JARIWALA SAHIL Y.

SEM: 5

SUB: OS

LAB : 1

AIM: Implement “cat” and “cp” command

SYSTEM CALL:

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.

A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system’s kernel.

System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.

System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Read System Call:

Needed Library For using read() System call:

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

Syntax :

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

here ssize_t is return type
fd = file descriptor
buf = buffer in which we read data
count = byte count which we want to read-only

Description:

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

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.

On error, -1 is returned, and errno is set appropriately.
Example: n = read(0,buff,sizeof(buff))
fd = 0 means read from stander input

Write System Call:

Needed Library For using write() System call:

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

Syntax :

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

here ssize_t is return type
fd = file descriptor
buf = buffer in which we have read data
count = byte count which we want to Write

Description:
write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.
On success, the number of bytes written is returned (zero indicates nothing was written).
On error, -1 is returned, and errno is set appropriately.
Example: write(1,buff,n)
fd = 1 means read from stander input

Open System Call:

Needed Library For using open() System call:

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

Syntax :

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

*pathname = path for file whiche we want to open
flags = defines the one or more access mode
modes(optional) = defines the permissions for new file creation

Description:

Given a path-name for a file, open() returns a file descriptor, a small, non negative integer for use in subsequent system calls.

The argument flags must include one of the following access modes:
O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read-only, write-only, or read/write, respectively.

In addition, zero or more file creation flags and file status flags can be bitwise-or'd in flags.
The file creation flags are O_CLOEXEC, O_CREAT, O_DIRECTORY, O_EXCL, O_NOCTTY, O_NOFOLLOW, O_TRUNC, and O_TTY_INIT.
mode specifies the permissions to use in case a new file is created. This argument must be supplied when O_CREAT is specified in flags; if O_CREAT is not specified, then mode is ignored.
Example: fd = open(“test.txt”,O_RDONLY)

Close System Call:

Needed Library For using open() System call:

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

Syntax:

int close(int fd);

fd = file descriptor which point to file which we want to close

Description:

close() closes a file descriptor, so that it no longer refers to any file and may be reused.

close() returns zero on success. On error, -1 is returned, and errno is set appropriately.

Example: close(fd)

