl(fd, F_DUPFD, 0);

    // Writing with different descriptors
    // write(fd,buff[],size);
    // this method is used to write in a file using FD
    // and 3rd arg is provided to specify the length og data we are going to wri
te

    write(fd, "Original data\n", 14);
    write(fd_dup, "dup data \n", 10);
    write(fd_dup2, "dup2 data\n", 10);
    write(fd_fcntl, "fcntl data\n", 11);
    // it is used to close the FD and release all the lock and resource which be
ing used by FD
    // on sucecss return 0 else 1
    close(fd); close(fd_dup); close(fd_dup2); close(fd_fcntl);

    // Check the content of the file
    // again opening file with readonly flag and default permissions and if file
 exist then update  fd
    fd = open(fileAdd, O_RDONLY);
    // declare buffer to store the data
    char buffer[1024];
    // ssize_t - it is a signet integer which repreasent the size of block or th
e number of bytes reads or writter
    // here we are reading data from fd and copies to buffer
    //  buffer is block of memory that we have allotted but
    // after reading fd we need to add null pointer manually to the end
    // we specify -1 in the end to store null pointer
    // last arg of read() tells us how many maximum bytes to read
    ssize_t bytes_read = read(fd, buffer, sizeof(buffer) - 1);
    // if ssize_t return less than 0 then it means there is an error
    // close fd and exit the program  with exit code
    if (bytes_read < 0) { perror("read"); close(fd); exit(EXIT_FAILURE); }
    // set the null pointer to the end wafter how muhc bytes we have been read
    buffer[bytes_read] = '\0';
    // just print it
    printf("File content:\n%s", buffer);
    // close fd
    close(fd);
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 12. Write a program to find out the opening mode of a file. Use fcntl.
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

void printFileMode(int flags) {
    // Check the access mode (O_RDONLY, O_WRONLY, O_RDWR)
    // O_ACCMODE is a bitmask , we use it to exrtact the access mode from flags
    // O_ACCMODE is only mask out and extract accessmode (read ,write , read/wri
tes)
    // for other flags we need to AND them with flag
    int accessMode = flags & O_ACCMODE;
    if (accessMode == O_RDONLY) {
        printf("O_RDONLY Flag : File is opened in read−only mode.\n");
    }
    else if (accessMode == O_WRONLY) {
        printf("O_WRONLY FLAG : File is opened in write−only mode.\n");
    }
    else if (accessMode == O_RDWR) {
        printf("O_RDWR Flag : File is opened in read−write mode.\n");
    }
    else {
        printf("Unknown access mode.\n");
    }

    // printf("%d \n",flags);
    //  printf("%d \n",O_APPEND);


    // Check additional flags
    // here other flags are needed to be AND with FLAGs(O_APPEND etc) to check
    // these flas are appended to the last are not the part of access mode of fi
le
    if (flags & O_APPEND) {
        printf("O_APPEND is set: Writes will append to the end of the file.\n");
    }
    if (flags & O_NONBLOCK) {
        printf("O_NONBLOCK is set: Non−blocking mode is enabled.\n");
    }
    if (flags & O_SYNC) {
        printf("O_SYNC is set: Writes are synchronized.\n");
    }
}

int main() {
    char fileAdd[]="Question13.c";
    // Open a file with specific flags
    // open file with write and read only and append flag
    // and adding O_CREAT flag if file not exist
    int fd = open(fileAdd, O_WRONLY | O_APPEND| O_CREAT, 0777);
    // if file not open then exit program
    if (fd < 0) {
        perror("Error while opening file");
        return 1;
    }

    // Use fcntl to get the file status flags
    // when we pass FD and F_GETFL parameter
    // fcntal return us the flag is currently is being used by FD
    int flags = fcntl(fd, F_GETFL);
```

```c
    // report error if any error found
    if (flags < 0) {
        perror("Error getting file flags");
        close(fd);
        return 1;
    }


    printFileMode(flags);

    // Close the file descriptor
    close(fd);
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 13. Write a program to wait for a STDIN for 10 seconds using select. Write a
proper print statement to
// verify whether the data is available within 10 seconds or not (check in $man
2 select).

