# Lab 2 – SYSTEM MONITORING

**OBJECTIVE**

- Learn the basic CLI and GUI tools to monitor system resources.
- Identify system calls used by programs written in C language.
- Learn to write simple batch/shell scripts.

**TIME REQUIRED** : 3 hrs

**PROGRAMMING LANGUAGE** : C

**SOFTWARE REQUIRED** : Ubuntu/Fedora, gcc/gc, Text Editor, Terminal

**HARDWARE REQUIRED** : Core i5 in Computer Labs

**SYSTEM MONITORING**

A **system monitor** is a hardware or software component used to monitor system resources and performance in a computer system. Among the management issues regarding use of system monitoring tools are resource usage and privacy.

**TASK 2.1**

Navigate to **System Monitor**

**How many cores are on your Desktop/Laptop? _____**

**What is the total RAM and how much is free? _____**

**What is the Sending and Receiving Rate of your network? _____**

**Screenshot Processes Tab and paste here: -**

**TASK 2.2:**

Open **Terminal.**

**Type in the following commands and show outcomes:**

To identify the available CPU, memory, and disk resources, we can used the following commands:

cat /proc/cpuinfo(read the CPU information)

cat /proc/meminfo(read the memory (RAM) information)

df -h( find out secondary storage (hard-disk) information)

topis a command line program provides a real-time view of the processes running in the system. It provides system summary and the list of tasks managing by Linux kernel. The program is useful to identify the processes running with CPU and memory utilization. Launch a terminal and execute top command. You can press q to exit from top program.

Does the outcome of top match with the outcome of with System Monitor? _____

**TASK 2.3**

straceis a tool that helps to run specified command and traces its interaction with operating system. Wecan run any program using strace and identify the system calls it makes.

Launch a terminal and run strace ls

Try to read the output generated by the program and identify the system calls.

## Exercise 2.1. write following code and show outcomes here:

```
#include <stdio.h>

int main(int argc, char argv[]) {

  // file handle
  FILE *fileGeek;

  // open a file called "strace_demo.txt", or create it
  fileGeek = fopen("strace_demo.txt", "w");

  // write some text to the file
  fprintf(fileGeek, "Write this to the file" );

  // close the file
  fclose(fileGeek);

  // exit from program
  return (0);

} // end of main
```

Save this into a file called “file-io.c” and compiled it with `gcc` into an executable called `stex`.

```
gcc -o stex file-io.c
```

To identify which sections of the output refer to the different parts of the internal workings of the program.

```
strace ./stex
```

We can clearly see the write system call sending the text “Write this to the file” to our opened file and the exit_group system call. This terminates all threads in the application and sends a return value back to the shell.

## Filtering the Output

```
strace -e write ./stex
paste output screenshot here
strace -e close,write ./stex
paste screenshot
```

## Sending the Output to a File

```
strace -o trace-output.txt ./stex
```

**SIMPLE BATCH/SHELL SCRIPT:**

In Linux we have a command interpreter known as shell. In this section, we practice writing a very simple shell script. The objective is to write set of commands in a file and run the file to understand the batch execution interface.

**TASK 2.5**

Create a File named **hello.sh.**

Type in the following code:

```
#!/bin/sh
echo "Hello! Lets Execute ls."
ls
```

Now type sh hello.sh and press enter.

**TASK 2.4**

Type the following code and compile this program: -

```
/*
The code is taken from http://www.daniweb.com/software-development/c/code/216411/reading-a-file-line-by-line */
#include <stdio.h>int main (
void ) {
    static const char filename[] = "file.txt"; FILE *file = fopen( filename, "r" );
    if ( file != NULL ) { char line [
        128 ];
            /* or other suitable maximum line size */ while ( fgets ( line,
        sizeof line, file ) != NULL ) {
            /* read a line */
            fputs ( line, stdout ); /* write the line */
        }
        fclose ( file );

}
    else{
        perror ( filename ); /* why didn't the file open? */
    }
    return 0;
}
```

Execute the code. What is the outcome?

What was the outcome? What did you understand?

**EXERCISE 2.2**

**[6]**

Search for any file-based code in C for Linux. Download it.

Where did you download it from? Compile and Execute code and show outcome here:

**RESOURCES:**

https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html https://en.wikipedia.org/wiki/Linux
https://www.youtube.com/watch?v=IVquJh3DXUA https://www.youtube.com/watch?v=oLjN6jAg-sY