Introduction

What is an OS

- It is a software that facilitates other software to run on the computer hardware.
- It provides interface between the hardware[a], and different types of utility and application programs[b].
- It works as a resource manager for different users and running programs.

---

[a] Instruction set architecture (ISA).

[b] Application programming interface (API) and application binary interface (ABI): see Wikipedia for ABI and API.

What is an OS

- It creates a virtual machines for every running code, creates an illusion of more resources than what is actually present in the system.
- It facilitates communication, if needed, between programs running in parallel.

What is an OS

- Special purpose OS is embedded in the controller of a gadget e.g. camera, washing machine, automobile etc.[a]
- General purpose OS is used on computers‘ and many modern devices such as smartphone, etc. They are visible to the user and supports 3^{rd}-party software.
- We shall talk about general purpose OS.

---

[a] Not visible to the user.

What is a General Purpose OS

- There are two different views.
- A complete package of software modules for resource management along with different utility software e.g. command interpreter, editor, compiler etc.
- The core software modules that manages the hardware resources and create facility to run programs on it.

Our Scope

- We shall talk about the core software modules known as OS Kernel.
- The support provided by the kernel to other software and its overall organization.
- But before that we need to define a few terms and issues.

Instruction Set Architecture (ISA)

- The set of machine instructions available to write a machine level program.
- It includes the set of accessible registers in the CPU, FPU, and MMU.
- It also includes other features e.g. interrupt structure, DMA, exception handling, memory management support etc.

Instruction Set Architecture (ISA)

- The I/O address mapping[a] is a part of it.
- Often the ISA of a CPU is not identical for the OS kernel and other software.
- The full instruction set is available when the CPU is in privileged mode. A subset is available in the non-privileged mode.

---

[a] Separate I/O space or memory mapped I/O space.

Instruction Set Architecture (ISA)

- The OS kernel runs in the privileged mode of the CPU.
- But other software run in the non-privileged mode, and do not have access to some of the instructions.
- This is essential for virtualization and protection provided by the OS.

Application Programming Interface (API)

- An OS hides the hardware details and provides a more user friendly but restricted view of the system.
- The services provided by the OS through a set of standard interface known as system calls[a] .

---

[a] See system call in Wikipedia.

Application Programming Interface (API)

- As an example if a software writer wishes to access the key-board, she does not require to program the key-board controller explicitly.
- In fact her code cannot access the keyboard directly. It uses appropriate system call to do the job.

Application Programming Interface (API)

- A system call to read data takes parameters e.g. logical device-file number, number of bytes to read, address of the buffer to store the data etc.
- The call transfers the control to specific code of OS to handle such request.

Application Programming Interface (API)

- An user program runs in the non-privileged mode of the CPU and does not have direct access to devices and many other hardware resources.
- The OS code runs in the privileged mode of the CPU, and have direct access to all hardware resources.

Application Programming Interface (API)

- The system call handler of OS translates the user specified parameters to device specific parameters, and takes appropriate actions.
- It also checks for the integrity of the request.
- Each system call uses special CPU instruction (software interrupt), it changes the mode of the CPU while transferring control to the OS.

Application Binary Interface (ABI)

- ABI is a convention of interface between machine level programs. As an example, which registers to use to pass parameters to a function, or where to get back the return value from it etc. This is decided by the compiler and its libraries.
- But it is also specified by the architecture and OS at the level of system call.

Resources and Management

- The resources of a computing system are CPU (time), memory (space), disk (persistent space), IO devices etc.
- OS allows different concurrently running programs (processes) to share the CPU by time sharing.
- The main memory is shared among different running, ready and blocked programs.

Resources and Management

- Different users use the disk space through file systems to store their files.
- IO devices and network communications are also shared by different programs.
- Kernel also uses the CPU, memory, disk and IO devices.

Resources and Management

- Sharing of resource requires allocation and protection (from another program or user).
- OS allocates resource following its policy, and ensures the protection with the support of the hardware architecture.

Resources and Management

- Execution of one particular code should not monopolize the CPU.
- One running code should not tamper the memory image of another running program or its data, or even that of OS.
- It is necessary to protect files of one user from another user's action.

Virtual Machine for Every Running Program

- Each running program gets an illusion that the computing system is available to it.
- But in reality there is only one or a few CPUs, one main memory etc.
- The OS facilitate the sharing of the CPU by different running programs over time so that each of them see a progress of their computation.

Machine for Every Running Program

- The memory is occupied by code and data of different running programs giving an illusion that the memory is available to each of them.
- It is not necessary to keep the entire code and data of every running program in the memory.

Machine for Every Running Program

- OS keeps the currently used portions of the code and data in the memory to accommodate more number of running programs.
- The remaining portions of data and code are kept in the bulk storage media. OS brings them back and forth as and when required.