#include <stdio.h>
#include <sys/select.h>
#include <unistd.h>

int main() {
    //it is datatype used to represent a set of FD
    // it allow use to track a FD to monitor for an activity. FD ie - such as so
cket,pipes or standard input/output
    fd_set readfds;
    // it is used to represent time intervel
    // first value define seconds and second value define microsecond
    // here {10,0} it is 10 sec and 0 ms
    struct timeval timeout = {10, 0};  // 10 seconds timeout

    // this function used to clear the fd_set and reset it to empty
    FD_ZERO(&readfds);
    //using this function will set fd to fd_set
    // there are 3 FD
    //     1. STDIN_FILENO (0){stdin} - take input from keyboard or pipe etc
    //     2. STDOUT_FILENO (1){stdout} -  write o/p
    //     3. STDERR_FILENO (2){stderr} - for error messages writing
    // by setting sdtin to set now program will understood that we want to monit
er the input
    FD_SET(STDIN_FILENO, &readfds);


    printf("Waiting for input on STDIN for 10 seconds...\n");

    // select syntex are alike
    // select(nfds,readfd,writefd,exceptfd,timeinterval)
    // STDIN_FILENO - {select() requires the highest FD number +1 because FDs ar
e zero based indecies
    // like if want to moniter 0 index then we need to specify 1 as nfds }
    // timeout -  it will wait till the time we mentioned
    // if data available before time mentioned then it will return positive valu
e
    // if no data is given it return 0
    // any error occur  then it will return -1
    int retval = select(STDIN_FILENO + 1, &readfds, NULL, NULL, &timeout);

    if (retval == -1) {
        perror("select()");
    } else if (retval == 0) {
        printf("No data within 10 seconds.\n");
    } else {
        printf("Data is available on STDIN.\n");
    }

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 14 Write a program to find the type of a file.
// a. Input should be taken from command line.
// b. program should be able to identify any type of a file.
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
// Function to print the file type
void printFileType(struct stat mode) {
    if (S_ISREG(mode.st_mode)) {
        printf("Regular file\n");
    } else if (S_ISDIR(mode.st_mode)) {
        printf("Directory\n");
    } else if (S_ISLNK(mode.st_mode)) {
        printf("Symbolic link\n");
    } else if (S_ISCHR(mode.st_mode)) {
        printf("Character device\n");
    } else if (S_ISBLK(mode.st_mode)) {
        printf("Block device\n");
    } else if (S_ISFIFO(mode.st_mode)) {
        printf("FIFO/pipe\n");
    } else if (S_ISSOCK(mode.st_mode)) {
        printf("Socket\n");
    } else {
        printf("Unknown file type\n");
    }

}

int main() {

    printf("Enter the file name with path \nor if file is in same directory just enter the name of file. \nFile name s
hould be less than 255 char\n");
    char fileName[255];
    fgets(fileName,sizeof(fileName),stdin);
    fileName[strcspn(fileName,"\n")] = '\0';
    struct stat file_stat;

    // Get file status
    // stat() function is used to obtain info about a file.
    // it fills struct stat
    // return on success 0 else 1
    // internal structure of struct stat is look like this
    //      struct stat {
    //      dev_t     st_dev;     /* Device ID of the file */
    //      ino_t     st_ino;     /* Inode number */
    //      mode_t    st_mode;    /* File type and mode (permissions) */
    //      nlink_t   st_nlink;   /* Number of hard links */
    //      uid_t     st_uid;     /* User ID of the owner */
    //      gid_t     st_gid;     /* Group ID of the owner */
    //      dev_t     st_rdev;    /* Device ID (if file is character or block spe
cial) */
    //      off_t     st_size;    /* Total size of the file in bytes */
    //      blksize_t st_blksize; /* Block size for filesystem I/O */
    //      blkcnt_t  st_blocks;  /* Number of 512-byte blocks allocated */
    //      time_t    st_atime;   /* Time of last access */
    //      time_t    st_mtime;   /* Time of last modification */
    //      time_t    st_ctime;   /* Time of last status change */
    // };
```

```c
    if (stat(fileName, &file_stat) != 0) {
        perror("stat");
        return 1;
    }

    // Print file type
    printFileType(file_stat);

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 15. Write a program to display the environmental variable of the user (use en
viron).
#include <stdio.h>


// environ is a global variable provided by the system
// it points to an array of string
// extern keyword used to declare the variable without defining it
// it indicate that 'environ' is a global variable defined in another source fil
e
// so store the reference of that file in our file
// it will link it during compile time
extern char **environ; // Declare the external variable environ

int main() {

    printf("Environmental Variables:\n");
    // declaring and initilising a Pointer to the environment variable list
    // Iterate through the environment variable list using for loop
    // Print each environment variable
    // Move to the next environment variable till it is available
    for (char **env = environ; *env; ++env) {
        printf("%s\n", *env);
    }

