Activity 5: System call with Assembly Language and Processes in Linux

Objective

1. The aim of this activity is to give students know the system call in Ubuntu (Linux OS).
2. The second is to give the experience writing a code with assembly language to call the system call.

5.1 What is System Call?

System call is a function provide by OS for the application request services. Linux in kernel 2.0 has number of the system call at 190 functions in 32bit OS, from table 5.1 shown only first 15 system calls. This activity uses system call no. 1 and 4.

Table 5.1 Example of system calls in Linux 32bit

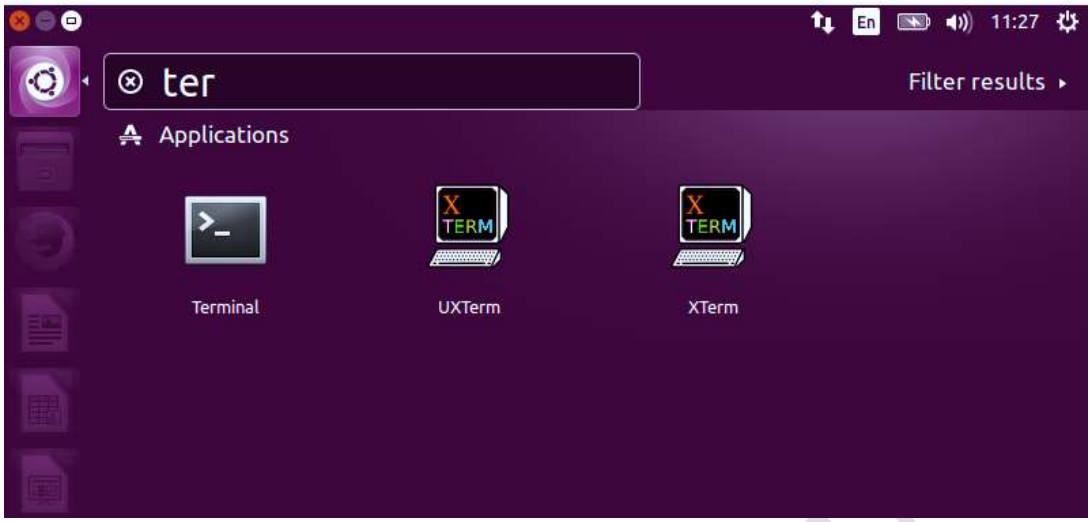
No / System call	Short description
0x00 sys_setup	Call function filesystem.c
0x01 sys_exit	Terminate the current process
0x02 sys_fork	Create a child process
0x03 sys_read	Read from a file descriptor
0x04 sys_write	Write to a file descriptor
0x05 sys_open	Open if possibly create a file or device
0x06 sys_close	Close a file descriptor
0x07 sys_waitpid	Wait for process termination
0x08 sys_creat	Create a file or device
0x09 sys_link	Make a new name for a file
0x0a sys_unlink	Delete a name and possibly the file it refers to
0x0b sys_execve	Execute program
0x0c sys_chdir	Change working directory
0x0d sys_time	Get time in seconds
0x0e sys_mknod	Create a directory or special or ordinary file
0x0f sys_chmod	Change permissions for a file

Before staring the activity, we should look at the related UNIX commands for this activity at table 5.2,

Table 5.2 List of Unix commands related in this activity.

Command pattern	Description
sudo apt install nasm	Sudo command allows permissions to the user or code scripts to run as a super user. Sudo command is used to install a module to the system. The example shows the command for installing nasm (The Netwide Assembler).
gedit file	Open editor to edit a file.
nasm –f elf hello.asm	Netwide Assembler converts hello.asm to an object file. The argument "-f" sets the format of the compiler by elf32: 32bits Linux system, elf64: 64bits Linux system, win32: 32bits Microsoft Windows win64: 64bits Microsoft Windows
ld –o executefile –m elf_i386 –s objfile	A command links the object file and archive files to create an execute file.
ls –l	Show list of filenames and details
./executedfile **&**	Running the execution file informs the background process
kill pid	Kill a process
kill –9 pid	Kill a process with strong command
ps	Display process information and the process ID
rm filname	Delete a file
cd directoryname	Move to a directory
gcc –o file_exe file_cpp	Compile a C language file to output an execute file.

1) The activity starts by opening Ubuntu OS (This activity material provided VirtualBox installed Ubuntu 16.3) and running Terminal app as shown in the figure below.



2) In some system not installed nasm, installation runs a command,

```
$  sudu  apt  install  nasm
```

3) Open gedit app is an editor to write the code from below.

```
$  gedit hello.asm
```

```
section .text
global _start
_start:
    mov edx,len                 ;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
    mov eax,1                   ;system call number (sys_exit)
    int 0x80                    ;call kernel

section .data
    msg db 'Hello, world!',0xa ;our dear string
    len equ $ - msg             ;length of our dear string
```

4) The assembly code is converted to machine language in from the hex code with commands as below. First command compiles hello.asm to hello.o called the object file and the second links hello.o and system library to make an execution file.

```
$ nasm -f elf hello.asm
$ ld -o hello_exe -m elf_i386 -s hello.o
```


5) Uses “ls” command to show the list of file the current directory.

```
$ ls -l
```

```
-rw-rw-r-- 1 csos csos 133 ต.ค.   9 13:52 hello2.cpp
-rw-rw-r-- 1 csos csos 179 ต.ค.   9 12:35 hello.asm
-rwxrwxr-x 1 csos csos 360 ต.ค.   9 14:43 hello_exe
-rw-rw-r-- 1 csos csos 624 ต.ค.   9 12:35 hello.o
csos@csos-VirtualBox:~/Desktop$
```

6) Run the execute file hello_exe with command,

```
$ ./hello_exe
```

7) Record the output of the program below,

_____

_____

5.2 Start writing C language in UNIX

This activity shows you in two things, writing C language in Linux and process managements. First starts with writing code in C language.

1) Open the terminal app and type,

```
gedit  hello2.cpp
```

2) Write the code the figure below.

```
#include <stdio.h>
int main(void)
{
      int i;
      for(i=0; i<10;i++)
            printf(“Hello World %d”,i);
      return 0;
}
```

3) Compile the code with gcc command,

```
gcc  -o  hello2_exe   hello2.cpp
```

4) Run program with command,

```
./hello_exe
```


5) Check output of the program and write the result.

_____

_____

5.3 The background processes and killing an unused process

1) Write the program as the code below by saving the execute file in name “nothing_exe”

```
#include <stdio.h>
int main(void)
{
	while(1)
	{
		int i;
		for(i=0; i<10000;i++)
			i=i;   //spending time by do something.
	}
	return 0;
}
```

2) Execute the program with a command below, by running four times.

```
./nothing_exe  &
```

3) Use **ps** and **top** command to check the running process, and recored PID number of each process.

```
$ ps
$ top
```

_____

_____

4) Use kill command to kill the running process. If some process is locked (it cannot terminate by the kill command), you try “kill -9 pid”

```
$ kill pid
$ kill -9 pid
```

5.4 Questions

1) What is the process detail that presented by top command?

2) What is effect when a user kills a process by typing wrong PID or giving PID of system process?

3) What is process of the program from the code at line 7 to 10 from the activity 4.1?

4) From activity 4.1, we study system call 0x01 and 0x04 that is in 32bit UNIX. Given student compare the system call of 64bit UNIX, by searching the list of 64bit system call from Internet, and give the detail at below.

===END===

