**Experiment No 3**

Title: Implementation of various operations on Files

Aim: To study and implement various file related system calls along with their option flags.

Objective: To study and implement following

1. OPEN system call

2. CREAT system call

3. READ system call

4. WRITE system call

5. FSTAST system call

1. APPEND system call
2. DUP system call

Relevance: To get knowledge about file commands in Linux

Theory:

System calls are the calls that a program makes to the system kernel to provide the services to which the program does not have direct access. For example, providing access to input and output devices such as monitors and keyboards. We can use various functions provided in the C Programming language for input/output system calls such as create, open, read, write, etc.

**Important Terminology**

* **What is the File Descriptor?**

The file descriptor is an integer that uniquely identifies an open file of the process.

* **File Descriptor table: A file** descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries. One unique file descriptors table is provided in the operating system for each process.
* **File Table Entry:** File table entries are a structure In-memory surrogate for an open file, which is created when processing a request to open the file and these entries maintain file position.
* **Standard File Descriptors**: When any process starts, then that process file descriptors table’s fd(file descriptor) 0, 1, 2 open automatically, (By default) each of these 3 fd references file table entry for a file named **/dev/tty**

**/dev/tty**: In-memory surrogate for the terminal.

* **Terminal**: Combination keyboard/video screen.
* **Read from stdin => read from fd 0**: Whenever we write any character from the keyboard, it reads from stdin through fd 0 and saves to a file named /dev/tty.
* **Write to stdout => write to fd 1**: Whenever we see any output to the video screen, it’s from the file named /dev/tty and written to stdout in screen through fd 1.
* **Write to stderr => write to fd 2**: We see any error to the video screen, it is also from that file write to stderr in screen through fd 2.

## **create() system call**

## The create() function is used to create a new empty file in C. We can specify the permission and the name of the file which we want to create using the create() function. It is defined inside **<unistd.h>** header file and the flags that are passed as arguments are defined inside **<fcntl.h>** header file.

### ****Syntax of create() in C****

int **create**(char \*filename, mode\_t mode);

### ****Parameter****

* **filename:** name of the file which you want to create
* **mode:** indicates permissions of the new file.
  + The mode is usually specified as an octal number such as 0666 that would mean read/write permission for owner, group, and others or the mode may also be entered using manifest constants defined in the "/usr/include/sys/stat.h" file.
  + The following is a sample of the manifest constants for the mode argument as defined in /usr/include/sys/stat.h:
    - #define S\_IRWXU 0000700 /\* -rwx------ \*/
    - #define S\_IREAD 0000400 /\* read permission, owner \*/
    - #define S\_IRUSR S\_IREAD #define S\_IWRITE 0000200 /\* write permission, owner \*/
    - #define S\_IWUSR S\_IWRITE
    - #define S\_IEXEC 0000100 /\* execute/search permission, owner \*/
    - #define S\_IXUSR S\_IEXEC #define S\_IRWXG 0000070 /\* ----rwx--- \*/
    - #define S\_IRGRP 0000040 /\* read permission, group \*/
    - #define S\_IWGRP 0000020 /\* write " " \*/
    - #define S\_IXGRP 0000010 /\* execute/search " " \*/
    - #define S\_IRWXO 0000007 /\* -------rwx \*/
    - #define S\_IROTH 0000004 /\* read permission, other \*/
    - #define S\_IWOTH 0000002 /\* write " " \*/
    - #define S\_IXOTH 0000001 /\* execute/search " " \*/

Example:

The mode=0764

or

S\_IRWXU | S\_IRGRP | S\_IWGRP | S\_IROTH

The above statement sets the file access permissions as-

* S\_IRWXU - Read, Write, Execute permission for user.
* S\_IRGRP - Read permission for Group members.
* S\_IWGRP - Read, Write permission for group members.
* S\_IROTH - Read only permission for all other people.

0 7 6 4

rwx rw r

u g o

### ****How C create() works in OS****

* Create a new empty file on the disk.
* Create file table entry.
* Set the first unused file descriptor to point to the file table entry.
* Return file descriptor used, -1 upon failure.

## **Open() system call-**

The open() function in C is used to open the file for reading, writing, or both. It is also capable of creating the file if it does not exist. It is defined inside **<unistd.h>** header file and the flags that are passed as arguments are defined inside **<fcntl.h>** header file.

### ****Syntax of open() in C****

int **open** (const char\* Path, int flags);

### ****Parameters****

* **Path:** Path to the file which we want to open.
  + Use the **absolute path** beginning with “/” when you are **not** **working in the same directory**as the C source file.
  + Use**relative path** which is only the file name with extension, when you are**working in the same directory**as the C source file.
* **flags:**It is used to specify how you want to open the file. We can use the following flags.

| **Flags** | **Description** |
| --- | --- |
| **O\_RDONLY** | Opens the file in read-only mode. |
| **O\_WRONLY** | Opens the file in write-only mode. |
| **O\_RDWR** | Opens the file in read and write mode. |
| **O\_CREAT** | Create a file if it doesn’t exist. |
| **O\_EXCL** | Prevent creation if it already exists. |
| **O\_ APPEND** | Opens the file and places the cursor at the end of the contents. |
| **O\_ASYNC** | Enable input and output control by signal. |
| **O\_CLOEXEC** | Enable close-on-exec mode on the open file. |
| **O\_NONBLOCK** | Disables blocking of the file opened. |
| **O\_TMPFILE** | Create an unnamed temporary file at the specified path. |

### ****How C open() works in OS****

* Find the existing file on the disk.
* Create file table entry.
* Set the first unused file descriptor to point to the file table entry.
* Return file descriptor used, -1 upon failure.

