**EXPERIMENT 4**

**Aim:** Study system calls related to file operations.

Study system calls: open(), read(), write(), stat(), fstat().

Problems

1. Implement file copy command using system call also handle all possible errors.
2. Print i-node information using stat/fstat. Also identify type of a file (device file, pipe, directory, link etc.)
3. Find out whether file descriptors and file read pointer are shared or not by parent and child process after fork()

**Tools:** Linux OS, gcc compiler

**Procedure:**

* **Details of the System calls:**

| NAME |
| --- |
|  | open - open and possibly create a file or device |
|  |  |
|  | SYNOPSIS |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  |  |
|  | int open(const char \*pathname, int flags); |
|  | int open(const char \*pathname, int flags, mode\_t mode); |
|  |  |
|  | DESCRIPTION |
|  | Given a pathname for a file, open() returns a file descriptor, a small, non-negative integer for |
|  | use in subsequent system calls. The file descriptor returned by a successful call will be the |
|  | lowest-numbered file descriptor not currently open for the process. |
|  | RETURN VALUE |
|  | open() return the new file descriptor, or -1 if an error occurred (in which case, |
|  | errno is set appropriately). |
|  | --------------------------------------------------------------------------------------------------------- |
|  | NAME |
|  | read - read from a file descriptor |
|  |  |
|  | SYNOPSIS |
|  | #include <unistd.h> |
|  |  |
|  | ssize\_t read(int fd, void \*buf, size\_t count); |
|  |  |
|  | DESCRIPTION |
|  | read() attempts to read up to count bytes from file descriptor fd into |
|  | the buffer starting at buf. |
|  |  |
|  | If count is zero, read() returns zero and has no other results. If |
|  | count is greater than SSIZE\_MAX, the result is unspecified. |
|  |  |
|  | RETURN VALUE |
|  | On success, the number of bytes read is returned (zero indicates end of |
|  | file), and the file position is advanced by this number. It is not an |
|  | error if this number is smaller than the number of bytes requested; |
|  | this may happen for example because fewer bytes are actually available |
|  | right now (maybe because we were close to end-of-file, or because we |
|  | are reading from a pipe, or from a terminal), or because read() was |
|  | interrupted by a signal. On error, -1 is returned, and errno is set |
|  | appropriately. In this case it is left unspecified whether the file |
|  | position (if any) changes. |
|  |  |
|  | -------------------------------------------------------------------------------------------------------- |
|  | NAME |
|  | write - write to a file descriptor |
|  |  |
|  | SYNOPSIS |
|  | #include <unistd.h> |
|  |  |
|  | ssize\_t write(int fd, const void \*buf, size\_t count); |
|  |  |
|  | DESCRIPTION |
|  | write() writes up to count bytes to the file referenced by the file |
|  | descriptor fd from the buffer starting at buf. POSIX requires that a |
|  | read() which can be proved to occur after a write() has returned |
|  | returns the new data. Note that not all file systems are POSIX con- |
|  | forming. |
|  |  |
|  | RETURN VALUE |
|  | On success, the number of bytes written are returned (zero indicates |
|  | nothing was written). On error, -1 is returned, and errno is set |
|  | appropriately. If count is zero and the file descriptor refers to a |
|  | regular file, 0 may be returned, or an error could be detected. For a |
|  | special file, the results are not portable. |
|  | --------------------------------------------------------------------------------------------------------- |
|  | NAME |
|  | stat, fstat, lstat - get file status |
|  |  |
|  | SYNOPSIS |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <unistd.h> |
|  |  |
|  | int stat(const char \*path, struct stat \*buf); |
|  | int fstat(int filedes, struct stat \*buf); |
|  | int lstat(const char \*path, struct stat \*buf); |
|  |  |
|  | DESCRIPTION |
|  | These functions return information about a file. No permissions are |
|  | required on the file itself, but — in the case of stat() and lstat() — |
|  | execute (search) permission is required on all of the directories in |
|  | path that lead to the file. |
|  |  |
|  | stat() stats the file pointed to by path and fills in buf. |
|  |  |
|  | lstat() is identical to stat(), except that if path is a symbolic link, |
|  | then the link itself is stat-ed, not the file that it refers to. |
|  |  |
|  | fstat() is identical to stat(), except that the file to be stat-ed is |
|  | specified by the file descriptor filedes. |
|  |  |
|  | RETURN VALUE |
|  | On success, zero is returned. On error, -1 is returned, and errno is |
|  | set appropriately. |
|  |  |
|  | --------------------------------------------------------------------------------------------------------- |
|  | NAME |
|  | fcntl - manipulate file descriptor |
|  |  |
|  | SYNOPSIS |
|  | #include <unistd.h> |
|  | #include <fcntl.h> |
|  |  |
|  | int fcntl(int fd, int cmd); |
|  | int fcntl(int fd, int cmd, long arg); |
|  | int fcntl(int fd, int cmd, struct flock \*lock); |
|  |  |
|  | DESCRIPTION |
|  | fcntl() performs one of the operations described below on the open file |
|  | descriptor fd. The operation is determined by cmd. |
|  |  |
|  | RETURN VALUE |
|  | For a successful call, the return value depends on the operation: |
|  |  |
|  | F\_DUPFD The new descriptor. |
|  |  |
|  | F\_GETFD Value of flags. |
|  |  |
|  | F\_GETFL Value of flags. |
|  |  |
|  | F\_GETOWN Value of descriptor owner. |
|  |  |
|  | F\_GETSIG Value of signal sent when read or write becomes possible, or |
|  | zero for traditional SIGIO behaviour. |
|  |  |
|  | All other commands |
|  | Zero. |
|  |  |
|  | On error, -1 is returned, and errno is set appropriately. |

