BAHIR DAR UNIVERSITY

FACALITY OF SOFTWARAE AND COMPUTING ENGINNERING

OSSP INDIVIDUAL ASSIGNMENT

NAME	***ID***
TSION MELESE	**1602642**

SUBMITTED TO : MR. WONDIMU B.

8/16/2017

TSION MELESE

A.Introduction

Logo of the mobile operating system ios by apple

Operating system

The operating system (OS) serves as the foundational software on a computer, orchestrating the management of all hardware and software resources while providing essential services to computer programs. Functioning as a central coordinator, it harmonizes the diverse components of the system. Absent an OS, the computer's hardware remains a mere assembly of ineffectual electronic parts.

It is the program that ,after being initially loaded into the computer by a boot program ,manages all of the other application programs in a computer the application programs make use of the OS by making requests for service through a defined applicatin program interface(API).

Specifically, the OS handles hardware management by acting as an intermediary between software applications and the physical components. This encompasses tasks such as CPU scheduling, which determines program access and duration of use; memory management, involving resource allocation and interference prevention; device management, which controls hardware through device drivers; and storage management, focused on file and directory organization. Resource allocation further ensures equitable and efficient distribution of system assets, preventing conflicts and guaranteeing access to necessary components.

Moreover, the OS provides a user interface enabling interaction with the computer, which can manifest as a graphical user interface (GUI) or a command-line interface (CLI). File system management structures files into directories, offering tools for manipulation and security. Security measures, including user authentication, access control, and firewalls, safeguard the system from threats.

The OS also undertakes process management, handling the execution of programs and offering mechanisms for inter-process communication.

iOS stands for **iPhone Operating System**. It is the software that powers Apple's mobile devices, acting as the **interface between the user and the hardware**. It manages hardware resources, enables app execution, handles input/output processes, and provides the platform for developers to build applications.

Core Characteristics of iOS:

1. **Closed Source & Proprietary:**
   iOS is **not open-source**. Apple maintains strict control over its ecosystem, which ensures security, performance, and consistent user experience.
2. **Touch-based User Interface:**
   It is primarily designed for **touch input** – taps, swipes, pinches – and includes gestures for multitasking and navigation.
3. **Optimized for Apple Hardware:**
   iOS is **specifically designed to run on Apple devices**, meaning it works efficiently with their custom chips.
4. **App Ecosystem:**
   Apps are delivered via the **Apple App Store**, with strict guidelines to ensure security, privacy, and performance.
5. **Security-Centric:**
   iOS has built-in security features like:
   - **Sandboxing apps**
   - **Secure Boot Chain**
   - **Touch ID / Face ID**
   - **Encrypted iMessage and FaceTime**

6. **Regular Updates:**
   Apple releases **frequent updates** that bring new features, patch security holes, and enhance user experience.

1.1Historical development of ios

The historical development of iOS is a fascinating journey that traces the evolution of one of the world's most popular mobile operating systems. Developed by Apple Inc., iOS has seen major transformations since its inception in 2007.

1. Origins: iPhone OS (2007–2009)
   2007 – Introduction with iPhone OS 1

- Focused on multitouch, visual voicemail, and web browsing.
  2008 – iPhone OS 2
- Third-party developers could now create apps using the iPhone SDK.
- Marked the beginning of the app economy.
  2009 – iPhone OS 3
- Introduced copy and paste, MMS, Spotlight search, and landscape keyboard.
- Supported iPhone 3GS and iPod Touch.

2. Rebranding and Expansion (2010–2012)
   2010 – iOS 4: Birth of the "iOS" Name

- Brought multitasking, folders, FaceTime, and iBooks.
- Introduced support for iPad, which debuted in April 2010.
  2011 – iOS 5
- Introduced Notification Center, iMessage, Newsstand, and iCloud.
- Removed dependency on iTunes for activation (PC-free experience).
  2012 – iOS 6
- Introduced Apple Maps, replacing Google Maps.
- Added Siri to iPad, Facebook integration, and Passbook (later Apple Wallet).

3. Design Overhaul and Modernization (2013–2016)
   2013 – iOS 7: Major Redesign

- Flat UI design replaced skeuomorphism.
- Introduced Control Center, AirDrop, and iTunes Radio.
  2014 – iOS 8
- Introduced HealthKit, HomeKit, Continuity, and Family Sharing.
- Opened up keyboards and widgets to third-party developers.

2015 – iOS 9

- Focused on performance and stability.
- Introduced Low Power Mode, multitasking on iPad, and Proactive Siri.

2016 – iOS 10

- Redesigned lock screen, enhanced iMessage with stickers and apps.
- Expanded Siri to support third-party apps.

4. Smarter and More Personalized (2017–2019)

2017 – iOS 11

- Introduced Files app, Augmented Reality (ARKit).
- Major update for iPads with drag and drop and a desktop-like dock.

2018 – iOS 12

- Focused on speed and stability, even for older devices.
- Added Screen Time, Memoji, and enhanced notifications.

2019 – iOS 13

- Introduced Dark Mode, Sign In with Apple, and new privacy controls.
- Split into iOS 13 for iPhone and iPadOS for iPad.

5. Intelligence and Privacy Focus (2020–2022)

2020 – iOS 14

- Added App Library, widgets on home screen, and Picture-in-Picture.
- Improved privacy transparency – apps had to disclose tracking behavior.

2021 – iOS 15

- Focused on FaceTime improvements, Focus modes, and Live Text (text recognition in images).

2022 – iOS 16

- Personalized lock screens, iCloud Shared Photo Library, and edit/unsend in iMessage.
- Introduced Passkeys, a passwordless login standard.