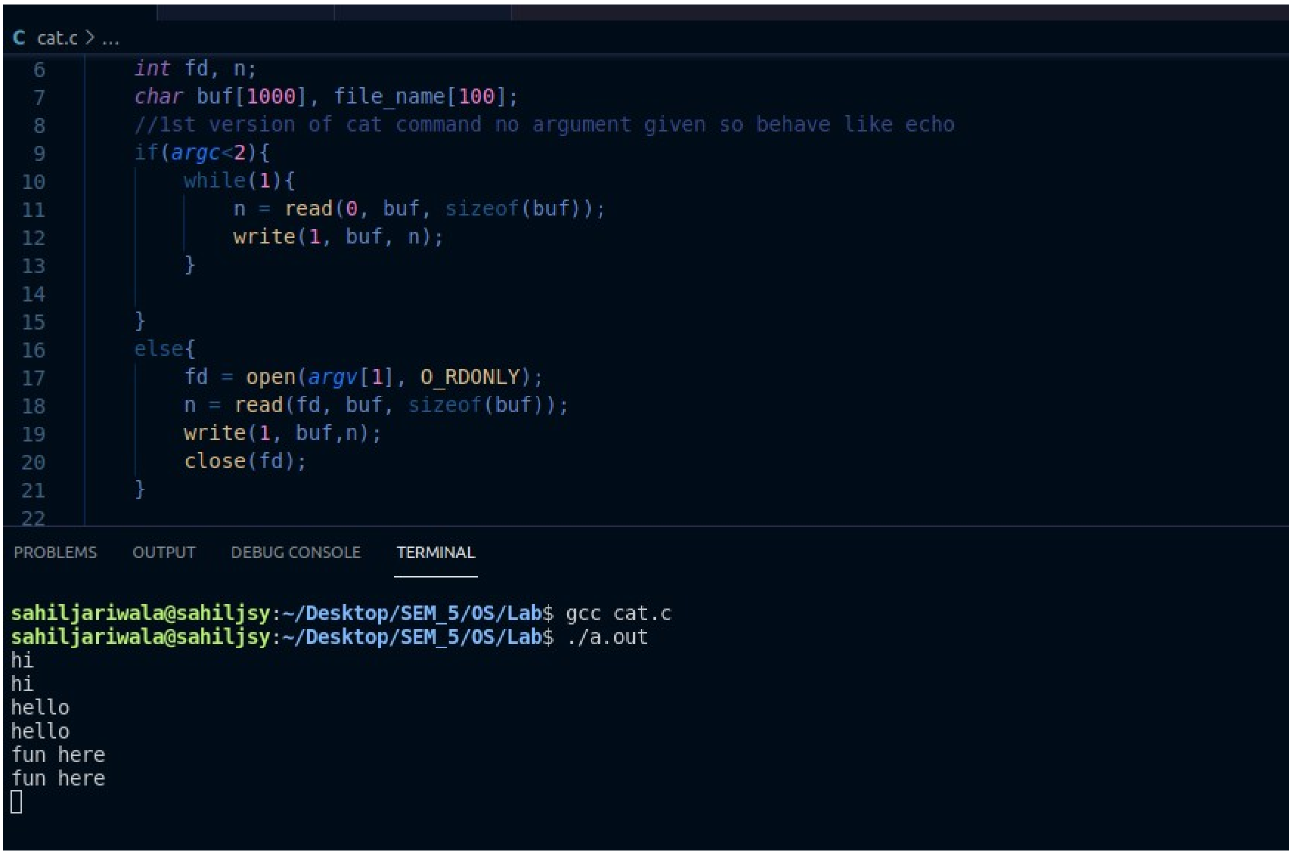
1. Implementation of “CAT”:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc, char *argv[] ){
   int fd, n;
   char buf[1000], file_name[100];
   //1st version of cat command no argument given so behave like echo
   if(argc<2){
      while(1){
         n = read(0, buf, sizeof(buf));
         write(1, buf, n);
      }

   }
   else{
      for(i=1; i< argc; i++)
       {
             fd = open(argv[i], O_RDONLY);
             n = read(fd, buf, sizeof(buf));
             write(1, buf,n);
             close(fd);
       }
```