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 6. Write a program to perform mandatory locking.
// a. Implement write lock
// b. Implement read lock

#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void releaselock(int fd,off_t offset, size_t size){
    struct flock lock;
    lock.l_type=F_UNLCK;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLK,&lock);

}
void setlock(int fd, int type,off_t offset,size_t size){
    struct flock lock;
    lock.l_type= type;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLKW,&lock);
}
void writelock(int fd,off_t offset,size_t size){
    setlock(fd,F_WRLCK,offset,size);
    printf("file is locked in write mode\n");
    char *data = "THE DATA IS WRITTEN BY WRITELOCK FUNCTION DEFINED BY ABHAY";
    printf("Press enter to write ");
    getchar();
    getchar();
    write(fd,data,strlen(data));
    printf("Write operation done. Releasing the lock");
    releaselock(fd,offset,size);
}
void readlock(int fd,off_t offset,size_t size){
     setlock(fd,F_RDLCK,offset,size);
     printf("File is locked in Reading mode \n");
     char buff[500];
     printf("Hit enter to read the file \n");
     getchar();
     getchar();
     lseek(fd,0,SEEK_SET);
     ssize_t bytes_read= read(fd,buff,sizeof(buff)-1);
     buff[bytes_read]='\0';
     printf("Data Read from file : \n\n%s \n",buff);

     printf("Now Releasing the lock\n");

    releaselock(fd,offset,size);
}
int main(){
    int fd = open("./Question16.txt",O_RDWR,0666);
    if(fd<0) {
        perror("open");
        exit(1);// fail to open
```

```c
    }
    int choice;
    printf("Enter your choice\n1).Write lock\n2).Read lock");
    scanf("%d",&choice);
    switch (choice)
    {
    case 1: writelock(fd,100,0); break;
    case 2: readlock(fd,100,0); break;
    default: break;
    }

    close(fd);
    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
//17. Write a program to simulate online ticket reservation. Implement write loc
k
// Write a program to open a file, store a ticket number and exit. Write a separ
ate program, to
// open the file, implement write lock, read the ticket number, increment the nu
mber and print
// the new ticket number then close the file.

#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void releaselock(int fd,off_t offset, size_t size){
    struct flock lock;
    lock.l_type=F_UNLCK;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLK,&lock);

}
void setlock(int fd, int type,off_t offset,size_t size){
    struct flock lock;
    lock.l_type= type;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLKW,&lock);
}
void writelock(int fd,off_t offset,size_t size){
    setlock(fd,F_WRLCK,offset,size);
    char buff[250];
    ssize_t bytesReaded =  read(fd,buff,sizeof(buff)-1);
    // if data present the truncate the file
    ftruncate(fd,0);
    lseek(fd,0,SEEK_SET);
    snprintf(buff,sizeof(buff),"%d",0);
    write(fd,buff,strlen(buff));
    printf("Ticket Number is stored to 0.\n");
    releaselock(fd,offset,size);
}
int main(){
    int fd = open("./Question17.txt",O_RDWR,0666);
    if(fd<0) {
        perror("open");
        exit(1);// fail to open
    }
    int choice;
    writelock(fd,100,0);
    close(fd);
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
//17. Write a program to simulate online ticket reservation. Implement write loc
k
// Write a program to open a file, store a ticket number and exit. Write a separ
ate program, to
// open the file, implement write lock, read the ticket number, increment the nu
mber and print
// the new ticket number then close the file.

#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void releaselock(int fd,off_t offset, size_t size){
    struct flock lock;
    lock.l_type=F_UNLCK;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLK,&lock);

