SYSTEM CALLS:

1) Open:

- Used to open file and directories.
- Syntax: int open (const char* Path, int flags [, int mode]);
- Parameters:

  Path: path to file which you want to use
  use absolute path begin with "/", when you are not work in same directory of file. Use relative path which is only file name with extension, when you are work in same directory of file.

  Flags: How you like to use
  O_RDONLY: read only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: create file if it doesn't exist, O_EXCL: prevent creation if it already exists
- Header Files:

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

2) Close:

- Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.
- Syntax: int close(int fd);
- Parameter

  fd :file descriptor
- Returns:

  0 on success.
  -1 on error.
- Header File:

  #include<fcntl.h>

3) Read:

- 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.
- Syntax: size_t read (int fd, void* buf, size_t cnt);
- Parameters

  fd: file descripter
  buf: buffer to read data from
  cnt: length of buffer

- Returns: How many bytes were actually read

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

4) Write:

- 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.
- Syntax: size_t write (int fd, void* buf, size_t cnt);
- Parameters:

  fd: file descripter

  buf: buffer to write data to

  cnt: length of buffer

- Returns: How many bytes were actually written

  return Number of bytes written on success
  return 0 on reaching end of file
  return -1 on error
  return -1 on signal interrupt
- Header File:

  #include<fcntl.h>

5) Creat:

- Used to create a new empty file.
- Syntax: int creat(char *filename, mode_t mode)
- Parameter :

  filename : name of the file which you want to create
  mode : indicates permissions of new file.
- Returns **:**

  return first unused file descriptor (generally 3 when first creat use in process beacuse 0, 1, 2 fd are reserved)
  return -1 when error

6) Lseek:

- The lseek() function repositions the offset of the open file associated with the file descriptor *fildes* to the argument *offset* according to the directive *whence*.
- Syntax: off_t lseek(int *fd*, off_t *offset*, int *whence*);
- Parameters:

  fd: file descriptor value
  offset: The offset of the pointer (measured in bytes).
  Whence: The method in which the offset is to be interpreted (relative, absolute, etc.). Legal values for this variable are provided at the end.
- Return:

  Returns the offset of the pointer (in bytes) from the beginning of the file. If the return value is -1, then there was an error moving the pointer.
- Header Files :

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

Programs:

1) Program to create a file

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
void main()
{
	int fd;
	fd=creat("first.txt",S_IRWXU);
	if(fd==-1)
	{
		printf("file not created");
	}
	else
	{
		printf("%d\n",fd);
	}
}
```

Output:

```
Rohits-MacBook-Pro:Practice rohitdandamudi$ subl creat.c
Rohits-MacBook-Pro:Practice rohitdandamudi$ cc creat.c
Rohits-MacBook-Pro:Practice rohitdandamudi$ ./a.out
3
```

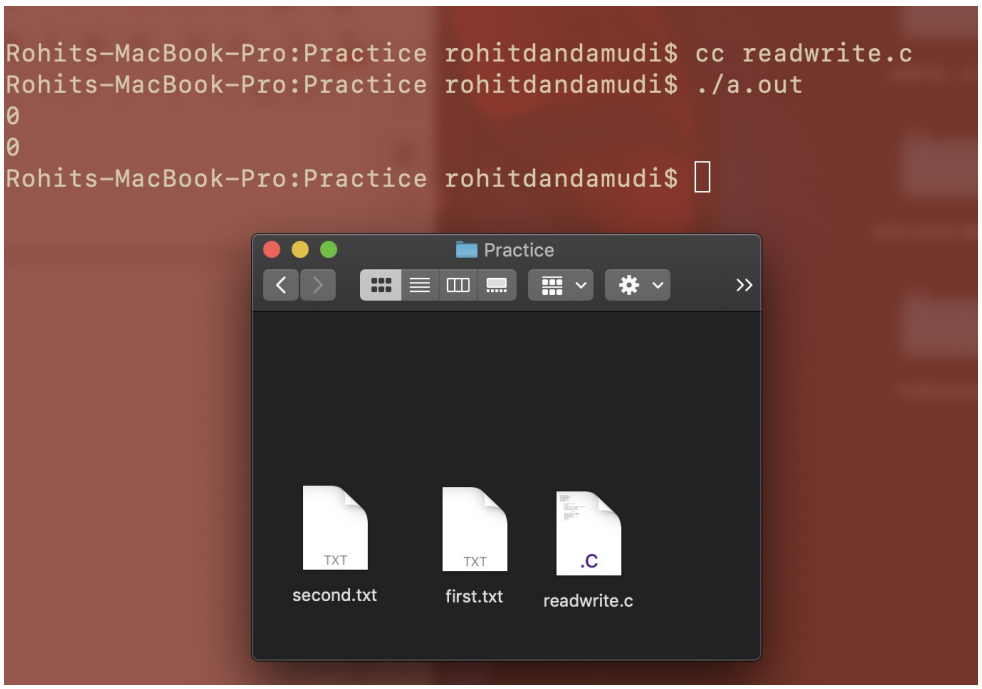
File Descriptor is returned

2) Program to read from a file and write into another file

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdlib.h>
#include<fcntl.h>
void main()
{

        int fd,fd1,a,b,charc=0;
        char ch;
        FILE *fptr;
        char *c = (char *) calloc(100, sizeof(char));
        fd=open("first.txt",O_RDONLY);
        fptr=fopen("first.txt","r");
        while((ch=fgetc(fptr))!=EOF)
        {
                charc++;
            }
        fd1=creat("second.txt",S_IRWXU);
        fd1=open("second.txt",O_WRONLY);
        a=read(fd,c,charc);
        b=write(fd1,c,charc);
        printf("%d\n%d\n",a,b);
        close(fd);
        close(fd1);
}
```