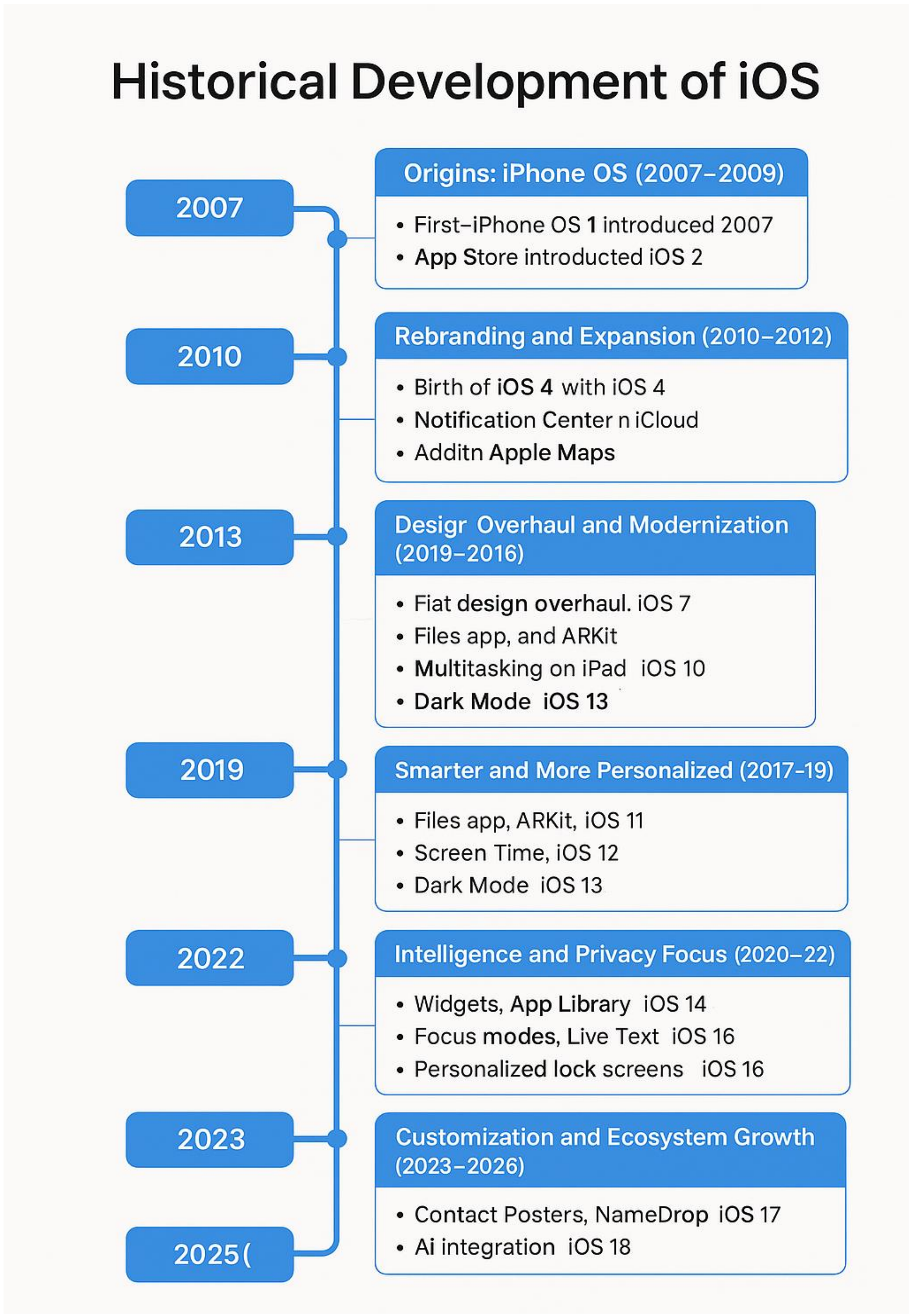
6. Customization and Ecosystem Growth (2023–2025)

2023 – iOS 17

- Introduced Contact Posters, NameDrop, and StandBy Mode.
- Boosted autocorrect and dictation, refined journaling and mental health tools.

2024 – iOS 18 (Expected)

- Stronger AI integration and further privacy too
- Even more customization features and ecosystemm



Historical Development of iOS

2007

Origins: iPhone OS (2007–2009)

- First–iPhone OS **1** introduced 2007
- **App Store** introducted iOS 2

2010

Rebranding and Expansion (2010–2012)

- Birth of **iOS 4** with iOS 4
- **Notification Center** n iCloud
- Additn **Apple Maps**

2013

Desigr Overhaul and Modernization (2019–2016)

- Fiat **design overhaul.** iOS 7
- Files app, and ARKit
- **Multitasking** on iPad iOS 10
- **Dark Mode iOS 13**

2019

Smarter and More Personalized (2017-19)

- Files app, ARKit, iOS 11
- Screen Time, iOS 12
- Dark Mode iOS 13

2022

Intelligence and Privacy Focus (2020–22)

- Widgets, **App Library** iOS 14
- **Focus modes,** Live Text iOS 16
- Personalized lock screens iOS 16

2023

Customization and Ecosystem Growth (2023–2026)

- Contact Posters, NameDrop iOS 17
- Ai integration iOS 18

2025(

•

B.Objectives

1. **Application Development & Testing**

   To develop iOS applications using Apple's **Xcode IDE** and test them on real or virtual iOS environments to ensure functionality.

- iOS must be installed on either a real device or emulator to **run compiled applications** .
- The iOS Simulator included with Xcode provides **emulated devices** for real-time testing.
- Without iOS, you cannot validate how the app **interacts with the OS**, such as notifications, camera, sensors, or location service.

2. UI/UX Validation

To confirm that the app's design meets iOS guidelines and provides a smooth, user-friendly interface on all supported devices.

- Apple's **Human Interface Guidelines (HIG)** set the gold standard for design.
- To test different **screen sizes**, **safe area insets**, and **device orientations** .
- For accessibility testing (e.g., **VoiceOver**, **Dynamic Type**) to ensure inclusivity.

3. Version Compatibility Checks

To ensure the application behaves correctly on different iOS versions, especially considering API deprecations and system behavior changes.

- Apple introduces **new APIs** and **deprecates old ones** regularly.
- Installing different versions of iOS (in simulators or on real devices) allows me to **simulate real-world fragmentation**.
- Backward compatibility testing is crucial to maintain a large user base and avoid crashes due to outdated system calls.

4. Security and Privacy Testing

To validate that the application adheres to iOS's strict security and privacy rules and handles user data responsibly.

- iOS enforces **app sandboxing**, meaning each app runs in its own environment.
- I tried to test permissions (camera, contacts, location), **secure storage using Keychain**, and **secure networking (HTTPS, SSL pinning)**.
- Apple reviews privacy labels; testing on iOS ensures proper configuration of **App Use Tracking Transparency (ATT)** prompts and **user consent flows**.

5. CI/CD Integration for iOS Builds

To integrate iOS into Continuous Integration/Deployment (CI/CD) workflows to automate testing, building, and deployment of applications.

- iOS builds can be automated using tools like **Fastlane**, **Jenkins**, **GitHub Actions**, **CircleCI**, or **Bitrise**.
- Simulators (or headless virtual devices) are used to **run test suites** during each commit.
- iOS is required to **sign builds using provisioning profiles**, which is part of the automated deployment process to the App Store or TestFlight.

6. Debugging and Performance Profiling

To investigate and resolve issues such as app crashes, slow UI rendering, memory leaks, or inefficient energy consumption.

