Operating System as a Service

Recap

We learned about the overview of OS role in the past week. The kernel is just one part of an operating system. The entire operating system software itself is a much **bigger** one, and it provides various apps and functionalities to **help users use the computer syste**m.

- User programs can make **system calls** whenever it needs to **elevate** its privileges and run in kernel mode to access the hardware or I/O devices.

> In other words, **the OS provides services for user programs**. The goal of making an operating system us to allow users to use the computer system in an easier and more efficient manner.

Below are the brief list of operating system services that are usually provided to the users via the Operating System **Interface**: terminal and GUI. We will learn more about each of them this week

Basic Support

Basic **support** for computer system usage via **system call** routines:

1. **Program execution**: The system must be able to load a program into memory and to run that program upon request. The program must be able to end its execution, either normally or abnormally (indicating error).
2. **I/O Operations**: Being able to interrupt programs and manage asynchronous I/O requests.
3. **File-system** manipulation: programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership.
4. **Process communication**: Processes run in virtual environment. Communications may be implemented via shared memory or through message passing, in which packets of information are moved between processes by the operating system.
    - This include communication protocol via the **internet**, where processes in different physical computers can communicate.
5. **Error detection**: The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices, etc. For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

Sharing Resources

Diagnostics report and computer **sharing** feature:

1. **Resource sharing**: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system: CPU cycles, main memory, file storage, I/O device routines
2. **Resource accounting**: This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.

Network and Security

Protection and **security** against external threats:

1. All access to system resources is controlled.
2. **Defenses**:
    1. defend against potential threats coming from external I/O devices, including modems and network adapters that may make invalid access attempts
    2. to record network traffic and connections for detection of break-ins.

Operating System User Interface

Users can utilise the OS services in two general ways:

1. Using the Operating system **GUI** or **CLI** (**User Interface**)
2. By writing instructions and making system calls within the it (**Programming Interface**)

OS User Interface

The OS User interface gives users **convenient** access to various OS services. They are programs that can execute specialised commands and help users perform appropriate system calls in order to navigate and utilise the computer system.

GUI

> The GUI or desktop environment is what we usually call our **home screen** or **desktop**. It characterises the feel and look of an operating system.

We use our mouse and keyboard everyday to interact with the OS GUI and make various system calls, for instance:

1. Opening or closing an app
2. File creation or deletion
3. Get attached device input or output
4. Install new programs, etc

When interacting with the OS through the GUI, users employ a mouse-based **window-and-menu** system characterized by a desktop **metaphor**:

- The user moves the mouse to position its pointer on images, or icons, on the screen (the desktop) that represent programs, files, directories, and system functions.
- Depending on the mouse pointer's location, clicking a button on the mouse can launch a program,
- Select a file or directory (folder) or pull down a menu that contains commands.

In fact, anything that is performed by the user that involves **resource allocation**, **memory management**, **access to I/O** and **hardware**, and **system security** requires **system calls**. Operations to perform various system calls are made easier with the OS interface and more **convenient** with the OS GUI.

> Obviously, the GUI is made such that general-purpose computers are user friendly.

You can **customise** your Ubuntu Desktop environment if you wish. By default, it comes with GNOME (3.36, at the time of current writing) desktop, but nothing can stop you from installing [other desktop environments].

CLI

The OS CLI (Command Line Interface) is what we usually know as the "terminal" or "command line".

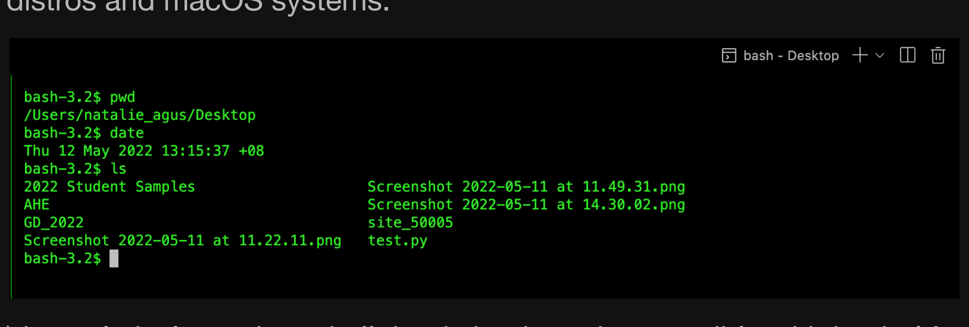
> CLI provides means of interacting with a computer program where the user issues **successive** commands to the program in the form of text. The program which handles this interface feature is called a command-line interpreter. We have experimented with this during Lab 1.

Command Line Interpreter

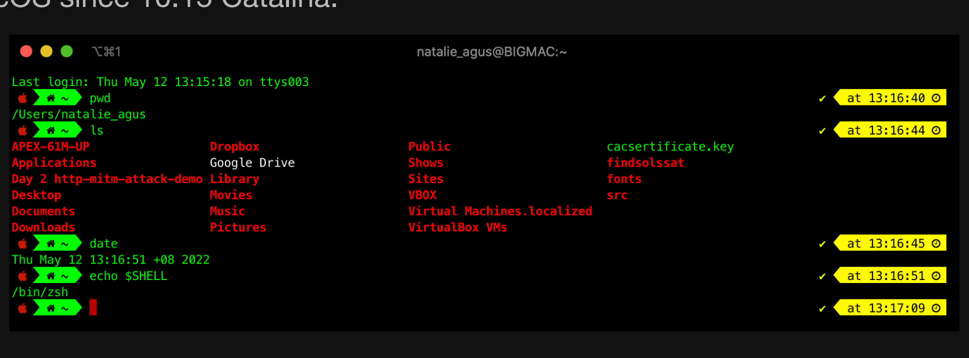
In `UNIX` systems, the particular program that acts as the **interpreters** of these commands are known as **shells**[1]. Users typically interact with a Unix shell via a [terminal emulator], or by **directly** writing a shell script that contains a bunch of successive commands to be executed.

For a system that comes with multiple command line interpreters (shells), a user may choose which one to use. Common shells are the Bourne shell, C-shell, Bourne-Again shell, and Korn shell.

- [Bourne-Again shell] (bash): written as part of the GNU Project to provide a superset of Bourne Shell functionality. This shell can be found installed and is the default interactive shell for users on most Linux distros and macOS systems.



- [Z shell] (zsh) is a relatively modern shell that is backward compatible with bash. It's the default shell in macOS since 10.15 Catalina.



- [PowerShell] – An object-oriented shell developed originally for Windows OS and now available to macOS and Linux.

In short, the shell primarily **interprets** a series of commands from the user and executes it. There are two ways to implement commands:

1. **Built-in**: the command interpreter itself contains the code to execute the command.
    - For example, a command to **delete** a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.
    - In this case, the number of commands that can be given determines the size of the command interpreter, since each command requires its own implementing code.
2. **System programs** (typically found in default `PATH` such at `/usr/bin`): command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and be executed. This is used by UNIX, among other operating systems.
    - You can type `echo $PATH` on your terminal to find out possible places on where these system programs are.

> You will be required to implement a Shell in Programming Assignment 1.

Terminal Emulator

A terminal emulator is a **text-based** user interface (UI) to provide easy access for the users to issue commands. Examples of terminal emulators that we may have encountered before are [iTerm], MacOS terminal, [Termius], and Windows Terminal.

Almost every system-administrative actions that we can perform on the OS GUI can be done via the CLI. For example, if we want to delete a file in different locations using the OS GUI, we need to perform the following steps:

1. Hover our mouse to click folder after folder until we arrive at a final folder where the target file resides
2. Right click, press delete (or use keyboard shortcut)
3. Repeat until all files are deleted in various paths

Equivalently, we can perform the same action using the CLI. In UNIX systems, we can write the following commands in our terminal emulator:

1. `cd <path>`
2. `rm <filename>`

Executing Commands

From Lab 1, you should've had the experience in trying out some simple shell commands. The implementation of the two commands `cd` and `rm` are as follows:

1. The first line is implemented within the shell program itself (**changing** directory with `chdir` system call).
2. The second line will tell the shell to **search** for a **system program** named `rm` and execute it with the parameter `<filename>`.
    - **System Programs** are simply programs that come with the OS to help users use the computer. They are run in **user** mode and will help users make the appropriate **system calls** based on the tasks given by the users. More about System Program will be explained in the latter part.
    - The function associated with the `rm` command (removing a file) would be defined completely within the code in the program called `rm`.
    - In this way, programmers can **add** new commands to the system easily by creating new system programs whose name matches the **command**.
    - The command-interpreter program, which can be small, does not have to be changed for new commands to be added.

> From Lab 1, you should have been able to find out where your system programs like `ls, mkdir, rm, pwd, ps` reside.

---

1. The most generic sense of the term shell means any program that users employ to type commands. A shell hides the details of the underlying operating system and manages the technical details of the operating system kernel interface, which is the lowest-level, or "inner-most" component of most operating systems. ↩