# CS 333: Operating Systems Lab
# Autumn 2018

# Lab 1: Hello OS!

### Important instructions

- Login as `labuser` on SL2 machines for this lab.
- This file is part of lab1.tar.gz archive which contains multiple directories with programs associated with exercise questions given below.

## Part 1: The OS view

The goal of this part of the lab is to get familiar with (Linux) tools and files used for system and process behaviour information, monitoring and control.

### Important tools

Following are some basic Linux tools. The first step of this lab is to get familiar with the usage and capabilities of these tools.
To know more about them use: `man <command>`. Start with `man man`.

- **top**
  `top` provides a continuous collective view of the system and operating system state. For example, list of all processes, resource consumption each process, system-level CPU usage etc. The system summary information displayed, order etc. has several configurable knobs. `top` also allows to send signals to processes (change priority, stop etc.)
- **ps**
  The `ps` command is used to view the processes running on a system. It provides a snapshot of the processes along with detailed per process information like process-id, cpu usage, memory usage, command name etc. Several has several flags to display different types of process information, e.g., executing `ps` without arguments will not show all processes on the system, but a combination of flags as input parameters will.
- **iostat**
  `iostat` is a command useful for monitoring and reporting CPU and statistics related to devices. `iostat` creates reports that can be used to change system configuration for better balance the input/output between physical disks. For example, the command reports total activity and rate of activities (read/write) to each disk/partition, can be configured to monitor continuously (after every specified interval).
- **strace**
  `strace` is a diagnostic and debugging tool used to monitor the interactions between processes and the Linux kernel. The tool traces the set of functions (system calls / calls of the Application Binary Interface) and signals (events) used by a program to communicate with the operating system.
- **lsof**
  `lsof` is a tool used to list open files. The tool list details of the file itself and details of users, processes which are using the files.
- **lsblk**
  `lsblk` is a tool used to list information about all available block devices such as hard disk, flash drives, CD-ROM etc.
- Also look up the following commands: `pstree, lshw, lspci, lscpu, dig, netstat, df, du, watch`.
  Note that some these programs may need root privileges.

### The /proc file system

The /proc file system is a mechanism provided by Linux, for communication between userspace and the kernel using the file system interface. Files in the /proc directory report values of several kernel parameters and also can be used for configuration and (re)initialization. The /proc file system is nicely documented in the man pages, — man proc. Understand the system-wide proc files such as meminfo, cpuinfo, etc and process related files such as status, stat, limits, maps etc. System related proc files are available in the directory /proc/, and process related proc files are available at /proc/<process-id>/

### Exercises

1. Collect the following basic information about your machine using the /proc file system and answer the following questions:
   (a) How many CPU sockets, cores, and CPUs does the machine have ?
   (b) What is the frequency of each CPU ?
   (c) How much memory does your machine have ?
   (d) How much of it is free and available? What is the difference between them?
   (e) What is total number of user-level processes in the system?
   (f) How many context switches has the system performed since bootup?
   (g) What is the size of files in the /proc directory? Frame a question of investigation based on the file size observation.
2. Run all programs in the subdirectory memory and identify memory usage of each program. Compare the memory usage of these programs in terms of VmSize & VmRSS and justify your results based on the code.
3. Run the executable subprocesses provided in the sub-directory subprocess and provide your roll number as command line argument.
   Find the number of sub processes created by this program. Describe how you obtained the answer.
4. Run strace along with the binary program of empty.c given in subdirectory strace.
   What do you think the output of strace indicates in this case? How many different (system call) functions do you see?

   Next, use strace along with another binary program of hello.c (which is in the same directory).
   Compare the two strace outputs,
   (a) Which part of the strace output is common, and which part has to do with the specific program?
   (b) List all unique system call functions for each program and look up functionality of each.
5. Run the executable openfiles in subdirectory files.
   List the files which are opened by this program, and describe how you obtained the answer.
6. Find all the block devices on your system, their mount points and file systems present on them. A mount point is a file system directory entry from where a disk can be accessed. A file system described how data is organized on a disk. Describe how you obtained the answer.

## Part 2: Booting unraveled

The goal of this part of the lab is to learn about how a computer boots, and write a dummy operating system. The question of interest here is, when a machine is powered on, how does it load the operating system and its components? where is the kernel stored? which files to read and execute? etc.
The answer to this lies with the idea of loading a portion of data from disk in memory, and executing the corresponding contents. The road to world peace via operating systems starts here. If we can find this special block of data on disk and make sure that contents of the disk contain the codes for world

peace, we are all set. In other works, this special block is the entry point to seize control of the hardware and for the operating system to perform its magic.
When a computer starts, a special program called the Basic Input/Output System (BIOS) is loaded from a chip in to the main memory. The BIOS detects connected hardware devices, resets them, tests them etc. and also looks for the *special* sector (the boot sector) on available disks to load the operating system.
The BIOS reads the first sector of each disk (one by one) and determines whether its is a boot disk (a disk with an operating system). A boot disk is detected via a magic number **0xaa55**, stored as the last two bytes of the boot sector of a disk.

7 The `boot_sector1.asm` file, in the `myos` directory, shows a sample assembly code that is supposed to do something. The idea is that this program produces machine instruction that would be copied on the boot sector and the computer powered-on.

Convert assembly (mnemonics) code to binary using the following,
`$nasm boot_sect1.asm -f bin -o boot_sect1.bin`

If you want to see what is exactly inside the `bin` file, the following command will help you.

```
$od -t x1 -A n boot_sector1.bin
```

The above binary can be used to setup (copy to) the first 512 bytes (the boot sector) of a disk. Instead of writing this boot sector to a physical hard disk, we can use an emulator. QEMU is a system emulator that provides a simple and nice method to run your boot sector directly from the `bin` file.

```
$qemu-system-i386 boot_sector1.bin
```

The above command emulates a system using the file provided as the attached disk (which in our case has the first 512 bytes of interest).

Compare the outputs of the booting process using the two programs, `boot_sector1.asm` and `boot_sector2.asm`, and justify your results. Submission should contain binary files and screen shots of QEMU along with an explanation.

8 Let's do something slightly more interesting. On boot, our custom OS should print out a message. Write a program, `hello.asm`, that prints `"Hello"` on the screen during boot-up.

To print a character on the screen, us the following code with appropriate repetitions and changes.

```
mov ah, 0x0e          ; set tele-type mode (output to screen)

mov al, 'H'           ; one ascii character hex code in register AL
int 0x10              ; send content of register to screen via an interrupt
```

Setup `hello.bin` as the input file for QEMU to use for booting and test output (capture screen shot in `hello.png`.

## Submission Guidelines

- All submissions via moodle. Name your submissions as: **<rollno_lab1>.tar.gz**
- The tar should contain the following files in the following directory structure:
  <roll_number_lab1>/
  |__part1/
  |____exercises_1_to_6.pdf
  |__part2/
  |____boot_sector1.bin
  |____boot_sector1.png
  |____boot_sector2.bin

```
|____boot_sector2.png
|____hello.asm
|____hello.bin
|____hello.png
```

- **Deadline: 16th July 5PM.**
  Expected time for completion of lab: 2.5 hours