- iOS comes with **Instruments**, a powerful profiling toolset in Xcode.
- Engineers monitor:
  - **Memory Usage**: to detect leaks and retain cycles.
  - **CPU Usage**: to catch performance bottlenecks.
  - **Energy Usage**: to optimize battery drain.
  - **Network Calls**: to monitor latency and response handling.
- Debugging on iOS gives precise insights **at the OS and hardware abstraction levels**.

Installing ios on pc

Virtualizing iOS directly on PC is legally and technically restricted. Instead, the process involves installing macOS in a virtual environment, which then provides access to iOS simulators via Xcode. This approach is essential for developers who lack Apple hardware and need a low-cost, educational alternative for iOS app testing and development.

Even using virtualization tools like virtualBox it is impossible to install ios on a non apple hardware .that is because virtualBox is designed to run x86 operating systems , while ios uses ARM architecture , which is fundamentally different .virtualBox does not support or emulate ARM architecture.

In addition , even if we try to install it in a vertualBox its ISO file is not possible to download because of legal restrictions that apples company has set about ios ISO file.

In today's mobile-centric world, iOS development has become an indispensable skill. However, setting up a proper iOS development environment is traditionally dependent on owning Apple hardware. This guide provides a thorough and practical approach to simulate the Apple ecosystem within a virtual environment. Using virtualization tools, developers can set up macOS on non-Mac hardware, install Xcode, and test iOS applications, thereby democratizing access to iOS development for learners, testers, and developers alike.

C.Requirements

To install iOS on a **PC (non-Mac computer)**, either **directly** (via rare emulators) or **indirectly** (via a virtual macOS + Xcode setup), we'll need to meet a set of **hardware**, **software**, and **system-level requirements**.

1.Hardware requirements

To install iOS in a virtual environment on a PC, the hardware must be powerful enough to handle a macOS virtual machine and the Xcode iOS simulator. At minimum, the system should have a quad-core Intel i5 or AMD Ryzen 5 processor that supports virtualization features like VT-x (for Intel) or AMD-V (for AMD). For smoother performance and responsiveness, especially when running resource-intensive tools like Xcode, a six-core or higher processor such as Intel i7 or Ryzen 7 is recommended. In terms of memory, 8 GB of RAM is the bare minimum, but 16 GB or more is ideal to run the host operating system, the virtual machine, and the simulator concurrently. Storage is also important—at least 80 GB of free disk space is needed, but 150 GB or more is preferred, particularly when downloading large macOS installers and Xcode components. Using a solid-state drive (SSD) instead of a traditional hard drive will significantly improve load times and performance. A dedicated

GPU isn’t mandatory, but it can help with smoother UI rendering inside the virtual environment. Lastly, the motherboard BIOS must support UEFI and have virtualization enabled to run macOS as a guest operating system.

2. Software Requirements

On the software side, the host PC should be running a 64-bit operating system—preferably Windows 10 or Windows 11. Advanced users may also use Linux as a host OS, though this requires additional manual configurations. The main tools needed to virtualize macOS are VMware Workstation or Oracle VM VirtualBox. VMware provides better performance and compatibility for macOS guests, especially when used with community patches such as Unlocker, which enables macOS options that aren’t normally available. For VirtualBox, EFI modifications and command-line tweaks are often necessary to allow the macOS image to boot and install correctly. In addition to virtualization software, you’ll need a legally acquired macOS image in .iso, .vmdk, or .vdi format—commonly, versions like macOS Monterey or Ventura are used. To build and simulate iOS apps, you’ll also need Xcode, Apple’s integrated development environment, which must be downloaded through the Mac App Store within the macOS virtual machine. Xcode includes everything required for iOS development: the SDK, build tools, a code editor, and most importantly, the iOS Simulator. An Apple ID is also necessary to access these developer tools legally.

3. Additional Tools & Utilities

Several supporting tools and utilities may be needed to complete the setup successfully. One of the most important is the “macOS Unlocker” tool for VMware, which unlocks hidden macOS guest options that are disabled by default. If you're using VirtualBox, you'll need to apply command-line modifications to set up EFI firmware, simulate an Apple-compatible CPU, and enable appropriate USB and graphics settings. If you’re planning to dual-boot rather than use a virtual environment, you might also work with bootloaders like Clover or OpenCore. These are often used in Hackintosh setups but can occasionally be helpful in boot-level configuration for VMs as well. An Apple Developer Account (free or paid) is needed to download macOS from the App Store and access Xcode’s full capabilities, including additional simulators for various iPhone and iPad models. You might also consider SSH tools or remote file sharing apps to manage the virtual machine more easily from your host system.

4. BIOS Configuration (Mandatory Steps)

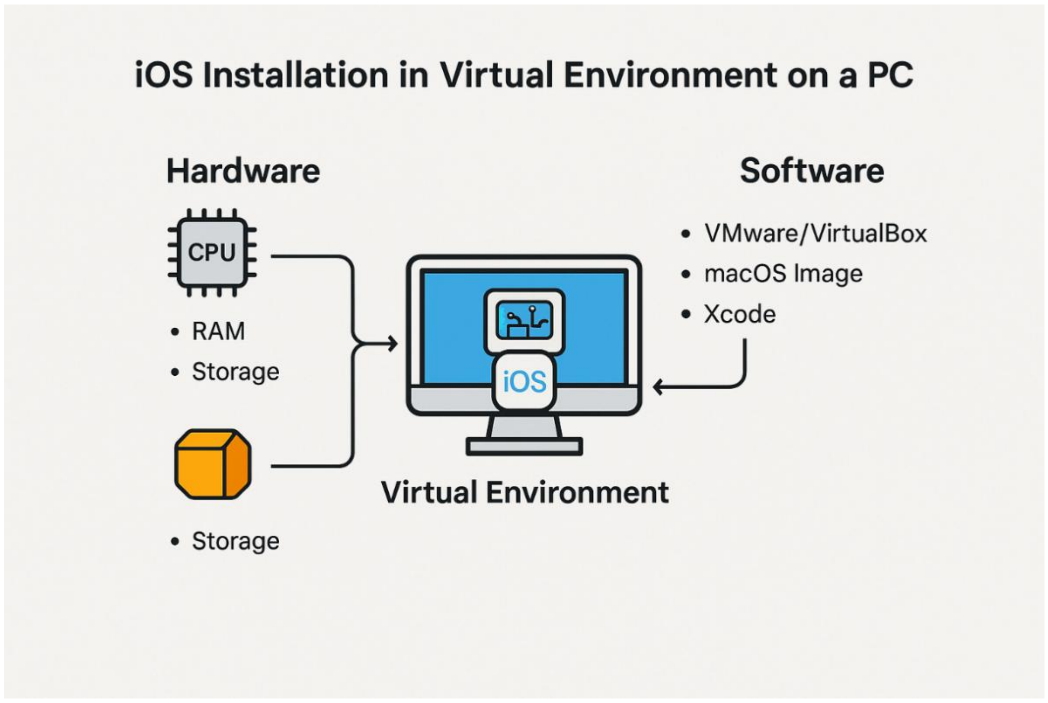
Before running any virtualized macOS system on a PC, it's crucial to adjust the BIOS settings. First, you need to enable virtualization technology—VT-x for Intel or AMD-V for AMD processors. Without this feature, 64-bit operating systems like macOS won't boot in a virtual machine. Next, you should ensure that the system is set to use UEFI boot mode, as macOS requires UEFI support for installation and operation. Many users also need to disable "Secure Boot," which can prevent modified or non-signed operating systems like macOS from loading properly. Some systems may require additional tweaks like enabling "VT-d" or "Hyper-Threading" for enhanced performance in certain virtual environments. If you're using VirtualBox, you'll likely need to manually tweak hardware acceleration and graphics options as well. These BIOS adjustments are mandatory and should be double-checked before proceeding with macOS installation.

