

***SYSTEM CALLS:-**

1. read

synopsis:

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

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

description:

It attempts to read up to count bytes from console or files into the buffer starting at buf.

int fd: This is file-descriptor. it can have different values like fd, 0, 1. It shows from where to read.

Void *buf: Whatever it reads will store into the buf.

size_t count: It is size of the buf that maximum how many bytes can be read.

On success, the number of bytes read is returned and the file position is advanced by this number.

2. write

synopsis:

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

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

description:

write() writes up to count bytes from the buffer starting at buf to the file referred to by the file descriptor fd.

Int fd: This is file pointer. It can have different values for where to write the content.

Const void *buf: The content that read from read() call.

size_t count: This represents the number of bytes read from read() call.

On success, the number of bytes written is returned. On error, -1 is returned, and errno is set to indicate the cause of the error.

3. open

synopsis:

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

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

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

```
int open(const char *pathname, int flags);
```

description:

The open() system call opens the file specified by pathname. This creates file-descriptor pointed to particular file.

Const char *pathname: This contains name of the file which we want to open.

Int flags: This provides different options to open file in different mode. Basic modes are O_RDONLY, O_WRONLY and O_RDWR.

4. close

Synopsis:

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

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

Description:

Close a file descriptor, so that it will no longer point to a file that was previously pointed.

On success return 0 and on error return(-1).

CODE:

1. Implement “cat” command using system calls.

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

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

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

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

```
int main(int argc, char *argv[])
{
    int n, fd, i, no;
    char arr[100], buf[1000];
    if(argc==1)
    {
        while((n=read(0, arr, sizeof(arr)))>0)
        {
            write(1, arr, n);
        }
    }
    else if(argc==2)
    {
        fd=open(argv[1], O_RDONLY);
        n=read(fd, arr, sizeof(arr));
```

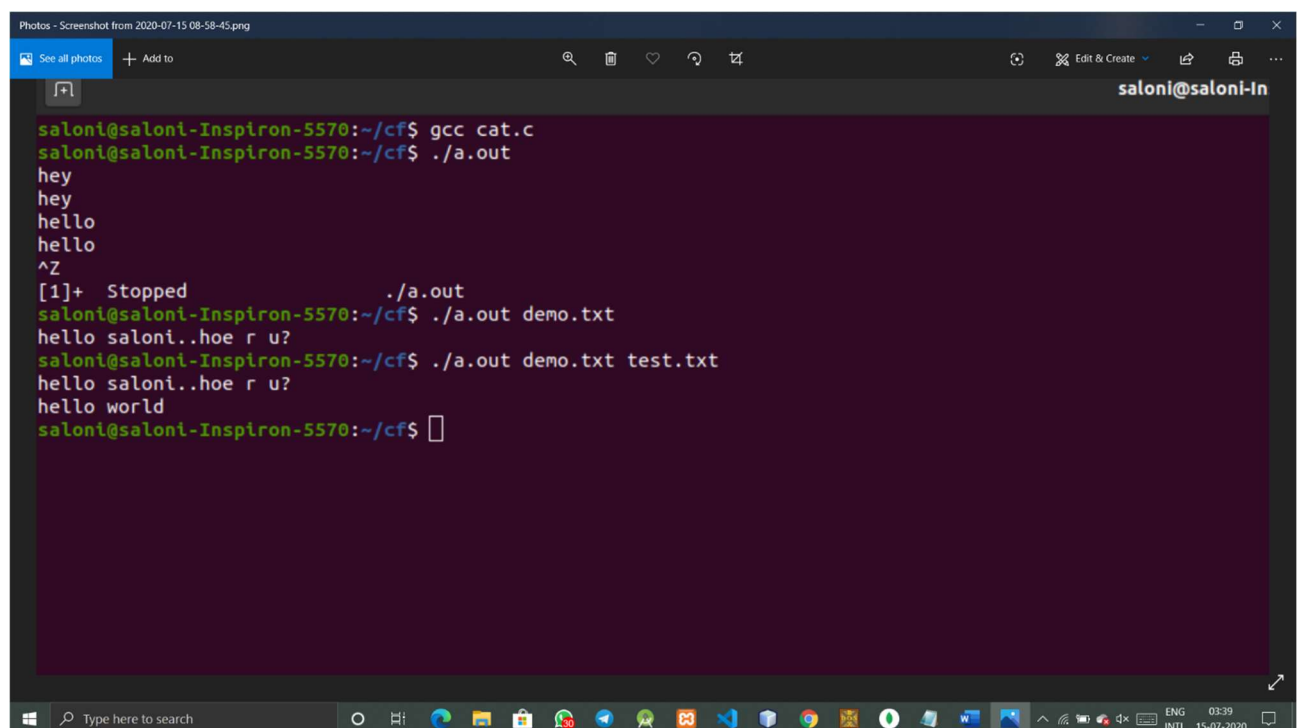
```

        write(1,&arr,n);
    }
    else
    {
        for (i=1;i<argc;i++)
        {
            fd=open(argv[i],O_RDONLY);
            no=read(fd,buf,sizeof(arr));
            write(1,&buf,no);
        }
    }

    return 0;
}