## **3. close() system call-**

The close() function in C tells the operating system that you are done with a file descriptor and closes the file pointed by the file descriptor. It is defined inside**<unistd.h>** header file.

### Syntax of close() in C

int close(int fd);

### ****Parameter****

* **fd: F**ile descriptor of the file that you want to close.

### ****Return Value****

* **0** on success.
* **-1** on error.

### ****How C close() works in the OS****

* Destroy file table entry referenced by element fd of the file descriptor table  
  – As long as no other process is pointing to it!
* Set element fd of file descriptor table to **NULL**

## **4. read() system call-**

From the file indicated by the file descriptor fd, the read() function reads the specified amount of bytes **cnt** of input into the memory area indicated by **buf**. A successful read() updates the access time for the file. The read() function is also defined inside the <unistd.h> header file.

### Syntax of read() in C

size\_t **read** (int fd, void\* buf, size\_t cnt);

### ****Parameters****

* **fd:** file descriptor of the file from which data is to be read.
* **buf:** buffer to read data from
* **cnt:** length of the buffer

### ****Return Value****

* return Number of bytes read on success
* return 0 on reaching the end of file
* return -1 on error
* return -1 on signal interrupt

### ****Important Points****

* **buf** needs to point to a valid memory location with a length not smaller than the specified size because of overflow.
* **fd** should be a valid file descriptor returned from open() to perform the read operation because if fd is NULL then the read should generate an error.
* **cnt** is the requested number of bytes read, while the return value is the actual number of bytes read. Also, some times read system call should read fewer bytes than cnt.

## **5. write() system call-**

Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT\_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

The write() is also defined inside **<unistd.h>** header file.

### Syntax of write() in C

size\_t **write** (int fd, void\* buf, size\_t cnt);

### ****Parameters****

* **fd:** file descriptor
* **buf:** buffer to write data from.
* **cnt:** length of the buffer.

### ****Return Value****

* returns the number of bytes written on success.
* return 0 on reaching the End of File.
* return -1 on error.
* return -1 on signal interrupts.

### ****Important Points about C write****

* The file needs to be opened for write operations
* **buf**needs to be at least as long as specified by cnt because if buf size is less than the cnt then buf will lead to the overflow condition.
* **cnt** is the requested number of bytes to write, while the return value is the actual number of bytes written. This happens when **fd** has a less number of bytes to write than cnt.
* If write() is interrupted by a signal, the effect is one of the following:
  + If write() has not written any data yet, it returns -1 and sets errno to EINTR.
  + If write() has successfully written some data, it returns the number of bytes it wrote before it was interrupted.

1. **dup() system call-**

* It uses the lowest-numbered unused descriptor for the new descriptor.
* If the copy is successfully created, then the original and copy file descriptors may be used interchangeably.
* They both refer to the same open file description and thus share file offset and file status flags.

**Syntax:**

int dup(int oldfd);

**oldfd:** old file descriptor whose copy is to be created.

1. **Fstat() system call-**

**Syntax**

int fstat(int *fildes*, struct stat \**buf*)

* The *fstat*() function shall obtain information about an open file associated with the file descriptor *fildes*, and shall write it to the area pointed to by *buf*.
* If *fildes* references a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be valid. The implementation may update other fields and flags.
* If *fildes* references a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be valid. The implementation may update other fields and flags.
* The *buf* argument is a pointer to a **stat** structure, as defined in [*<sys/stat.h>*](https://pubs.opengroup.org/onlinepubs/009696699/basedefs/sys/stat.h.html), into which information is placed concerning the file.
* The structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*, *st\_ctime*, and *st\_mtime* shall have meaningful values for all other file types defined in this volume of IEEE Std 1003.1-2001. The value of the member *st\_nlink* shall be set to the number of links to the file.

**Return value-**

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

1. **Append-**

On Linux, while working with files in a terminal sometimes we need to append the same data of a command output or file content. Append means simply add the data to the file without erasing existing data.

### **Using >> Operator**

The >> operator redirects output to a file. If the mentioned file doesn’t exist the file is created and then the text is appended to the file.

echo "Which is the best Linux Distro?" >> file.txt

1. printf command to append text into a file

printf "Which is the best Linux Distro?\n" >> file.txt

1. using cat command

cat file1.txt >> file2.txt

1. using tee command

echo "Which is the best Linux Distro?" | tee -a file.txt