5. Indirect iOS Installation Path (Recommended Method)

Since iOS is built for ARM processors and cannot be installed directly onto traditional PC hardware, the recommended method is to install macOS in a virtual machine and then use Xcode's iOS Simulator. This approach gives developers access to the official development tools and environments without violating Apple's hardware restrictions. After installing macOS in VMware or VirtualBox, the next step is to download Xcode through the Mac App Store. Once installed, you can launch the iOS Simulator, which behaves like a virtual iPhone or iPad, allowing you to test user interfaces, run apps, and debug code. This method replicates the iOS experience without needing a physical Apple device and is completely supported by Apple as long as it runs on genuine macOS. Although this isn't true iOS installation on the hardware, it's functionally equivalent for software development and testing, and it's the standard approach used by professional developers who don't own multiple Apple devices.

What WE CANNOT Do

Despite the flexibility of virtual environments, there are still clear limitations to what's possible when installing iOS-related environments on a PC. You cannot install iOS as a standalone OS directly onto the PC because it's designed for Apple's ARM-based processors, and PC processors are generally x86-based. You also can't use the App Store or install iOS .ipa files directly into the simulator unless they are built through Xcode. Real-device features like Face ID, GPS, the camera, and the Taptic Engine cannot be emulated accurately, though the simulator can mimic some of their behaviors. Additionally, due to Apple's licensing restrictions, running macOS or iOS on non-Apple hardware can violate the End User License Agreement, so these methods should only be used for educational and development purposes on Apple-owned devices when possible.



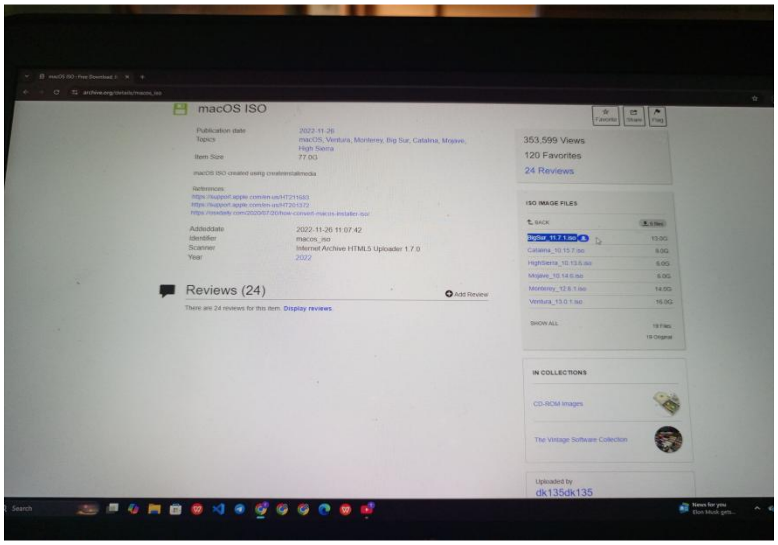
D. IOS INSTALLATION STEPS

NOTE
Under are listed installation steps for ios using virtualBox and macOS simulator under Xcode . from the steps 1-7 I was able to do up to step five but the preceding steps are taken from different resources that could be helpful for the full instillation of the operating system under the full support of requirements.
The main reasons which triggered me from proceeding were mainly the impossibility to download the ISO file of ios which affects the attaching file stage of step 3 .so I tried to install macOS in my virtualBox by which I was able to proceed upto step 5 ,but still because of hardware limitation which macOS needs a 64-bit processor and unfortunately me having a 32-bit intel processor made me stop at step 5.

Step 1: Check PC Requirements

- Ensure our PC has:
  - A 64-bit processor (Intel/AMD) with **VT-x/AMD-V** enabled.
  - At least **8–16 GB RAM**.
  - **150 GB or more** of free SSD space.
  - Virtualization support enabled in **BIOS/UEFI**.
  - **UEFI boot mode** (disable Secure Boot if needed).

Step 2: Download Required Files

- macOS image ISO bigsur   of mac
- 

macOS ISO

Publication date	2022-11-26
Topics	macOS, Ventura, Monterey, Big Sur, Catalina, Mojave, High Sierra
Item Size	77.0G

macOS ISO created using createinstallmedia

References
https://support.apple.com/en-us/HT211683
https://support.apple.com/en-us/HT201372
https://osxdaily.com/2020/07/20/how-convert-macos-installer-iso/

Addeddate	2022-11-26 11:07:42
Identifier	macos_iso
Scanner	Internet Archive HTML5 Uploader 1.7.0
Year	2022

Reviews (24)

