ASSEMBLY - SYSTEM CALLS

System calls are APIs for the interface between the user space and the kernel space. We have already used the system calls. sys_write and sys_exit, for writing into the screen and exiting from the program, respectively.

Linux System Calls

You can make use of Linux system calls in your assembly programs. You need to take the following steps for using Linux system calls in your program −

- 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.

There are six registers that store the arguments of the system call used. These are the EBX, ECX, EDX, ESI, EDI, and EBP. These registers take the consecutive arguments, starting with the EBX register. If there are more than six arguments, then the memory location of the first argument is stored in the EBX register.

The following code snippet shows the use of the system call sys_exit −

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

The following code snippet shows the use of the system call sys_write −

```
mov edx,4  ; message length
mov ecx,msg  ; message to write
mov ebx,1  ; file descriptor (stdout)
mov eax,4  ; system call number (sys_write)
int 0x80  ; call kernel
```

All the syscalls are listed in */usr/include/asm/unistd.h*, together with their numbers $thevaluetoputinEAXbeforeyoucallint80h$.

The following table shows some of the system calls used in this tutorial −

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	int	-	-	-	-
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	-	-	-	-

Example

The following example reads a number from the keyboard and displays it on the screen −

```
section .data                           ;Data segment
```

```
   userMsg db 'Please enter a number: ' ;Ask the user to enter a number
   lenUserMsg equ $-userMsg             ;The length of the message
   dispMsg db 'You have entered: '
   lenDispMsg equ $-dispMsg

section .bss           ;Uninitialized data
   num resb 5

section .text          ;Code Segment
   global _start

_start:                ;User prompt
   mov eax, 4
   mov ebx, 1
   mov ecx, userMsg
   mov edx, lenUserMsg
   int 80h

   ;Read and store the user input
   mov eax, 3
   mov ebx, 2
   mov ecx, num
   mov edx, 5          ;5 bytes (numeric, 1 for sign) of that information
   int 80h

   ;Output the message 'The entered number is: '
   mov eax, 4
   mov ebx, 1
   mov ecx, dispMsg
   mov edx, lenDispMsg
   int 80h

   ;Output the number entered
   mov eax, 4
   mov ebx, 1
   mov ecx, num
   mov edx, 5
   int 80h

   ; Exit code
   mov eax, 1
   mov ebx, 0
   int 80h
```

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

```
Please enter a number:
1234
You have entered:1234
```