CO313 – Operating Systems **Lab1 – Simple system library functionality**	
Lab date: 7th March 2013	**Due date: 15th March 2013 @ 12:00PM** **4% penalty per day there after until 22nd March**

Aim: The aim of this lab is to explore the calling convention of a C language program and the system call interface of the Linux kernel.

Objectives: at the end of this practical you should be able to:
- Write simple assembly programs for x86
- Link assembly programs with C programs
- Understand the argument passing mechanism used by the Linux system calls/C-programs,
- Understand the need for system libraries.

There are two parts for this practice: written questions and coding exercise. For both parts you should use the *lab1.tgz* tarball available via CMS.

Part I (Written questions):
Answer the following questions. They relate to the code given in the l*ab1.tgz* tarball.
1. Read and try to understand the *Makefile*. What is the purpse of make?
2. What will happen if you type "*make clean*"? How is this specified? (hint: http://www.gnu.org/software/make/manual/make.html)
3. Read the *lib.S* file. What is the language used to write this?
4. What is the purpose of "global" inside the *lib.S?*
5. Can you call an assembly function from C? If so what are the things that you need to worry about?

Part II (Coding exercise):

Your main function (see *main.c*) calls a function called *my_print.* The *my_print* function takes two arguments: a pointer to a string that is to be printed and the number of characters in that string.

Since *main.c* is written in C, it uses the C calling convention while passing arguments. For x86 the C calling convention passes arguments on the stack. So when you reach the *my_print* function the arguments are placed on the stack – current stack pointer points to the return address, first argument is 4bytes above the current stack (that is %esp + 4 this 4 is because the return address is 32bits) the second is 8bytes above the current stack (because the first argument is 32bits).

The Linux kernel on the other hand expects system call arguments to be in registers. For the write system call the register contents should be as follows:

Register	Value	General usage of register
%eax	System call number	Should contain *syscall*

	These numbers are given in /usr/include/asm/unistd_32.h file	number in call cases
%ebx	Should contain the file descriptor telling where to write. If it is the screen (standard out) file descriptor is 1	1st argument for (write) system call
%ecx	Should contain the pointer of the string which is to be written.	2nd argument for (write) system call
%edx	Should contain the size of the string which is to be written	3rd argument for (write) system call

Once arguments are loaded into registers you can perform the system call by executing *int $0x80* instruction.

Task1:
Your first task is to complete the *my_print* function. It should print the given string on to the screen and return 0 on success or -1 otherwise.

Here you may assume that print was successful if *write* system call has written the same amount of characters as what was asked of it.

Task2:
There are number of system calls that requires no argument except the system call number. For example, fork(), getpid(), getppid() etc. All these system calls are *void* and returns a 32bits (which can be interpreted in different ways by the calling program).

In the *lib.S* file there is a function call *generic_syscall.* The interface for *generic_syscall* is given in *mylib.h*. You should decide how to implement the *generic_syscall* function in the *lib.S* taking into consideration the C calling convention.

Have a look at functions implemented in *syscalls.c.* Your objective is to implement them using the *generic_syscall* function provided by *lib.S.* (You are not allowed to use the standard libraries that comes with the C distribution).

You can test your implementation with the *fork.c* program. To compile just type “make fork”.

Submission

Submit the completed *lib.S* file and *syscalls.c* to CMS on or before the deadline. You should make sure the code is working before submitting. Test the code by modifying the code.