There are 24 reviews for this item. Display reviews

353,599 Views
120 Favorites
24 Reviews

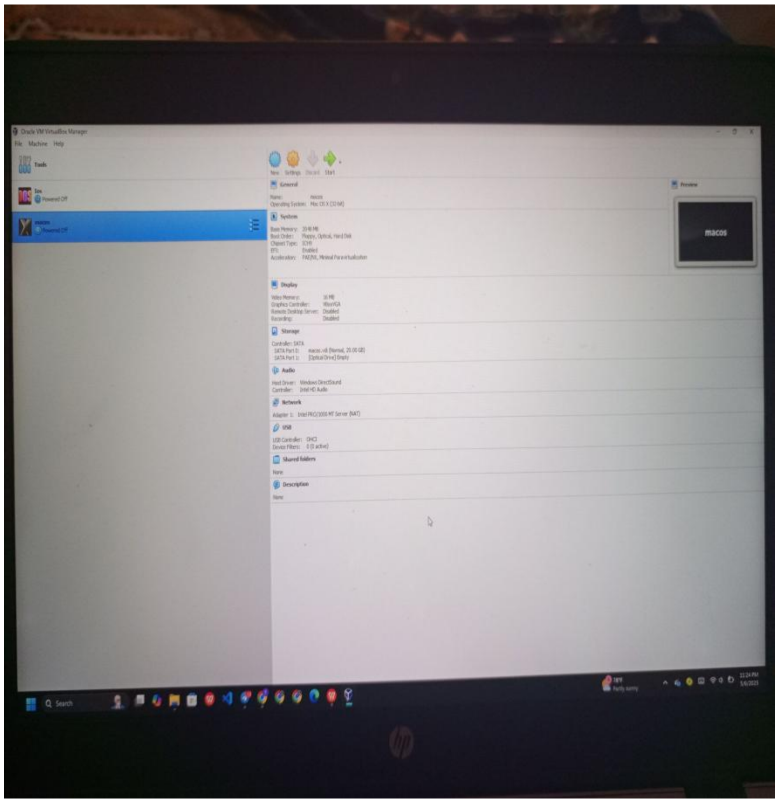
ISO IMAGE FILES

BigSur_11.7.1.iso	13.0G
Catalina_10.15.7.iso	8.0G
HighSierra_10.13.6.iso	6.0G
Mojave_10.14.6.iso	6.0G
Monterey_12.6.1.iso	14.0G
Ventura_13.0.1.iso	16.0G

SHOW ALL — 19 Files, 19 Original

IN COLLECTIONS
CD-ROM Images
The Vintage Software Collection

Uploaded by
dk135dk135

- Download **VirtualBox**.
- 

Oracle VM VirtualBox Manager

General
Name: macos
Operating System: Mac OS X (32-bit)

System
Base Memory: 2048 MB
Boot Order: Floppy, Optical, Hard Disk
Chipset Type: ICH9
EFI: Enabled
Acceleration: PAE/NX, Minimal Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: SATA
SATA Port 0: macos.vdi (Normal, 20.00 GB)
SATA Port 1: [Optical Drive] Empty

Audio
Host Driver: Windows DirectSound
Controller: Intel HD Audio

Network
Adapter 1: Intel PRO/1000 MT Server (NAT)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Description
None

- Extention : **Bootloader files or scripts**

Step 3: Set Up Virtual Machine

In my VirtualBox:

1. Install VirtualBox and create a new VM:
   - OS: macOS 64-bit

- Memory: 8–16 GB RAM
- Attach macOS image to virtual drive.

2. Run **command-line scripts** to tweak EFI and system flags.
3. Enable USB 3.0 and 128 MB video memory.
4. Start the VM and install macOS.

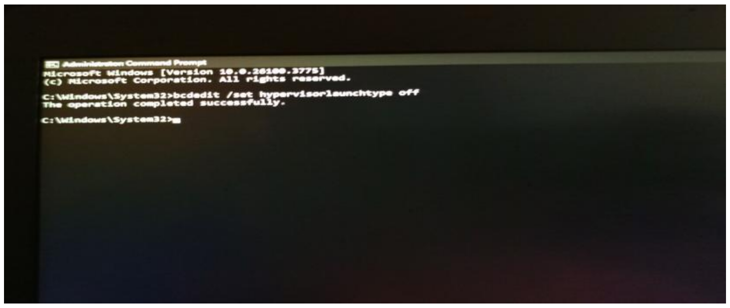
Step 4: command prompt

1. Disable windowshyper-v vertualization system to avoid further conflicts with virtualBox.

- Command prompt administrator

Run: bcdedit /set hypervisorlaunchtype off

- launch windows security >device security >core isolation details >turn off memory integrity.



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>bcdedit /set hypervisorlaunchtype off
The operation completed successfully.