1. **Implement file copy command using system call also handle all possible errors.**

#include<sys/stat.h>

#include<fcntl.h>

#include<stdio.h>

#define MAX\_LEN 50

int main()

{

int fd1;

fd1 = open("file1.txt",O\_RDWR);

if(fd1>0)

printf("File 1 opened successfully!\n");

else{

printf("Error!\n");

//exit(0);

}

int fd2 = open("file2.txt",O\_RDWR);

if(fd2>0)

printf("File 2 opened successfully!\n");

else{

printf("Error!\n");

//exit(0);

}

char buff[MAX\_LEN];

int nbytes=read(fd1,buff,MAX\_LEN);

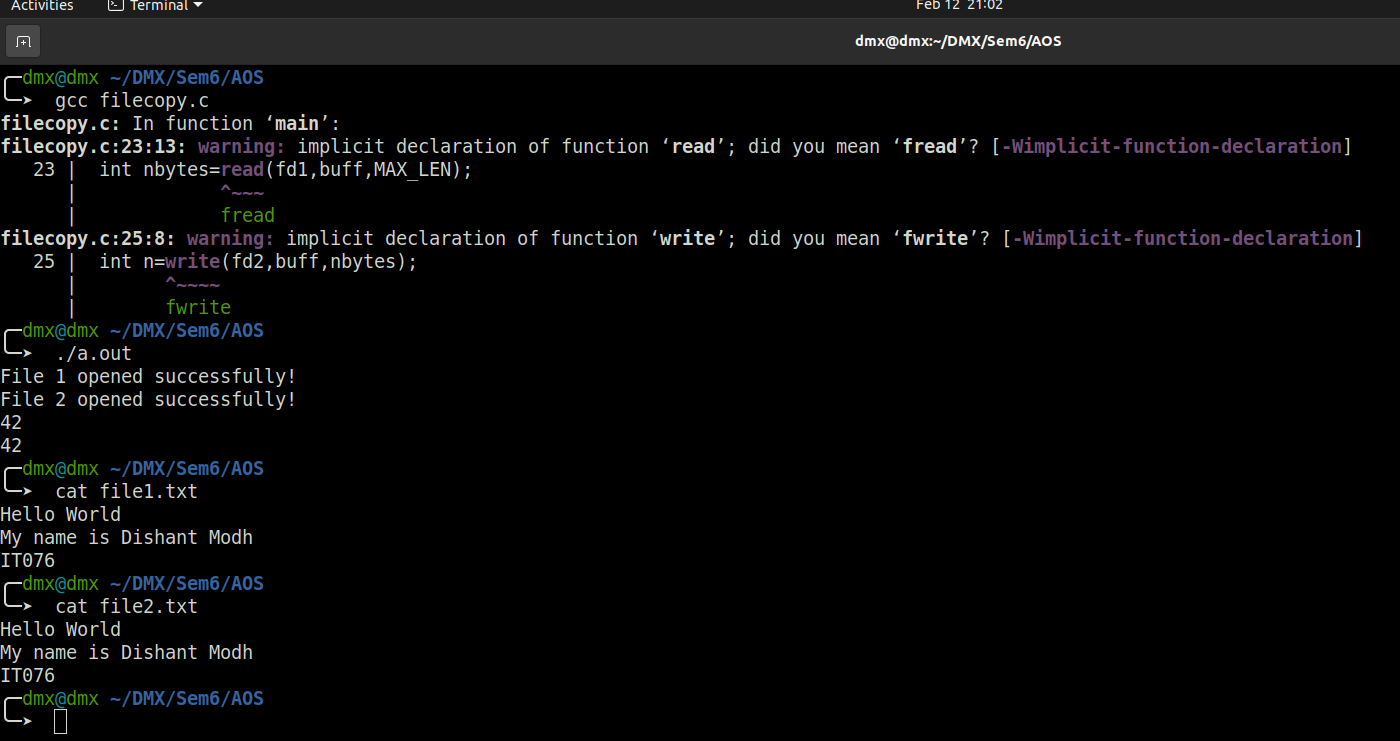
printf("%d\n",nbytes);

int n=write(fd2,buff,nbytes);

printf("%d\n",n);

return 0;

}



1. **Print i-node information using stat/fstat. Also identify type of a file (device file, pipe, directory, link etc.)**

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <time.h>

int main()

{

struct stat mystat;

int x = stat("file1.txt",&mystat);

if(x == 0)

{

//file=open("file1",O\_RDONLY))

printf("File Size: \t\t%d bytes\n",mystat.st\_size);

