

Practical-1

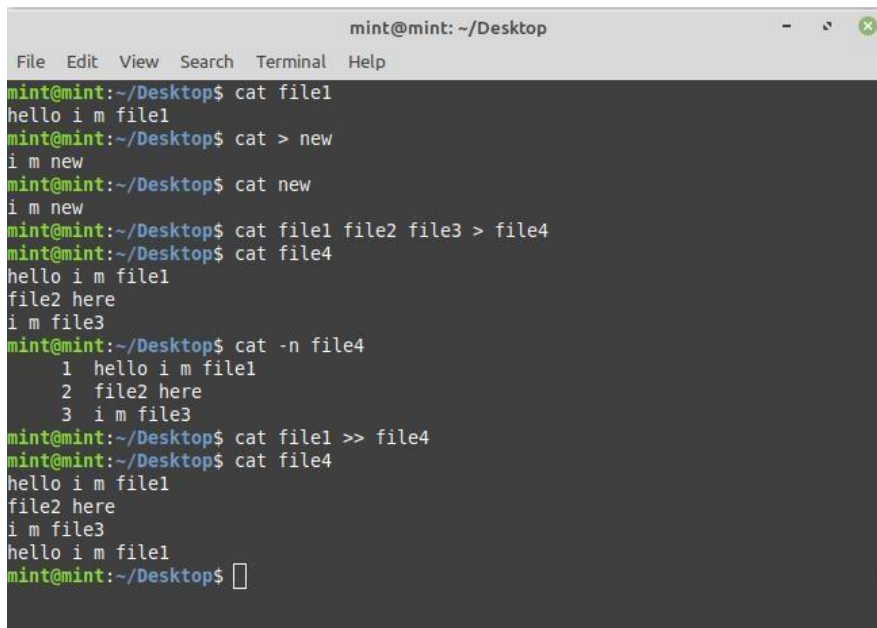
Aim: Implementation of “cat” and “cp” command in C. (use of open, read, write, and close system calls).

Cat command:

It allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

synopsis : cat [OPTION]... [FILE]...

demonstration of cat command,



```
mint@mint: ~/Desktop
File Edit View Search Terminal Help
mint@mint:~/Desktop$ cat file1
hello i m file1
mint@mint:~/Desktop$ cat > new
i m new
mint@mint:~/Desktop$ cat new
i m new
mint@mint:~/Desktop$ cat file1 file2 file3 > file4
mint@mint:~/Desktop$ cat file4
hello i m file1
file2 here
i m file3
mint@mint:~/Desktop$ cat -n file4
 1 hello i m file1
 2 file2 here
 3 i m file3
mint@mint:~/Desktop$ cat file1 >> file4
mint@mint:~/Desktop$ cat file4
hello i m file1
file2 here
i m file3
hello i m file1
mint@mint:~/Desktop$
```

Cp command:

This command is used to copy files or group of files or directory.

synopsis : cp [OPTION]... SOURCE... DIRECTORY

Demonstration of cp command,

```
mint@mint: ~/Desktop
File Edit View Search Terminal Help
mint@mint:~/Desktop$ ls
a.out cat.png file1 file3 new sample.c
cat.c cp.c file2 file4 pwd.c ubiquity.desktop
mint@mint:~/Desktop$ cp new sample
mint@mint:~/Desktop$ ls
a.out cat.png file1 file3 new sample ubiquity.desktop
cat.c cp.c file2 file4 pwd.c sample.c
mint@mint:~/Desktop$ cat new
i m new
mint@mint:~/Desktop$ cat sample
i m new
mint@mint:~/Desktop$ cp -i file1 sample
cp: overwrite 'sample'? y
mint@mint:~/Desktop$ cat sample
hello i m file1
mint@mint:~/Desktop$
```

Read system call:

From the file indicated by the file descriptor `fd`, the `read()` function reads `cnt` bytes of input into the memory area indicated by `buf`. A successful `read()` updates the access time for the file. Synopsis: `#include<unistd.h>`

```
ssize_t read(int fd, void *buf, size_t count);
```

`read()` attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`.

On files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the number of bytes read. If the current file offset is at or past the end of file, no bytes are read, and `read()` returns zero.

If `count` is zero, `read()` may detect the errors described below. In the absence of any errors, or if `read()` does not check for errors, a `read()` with a count of 0 returns zero and has no other effects.

Write system call:

Writes cnt bytes from buf to the file or socket associated with fd. If cnt is zero, write() simply returns 0 without attempting any other action.

Synopsis: `#include<unistd.h>`

`Ssize_t write(int fd,const void *buf,size_t count);`

On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned, and errno is set appropriately.

If count is zero and fd refers to a regular file, then write() may return a failure status if one of the errors below is detected. If no errors are detected, 0 will be returned without causing any other effect. If count is zero and fd refers to a file other than a regular file, the results are not specified.