Machine for Every Running Program

- Sharing of CPU, main memory and other resources is also important for their better utilization.
- As an example when a running program waits for an IO, it does not use the CPU.
- In this time slot the CPU is used by other program in execution.

Machine for Every Running Program

- It makes little sense to fill the whole memory with a large program and its data.
- Execution of code often fetches nearby instructions[a] and often repeats the same code[b]. Data access also has some pattern.
- OS tries to keep the active portion of code and data in the memory.

---

[a] Spacial locality of reference.

[b] Temporal locality of reference.

Illusion Management

- Sharing of resource among different running code has its cost.
- It is necessary to save the State of the CPU after an unfinished computation. Then the saved state is restored to restart the computation.
- The memory mapping and other information are also saved and restored.

Illusion Management

- It may be necessary to transfer a part of memory image of a running program between the bulk storage and the main memory.
- This memory swapping is time consuming and also requires house keeping by OS.
- OS maintains data structures for different running programs and different resources.

OS: External and Internal Views

- The OS may be viewed in two different ways: external or user-view and internal or designer-view.
- The external view is for users who use the facilities provided by the OS through its system calls.

OS: External and Internal Views

The internal view is mainly for the OS-programmers, who writes and maintains the OS code and its algorithms e.g. scheduling policy, memory management, and data structures related to running codes, open file etc.

Important OS Modules

- Creation and Scheduling of Processes.
- Communication and Synchronization among cooperating concurrent processes.
- Management of memory.
- Management of persistent mass storage and file system.
- I/O system management.
- System protection and security.

Important Sub-Modules

- Interrupt and exception handling.
- System call handling.
- Virtual file system.
- Device drivers.

Process: a Code in Execution

- The main job of an OS is to facilitate the running of programs on a computer hardware.
- An execution instance of a program is called a process[a].
- A program can be used to create several processes running concurrently.

---

[a] There is also an important similar notion called thread

Process: a Code in Execution

- When a computer is on, some process is running on the machine. It may be a user process, system process, system idle process[a].
- OS creates, maintains, schedules and destroys a process

---

[a] `https://en.wikipedia.org/wiki/System_Idle_Process`

Process: a Code in Execution

- An executable code is already bound to different types of logical memory segments e.g. code, static data etc.
- When a processes is created some other memory segments e.g. execution stack, heap etc. are attached to it.
- OS creates a mapping of different logical memory segments to the physical memory.

Process: a Code in Execution

- When a process is scheduled, the state of the CPU preserved in the OS data structure is loaded to the CPU hardware.
- The program counter holds the address of the next instruction. The stack pointer points to the top of the execution stack.
- The CPU starts executing the code of the process.

Process Switching

- Once the time slice of the running process is over. It is suspend and a process from the ready-queue is dispatched for execution.
- A running process needs to read data. Often the IO operation is slow. So it is suspend until the IO is complete.
- Some other event may also cause suspension of running process.

Process Switching

- When the current time slice of the running process is over, an external device called timer initiates an event known as an interrupt.
- An interrupt is a hardware signal that comes from some external device to the CPU.

Process Switching

- After receiving the timer interrupt, the CPU hardware saves a minimal portion of its state and transfers the control to a piece of OS software known as an interrupt handler.
- The interrupt handler initiates a sequence of actions that changes the state of the process from running to ready, and starts another process from the ready-queue.

Process Switching

- A running process P_r needs to read data from the keyboard or disk.
- P_r sends a request to the OS through a system call.
- The computation cannot advance without the data, which may be available after a significantly large amount of time[a].

---

[a] Compared to the instruction cycle of the CPU.

Process Switching

- User enters data through the keyboard at a much slower rate. Even the disk access, if necessary[a], is also several order of magnitude slower than the CPU cycle.
- OS may suspends/blocks the process P_r, initiates the IO required by it, and dispatches a process P_q from the ready-queue.

---

[a] The data from the disk may be available in the kernell buffer

Process Switching

- Once the data requested by P_r is available, the currently running process P_c (may or may not be P_q) is interrupted[a].
- The control is transferred to OS, and it changes the status of P_r from suspended to ready.
- The control is returned to P_c after the interrupt is serviced.

---

[a] May be by the IO device controller.

Process Switching

- When two or more cooperating concurrent processes are running, one or more of them may be suspended at a synchronization point.
- Different mechanisms are developed to implement synchronization, exclusion etc.

Context Switching

- The process switching is called a context switching.
- To suspend a running process P_r it is necessary to save the state of P_r e.g. content of the CPU registers, status word, memory mapping information, file mapping information etc. to a OS data structures.

Context Switching

- Similarly, to (re)start a ready process it is necessary to load its saved state from the OS data structure to the CPU and to other places.
- For every context switching the OS code known as scheduler is involved.

