Assembly - File Management

The system considers any input or output data as stream of bytes. There are three standard file streams −

- Standard input (stdin),
- Standard output (stdout), and
- Standard error (stderr).

File Descriptor

A **file descriptor** is a 16-bit integer assigned to a file as a file id. When a new file is created or an existing file is opened, the file descriptor is used for accessing the file.

File descriptor of the standard file streams - **stdin, stdout** and **stderr** are 0, 1 and 2, respectively.

File Pointer

A **file pointer** specifies the location for a subsequent read/write operation in the file in terms of bytes. Each file is considered as a sequence of bytes. Each open file is associated with a file pointer that specifies an offset in bytes, relative to the beginning of the file. When a file is opened, the file pointer is set to zero.

File Handling System Calls

The following table briefly describes the system calls related to file handling −

%eax	Name	%ebx	%ecx	%edx
2	sys_fork	struct pt_regs	-	-
3	sys_read	unsigned int	char *	size_t
4	sys_write	unsigned int	const char *	size_t
5	sys_open	const char *	int	int
6	sys_close	unsigned int	-	-
8	sys_creat	const char *	int	-
19	sys_lseek	unsigned int	off_t	unsigned int

The steps required for using the system calls are same, as we discussed earlier −

- Put the system call number in the EAX register.
- Store the arguments to the system call in the registers EBX, ECX, etc.
- Call the relevant interrupt (80h).
- The result is usually returned in the EAX register.

Creating and Opening a File

For creating and opening a file, perform the following tasks −

- Put the system call sys_creat() number 8, in the EAX register.
- Put the filename in the EBX register.
- Put the file permissions in the ECX register.

The system call returns the file descriptor of the created file in the EAX register, in case of error, the error code is in the EAX register.

Opening an Existing File

For opening an existing file, perform the following tasks −

- Put the system call sys_open() number 5, in the EAX register.
- Put the filename in the EBX register.
- Put the file access mode in the ECX register.
- Put the file permissions in the EDX register.

The system call returns the file descriptor of the created file in the EAX register, in case of error, the error code is in the EAX register.

Among the file access modes, most commonly used are: read-only (0), write-only (1), and read-write (2).

Reading from a File

For reading from a file, perform the following tasks −

- Put the system call sys_read() number 3, in the EAX register.
- Put the file descriptor in the EBX register.
- Put the pointer to the input buffer in the ECX register.
- Put the buffer size, i.e., the number of bytes to read, in the EDX register.

The system call returns the number of bytes read in the EAX register, in case of error, the error code is in the EAX register.

Writing to a File

For writing to a file, perform the following tasks −

- Put the system call sys_write() number 4, in the EAX register.
- Put the file descriptor in the EBX register.
- Put the pointer to the output buffer in the ECX register.
- Put the buffer size, i.e., the number of bytes to write, in the EDX register.

The system call returns the actual number of bytes written in the EAX register, in case of error, the error code is in the EAX register.

Closing a File

For closing a file, perform the following tasks −

- Put the system call sys_close() number 6, in the EAX register.
- Put the file descriptor in the EBX register.

The system call returns, in case of error, the error code in the EAX register.

Updating a File

For updating a file, perform the following tasks −

- Put the system call sys_lseek () number 19, in the EAX register.

- Put the file descriptor in the EBX register.
- Put the offset value in the ECX register.
- Put the reference position for the offset in the EDX register.

The reference position could be:

- Beginning of file - value 0
- Current position - value 1
- End of file - value 2

The system call returns, in case of error, the error code in the EAX register.

Example

The following program creates and opens a file named *myfile.txt*, and writes a text 'Welcome to Tutorials Point' in this file. Next, the program reads from the file and stores the data into a buffer named *info*. Lastly, it displays the text as stored in *info*.

```
section .text
   global _start         ;must be declared for using gcc

_start:                  ;tell linker entry point
   ;create the file
   mov  eax, 8
   mov  ebx, file_name
   mov  ecx, 0777        ;read, write and execute by all
   int  0x80             ;call kernel

   mov [fd_out], eax

   ; write into the file
   mov  edx,len          ;number of bytes
   mov  ecx, msg         ;message to write
   mov  ebx, [fd_out]    ;file descriptor
   mov  eax,4            ;system call number (sys_write)
   int  0x80             ;call kernel

   ; close the file
   mov eax, 6
   mov ebx, [fd_out]

   ; write the message indicating end of file write
```

```
   mov eax, 4
   mov ebx, 1
   mov ecx, msg_done
   mov edx, len_done
   int  0x80

   ;open the file for reading
   mov eax, 5
   mov ebx, file_name
   mov ecx, 0              ;for read only access
   mov edx, 0777           ;read, write and execute by all
   int  0x80

   mov  [fd_in], eax

   ;read from file
   mov eax, 3
   mov ebx, [fd_in]
   mov ecx, info
   mov edx, 26
   int 0x80

   ; close the file
   mov eax, 6
   mov ebx, [fd_in]
   int  0x80

   ; print the info
   mov eax, 4
   mov ebx, 1
   mov ecx, info
   mov edx, 26
   int 0x80

   mov  eax,1              ;system call number (sys_exit)
   int  0x80               ;call kernel

section .data
file_name db 'myfile.txt'
msg db 'Welcome to Tutorials Point'
len equ  $-msg
```

```
msg_done db 'Written to file', 0xa
len_done equ $-msg_done

section .bss
fd_out resb 1
fd_in  resb 1
info resb  26
```

When the above code is compiled and executed, it produces the following result −

```
Written to file
Welcome to Tutorials Point
```