}
void setlock(int fd, int type,off_t offset,size_t size){
    struct flock lock;
    lock.l_type= type;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLKW,&lock);
}
void writelock(int fd,off_t offset,size_t size){
    setlock(fd,F_WRLCK,offset,size);
    printf("Locking the record file to book a ticket\n");
    char buff[250];
    ssize_t bytesReaded =  read(fd,buff,sizeof(buff)-1);
    buff[bytesReaded]='\0';
    int number = atoi(buff);
    printf("Current no of Ticket Booked : %d \n",number);
    printf("Press enter to Book Your ticket \n");
    getchar();
    getchar();
    number++;
    lseek(fd,0,SEEK_SET);
    snprintf(buff,sizeof(buff),"%d",number);
    write(fd,buff,strlen(buff));
    printf("Ticket is Booked.\nYour Ticket no : %d\n Releasing the lock",number);
    releaselock(fd,offset,size);
}
void readlock(int fd,off_t offset,size_t size){
     setlock(fd,F_RDLCK,offset,size);
     printf("Ticket record are locked in Reading mode ,\n");
     char buff[500];
     printf("Hit enter to get current no of ticket booked \n");
     getchar();
     getchar();
     lseek(fd,0,SEEK_SET);
     ssize_t bytes_read= read(fd,buff,sizeof(buff)-1);
```

```c
     buff[bytes_read]='\0';
     printf("Ticket booked : \n\n%s \n",buff);
     printf("Now Releasing the lock\n");

    releaselock(fd,offset,size);
}
int main(){
    int fd = open("./Question17.txt",O_RDWR,0666);
    if(fd<0) {
        perror("open");
        exit(1);// fail to open
    }
    int choice;
    printf("Enter your choice\n1). Book Ticket (According to Question Press 1)\n2). Read Total Ticket Booked(J
ust to lookat total Tickets)\n");
    scanf("%d",&choice);
    switch (choice)
    {
    case 1: writelock(fd,100,0); break;
    case 2: readlock(fd,100,0); break;
    default: break;
    }

    close(fd);
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 18. Write a program to perform Record locking.
// a. Implement write lock
// b. Implement read lock
// Create three records in a file. Whenever you access a particular record, firs
t lock it then modify/access
// to avoid race condition


#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define FILE_PATH "./Question18.txt"
#define RECORD_SIZE 16

int max(int a, int b) {
    return (a > b) ? a : b;
}
void increment_and_format(char *str) {
    int number = atoi(str);
    number++;
    snprintf(str, RECORD_SIZE-1, "%d", number);
    int len = strlen(str);
    int zeros_to_add = 15 - len;
    if (zeros_to_add > 0) {
        memmove(str + zeros_to_add, str, len + 1);
        memset(str, '0', zeros_to_add);
    }
}
void releaselock(int fd,off_t offset, size_t size){
    struct flock lock;
    lock.l_type=F_UNLCK;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLK,&lock);

}
void setlock(int fd, int type,off_t offset,size_t size){
    struct flock lock;
    lock.l_type= type;
    lock.l_whence=SEEK_SET;
    lock.l_start=offset;
    lock.l_len = size;
    fcntl(fd,F_SETLKW,&lock);
}
void createNewArr(char *str,char *src,int from , int end){
    int j=0;
     for(int i=from;i<end;i++){
        str[j++]=src[i];
     }
}
void writelock(int fd,int rNo){
    off_t offset = rNo*(RECORD_SIZE-1);
    size_t size = RECORD_SIZE-1;
    setlock(fd,F_WRLCK,offset,size);
```

```c
    printf("Locking the record file %d\n",rNo);
    char buff[rNo*RECORD_SIZE];
    char newBuff[RECORD_SIZE];
    ssize_t bytesReaded =  read(fd,buff,sizeof(buff)-1);
    if (bytesReaded == 0) {
        printf("Record is empty. Initializing with 0.\n");
        memset(buff,'0',sizeof(buff));
    }else{
     createNewArr(newBuff,buff,max(0,((rNo-1)*RECORD_SIZE-(rNo-2))),(RECORD_SIZE
*rNo-rNo));
    }
    newBuff[RECORD_SIZE]='\0';
    int number = atoi(newBuff);
    printf("Current entry in Record no : %d  is  : %d\n",rNo,number);
    printf("Press enter to update the Record no :%d \n",rNo);
    getchar();
    getchar();
    number++;
    increment_and_format(newBuff);
    if(bytesReaded>0) lseek(fd,max(0,((rNo-1)*RECORD_SIZE-(rNo-1))),SEEK_SET);
    else   lseek(fd,0,SEEK_SET);
    write(fd,newBuff,strlen(newBuff));
    printf("Your Record is updated: %d\nReleasing the lock on Record no : %d\n",number,rNo);
    releaselock(fd,offset,size);
}
void readlock(int fd,int rNo){
    off_t offset = (rNo-1)*(RECORD_SIZE-1);
    size_t size = RECORD_SIZE-1;
     setlock(fd,F_RDLCK,offset,size);
     printf("Locking the record file  : %d in reading mode\n",rNo);
     char buff[rNo*RECORD_SIZE];
    char newBuff[RECORD_SIZE];
     printf("Hit Enter to get the data of current record \n");
     getchar();
     getchar();
     lseek(fd,offset,SEEK_SET);
     ssize_t bytes_read= read(fd,buff,sizeof(buff)-1);