Note

- Every interrupt does not lead to context switching.
- On receiving an interrupt the context of the running process is saved.
- The same context is loaded once the interrupt is serviced.

Thread: a Light-weight Process

- Context switching is costly due to the volume of state information of a process that is to be saved and restored.
- A thread of execution is a sequence of instructions within a process that can be scheduled for execution by the OS.
- A process may have more than one threads of execution in it.

Thread: a Light-weight Process

- Two different process have two different logical address spaces.
- But two different threads of execution within a process share the same logical address space.
- But different threads have different CPU states and execution stacks that are to be saved.

Can a User Process Modify OS?

- Both a user program and the OS use the ISA of the underlying hardware.
- How does the OS protects itself from a user program?
- OS after all uses a sequence of instructions to modify its data structures.

Can a User Process Modify OS?

- A similar sequence of instructions may be used by a user program to modify/tamper the OS data structure.
- The support for protection comes from the hardware. Any modern CPU has more than one modes of operation - privileged or kernel mode and user mode[a]

---

[a] There may be more than two modes in a processor.

Can a User Process Modify OS?

- The complete ISA is available in the privileged mode.
- The ISA of the user mode is restricted. Following are a few such restrictions:

Can a User Process Modify OS?

In user mode,

- IO instructions (for machine with separate IO space) are not available.
- Registers related to memory management and mapping are not accessible.
- Interrupt related data and psw cannot be modified in the user mode.

Can a User Process Modify OS?

- Every user process runs on the CPU in the user mode.
- The CPU is automatically switched to privileged mode, when the control gets transferred from a user process to OS through interrupt, software interrupt, or exception[a]

---

[a] Divide-by-zero, illegal op-code, memory violation etc.

Can a User Process Modify OS?

- When the control is returned back to the user process, the state of the CPU is switched to user mode.
- This ensures the protection of OS from the user and one user from another.

Cooperating Processes

- Two or more user processes are said to be cooperating if the computation of one is affected by the computation of another.
- Cooperation requires communication between processes. Typical mechanisms for communication are through shared memory, message passing etc.

Cooperating Processes

- Cooperation requires ordering and synchronization of events.
- OS facilitates and manages inter-process communication.
- It also creates facilities for synchronization and atomicity of the critical sections of code segments.

Linux System Boot up

- How does a OS like Linux get started when the power is On?
- The process is complicated. Following is a broad outline of steps.

Process of Linux System Boot up

1. When the power of the PC is on, the system BIOS starts automatically! It loads the MBR from the bootable disk.
2. MBR is executed to loads more sophisticated boot loader e.g. GRUB (GRand Unified Bootloader).
3. GRUB loads the kernel and initrd (initial ram disc) image.

Process of Linux System Boot up

4. After a setting page table etc, kernel starts the first process init.
5. init starts other run-level programs.

BIOS: Basic Input/Output System

- When the power of the system is ON, the CPU receives some signal (Power Good).
- The hardware loads the PC with the address of the first instruction of BIOS and the execution of POST (power on self test) starts.

System BIOS

- Different hardware e.g. CMOS RAM, CPU, video card, hard disk etc. are checked and initialized.
- If there is no error, BIOS loads the MBR (master boot record) from the boot device (boot device sequence is available in the CMOS).
- It transfers the control to the MBR code.

MBR: Master Boot Record

- MBR is contained in the first sector of the boot device e.g. hard disk. Historically the size of the sector is 512 bytes.
- The image of modern OS kernel is often stored in the file system.
- The 512 byte space of MBR is insufficient to store the code to load the OS image from the file system.

MBR: Master Boot Record

- Also a boot loader displays a menu of different boot options along with a default option. This too requires space.
- Modern boot loaders e.g. LILO or GRUB splits the required start up code in two parts.
- The smaller part resides in the MBR, which invokes the main loader code.

GRUB: GRand Unified Boot loader (GNU)

- Its first stage (`boot.img`) is stored in MBR. This invokes the second stage (there is a stage 1.5 also).
- GRUB can access the underlying file system(s)[a] and can read its configuration file.
- GRUB loads the executable image of kernel and the initrd image following the path in the file system.

---

[a] Unlike old LILO (Linux Loader).

Kernel

- The compressed Kernel image is decompressed in the memory.
- Kernel start up process mounts the initial file system, initrd.
- It does different types of initialization e.g. page table etc.
- It runs init, the first process run by the system.

Bibliography

1. http://www.thegeekstuff.com/2011/02/linux-boot-process
2. https://en.wikipedia.org/wiki/Linux_startup_process
3. https://en.wikipedia.org/wiki/BIOS
4. https://en.wikipedia.org/wiki/Master_boot_record
5. https://en.wikipedia.org/wiki/GNU_GRUB
6. https://en.wikipedia.org/wiki/Initrd