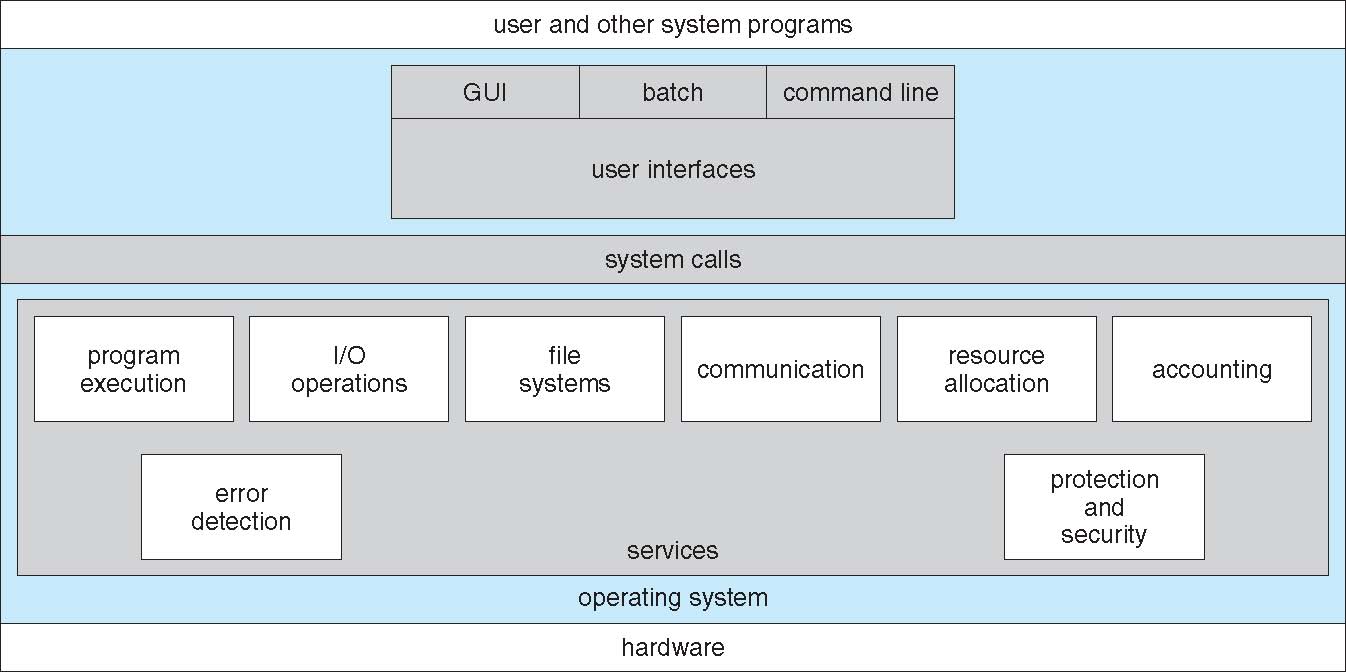
**Operating System Services**

Operating systems provide an environment for execution of programs and services to programs and users

* **User interface –**
  + **Command-Line (CLI)**
  + **Graphics User Interface (GUI)**
  + **Batch**
* **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
* **I/O operations -** involve a file or an I/O device
* **File-system manipulation**
* **Communications** processes may exchange information, on the same computer or between computers over a network
* **Error detection**
* **Resource allocation**
* **Accounting -** To keep track of which users use how much and what kinds of computer resources
* **Protection and security**
  + **Protection** involves ensuring that all access to system resources is controlled
  + **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



* CLI or **Command Interpreter** allows direct command entry
  + **Shells**
* GUI or **Graphical User Interface**
  + Icons
  + User-friendly **desktop**
* Many systems now include both CLI and GUI interfaces
* **Microsoft Windows** is GUI with CLI “command” shell
* **Apple Mac OS X** is “Aqua” GUI interface with UNIX kernel underneath and shells available
* **Unix and Linux** have CLI with optional GUI interfaces (CDE, KDE, GNOME)
* **Touchscreen Interfaces**
  + Mouse not possible or not desired
  + Actions and selection based on gestures
  + Virtual keyboard for text entry

**System Calls**

* Programming interface to the services provided by the OS
* Typically written in a high-level language (C or C++)
* **Application Programming Interface (API).** Accessed using API rather than direct system call use
* Three most common APIs are
  + **Win32 API** for Windows,
  + **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  + **Java API** for the Java virtual machine (JVM)

**Types of System Calls**

* **Process control**
  + create process, terminate process
  + end, abort
  + load, execute
  + get process attributes, set process attributes
  + wait for time
  + wait event, signal event
  + allocate and free memory
  + Dump memory if error
  + **Debugger** for determining **bugs, single step** execution
  + **Locks** for managing access to shared data between process
* **File management**
  + create file, delete file
  + open, close file
  + read, write, reposition
  + get and set file attributes
* **Device management**
  + request device, release device
  + read, write, reposition
  + get device attributes, set device attributes
  + logically attach or detach devices
* **Information maintenance**
  + get time or date, set time or date
  + get system data, set system data
  + get and set process, file, or device attributes
* **Communications**
  + create, delete communication connection
  + send, receive messages if message passing model to host name or process name
    - From client to server
  + Shared-memory model create and gain access to memory regions
  + transfer status information
  + attach and detach remote devices
* **Protection**
  + Control access to resources
  + Get and set permissions
  + Allow and deny user access