C:\Windows\System32>
```

- finally I restarted my pc to start the new commands.

Step 5: Install macOS

- Boot into macOS installer.
- Use **Disk Utility** to format the virtual disk (APFS).
- Proceed with macOS installation.
- Set up user account and region once installation finishes.

Step 6: Install Xcode

1. Open the macOS virtual machine.
2. Sign in with your **Apple ID**.
3. Open **App Store** and download **Xcode**.

Step 7: Launch iOS Simulator

- Open **Xcode** after installation.
- Go to **Xcode > Open Developer Tool > Simulator**.
- Choose a device (e.g., iPhone 15, iPad Air).
- The iOS simulator boots and functions like an iPhone.

Notes & Limitations:

- You cannot install iOS directly on a PC like Android.
- No access to real-device hardware features (Face ID, camera).
- Performance may be slower than real Macs.

E.ISSUES OR PROBLEMS AND SOLUTIONS

Problems

1. Virtualization Disabled in BIOS

This setting is one of those often overlooked but is crucial.even if my CPU support virtualization I receive error "VT-x is not vailable .this might be BIOS or UEFI setting problem.

2. Secure Boot & UEFI Conflicts

Many modern systems come with Secure Boot enabled by default. This blocked the virtual machine from booting macOS properly, especially when using custom EFI configurations. Secure Boot is not disabled,so the virtual machine hang at a black screen .

3. macOS Installation Freezes or Fails

the macOS installer freezes at the Apple logo, loading bar, or "less than a minute remaining" stage.

This can be caused by:

- Incompatible macOS version with your virtualization software.
- Incorrect VM configuration (like CPU cores or RAM).
- Damaged or corrupted macOS image (.iso/.vmdk).
- Missing EFI or bootloader configurations in VirtualBox.

4. Post-Installation Boot Loop

After installing macOS, I face a **boot loop** when restarting. This can be due to:

- Missing or incorrect EFI configuration.
- Unsupported macOS updates.
- Misconfigured VM settings.
- System kernel panic errors from virtualization conflicts.

4. Ios ISO file impossibility to download

- Though its not a problem which can be resolved in system level my biggest problem was that it is impossible to download ISO file or image of ios because of legal restrictions that apple has set.

F. Solutions

1. Virtualization Disabled in BIOS

Solution:
Virtualization is usually disabled by default in most BIOS settings.

1. Restart my PC and enter the BIOS/UEFI setup (press **F2**, **DEL**, **ESC**, or **F10** during boot—depends on manufacturer).
2. Locate the **Virtualization Technology** setting:
   - On Intel CPUs: **Intel VT-x**
3. Enable the setting, then save and exit BIOS.

2 . Secure Boot or UEFI Boot Mode Issues

Solution:
Secure Boot can block unsigned operating systems like macOS.

1. Go to BIOS/UEFI settings.
2. Locate and **disable Secure Boot** under Boot or Security options.
3. Set the boot mode to **UEFI only**, not Legacy.

3. macOS Installation Freezes or Fails

Solution:
If macOS gets stuck at the Apple logo or freezes:

- Recheck my VM settings:
  - Assign **4 CPU cores**, **8–12 GB RAM**, and **128 MB VRAM**.
  - Enable **EFI**, **I/O APIC**, and **Virtualization** in VM configuration.
- Recheck, make sure I've entered the **command-line tweaks** properly (like VBoxManage commands).
- Tried to re-download the macOS image if it was corrupt.
- Use **Big Sur** images, as they are more stable with VMs.

4. Boot Loop After Installation

Solution:
A common error after installation is getting stuck in a reboot cycle:

- Reheck the macOS drive is selected as the boot device in VM settings.
- Reapply **EFI boot files**
- Start the VM in **Safe Mode** by holding **Shift** during boot.
- Roll back to a **previous VM snapshot**
- redo the installation if files were damaged.

G. FILESYSTEM SUPPORT

1. APFS (Apple File System)
- IOS supports this system (primary macOS/iOS file system)

APFS is the default file system for macOS (from High Sierra onward) and iOS. It is **required** when installing modern versions of macOS in a virtual machine, particularly if you want to install Xcode or run iOS simulators. It is not just a file system support it is an integral part of the operating system`s architecture and it enables many of the features that make **ios** a modern and reliable mobile operating system.
It offers:

1. The foundation for storage : boot volume,containerization,Fast SSD performance
2. Flash storage : performance, space sharing ,Snapshot support
3. Security and data integrity : native encryption ,copy-on-write, snapshots
4. Cloning
5. Under pining system features : system integrity protection(SIP), software updates

2. HFS+ (Mac OS Extended / Journaled)
-IOS supports this system

HFS+ is the **older macOS file system**, still usable in macOS but replaced by APFS in modern versions. It may be useful if you're using macOS versions older than High Sierra or creating compatibility partitions.

3. NTFS (New Technology File System)
- IOS Supports this system but Not natively writable on macOS

macOS can read **NTFS**, but cannot write to it without third-party drivers (e.g., Paragon NTFS or Tuxera). It's a Windows file system and not usable for formatting the internal virtual disk of macOS.

But still it is useful for **external drives** or **shared folders** between Windows host and macOS guest with read-only access.

4. FAT32 (File Allocation Table 32)
-IOS supports this system especially for interoperability.

macOS **can read and write** to FAT32 drives. However, it has:

- File size limit of **4 GB**
- Partition size limit of **32 GB**

It is great for transferring small files between macOS VM and Windows host, but **not suitable** for installing macOS or Xcode.

5. exFAT
- IOS supports this system.

exFAT is supported by both Windows and macOS with **no 4 GB file size limit**, making it ideal for **file exchange** between the two systems.
It is the best choice for **external drives** or shared virtual folders between macOS and Windows.

6. ext4 (Extended File System 4)
- This system is not supported by IOS.