     buff[RECORD_SIZE-1]='\0';
     printf("Data of current Record : %d is : %d\n",rNo,atoi(buff));
     printf("Now Releasing the lock on %d Record\n",rNo);

    releaselock(fd,offset,size);
}

int selectRecord(){
    int select;
    printf("Enter the record no you want to go with\n1\n2\n3\n");
    scanf("%d",&select);
    return select;
}
int main(){
    int fd = open(FILE_PATH,O_RDWR,0666);
    if(fd<0) {
        perror("open");
        exit(1);// fail to open
    }
    int choice;
    printf("Enter your choice\n1). Write Mode\n2). Read Mode\n");
    scanf("%d",&choice);
    switch (choice)
    {
```

```
    case 1: writelock(fd,selectRecord()); break;
    case 2: readlock(fd,selectRecord()); break;
    default: break;
    }

    close(fd);
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 19. Write a program to find out time taken to execute getpid system call. Use
 time stamp counter.
#include <stdio.h>
#include <unistd.h>  // For getpid()

// this function will read CPU time stamp counter - this counter is a 64 bit reg
ister
// that increment with each clock cycle - aka timestamp counter (TSC).
// it count the #cpu cycle since last reset.
// Location :- hardware -> TSC is located within each CPU core.
//                         it is a part of cpu control and status register (CSRs
)
//                         it is accessed via "rdtsc" (Read time Stamp counter)
assembly instruction
//             Accessing : 1) using "RDTSC" instruction x86 assembly language
//                           this instruction loads the current value of TSC int
o EDX:EAX pair of integer in 64 bit mode
//                         2) in C instruction is accessed through inline assemb
ly,
unsigned long long rdtsc(){
    // TSC give us 64 bit code but unsigend int can only hold 32 bit. so it will
 store in 2 variable
    unsigned int lo, hi;
    // __asm__ it used to insert the assembly language code directlr within the
c program aka- inline assembly
    // __volatile__ it prevents the compiler ot to optimize away this assembly c
ode, ensuring that it is executed exactly as written.
    //          and  the value of the variable will not change unexpectedly.
    // "rdtsc" - it reads the current value of TSC into two register
    //            1. EAX -  will hold lower 32 bit of TSC
    //            2. EDX - hold upper 32 bith
    //        =a,=d means the value present in EAX and EDX store in lo an hi
    //      now shifting hi value by 32 bit so making it as upper hafl
    //       and using | 'or' operator combine both lo and hi
    __asm__ __volatile__ (
        "rdtsc"
        : "=a" (lo), "=d" (hi)
    );
    return ((unsigned long long)hi << 32) | lo;
}

int main() {
    unsigned long long start, end;

    // get the time before calling getpid() function
    start = rdtsc();

    // Execute the getpid() system call
    getpid();

    // Get time time after calling getpid() function
    end = rdtsc();

    // now calculate the diffence between start and end, and this is the answer
    printf("getpid() takes : %llu cycles\n", end - start);

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
//20. Find out the priority of your running program.
//    Modify the priority with nice command.

#include <stdio.h>
#include <unistd.h>
int main(){

    while(1){
        // now this will run infinatly

        sleep(1);
    }
    return 0;
}
// just run this program either back or foreground

// now to check the priority of our running program
// $ps -l -p <pid>
// entter this and it will return us below data
//  │  abhay@abhay-pc:~/Desktop/handsOnList$ ps -l -p 12689
//  │ F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
//  │ 0 S  1000   12689    9069  0  80   0 -   638 hrtime pts/2    00:00:00 a.ou
t

// PRI - PRiority
// NI : nice value  (default val is 0)

// now stop program and change the nice value
// nice value change on starting
// use 'nice' command

// $ nice -n 10 ./prog_name

// if you want to change the nice value while running
// use 'renice' command

// renice -n 25 -p <pid>
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 21. Write a program, call fork and print the parent and child process id.
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    // Create a new process
    int pid = fork();

    if (pid < 0){
        printf("Fork is failed to create\n");
        return 1;
    } else if (pid == 0) {
        // Child process created
        printf("Child process. Process ID: %d\n", getpid());
        printf("Parent Process ID: %d\n", getppid());
    } else {
        // Parent process
        printf("This is the parent process. Process ID: %d\n", getpid());
        printf("Child Process ID: %d\n", pid);
        wait(NULL);
    }

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 22. Write a program, open a file, call fork,
// and then write to the file by both the child as well as the
// parent processes. Check output of the file.