**System Programs**

* **File management**
* **Status information**
  + detailed performance, logging, and debugging information
  + ask the system for info
  + print the output to the terminal or other output devices
  + **registry** - used to store and retrieve configuration information
* **File modification**
* **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
* **Program loading and execution**
* **Communications -** for creating virtual connections among processes, users, and computer systems
* **Background Services**
  + **services**, **subsystems**, **daemons**
* **Application programs**
  + Don’t pertain to system
  + Run by users
  + Not typically considered part of OS
  + Launched by command line, mouse click, finger poke
* OS is Affected by choice of hardware, type of system
* **User goals** – operating system should be convenient to use, easy to learn, reliable, safe, and fast
* **System goals** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
* **software engineering** . Specifying and designing an OS is highly creative task

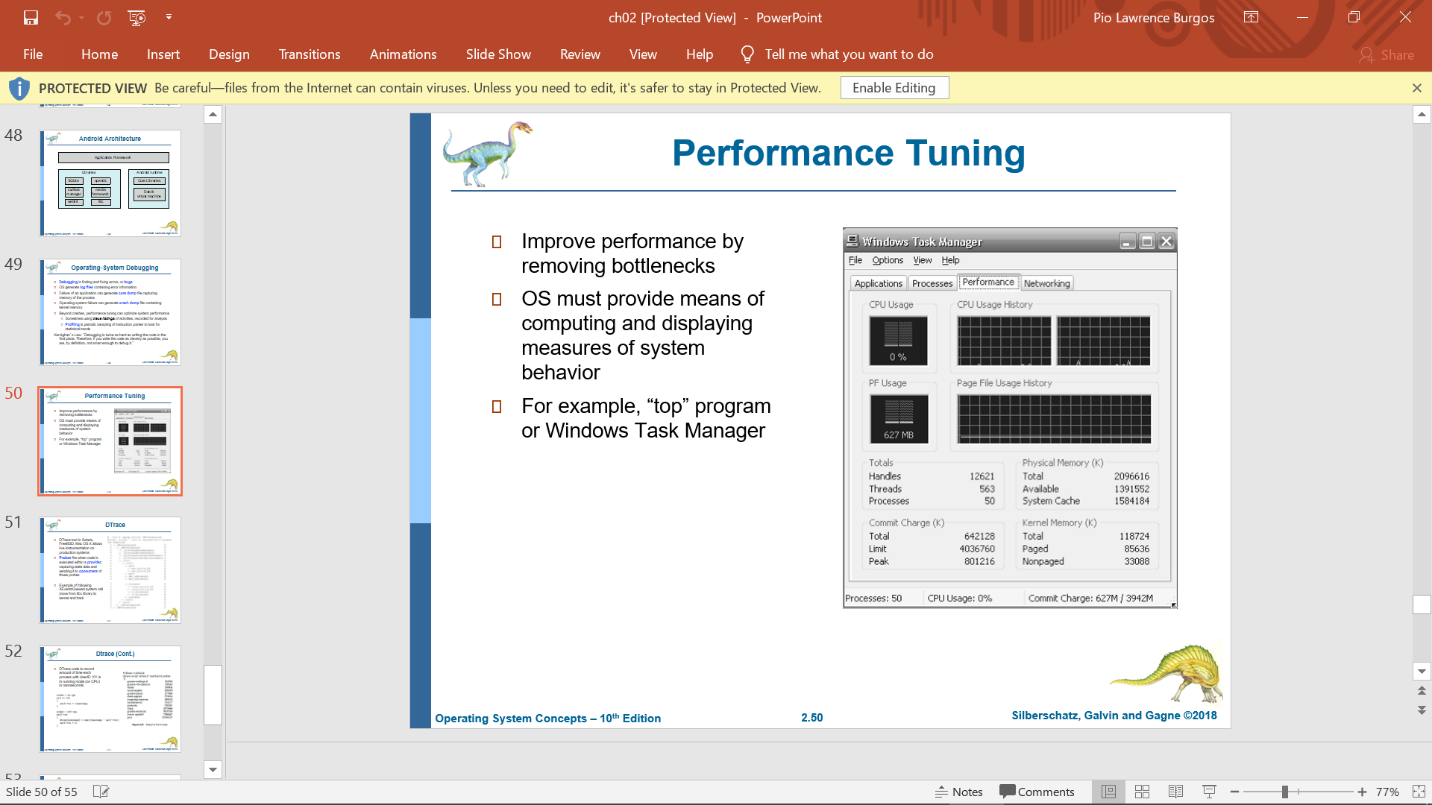
**Implementation**

* mix of languages
* Lowest levels in assembly
* Main body in C
* Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
* More high-level language easier to port to other hardware
  + But slower
* **Emulation** can allow an OS to run on non-native hardware

**iOS**

* Apple mobile OS for iPhone, iPad
  + Structured on Mac OS X, added functionality
  + Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)

**Android**

* Developed by Open Handset Alliance (mostly Google)
* Open Source
* Similar stack to IOS
* Based on Linux kernel but modified
  + Provides process, memory, device-driver management
  + Adds power management

**Operating-System Debugging**

* **Debugging** is finding and fixing errors, or ***bugs***
* OS generate **log files** containing error information
* Failure of an application can generate **core dump** file capturing memory of the process
* Operating system failure can generate **crash dump** file containing kernel memory