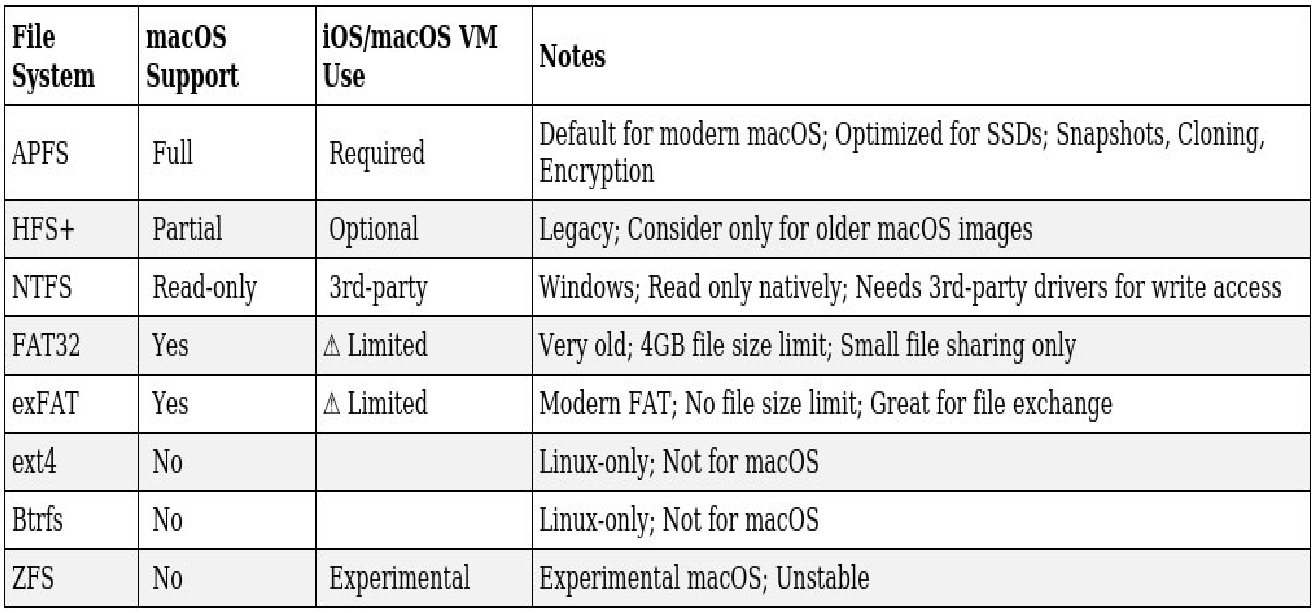
ext4 is the standard Linux file system. macOS does not natively support reading or writing to it. You'd need third-party drivers or mounting tools.

7. Btrfs (B-tree File System)
- This system is not supported by IOS.

Btrfs is a Linux-focused advanced file system. macOS does not support it at all.

8. ZFS (Zettabyte File System)
- This system is not supported by IOS.

ZFS was never officially adopted by macOS. Some early experimental versions existed, but it is not supported in modern macOS. Third-party ZFS support is highly experimental.
But it is not recommended or supported for macOS VMs.



File System	macOS Support	iOS/macOS VM Use	Notes
APFS	Full	Required	Default for modern macOS; Optimized for SSDs; Snapshots, Cloning, Encryption
HFS+	Partial	Optional	Legacy; Consider only for older macOS images
NTFS	Read-only	3rd-party	Windows; Read only natively; Needs 3rd-party drivers for write access
FAT32	Yes	⚠ Limited	Very old; 4GB file size limit; Small file sharing only
exFAT	Yes	⚠ Limited	Modern FAT; No file size limit; Great for file exchange
ext4	No		Linux-only; Not for macOS
Btrfs	No		Linux-only; Not for macOS
ZFS	No	Experimental	Experimental macOS; Unstable

H.ADVANTAGES AND DISADVANTAGS

Advantages of iOS

1. Stable and Consistent Performance

- Apple optimizes both hardware and software, resulting in a seamless and **high-performance ecosystem**.
- Rarely experiences crashes or performance lags compared to fragmented Android systems.

2. Strong Security and Privacy

- iOS uses **sandboxing, encryption, and secure boot chains**, making it one of the most secure mobile operating systems.
- Regular updates and **strict App Store policies** prevent malicious software.
- Face ID and Touch ID are deeply integrated for secure authentication.

3. Regular and Timely Updates

- All supported devices receive updates **simultaneously**—no delay across carriers or manufacturers.
- Enables developers to build apps that use the latest APIs without worrying about version fragmentation.

4. High-Quality Developer Tools

- Tools like **Xcode**, **Swift**, and **UIKit** are powerful and well-maintained.
- Includes built-in simulators, storyboards, and accessibility tools.
- **SwiftUI** allows for rapid interface design using modern declarative syntax.

5. Strong Monetization Opportunities

- iOS users are known to **spend more** on apps and in-app purchases than Android users.

- Higher chance of **profitability** for premium and subscription-based apps.

6. Ecosystem Integration

- Seamless interaction with **MacBooks, Apple Watch, iPad, Apple TV**, etc.
- Features like **Handoff, Universal Clipboard, AirDrop**, and **iCloud Sync** enhance user experience across devices.

Disadvantages of iOS

1. Closed Source / Less Customization

- iOS is a **closed ecosystem**, meaning developers and users have limited control over the system internals.
- You can’t customize the UI or file system deeply like you can in Android or Linux-based platforms.

2. Strict App Store Policies

- Apple’s review process is known for being **slow and strict**.
- Even small violations of guidelines can lead to app rejections or bans.
- Developers must pay an **annual $99 fee** for the Apple Developer Program.

3. Limited Device Variety

- Fewer device models compared to Android means **less hardware diversity** to innovate with.
- You’re restricted to Apple’s hardware choices (e.g., no expandable storage or headphone jacks on many devices).

4. High Cost of Entry

- Apple devices are expensive, especially for beginners or students.
- Requires a **Mac** for development (to use Xcode), increasing upfront costs.
- Virtual macOS setups are limited and often unstable compared to native Apple machines.

5. No Default App Replacements

- Until recent versions, iOS didn’t allow changing default apps like browser or mail client.
- Still limited in terms of **inter-app communication** compared to Android.

I.conclusion

The iOS operating system stands as a benchmark of **stability, security, and premium user experience** in the mobile industry. Its tightly integrated hardware-software architecture ensures consistently high performance and long-term software support, which benefits both users and developers. iOS’s strong emphasis on **privacy, data security, and regular updates** sets it apart as one of the most reliable platforms available today.

From a software engineering perspective, iOS offers a polished development environment through tools like **Xcode and Swift**, facilitating efficient app development and testing. Its cohesive ecosystem also enables cross-device continuity and a loyal user base with higher monetization potential.
However, iOS's **closed-source nature**, strict App Store regulations, and the **high cost of entry** pose limitations—especially for beginners, hobbyists, or those seeking deep system-level customization. Despite these constraints, iOS remains a **leading platform for developing secure, high-quality, and revenue-generating applications**, especially in commercial and enterprise environments.

J.Future Outlook of iOS

As Apple continues to evolve its ecosystem, the future of iOS looks exceptionally promising. The platform is gradually merging more tightly with **macOS, iPadOS, and visionOS**, creating a **unified development environment** across devices. With the rise of technologies like **SwiftUI, ARKit, CoreML (Machine Learning), and RealityKit**, iOS development is moving toward a **more immersive, AI-powered, and declarative future**.
Apple's focus on **sustainability, privacy-first innovation, and on-device intelligence** ensures iOS will remain a key player in the next era of mobile computing. In addition, the expansion of **Apple Silicon** (M1/M2/M3 chips) across its entire product line enables performance parity between desktop and mobile platforms, paving the way for deeper app capabilities.
From a developer's standpoint, **Apple's push toward spatial computing (e.g., Vision Pro)** will create new markets for apps in **mixed reality**, making now an ideal time to get comfortable with Swift, Xcode, and the iOS ecosystem.