Task 1: Write a program to achieve following:

1. Read input from terminal

2. Display the information read on the terminal.

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
#include<sys/stat.h>
```

```
#include<fcntl.h>
```

```
int main()
```

```
{
```

```
    int n;
```

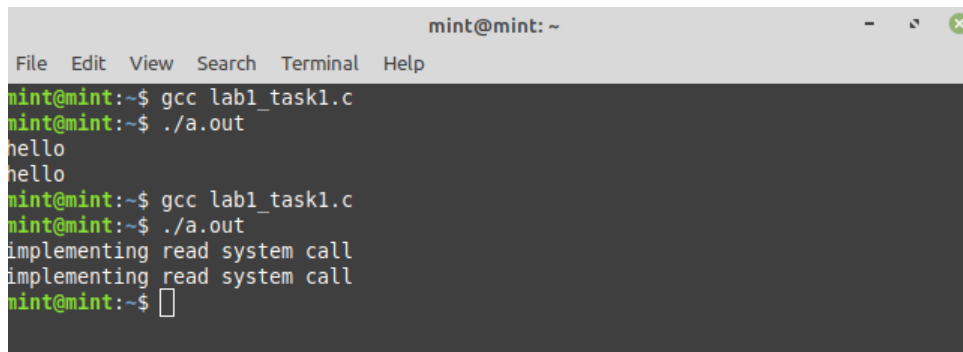
```
    char buff[100];
```

```
    n=read(0,buff,sizeof(buff));
```

```
        write(1,buff,n);
```

}

Output:

A terminal window titled 'mint@mint: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
mint@mint:~$ gcc lab1_task1.c
mint@mint:~$ ./a.out
hello
hello
mint@mint:~$ gcc lab1_task1.c
mint@mint:~$ ./a.out
implementing read system call
implementing read system call
mint@mint:~$
```

Open system call:

Used to Open the file for reading and writing.

Synopsis: `#include<sys/types.h>`

`#include<sys/stat.h>`

`#include<fcntl.h>`

`Int open(const char* path, int flags);`

Flags: `O_RDONLY`: read only, `O_WRONLY`: write only, `O_RDWR`: read and write, `O_CREAT`: create file if does not exist, `O_EXCL`: prevent creation if it exists.

Close system call:

Close the file which pointed by fd.

Synopsis: `#include<unistd.h>`

```
int close(int fd);
```

close() returns zero on success. On error, -1 is returned

Task 2: Write a program to achieve following:

- 1. Read a file name from terminal.**
- 2. Open the file and read the contents from the file.**
- 3. Write the file contents on the terminal.**

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
#include<sys/stat.h>
```

```
#include<fcntl.h>
```

```
int main(int argc,char *argv[])
```

```
{
```

```
    int fd,n;
```

```
    char buff[100];
```

```
    if(argc==2)
```

```
    {
```

```
        fd=open(argv[1],O_RDONLY);
```

```
        while((n=read(fd,buff,sizeof(buff)))>1)
```

```
        {
```

```
            write(1,buff,n);
```

```
    }  
    close(fd);  
}  
}
```

Output:

A screenshot of a terminal window titled 'mint@mint: ~'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The command history shows: 'mint@mint:~\$ gcc lab1 task2.c', 'mint@mint:~\$./a.out file1', the program output 'hello you are reading file1' and 'bye', and the prompt 'mint@mint:~\$' with a cursor.

```
mint@mint:~$ gcc lab1 task2.c  
mint@mint:~$ ./a.out file1  
hello you are reading file1  
bye  
mint@mint:~$
```

ASSIGNMENTS:

1.Implement basic “cat” command using system calls.

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<sys/types.h>
```

```
#include<sys/stat.h>
```

```
#include<fcntl.h>
```

```
int main(int argc,char *argv[])
```

```
{
```

```
int fd,n,i;

char buff[100];

if(argc==1)

{

    n=read(0,buff,sizeof(buff));

    write(1,buff,n);

}

else if(argc==2)

{

    fd=open(argv[1],O_RDONLY);

    while((n=read(fd,buff,sizeof(buff)))>1)

    {

        write(1,buff,n);

    }

    close(fd);

}

else if(argc>2)

{

    for(i=1;i<argc;i++)

    {

        fd=open(argv[i],O_RDONLY);
```

```

while((n=read(fd,buff,sizeof(buff)))>1)
{
    write(1,buff,n);
}

close(fd);
}
}
}

```

Output:

```

mint@mint: ~/Desktop
File Edit View Search Terminal Help
mint@mint:~/Desktop$ gcc -Wall cat.c
mint@mint:~/Desktop$ ./a.out
implementing cat command in C
implementing cat command in C
mint@mint:~/Desktop$ cat
hello
hello
mint@mint:~/Desktop$ ./a.out file1
hello i m file1
mint@mint:~/Desktop$ ./a.out file1 file2 file3
hello i m file1
file2 here
i m file3
mint@mint:~/Desktop$ 

```

2.Implement basic “cp” command using system calls.

```
#include<stdio.h>
```

```
#include<unistd.h>
```



```
#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

int main(int argc,char *argv[])
{
    int fd1,fd2,n;

    char buff[100];

    if(argc==3)
    {
        fd1=open(argv[1],O_RDONLY);

        fd2=open(argv[2],O_RDWR | O_CREAT);

        while((n=read(fd1,buff,sizeof(buff)))>1)
        {
            write(fd2,buff,n);
        }

        close(fd1);

        close(fd2);
    }
    else
    {
        printf("enter proper source file and destination file\n");
    }
}
```

}

Output:

```
mint@mint: ~/Desktop
File Edit View Search Terminal Help
mint@mint:~/Desktop$ gcc -Wall cp.c
mint@mint:~/Desktop$ cat file3
i m file3
mint@mint:~/Desktop$ ./a.out file1 file3
mint@mint:~/Desktop$ cat file3
hello i m file1
mint@mint:~/Desktop$ ls
a.out  cat.png  cp.png  file2  file4  pwd.c
cat.c  cp.c     file1   file3  pl.png  ubiquity.desktop
mint@mint:~/Desktop$ ./a.out file1 new
mint@mint:~/Desktop$ ls
a.out  cat.png  cp.png  file2  file4  pl.png  ubiquity.desktop
cat.c  cp.c     file1   file3  new    pwd.c
mint@mint:~/Desktop$ chmod u+r new
mint@mint:~/Desktop$ cat new
hello i m file1
mint@mint:~/Desktop$
```