

What is an OS

An OS is a **layer of software** that **interfaces** between hardware resources and one or multiple applications running on the machine. It is a special program that should **never stop** and **never fail**. It is fully **trusted** by hardware and applications.

[[Pasted_image_20240312210618.png|150]]

The OS Provides

Protection, isolation and sharing of resources

Fault isolation Fault isolation is intended to separate [[The Process Abstraction|Processes]] from each-other as well as from the OS. Un-allowed access results in **Segmentation fault**

[[**Resource sharing**]] [[Scheduling]] The OS needs to choose which [[The Process Abstraction|process]] to execute, as well as how much of each physical resource to allocate to each [[The Process Abstraction|process]].

Communication The OS allows processes to communicate with each other in a protected manner

Clean and easy to use abstraction of physical resources

Virtualization is used to give applications the illusion of exclusive CPU access along with the illusion of infinite hardware resources.

Higher level objects such as files, users and messages are used.

Process Abstraction The OS provides process abstraction to every program, enabling each process to execute in a restricted execution environment

[[The Process Abstraction|Processes]] have a nicer interface than raw hardware: Threads, Address space, Files and Sockets. The OS translates from hardware interface to application interface. It provides each running program with its own process. [[Pasted_image_20240312211634.png]]

A set of common services

The OS provides a set of common services to standardise the design of applications. This makes sharing easier and maximises reuse as all applications are built upon the same base. Decouples application development from hardware.

For example: - File Systems - User Interface - Network

How OS handles processes

The OS keeps a tree of all processes - Every process has a parent except “init” (pid 1 = init) - The OS Scheduler maintains the set of schedulable processes
Running a process [[Pasted_image_20240318141157.png|500]]

Keeping Control: Interrupts

- provided by hardware as a signaling mechanism for OS to maintain Control
- **Asynchronous**, signals some external event has happened and needs attention
- Interrupts are disabled when an interrupt handler is running ##### Handling
- Detected by CPU
- Suspends running process
- Executes interrupt handling code in kernel mode
- Restores the user process [[InterruptHandling.png|500]]


Example

Timer Interrupt: Interrupts every 10 ms for OS to regain control - Every 10 ms CPU switches from user to kernel mode - Now OS can choose which process to execute

Scheduling

Mechanism

Context Switch The act of stopping one process and running another one. Happens when the OS is in kernel mode and:
- Cannot return to the same process - Process has terminated - Process is waiting on I/O - Does not want to return to same process - Process has run for too long - Other processes are present and need to be scheduled

Procedure Context of process is maintained in Process Control Block (PCB)
OS maintains process control block for each process PCB contains: - All process specific registers - All general registers  De-scheduled process is either in **Ready** or **Blocked** state

Preemption

OS sets a timer before scheduling a process to avoid process running for ever
- Hardware sends interrupt after timer expires - Interrupts process execution - Switch to kernel mode - OS decides if process can continue or if it performs a context switch

Metrics

Utilization Fraction of the time CPU is executing a job **Goal:** Maximize CPU utilization

Turnaround Time Total time from job arrival to job completion **Goal:** Minimize turnaround time

Response Time Total time from job arrival until first time job is scheduled

Policies

Non preemptive

FIFO (First in First Out) Complete jobs in chronological orders of arrival
Suffers from Convoy effect ##### Convoy effect Short jobs get queued behind a long job that arrived first.

SJF (Shortest Job First) Execute shortest job first assuming they arrive at the same time, otherwise execute first arrived

Long running jobs cannot be interrupted for shorter jobs that arrive after

Preemptive

STCF (Shortest Time to Completion First) Extends SJF by adding preemption. Each time a new job enters the system: - Scheduler determines which of remaining jobs has shortest completion time - Schedules the shortest job first

Not great response time

RR (Round-Robin) Runs jobs for fixed time-slice and then switches to next job

Taking I/O into account I/O is generally slow so scheduler should schedule other jobs during I/O

Multi-Level Feedback Queue (MLFQ) **Goal:** General Purpose Scheduler
Challenge: Supporting long running, CPU intensive jobs (Batch processing) and interactive, low latency foreground tasks (interactive processes)

First *optimizes* average turnaround time - Important for batch processes Then *minimizes* response time - Important for interactive processes

Approach: Multiple levels of Round Robin Each level has higher priority and preempts lower level Process at a higher level will always be scheduled first High levels have short time slices, lower levels run longer

Rule 1: if $\text{priority}(A) > \text{priority}(B)$ then A runs Rule 2: if $\text{priority}(A) == \text{priority}(B)$ then A & B run in RR Rule 3: Processes start at top priority Rule 4: If a process uses its total time slice, the scheduler lowers its priority Rule 5: Periodically moves all jobs to topmost queue (**priority boosting**)

Boosting avoids starvation (long tasks never being scheduled)

Idle process Process that runs if all processes are blocked Has low priority Never blocks or does I/O