Output:

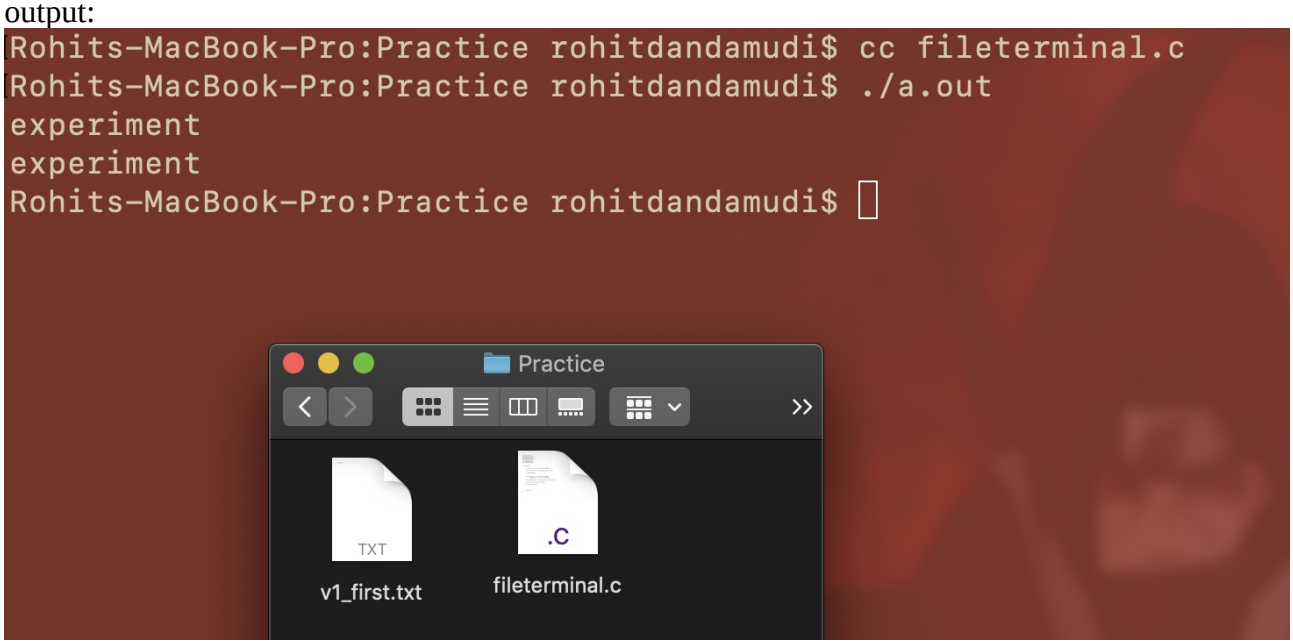


3) Program to take code from terminal

Code:

```
#include<stdio.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>


int main(){

    int fd1= creat("v1_first.txt",S_IRWXU);

    char *msg= (char *) calloc(100, sizeof(char));

    read(0,msg,100);

    int bytes=write( fd1, msg,strlen(msg));
        fd1=open("v1_first.txt",O_RDONLY);

    char *msg1= (char *) calloc(bytes, sizeof(char));

    bytes=read(fd1,msg1,strlen(msg));

    write(1,msg1,bytes);
    return 0;
}
```

output:



4) Program using lseek system call

Code:

```
// C program to read nth byte of a file and
// copy it to another file using lseek
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

void func(char arr[], int n)
{
	// Open the file for READ only.
	int f_write = open("start.txt", O_RDONLY);

	// Open the file for WRITE and READ only.
	int f_read = open("end.txt", O_WRONLY);

	int count = 0;
	while (read(f_write, arr, 1))
	{
		// to write the 1st byte of the input file in
		// the output file
		if (count < n)
		{
			// SEEK_CUR specifies that
			// the offset provided is relative to the
			// current file position
			lseek (f_write, n, SEEK_CUR);
			write (f_read, arr, 1);
			count = n;
		}

		// After the nth byte (now taking the alternate
		// nth byte)
		else
		{
			count = (2*n);
			lseek(f_write, count, SEEK_CUR);
			write(f_read, arr, 1);
		}
	}
	close(f_write);
	close(f_read);
}
```

```
int main()
{
        char arr[100];
        int n;
        n = 5;

        // Calling for the function
        func(arr, n);
        return 0;
}
```

Output: