Department of Computer Science and Engineering
Indian Institute of Technology Bombay

CS 333: Operating Systems Lab
Autumn 2023
Lab 1: Hello OS!

Instructions

- Login with username **labuser** on software lab machines for this lab.
- This file is part of the **lab1.tar.gz** archive which contains multiple directories with programs associated with the exercise questions listed below.
- Most questions of this lab are tutorial style and are aimed to provide an introduction to tools and files related to system and process information and control.

Part 1: The OS view

(a) 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 of each process, system-level CPU usage etc. The system summary information displayed, order etc. has several configurable knobs.
`top` also allows you to send signals to processes (change priority, stop etc.).

Sample tasks: (Hint**: man top)**
- Display processes of specific user (e.g., labuser)
- Add **ppid** information to the displayed information (What is **pid**? What is **ppid**?)

- **`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.

Sample tasks: (Hint: **man ps**)
- List all the processes (of all users) in the system
- List all process whose **ppid** is 2
- Which process has **pid** 2 ?

- **`iostat`**

  `iostat` is a command useful for monitoring and reporting CPU and device usage statistics. For example, the command reports total activity and rate of activities (read/write) to each disk/partition and can be configured to monitor continuously (after every specified interval).

  Sample tasks: (man iostat)
  - Display average cpu utilization of your system
  - Display average disk and cpu utilization every 3 seconds for 10 times

- **`strace`**

  `strace` is a diagnostic and debugging tool used to monitor the interactions between processes and the operating system (Linux). 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.

  Sample task: (man starce)
  - Display all system calls and signals made by any command (for example display the system calls made by **ls** command).
  - Display summary of total time taken by each system call, time taken per system call during a program execution and the number of times a system call was invoked.

- **`lsof`**

  `lsof` is a tool used to list open files. The tool lists details of the file itself and details of users, processes which are using the files.

  Sample task: (man lsof)
  - Display all opened files of a specific user.

- **`lsblk`**

  `lsblk` is a tool used to list information about all available block devices such as hard disks drives (HDD), solid-state drive (SDD), flash drives, CD-ROM etc.

  Sample task:
  - Display all device permissions (read,write,execute) and owners.

- Also look up the following commands/tools:
  `pstree, lshw, lspci, lscpu, dig, netstat, df, du, watch`.

(b) The `proc` file system

The **`proc`** file system is a mechanism provided by Linux, for communication between userspace and the kernel (operating system) using the file system interface. Files in the `/proc` directory report values of several OS parameters and also can be used for configuration and (re)initialization. The **`proc`** file system is very well 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 the tools listed above and answer the following questions. Also, mention the tool and file you used to get the answers.
    a. Find the Architecture, Byte Order and Address Sizes of your CPU.
    b. How many CPU sockets, cores, and CPU threads does the machine have?
    c. Find the sizes of L1, L2 and L3 cache.
    d. What is the total main memory and secondary memory of your machine and how much of it is free?
    e. Find the number of total, running, sleeping, stopped and zombie processes. A zombie process is a stopped/terminated process waiting to be cleaned up.
    f. How many context switches has the system performed since bootup? A context switch is the process of storing the state of a process or thread so that it can be restored and resume execution at a later point, and then restoring a different, previously saved, state. This allows multiple processes to share a single CPU and is an essential feature of a multitasking operating system.

2. Run all programs in the subdirectory named **`memory`** and identify the memory usage of each program. Compare the memory usage of these programs in terms of **`VmSize`** & **`VmRSS`** and justify your observations based on the code.

3. Run the executable **`subprocesses`** provided in the sub-directory **subprocess** and provide your roll number as a command line argument. Find the number of subprocesses created by this program. Describe how you obtained the answer.

4. Run **`strace`** along with the binary program of **`empty.c`** (file located in subdirectory **`strace`**). What do you think the output of **`strace`** indicates in this case? How many different system calls can you identify?

    Next, use **`strace`** along with the binary program of **`hello.c`** (which is in the same directory).
    Compare the two **`strace`** outputs,

- Which part of the output is common, and which part has to do with the specific program?
- List all unique system calls for each program and look up the 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 describes how data is organized on a disk. Describe how you obtained the answer.

(c) Object Files

An object file is a file binary information (object code) a sequence of hardware instructions representing a program. Object files store information about data and code and also information about sections (text, data, etc.), and information used to relocate code and data of binary. An object file is generated by a compiler or an assembler and represents an executable file or a shared library.

ELF (Executable and Linkable Format) is an universally used file format for object files on Unix-like machine, More about ELF here: https://wiki.osdev.org/ELF

We will use objdump to analyze and understand object file formats and information.

- **`objdump`**

`objdump` command in Linux is used to provide thorough information for object/exectuable files.

In the folder **`object`**, find a binary (executable) program named **`function.out.`**
the file is made by a
company for their aptitude test. It takes
The program takes an input as variable x and computes and output $y = f(x, a, b, c, d)$.
a, b, c, d are secret parameters and are stored as global variables in the program.
The task is to find values of the parameters a, b, c, d.

Use the **`objdump`** tool to analyze the secret binary file and see if you can figure out the values easily without using your "Math" skills.

Sample `objdump` commands:

- Display all sections' information of the object file.
  `objdump -h <object-file>`
- Display the symbol table entries of the object file.
  `objdump -t <object-file>`
- Display all headers information.
  `objdump -x <object-file>`
- Display all the assembler contents of the executable section.
  `objdump -d <object-file>`
- Display the contents of all sections.
  `objdump -s <object-file>`
- Display the content of a specific section in hex representation.
  `objdump -s -j <section-name> <object-file>`

Part 2: Booting unraveled

The goal of this part of the lab is to learn about how a computer boots, and work with 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 the contents of the disk contain the codes for world peace, we are all set. In other words, 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 into 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 it 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.

1. 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 instructions that would be copied on the boot sector when the computer is powered-on.

   Convert assembly (mnemonics) code to binary using the following,

   `$nasm boot_sector1.asm -f bin -o boot_sector1.bin`

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

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

   The above binary can be used to set up (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 load and execute the 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. Submissions should contain binary files and screenshots of QEMU along with an explanation.

2. Let’s do something slightly more interesting. On boot, our custom OS should print out a message.

Write a program, **`hello.asm`**, that prints custom text (your name?) on the screen during boot-up, for example — “BuzzLightyear”.

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

```
mov ah, 0x0e        ; set tele-type mode (output to screen)
mov al, 'B'         ; 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 screenshot and save in a file named **`hello.png`**.)

Submission Guidelines

- All submissions via moodle. Name your submissions as: <rollno_lab1>.tar.gz
- For example; if your roll number is 123456789 then the submission will be 123456789.tar.gz and if your roll number is 12D345678 then the submission will be 12d345678.tar.gz
- The tar should contain the following files in the following directory structure:

```
<rollnumber_lab1>/
|__part1/
       |____exercises_1_to_7.pdf
|__part2/
       |____boot_sector1.bin
       |____boot_sector1.png
       |____boot_sector2.bin
       |____boot_sector2.png
       |____hello.asm
       |____hello.bin
       |____hello.png
```

- Deadline: **3rd August 2023, 5 pm.**