

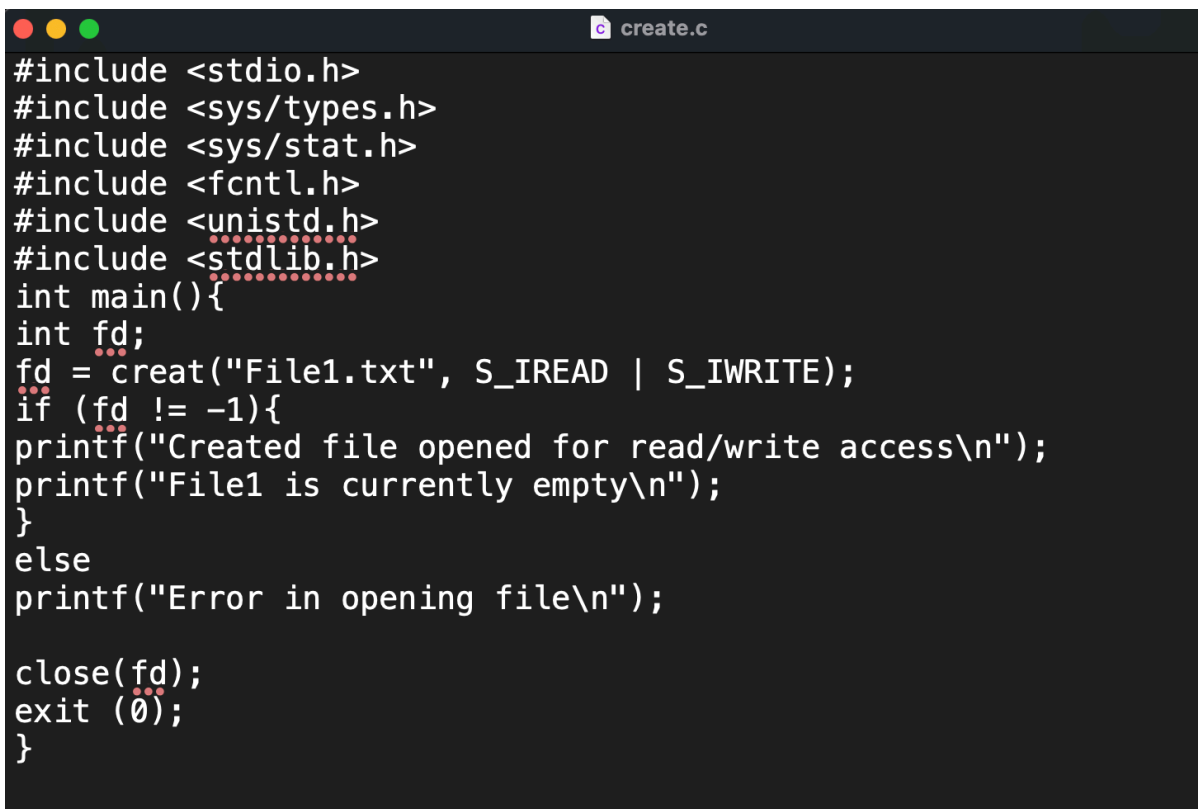
## Lab Experiment No. 1

### File-related System Calls Implementation

**Aim:** To get acquainted with how the system calls are invoked for file-related functionalities.

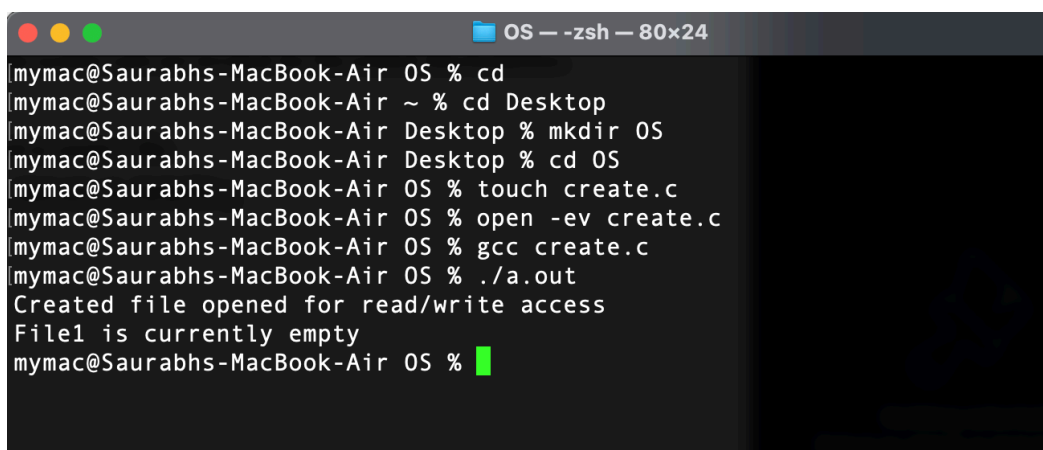
**1). creat () :** This system call is used to create a file.

**Syntax :** `int fd = creat( "filename" ,S_IREAD | S_IWRITE );`

A screenshot of a code editor window titled 'create.c'. The code is a C program that uses the 'creat' system call to create a file named 'File1.txt' with read and write permissions. It includes headers for stdio, sys/types, sys/stat, fcntl, unistd, and stdlib. The main function declares an integer 'fd', calls 'creat' with 'File1.txt' and 'S\_IREAD | S\_IWRITE', checks if the return value is not -1, and prints success or error messages. It then closes the file and exits.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
int main(){
    int fd;
    fd = creat("File1.txt", S_IREAD | S_IWRITE);
    if (fd != -1){
        printf("Created file opened for read/write access\n");
        printf("File1 is currently empty\n");
    }
    else
        printf("Error in opening file\n");

    close(fd);
    exit (0);
}
```

A screenshot of a terminal window titled 'OS - zsh - 80x24'. It shows a series of commands being executed to compile and run the 'create.c' program. The output shows the file 'File1.txt' being created and opened successfully.

