

COMP 8567 Advanced Systems Programming

Unix File Input/Output

Winter 2023

Outline

- Introduction
- File Descriptors
- `open()` system call
- `read()` system call
- `write()` system call
- `lseek()` system call
- **`umask()`** system call and **`umask`** command
- Summary

Introduction

Most Unix File I/O can be performed using the system calls :

open(): to open a file

creat(): to create a new file or rewrite an existing one The **creat()** function is redundant. Its services are also provided by the **open()** function. It has been included primarily for historical purposes

read(): to read a specified number of bytes

write(): to write a specified number of bytes

lseek(): to explicitly position at a file offset

close(): to close a file

In contrast to the std I/O functions, the Unix I/O system calls are unbuffered (All characters are read or written in **one system call** and are not read/written character by character)

File Descriptors

- File descriptors : The kernel refers to any open file by a file descriptor- a nonnegative integer
- In particular, the standard input, standard output and standard error have descriptors 0, 1 and 2 reserved for them respectively (The file descriptors 0,1 and 2 **cannot** be allocated to a file created by user/programs)
- The symbolic constants, defined in <unistd.h> for these values are
 - STDIN_FILENO //0
 - STDOUT_FILENO //1
 - STDERR_FILENO // 2
- File descriptors range from 0 through FOPEN_ MAX (The maximum no of files that can be opened by a **process**)

```
#include <stdio.h>
#include <unistd.h>
//fmax.c
//Print some of the i/o constants defined in unistd.h

main()
{
printf("\n%d",STDIN_FILENO);
printf("\n%d",STDOUT_FILENO);
printf("\n%d",STDERR_FILENO);
printf("\n%d",FOPEN_MAX);

}
```

open() system call

- **Synopsis** : `int open(const char * pathname, int oag , [int mode])`
- Opens the file specified by *pathname* (can be absolute or relative)

Returns the file descriptor if OK, -1 otherwise.

The argument *oag* is formed by OR'ing together 1 or more of the following constants (in `<fcntl.h>`)/file control options

- `O_RDONLY`: Open for reading only
- `O_WRONLY`: Open for writing only
- `O_RDWR`: Open for reading and writing only
- `O_APPEND`: Open for writing after the end of file (For all write operations)
- `O_CREAT`: Create a file

Note that the third argument, **only** used when a file is created, supplies the file's **initial permission flag settings, as an octal value** (Ex: 0700)

Examples :

```
if ((d=open("/home/pranga/chapter4/check.txt", O_RDONLY))!=-1)
error_routine();
```

Both absolute and relative paths (for the filename) can be used

```
if ((d=open("check.txt", O_RDONLY))!=-1)//Continue with file operations;
```

```
d=open(name,O_CREAT| O_RDWR, 0700)
```

File Permissions:

User Group Others

RWE RWE RWE

111 000 000 (0700)

In this case, 0700 value for mode provides all rights to the owner of this file and **no permission to group and others**

Other Example values for mode :

0400: Allows read by owner

0200: Allows write by owner

0100: Allows execute by owner

0040: Allows read by group

0004: Allows read by others

0777: Allows read/write/execute by all

Check umask and chmod later.

```
//open.c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
//Tries to open a file and displays an error if the file cannot be opened in the specified mode
//Prints the file descriptor (a non-negative integer) if open() is successful 1

int main(void)
{
int fd1=open("new.txt",O_CREAT|O_RDWR,0700);
if(fd1==-1)
printf("\n The operation was not successful\n");
else
printf("\n The file descriptor is %d\n",fd1);
close(fd1);
}
```


read() and write() system calls

read() synopsis:

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

Reads as many bytes as it can, possibly up to nbyte, and returns the number of bytes actually read.

ssize_t and size_t are usually defined as **long integers**

Example: `long int n= read(fd3,buff1,200);` // fd3 is the file descriptor obtained by opening check.txt successfully

The value returned by read() can be :

- -1 : in case of an error
- smaller than nbyte : the number of bytes remaining before the end of file was less than the nbyte specified //Ex: if the no of bytes remaining before the end of the file is 150
- 0 : (if the file exists, but has no characters)

write() synopsis:

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

Example: `long int n= write(fd3,buff1,200);` //writes the contents of buff1 into check.txt

write returns nbyte if OK and -1 otherwise.

```
//br1.c
```

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

```
int main(void)  
{
```

```
//Reads from check.txt into an array of characters
```

```
int fd3=open("check.txt",O_RDONLY);  
char *buff1;  
long int n;  
n=read(fd3,buff1,300);  
printf("\the number of bytes read is %d\n", n);
```

```
printf("%s", buff1);
```

```
close(fd3);  
}
```

//bw1.c

//Writes an array of characters into check.txt

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

int main(void)

{

int fd3=open("check.txt",O_RDWR);

char *buff1="Hello";

long int n;

n=write(fd3,buff1,5);

printf("\n\nThe number of bytes written were %ld\n",n);

n=write(fd3,buff1,5);

printf("\n\nThe number of bytes written were %ld\n",n);

n=write(fd3,buff1,5);

printf("\n\nThe number of bytes written were %ld\n",n);

close(fd3);