#include <stdio.h>
#include <unistd.h>  // For fork() and getpid()
#include <fcntl.h>   // For open()
#include <sys/types.h> // For pid_t
#include <sys/wait.h>  // For wait()
#include <string.h>  // For strlen()
int main() {
    int fd;
    int pid;
    char fileName[250];
    char dataToWrite[1000];
    printf("Enter the filename \n");
    fgets(fileName,sizeof(fileName),stdin);
    // Open a file, if it is not available then create new file
    fd = open(fileName, O_WRONLY | O_CREAT | O_TRUNC, 0777);
    if (fd < 0) {
        perror("there is an error while opening file");
        return 1;
    }

    // Create a child process
    pid = fork();

    if (pid < 0) {
        printf("Fork not created");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("enter a message for child process \n");
        fgets(dataToWrite,sizeof(dataToWrite),stdin);
        write(fd, dataToWrite, strlen(dataToWrite));
    } else {
         // Wait for the child process to finish
           wait(NULL);
        // Parent process
        printf("enter a message for Parent Process \n");
        fgets(dataToWrite,sizeof(dataToWrite),stdin);
        write(fd, dataToWrite, strlen(dataToWrite));



    }

    // Close the file
    close(fd);

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 23. Write a program to create a Zombie state of the running program.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>  // For wait()

int main() {
    pid_t pid = fork();  // Create a new process

    if (pid < 0) {
        // Fork failed
        perror("There is an error in fork – process is not created");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Child process  with process ID: %d is completed now exiting.\n", getpid());
        exit(0);  // Exit immediately, becoming a zombie
    } else {
        // Parent process
        printf("Parent process with Process ID: %d is running will now sleep.\n", getpid());
        // Sleep for 30 seconds to check the zombie state in terminal
        // run this process FG or BG use below command to check zombie process
        // $ps aux | grep Z
        // here parent is sleeping for 30 sec without calling wait()
        // so child become zombie because it is exited but still not yet acknowl
edged by parent
        sleep(30);


        printf("Parent process with process ID: %d is exiting.\n", getpid());
    }

    return 0;
}

// zombie process is a process that is completed but still has an entry in proce
ss table.
// this happen due to parent process did not use 'wait()' or 'waitpid()' functio
n to check exit status of child
//
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 24. Write a program to create an orphan process.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int pid = fork();  // creating a child process using fork()

    if (pid < 0) {
        printf("There is an error while creating process using fork");
        return 1;
    } else if (pid == 0) {
        // Child
        // here we set our child sleep for few secs
        // then let parent process complete its executation
        sleep(10);
        printf("Child process with process ID: %d  is now became an orphan.\nAs parent process is exited.\n"
, getpid());
        // after parent process exited. the init process takeover the orphan and
 let child compele its execution
        // here in linux-Ubuntu 24.04 LTS orphan is handled by systemd
        // please check the to check systemd process
        // pidof systemd
        printf("Parent process ID after child become orphan : %d\n", getppid());
    } else {
        // Parent
        // set parent process to sleep so our child can enter and get sleep
        // as soon child process set to sleep the parent start its execution jsu
t after 2 sec
        // and prints it process id and exit from execution leaving child proces
s in sleeping state
        sleep(2);
        printf("Parent process  with process ID: %d is now exiting.\n", getpid());
        exit(0);
    }

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 25. Write a program to create three child processes. The parent should wait f
or a particular child (use
// waitpid system call)

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>  // For waitpid()

int main() {
    int child1, child2, child3;

    // Create three child processes
    // child1 = fork();
    // child2 = fork();
    // child3 = fork();
    if ((child1 = fork()) == 0) {
        printf("Child 1 is running : %d\n", getpid());
        exit(0);
    }

   if ((child2 = fork()) == 0) {
        printf("Child 2 is running : %d\n", getpid());
        exit(0);
    }

    if ((child3 = fork()) == 0) {
        printf("Child 3  is running : %d\n", getpid());
        exit(0);
    }