```
    }
}
```

OUTPUT:

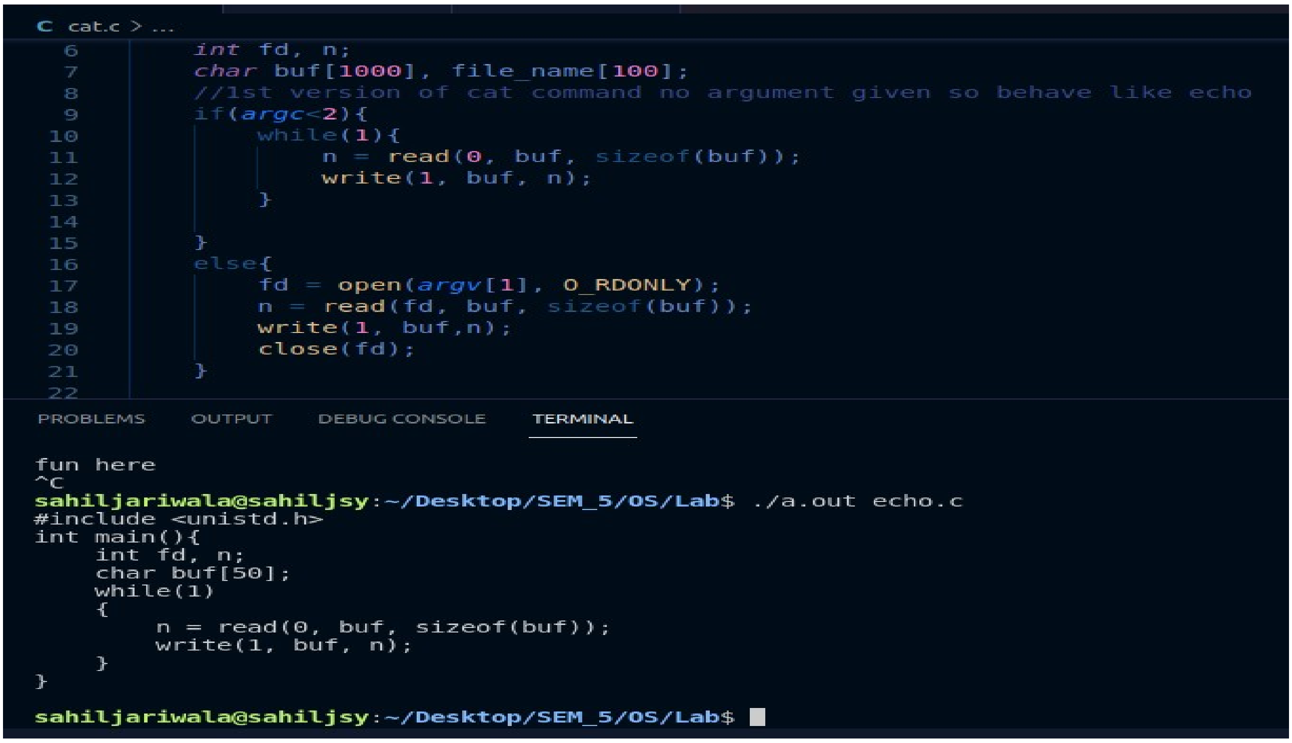
1. cat without argument



```c
    int fd, n;
    char buf[1000], file_name[100];
    //1st version of cat command no argument given so behave like echo
    if(argc<2){
        while(1){
            n = read(0, buf, sizeof(buf));
            write(1, buf, n);
        }

    }
    else{
        fd = open(argv[1], O_RDONLY);
        n = read(fd, buf, sizeof(buf));
        write(1, buf,n);
        close(fd);
    }

```

```
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ gcc cat.c
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out
hi
hi
hello
hello
fun here
fun here
```

2. cat with argument



```c
    int fd, n;
    char buf[1000], file_name[100];
    //1st version of cat command no argument given so behave like echo
    if(argc<2){
        while(1){
            n = read(0, buf, sizeof(buf));
            write(1, buf, n);
        }