```
mymac@Saurabhs-MacBook-Air OS % cd
mymac@Saurabhs-MacBook-Air ~ % cd Desktop
mymac@Saurabhs-MacBook-Air Desktop % mkdir OS
mymac@Saurabhs-MacBook-Air Desktop % cd OS
mymac@Saurabhs-MacBook-Air OS % touch create.c
mymac@Saurabhs-MacBook-Air OS % open -ev create.c
mymac@Saurabhs-MacBook-Air OS % gcc create.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
Created file opened for read/write access
File1 is currently empty
mymac@Saurabhs-MacBook-Air OS %
```

**2). open() :** Used to open a file. Returns a file descriptor specifying the position of this opened file in table of open files for current process.

**Syntax :** fd = open( path , flags , mode(optional));

Flags: O\_RDONLY, O\_RDWR, O\_APPEND, O\_WRONLY.

Mode: for permissions(User,Groups, other)

**3). read() :** Read from the file descriptor. returns the bytes read on success. Returns 0 when end of file is reached, returns -1 when error is occurred.

**Syntax :** size\_t read( int fd , void \* buff , size\_t count );

==> It attempts to read upto count bytes from file descriptor in to the buffer buff.

```
opening.c
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
static char message[] = "HELLO WORLD";
int main()
{
    int fd;
    char buffer[80];
    fd = open("file2.txt", O_RDWR | O_CREAT | O_EXCL, S_IRREAD | S_IWRITE);
    if (fd != -1)
    {
        printf("File has been opened for READ/WRITE access\n");
        write(fd, message, sizeof(message));
        lseek(fd, 0, 0); /* go back to the beginning of the file */
        if (read(fd, buffer, sizeof(message)) == sizeof(message))
            printf("\'%s\' Message has been written to file \n", buffer);
        else
            printf("error in reading file \n");
        close (fd);
    }
    else
        printf("File already exists\n");
    exit(0);
}
```

```
mymac@Saurabhs-MacBook-Air OS % gcc opening.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
File already exists
mymac@Saurabhs-MacBook-Air OS % gcc opening.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
File has been opened for READ/WRITE access
"HELLO WORLD" Message has been written to file
mymac@Saurabhs-MacBook-Air OS %
```

4). **lseek** : Position a pointer to a specified position in a file (Used for random access when reading or writing). Reposition read/write file affect.

**Syntax** : `lseek(int fd , off_t offset , int whence );`

```
seeking.c
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    int fd;
    long position;
    fd = open("file2.txt", O_RDONLY);
    if ( fd != -1)
    {
        position = lseek(fd, 0, 2); /* seek 0 bytes from end-of-file */
        if (position != -1)
            printf("The length of file2.txt is %ld bytes.\n", position);
        else
            perror("lseek error");
    }
    else
        printf("can't open file2.txt\n");
    close(fd);
}
```

```
mymac@Saurabhs-MacBook-Air OS % touch seeking.c
mymac@Saurabhs-MacBook-Air OS % open -ev seeking.c
mymac@Saurabhs-MacBook-Air OS % gcc seeking.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
The length of file2.txt is 12 bytes.
mymac@Saurabhs-MacBook-Air OS %
```

5). **Fork** : Creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.

```
forking.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    fork();
    printf("HELLO My Friends \n");
    return 0;
}
```

```
mymac — -zsh — 80x24
mymac@Saurabhs-MacBook-Air ~ % gcc forking.c
mymac@Saurabhs-MacBook-Air ~ % ./a.out
Hello from Parent!
Hello from Child!
mymac@Saurabhs-MacBook-Air ~ %
```

6). **pipe()** : Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a “virtual file”.

```
piping.c — Edited
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{
    int MSGSIZE = 16;
    char inbuf[MSGSIZE];
    int p[2];

    if (pipe(p) < 0)
        exit(1);

    write(p[1], msg1, MSGSIZE);
    write(p[1], msg2, MSGSIZE);
    write(p[1], msg3, MSGSIZE);

    for (int i = 0; i < 3; i++) {
        /* read pipe */
        read(p[0], inbuf, MSGSIZE);
        printf("%s\n", inbuf);
    }
    return 0;
}
```

```
1 error generated.
mymac@Saurabhs-MacBook-Air ~ % gcc piping.c
mymac@Saurabhs-MacBook-Air ~ % ./a.out
hello, world #1
hello, world #2
hello, world #3
mymac@Saurabhs-MacBook-Air ~ %
```

7). **wait()** : A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

```
waiting.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    pid_t cpid;
    if (fork()== 0)
        exit(0);          /* terminate child */
    else
        cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    return 0;
}
```

```
mymac — -zsh — 84x24
mymac@Saurabhs-MacBook-Air ~ % touch waiting.c
mymac@Saurabhs-MacBook-Air ~ % open -ev waiting.c
mymac@Saurabhs-MacBook-Air ~ % gcc waiting.c
mymac@Saurabhs-MacBook-Air ~ % ./a.out
Parent pid = 7969
Child pid = 7971
mymac@Saurabhs-MacBook-Air ~ %
```