printf(" PROTECTION MODE: \t%d\n",mystat.st\_mode);

printf("LAST MODIFIED TIME: \t%s\n",ctime(&mystat.st\_mtime));

printf("Number of Links: \t%d\n",mystat.st\_nlink);

printf("File inode: \t\t%d\n",mystat.st\_ino);

printf( (S\_ISREG(mystat.st\_mode)) ? "Regular file" : "Not Regular File");

printf("File Permissions: \t");

printf( (S\_ISDIR(mystat.st\_mode)) ? "d" : "-");

printf( (mystat.st\_mode & S\_IRUSR) ? "r" : "-");

printf( (mystat.st\_mode & S\_IWUSR) ? "w" : "-");

printf( (mystat.st\_mode & S\_IXUSR) ? "x" : "-");

printf( (mystat.st\_mode & S\_IRGRP) ? "r" : "-");

printf( (mystat.st\_mode & S\_IWGRP) ? "w" : "-");

printf( (mystat.st\_mode & S\_IXGRP) ? "x" : "-");

printf( (mystat.st\_mode & S\_IROTH) ? "r" : "-");

printf( (mystat.st\_mode & S\_IWOTH) ? "w" : "-");

printf( (mystat.st\_mode & S\_IXOTH) ? "x" : "-");

printf("\n\n");

printf("The file %s a symbolic link\n\n", (S\_ISLNK(mystat.st\_mode)) ? "is" : "is not");

}

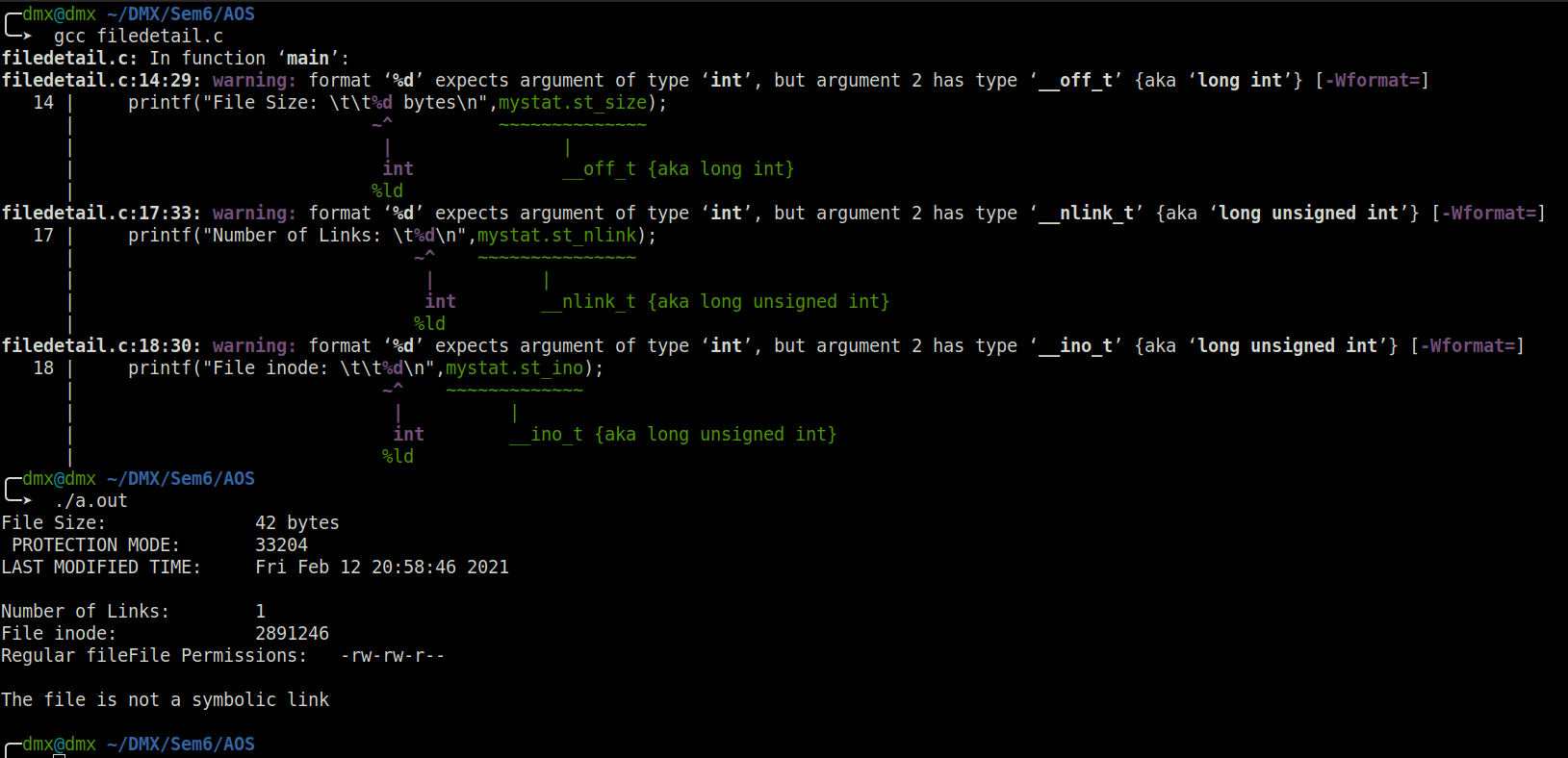
else{

printf("Error!");

}

return 0;

}



1. **Find out whether file descriptors and file read pointer are shared or not by parent and child process after fork()**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

int main()

{

int fd;

fd = open("file1.txt",O\_RDWR);

if(fd > 0)

{

char buff[50];

pid\_t a = fork();

if(a == 0){

int nbytes=read(fd,buff,50);

printf("child : %d\n",nbytes);

}

else{

int n=read(fd,buff,50);

printf("parent : %d\n",n);

}

}

return 0;

}