    }
    else{
        fd = open(argv[1], O_RDONLY);
        n = read(fd, buf, sizeof(buf));
        write(1, buf,n);
        close(fd);
    }

```

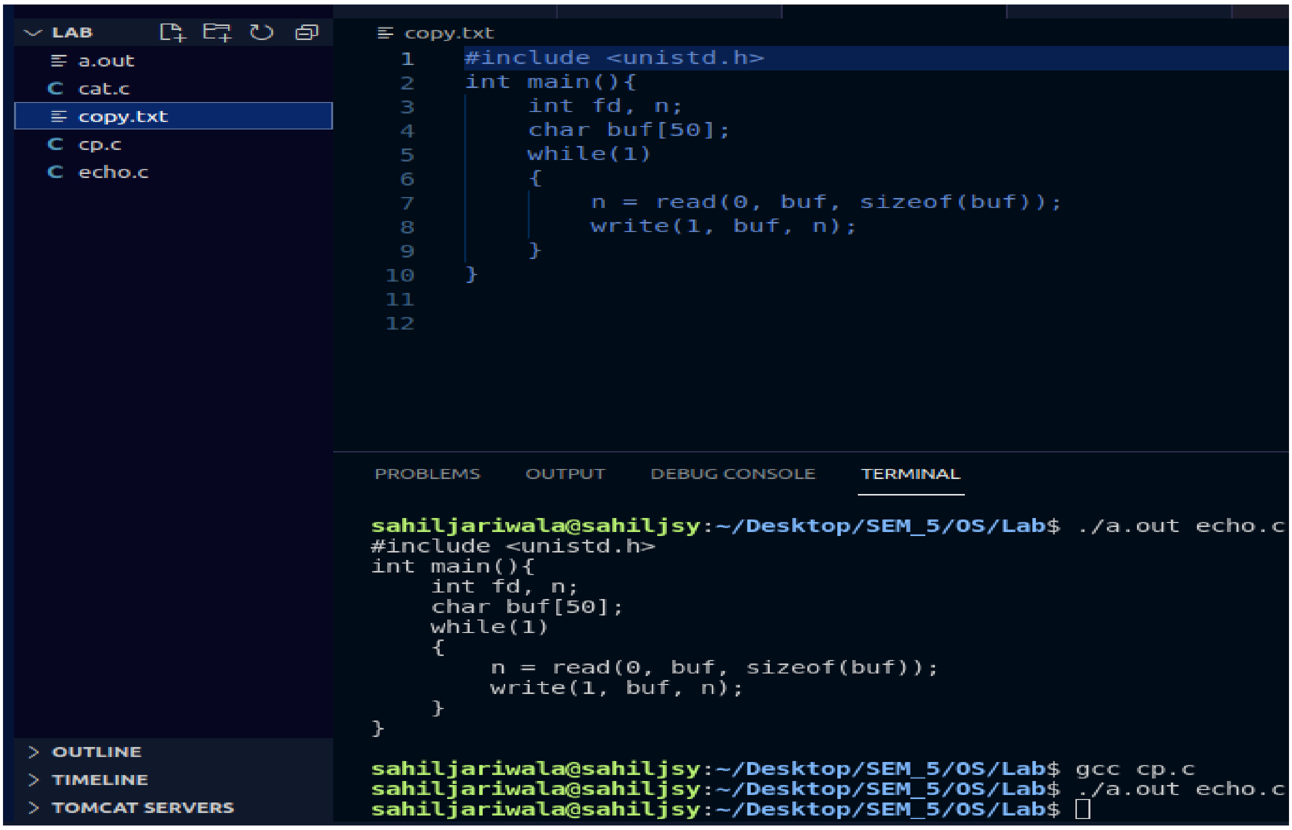
```
fun here
^C
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out echo.c
#include <unistd.h>
int main(){
    int fd, n;
    char buf[50];
    while(1)
    {
        n = read(0, buf, sizeof(buf));
        write(1, buf, n);
    }
}

sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$
```

2. Implement “CP”

```c
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc, char *argv[]){
    int fd, n;
    char buf[1000];
    fd = open(argv[1], O_RDONLY);
    n = read(fd, buf, sizeof(buf));
    close(fd);
    fd = open(argv[2], O_WRONLY | O_CREAT, 777);
    write(fd, buf,n);
    close(fd);
}
```

OUTPUT:

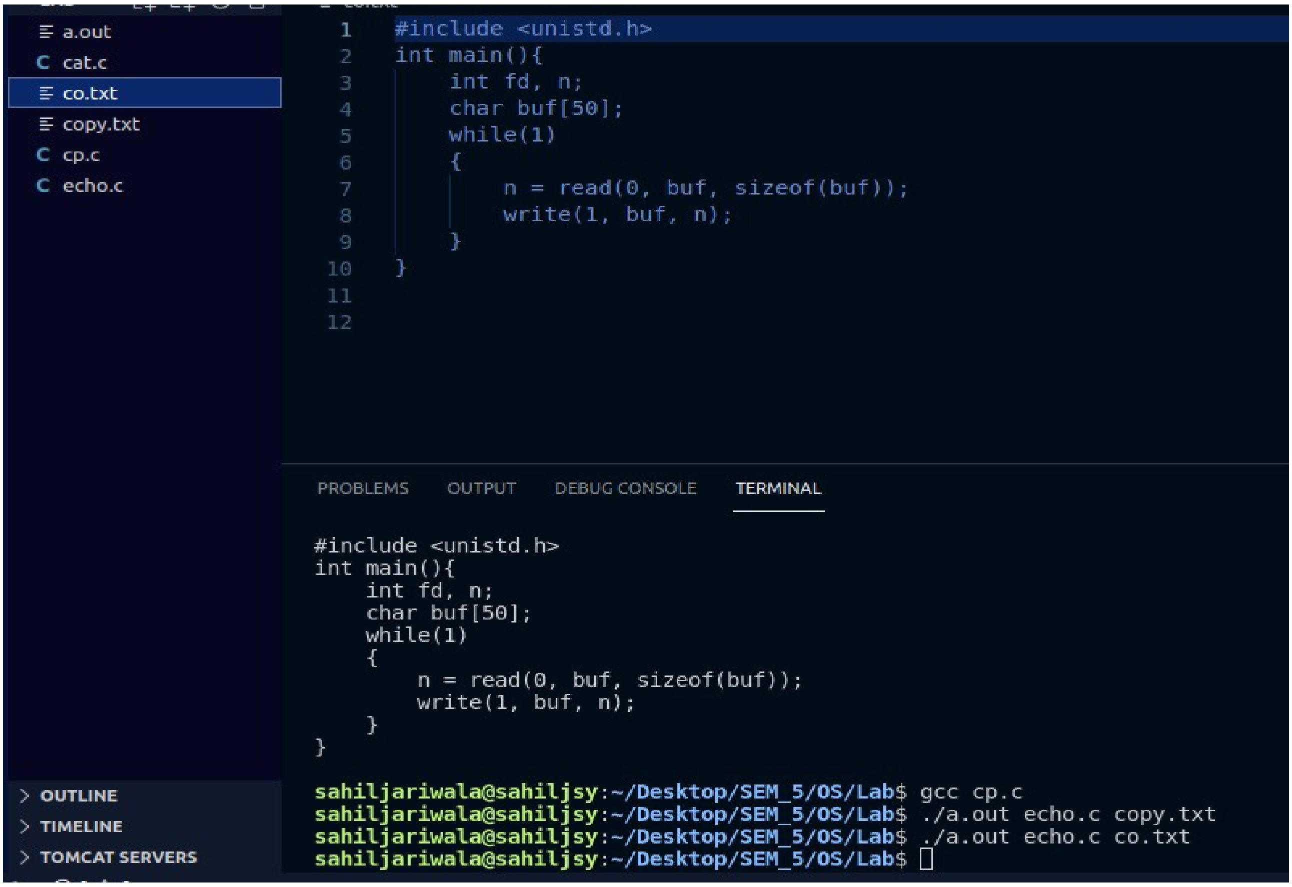
1. Destination file is present



```
#include <unistd.h>
int main(){
    int fd, n;
    char buf[50];
    while(1)
    {
        n = read(0, buf, sizeof(buf));
        write(1, buf, n);
    }
}
```

```
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out echo.c
#include <unistd.h>
int main(){
    int fd, n;
    char buf[50];
    while(1)
    {
        n = read(0, buf, sizeof(buf));
        write(1, buf, n);
    }
}
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ gcc cp.c
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out echo.c
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ 
```

2. If File does not exists



```
#include <unistd.h>
int main(){
    int fd, n;
    char buf[50];
    while(1)
    {
        n = read(0, buf, sizeof(buf));
        write(1, buf, n);
    }
}
```

```
#include <unistd.h>
int main(){
    int fd, n;
    char buf[50];
    while(1)
    {
        n = read(0, buf, sizeof(buf));
        write(1, buf, n);
    }
}
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ gcc cp.c
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out echo.c copy.txt
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ ./a.out echo.c co.txt
sahiljariwala@sahiljsy:~/Desktop/SEM_5/OS/Lab$ 
```