```

OUTPUT:



```

saloni@saloni-Inspiron-5570:~/cf$ gcc cat.c
saloni@saloni-Inspiron-5570:~/cf$ ./a.out
hey
hey
hello
hello
^Z
[1]+  Stopped                  ./a.out
saloni@saloni-Inspiron-5570:~/cf$ ./a.out demo.txt
hello saloni..hoe r u?
saloni@saloni-Inspiron-5570:~/cf$ ./a.out demo.txt test.txt
hello saloni..hoe r u?
hello world
saloni@saloni-Inspiron-5570:~/cf$ 

```

DESC:

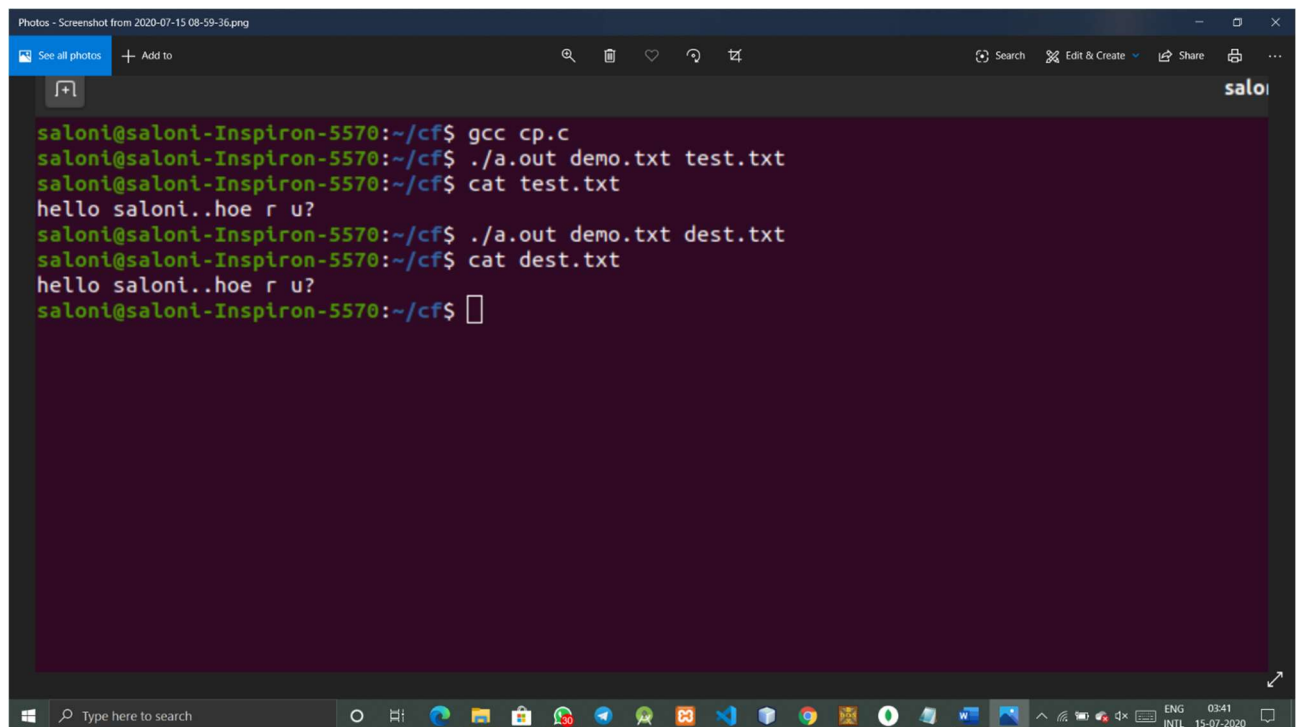
here we can see that when we run the code without any arguments then it is working as “echo”.

When we are providing one or more than one arguments than it will shows the content of the file specified in the command line arguments.

2. Implement “cp” command using system calls.

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main(int argc,char *argv[])
{
    int fd,fde,n,fdw;
    char arr[100];
    fd=open(argv[1],O_RDONLY);
    n=read(fd,arr,sizeof(arr));
    fde=open(argv[2],O_CREAT,0777);
    fdw=open(argv[2],O_WRONLY);
    write(fdw,arr,n);
    return 0;
}
```

OUTPUT:



```
saloni@saloni-Inspiron-5570:~/cf$ gcc cp.c
saloni@saloni-Inspiron-5570:~/cf$ ./a.out demo.txt test.txt
saloni@saloni-Inspiron-5570:~/cf$ cat test.txt
hello saloni..hoe r u?
saloni@saloni-Inspiron-5570:~/cf$ ./a.out demo.txt dest.txt
saloni@saloni-Inspiron-5570:~/cf$ cat dest.txt
hello saloni..hoe r u?
saloni@saloni-Inspiron-5570:~/cf$
```

DESC:

In cp command we need to specify two files . one as source and other as destination. After executing source file's contents are copied to destination file.

If destination files doesn't exist then it will be created by O_CREAT and the contents of source will be copied to newly created destination file.