}

//bow.c

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

int main(void)
{
    //writes and array of characters (overwrites if already present)
    int fd3=open("check.txt",O_RDWR);
    char *buff1="*****";
    long int n;
    n=write(fd3,buff1,20);

    printf("\n\nThe number of bytes written were %ld\n",n);
    close(fd3);

}
```

close(fd)

Note that **int close(int fd)** frees the file descriptor fd.

close() returns 0 when OK and -1 otherwise. For example, -1 will be returned if fd was already closed.

lseek() system call

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

Returns the **resulting offset** if OK, -1 otherwise.

//Resulting offset is always from the beginning of the file

The return type `off_t` is a long integer.

it sets the file pointer(position) associated with the open file descriptor specified by the file descriptor `fd` as follows:

- If `whence` is `SEEK SET`, the pointer is set to offset bytes //From the beginning of the file
- If `whence` is `SEEK CUR`, the pointer is set to its current location plus offset.
- If `whence` is `SEEK END`, the pointer is set to the **size of the file** plus offset
 - // Includes the **Coded character set identifier (CCSID)** which is an 8-bit (1 Byte) code for UTF encoding
 - // UTF (Unicode transformation format)

These three constants are defined in `<unistd.h>`

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

//ls1.c with SEEK_SET

```
main()
{
int fd3=open("check.txt",O_RDWR);
int long n=lseek(fd3,10,SEEK_SET);
printf("\nThe resulting offset is %d\n",n);
char * buff1="COMP 8567";
n=write(fd3,buff1,9);
printf("\nThe no of bytes written from the resulting offset is
%d\n",n);
close(fd3);
}
```

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

//ls2.c with SEEK_CURR

```
main() {

    int fd3=open("check.txt",O_RDWR);
    int long n=lseek(fd3,10,SEEK_SET);
    printf("\nThe resulting offset is %d\n",n);
    char * buff1="COMP 8567";
    n=write(fd3,buff1,9);
    printf("\nThe no of bytes written from the resulting offset is
    %d\n",n);

    //SEEKCUR
    n=lseek(fd3,5,SEEK_CUR);
    printf("\nThe resulting offset is %d\n",n);
    n=write(fd3,buff1,9);
    printf("\nThe no of bytes written from the resulting offset is
    %d\n",n);
    close(fd3);
}
```



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

//ls3.c with SEEK_END

```
main()
{
int fd3=open("check.txt",O_RDWR);
int long n=lseek(fd3,10,SEEK_SET);
printf("\nThe resulting offset is %d\n",n);
char * buff1="COMP 8567";

n=write(fd3,buff1,9);
printf("\nThe no of bytes written from the resulting offset is
%d\n",n);

//SEEKCUR
n=lseek(fd3,5,SEEK_CUR);
printf("\nThe resulting offset is %d\n",n);
n=write(fd3,buff1,9);
printf("\nThe no of bytes written from the resulting offset is
%d\n",n);
```

```
//SEEKEND
```

```
n=lseek(fd3,10,SEEK_END);

printf("\nThe resulting offset is %d\n",n);

n=write(fd3,buff1,9);

printf("\nThe no of bytes written from the
resulting offset is %d\n",n);

close(fd3);

} //end main
```

umask() system call and umask command

- System call `umask()` and command `umask` allow the settings of the user mask that controls newly created file permissions.
- Each mask digit is negated, then applied to the file permission/default permission using a **logical AND** operation.

Ex: If the user has requested the file permission 0666 (110 110 110) in `open()`

and If `umask` is 0022 (000 010 010), permission of the newly created file would be (110 100 100)

i.e. //Negation of mask (111 101 101) **AND** (110 110 110) = 110 100 100

- **umask() system call Synopsis:**
- `mode_t umask(mode_t mask);`
 - `umask()` sets the calling process's file mode creation to `(!mask & mode)`
 - Ex: if mask is 0055 (000 101 101) and the mode is 0777 (111 111 111) , the new file would be created with permission 0755
(111 101 101) 111 010 010 111 010 010 0722
- Need header files:
 - `<sys/types.h>` and `<sys/stat.h>`

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

```
//umaskex.c
```

```
main()
```

```
{
```

```
int fd1=open("check24.txt",O_CREAT|O_RDWR,0777);
```

```
}
```

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

```
//umaskex1.c
```

```
main()
```

```
{
```

```
umask(0000); //system call within a program, overrides the previously set mask (in the
command line)
```

```
int fd1=open("check24.txt",O_CREAT|O_RDWR,0777);
```

```
}
```

umask –Linux command

- Command umask does a similar job as the system call umask()
- Synopsis: umask [-S] [mask]
- When option -S is present, accept symbolic representation of mask // **\$umask -S**
- When no mask is provided, umask returns the current user mask.
- Examples: umask -S g+w: allows write permission for my group, if requested.
- umask 0000: makes your mask neutral
- umask 0077: no permission for your group and others.
- umask acts as a safety measure that disables some permissions when files are created (**however, they can be changed later using chmod**)

Sample chmod and umask commands

- `$ chmod g+w check24.txt`
- `$ chmod g-w check24.txt`
- `$ umask -S u-w`
- `$ umask -S u+w`
- `$ umask -S g+w`

Summary

- Introduction
- File Descriptors
- `open()` system call
- `read()` system call
- `write()` system call
- `lseek()` system call
- **`umask()`** system call and **`umask`** command