    //  waitpid will wait for particular process
    //  and setting 0 means it will block the process untill its
    //  targeted child is executed successfully and changes its state
    //  &status varibale hold the return status of child
    waitpid(child2, NULL, 0);
    printf("Child %d exited\n", child2);

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 26. Write a program to execute an executable program.
// a. use some executable program
// b. pass some input to an executable program. (for example execute an executab
le of $./a.out name)
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc > 1) {
        printf("Argument passed by you : %s!\n", argv[1]);
    } else {
        printf("No argument is passed!\n");
    }
    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 26. Write a program to execute an executable program.
// a. use some executable program
// b. pass some input to an executable program. (for example execute an executab
le of $./a.out name)

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {


    int pid1, pid2;
    int status;

    // i need to create 2 different child
    // to run execl because it will replace the current process image with new i
mage
    // so we will just create 2 new child processes to do this work
    if ((pid1 = fork())== 0) {
        // Child process 1
        //in execl() 1st argument is path to file
        //           2nd argument is name of file
        //           3rd argument is argument which is needed to be passed
        printf("Running program without arguments\n");
        execl("./Question26a.out", "./Question26a.out", NULL);
    }

    if ((pid2 = fork()) == 0) {
        // Child process 2
        sleep(1);
    printf("Running program with arguments\n");
    execl("./Question26a.out", "./Question26a.out", "Abhay", NULL);
    }
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);
    printf("Both child processes have finished.\nAnd executed both program with or without argument.\n");
    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
//27. Write a program to execute ls -Rl by the following system calls
// a. execl
// b. execlp
// c. execle
// d. execv
// e. execvp


#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid;

    // 1. Using execl
    // Syntax: int execl(path, filename ..., NULL)
    //in execl() 1st argument is path to file
    //           2nd argument is name of file
    //           3rd argument is argument which is needed to be passed
    // in this we need to provide full executable path
    // it does not use the PATh enviromant variable to locate the executable
    // it replace the current process image with a new process image. the curren
t
    // process will completely replaced.
    printf("\n\n------------------------------------------------------------\n\n");
    printf("Executing excel() Function \n");
    printf("\n\n------------------------------------------------------------\n\n");
    if (fork() == 0)     execl("/bin/ls", "ls", "-Rl", NULL);
    else     wait(NULL); // Wait for the child process to finish


    // 2. Using execlp
    // int execlp(file, *arg0, ...,NULL)
    // it  searches file in the directory listed in the PATH envrionment variabl
e
    // argument list must be terminated by a 'NULL'
    // is also replaces process image
    printf("\n\n------------------------------------------------------------\n\n");
    printf("Executing execlp() Function \n");
    printf("\n\n------------------------------------------------------------\n\n");
    if (fork() == 0) execlp("ls", "ls", "-Rl", NULL);
    else wait(NULL); // Wait for the child process to finish


    // 3. Using execle
    // it is also same as execl but it allows specifying a custom
    // environment list for the new process. environment list is
    // passed as an array of strings, ending with a 'NULL' pointer
    // envp - array of environment variavles for the new process
    printf("\n\n------------------------------------------------------------\n\n");
       printf("Executing execle() Function \n");
       printf("\n\n------------------------------------------------------------\n\n");
    if (fork() == 0) {
        char *envp[] = {NULL}; // Passing an empty environment
        execle("/bin/ls", "ls", "-Rl", NULL, envp);
    } else wait(NULL); // Wait for the child process to finish

    // 4. Using execv
    // execv execute a program specified by path with arguments passed as an arr
```

```
ay of strings
    // the array must be terminated by a NULL pointer
    // path to executable file
    // argv - an array of string representing the command line argumets for exec
tubale
    // first element should be the name of the program itself
    printf("\n\n------------------------------------------------------------\n\n");
   printf("Executing execv() Function \n");
   printf("\n\n------------------------------------------------------------\n\n");
    if (fork() == 0) {
        char *args[] = {"ls", "-Rl", NULL};
        execv("/bin/ls", args);
    } else     wait(NULL); // Wait for the child process to finish


    // 5. Using execvp
    // it is similer to execv but it searches for the executbale file in the dir
ectories listed
    // in the 'PATH' envrionment variable
    // the argument are passed as an array of strings
    // the name of file to execute it searched in the directories  specified in
path
    // args should be in array of string
    printf("\n\n------------------------------------------------------------\n\n");
   printf("Executing execvp() Function \n");
   printf("\n\n------------------------------------------------------------\n\n");
    if (fork() == 0) {
        char *args[] = {"ls", "-Rl", NULL};
        execvp("ls", args);
    } else {
        wait(NULL); // Wait for the child process to finish
    }

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
//28. Write a program to get maximum and minimum real time priority.

