# Lab 5: Reading Material

- Read the chapter Writing Procedures from the book Dandamudi, Sivarama P.: Guide to Assembly Language Programming in Linux.
- A list of the instruction set for IA-32.
- Introduction to system calls.
- Linux system calls table

# Functions and System Calls

## Objectives

- Function calls in assembly.
- System calls in assembly.
- Handling command line arguments in assembly.
- Addressing modes.
- Using the stack.

## Task 0: Basic interaction with Unix system calls

You should accomplish this task **before** attending the lab session.

Write a program, *prog*, in C programming language that receives a file name as an argument from the command line and prints it's corresponding file descriptor with an informative sentence. The program must close the file associated with this file descriptor and prints CLOSING DONE if the closing process succeeded, otherwise, it prints CLOSING FAILED.

For example:

```
$> prog makefile
 The corresponding file descriptor is 3
 CLOSING DONE

Assumption: the current directory contains a file with the name 'makefile'
```

To make the task more challenging, the implementation of the program MUST:

- Do the file opening and closing using open and close Unix system calls. (NOT fopen and fclose) Your c code calls these 2 functions, which you must implement in assembly, see system calls in the reading material.
- System calls work with the CPU registers, so you need to initialize the appropriate registers. Use Assembly code to initialize the relevant registers for `open` and `close`.

- Do NOT use any standard library function except *printf* function (for debugging purposes) from *stdio* library.

## Lab instructions

In this lab you are going to learn how to mix Assembly and C code. You will build modules in both languages and make them interact with each other through procedures.

## Task 1a: Unix system calls and command line parameters handling

In the lwc.zip archive file you will find a program that counts lines, words and characters in text files. The program is written in C programming language.

```
$> lwca lwc.c
102 lines
251 words
1774 chars
```

In this task you are required to complete the calls for the Unix system calls. Moreover, you need to complete the implementation of the ***_start*** label in the ***start.asm*** file. Your code is expected to prepare the arguments that main expects on the stack, namely, your code will mimic the gcc behavior.

To test your code, use the provided makefile which uses the linker only instead of using the ***gcc***. If your implementation is not accurate, the C program will not receive the command line parameters appropriately, and FAIL as a consequence.

### Task 1b

Implement the functions utoa_s and atou_s in assembly. The function utoa_s receives an int (one digit or more) and converts it into a string. Change your c program to use the function you just implemented instead of utoa, and make sure it runs as expected.
The function atou_s receives a string, and converts it to int. You may assume that the string represents valid integer number, in the range of int (i.e. 4-bytes signed integer range).

### Task 1c

Take your implementation of utoa_s, and run it with 123, illustrating the changes that happen to the stack and registers. Yes, you need to draw the stack and the registers that you use.
Take your implementation of atou_s, and run it with '5678', illustrating the changes that happen to the stack and registers.

## Task 2: Functions, Arguments and the Stack

In this task you are expected to write a pure Assembly implementation of (partial) lwc. You already implemented the necessary system calls, _start, and the utoa function.

### Task 2a: Read from file and output to stdout

Write a pure assembly code that reads the content of the file using a buffer of size>=50 (you need to read in a loop until there's nothing left to read), and writes each buffer it reads to the standard output (stdout).

### Task 2b

Write a pure assembly code that reads the content of a file in a buffer of size>= 50, like in task 2a, if '-w' is received, counts the number of words that appear in the file and outputs it to the standard output. You can assume that '-w' comes before the filename (if -w is not received - execute task 2a).
Add an option ‘-ws <word>’ to count some specific word appearances.
For example, ‘-ws Hi’ should count how many times a word ‘Hi’ (note that this is case sensitive) appears in the given file.

Mandatory Requirements:

- Print an appropriate message and exit the program, using the system call exit, if any of the system calls fail.
- Print the count using the system call write, and the utoa function you implemented in task 1b.
- After creating an o file with nsam, use the following line to run the linker and create an executable: `ld -melf_i386 -o <runnable file name> <o file>`.

## Submission

- SUBMIT a ZIP file at the end of the lab session .
- Your zip file should include the following files: task1a.s, task1b.s, lwc.c, task2a.s, task2b.s, makefile (which contains targets for creating each of the tasks and subtasks).

Good luck!