#include <stdio.h>
#include <sched.h>
#include <unistd.h>
void printPri(char *string,int SCHED_POLICY){
printf("Max realtime priority of %s Scheduler is : %d\n",string, sched_get_priority_max(SCHED_
POLICY));
printf("Min realtime priority of %s Scheduler is : %d\n",string, sched_get_priority_min(SCHED_P
OLICY));
}

int main() {
    int pid;
    int policy;
    // getting the process ID of current process to retrive the SCHEDULING polic
y
    pid = getpid();  // Get current process ID
    // using below function by passing PID of current function we can get schedu
ling policy
    policy = sched_getscheduler(pid);
    // why iam doing this because OS uses multiple type of policies like below
    // SCHED_FIFO
    // SCHED_RR
    // SCHED_OTHER
    // so to get correct priory we need current scheduling algo
    printPri("SCHED_FIFO",SCHED_FIFO);
    printPri("SCHED_RR",SCHED_RR);
    printPri("SCHED_OTHER",SCHED_OTHER);

    return 0;
}
```

```
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 29. Write a program to get scheduling policy and modify the scheduling policy
 (SCHED_FIFO,SCHED_RR).



#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sched.h>
#include <errno.h>

void printCurrentPolicy(int policy) {
    switch (policy) {
        case SCHED_OTHER:
            printf("Current : SCHED_OTHER\n");
            break;
        case SCHED_FIFO:
            printf("Current : SCHED_FIFO \n");
            break;
        case SCHED_RR:
            printf("Current : SCHED_RR\n");
            break;
        default:
            printf("Unknown scheduling policy\n");
            break;
    }
}

int main() {
    int pid = getpid();
    struct sched_param param;
    printCurrentPolicy(sched_getscheduler(pid));
    param.sched_priority = 10;
    sched_setscheduler(pid, SCHED_FIFO, &param);
    printf("Scheduling policy changed to SCHED_FIFO with priority %d\n", param.sched_priority);
    printCurrentPolicy(sched_getscheduler(pid));
    param.sched_priority = 15;
    sched_setscheduler(pid, SCHED_RR, &param);
    printf("Scheduling policy changed to SCHED_RR with priority %d\n", param.sched_priority);
    printCurrentPolicy(sched_getscheduler(pid));

    return 0;
}
```

```c
// Roll no :- MT2024003
// Name :- Abhay Bhadouriya
// 30. Write a program to run a script at a specific time using a Daemon process
.


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

// script name which we are going to run
#define SCRIPT_PATH "./alert.sh"
// time in hour
#define TARGET_HOUR 23
// time in minute
#define TARGET_MINUTE 06

void createDaemonProcess() {
    pid_t pid;

    pid = fork();

    if (pid < 0) {
        // child not created
        exit(1);
    }

    if (pid > 0) {
        // terminate parent
        exit(0);
    }

    // Set the child process as the session leader
    if (setsid() < 0) {
        // exit if session is not created on child
        exit(1);
    }

    pid = fork();
    if (pid < 0) {
        exit(EXIT_FAILURE);
    }

    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    umask(0);
    chdir("/");

    // Close standard input, output, and error file descriptors
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);
}
```

```c
int main() {

    // run for infinite for checking if time is there
    int hour;
    int min;
    printf("Enter Hour in 24hour format \n");
    scanf("%d",&hour);
    printf("Enter minute \n");
    scanf("%d",&min);
     createDaemonProcess();
    while (1) {
        // getting time in UTC
        time_t now = time(NULL);
        // converting time to local machine
        struct tm *tm_info = localtime(&now);
        if (tm_info->tm_hour == hour && tm_info->tm_min == min) {
                // when time is same then just trigger the execl
                    system("/home/abhay/Desktop/handsOnList/alert.sh");
                    return 0;
        }
        sleep(5);
   }
    return 0;
}
```