Recommendations for Developers

1. **Invest in Learning Swift & SwiftUI**
   The iOS development stack is becoming increasingly centered on Swift and SwiftUI. Mastering these early gives you a competitive edge and prepares you for app development across **all Apple platforms**.
2. **Use Virtual Environments for Prototyping**
   If a real Mac isn't available, use virtualized macOS for initial development and testing. But keep in mind: a **physical Mac** is still essential for final testing, app store submission, and performance optimization.

3. **Prepare for Cross-Platform Growth**
   Apple is unifying APIs across iPhone, iPad, Mac, and even wearables. Learn how to build adaptive UIs and cross-device experiences using tools like **Universal Apps and Catalyst**.
4. **Leverage New Technologies**
   Explore **ARKit**, **CoreML**, and **HealthKit**—these frameworks are growing in demand across industries like health, finance, education, and entertainment.
5. **Stay Informed on Legal & Licensing Boundaries**
   When using macOS in virtual machines, understand that it's mainly for **educational/testing purposes** due to Apple's licensing policies. For production or deployment work, consider cloud-based Mac services or invest in a Mac.
6. **Focus on Performance and UX**
   iOS users expect high responsiveness and intuitive interfaces. Make full use of Apple's **Human Interface Guidelines** to build polished, user-centered apps.

Virtualization in Modern Operating Systems: What, Why, and How

1. What is Virtualization?

Virtualization is a technology that enables the creation of multiple simulated environments or dedicated resources from a single, physical hardware system. In the context of operating systems, virtualization allows one physical machine (host) to run multiple operating systems (guests) simultaneously, each within its own isolated environment called a Virtual Machine (VM).

At the heart of virtualization lies a software layer known as a hypervisor, which sits between the hardware and the virtual machines. The hypervisor manages and allocates hardware resources (CPU, memory, disk space, etc.) to each VM and ensures they operate independently.

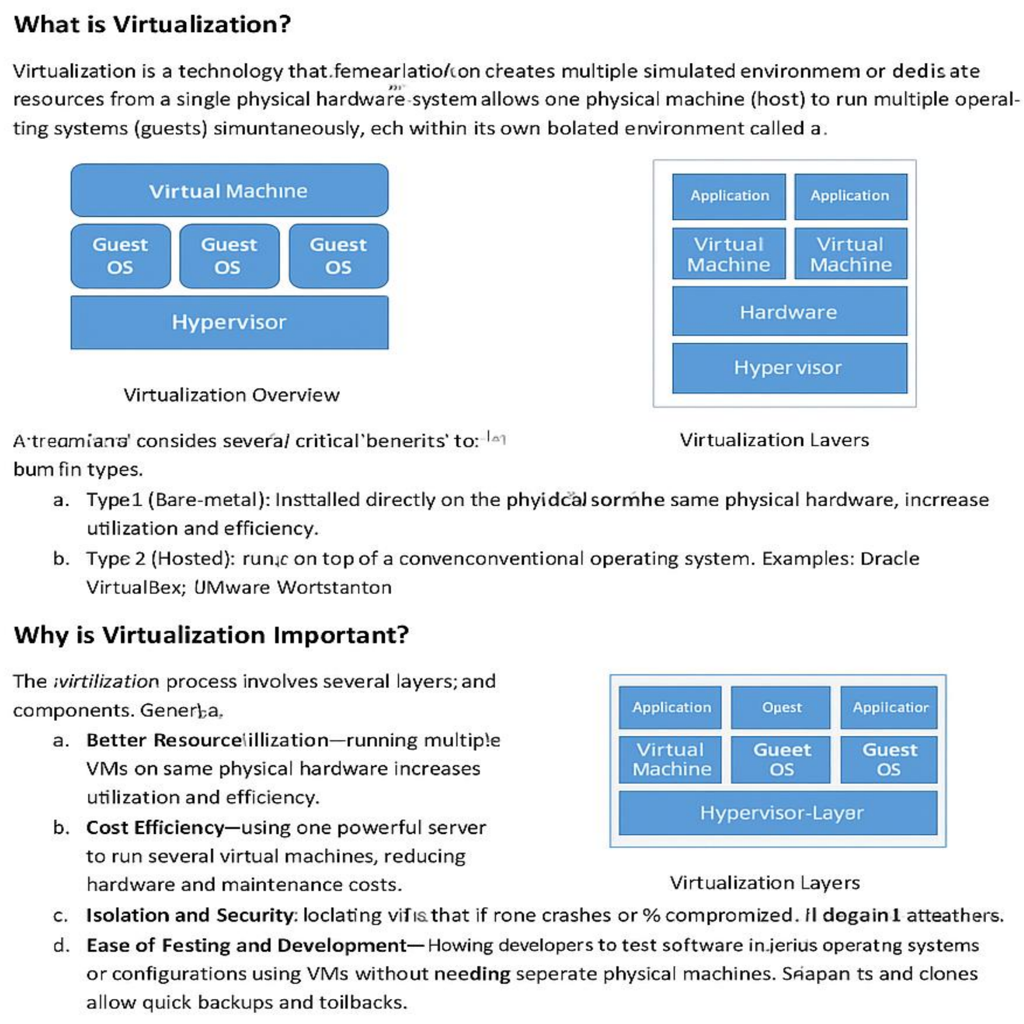
There are two main types of hypervisors:

• Type 1 (Bare-metal): default

Installed directly on the physical hardware. Examples: VMware ESXi, Microsoft Hyper-V, Xen.

• Type 2 (Hosted):needs installation

Runs on top of a conventional operating system. Examples: Oracle VirtualBox, VMware Workstation.



What is Virtualization?

Virtualization is a technology that.femearlatio/(on creates multiple simulated environmem or dedis ate resources from a single physical hardware system allows one physical machine (host) to run multiple operal-ting systems (guests) simuntaneously, ech within its own bolated environment called a.

Virtual Machine
Guest OS
Guest OS
Guest OS
Hypervisor

Virtualization Overview

Application
Application
Virtual Machine
Virtual Machine
Hardware
Hyper visor

Virtualization Lavers

A·treamiana' consides several critical`benerits' to:
bum fin types.

a. Type1 (Bare-metal): Insttalled directly on the phyidcal sormhe same physical hardware, incrrease utilization and efficiency.
b. Type 2 (Hosted): run.c on top of a convenconventional operating system. Examples: Dracle VirtualBex; UMware Wortstanton

Why is Virtualization Important?

The *ivirtilization* process involves several layers; and components. Generba.

a. **Better Resource**illization—running multiple VMs on same physical hardware increases utilization and efficiency.
b. **Cost Efficiency—**using one powerful server to run several virtual machines, reducing hardware and maintenance costs.
c. **Isolation and Security**: loclating vif is that if rone crashes or % compromized. **Il dogain1** atteathers.
d. **Ease of Festing and Development—** Howing developers to test software in.jerius operatng systems or configurations using VMs without **needing** seperate physical machines. Sńapan ts and clones allow quick backups and toilbacks.

Virtualization Layers

2. Why is Virtualization Important?

Better Resource Utilization

Physical machines often do not use their resources (CPU, RAM) to full capacity. Virtualization allows multiple VMs to share the same physical hardware, increasing overall utilization and efficiency.

Cost Efficiency
Instead of purchasing multiple physical servers, organizations can use one powerful server to run several virtual machines, reducing hardware and maintenance costs.
Isolation and Security
Each VM runs in its own isolated environment. If one VM crashes or is compromised, it doesn’t affect others.
Ease of Testing and Development

Developers can test software in various operating systems using VMs without separate physical machines.

Scalability and Flexibility

Virtualization makes it easier to scale systems up or down and supports cloud computing.

Disaster Recovery

VMs can be easily backed up and restored, aiding in disaster recovery scenarios.

3. How Does Virtualization Work?

Hardware Layer

This is the actual physical system – the CPU, RAM, storage, network interfaces, etc.

Hypervisor Layer

The hypervisor abstracts the hardware and provides virtual hardware to each VM, managing resource allocation and ensuring isolation.

Guest Operating Systems

Each virtual machine runs its own guest OS, which may differ from the host OS.

Virtual Hardware

Each VM is given virtual versions of CPU, RAM, hard drives, and network cards.

Host Operating System

In Type 2 virtualization, the hypervisor runs on a regular OS that manages the hardware.

Real-World Use Cases of Virtualization

- Cloud Computing: Services like AWS, Azure, and Google Cloud use virtualization for scalable infrastructure.
- Software Testing: Developers test applications in multiple OS environments using VMs.
- Education and Training: Virtual machines simulate different systems for lab use.
- Legacy System Support: Old software can run in VMs without old hardware.

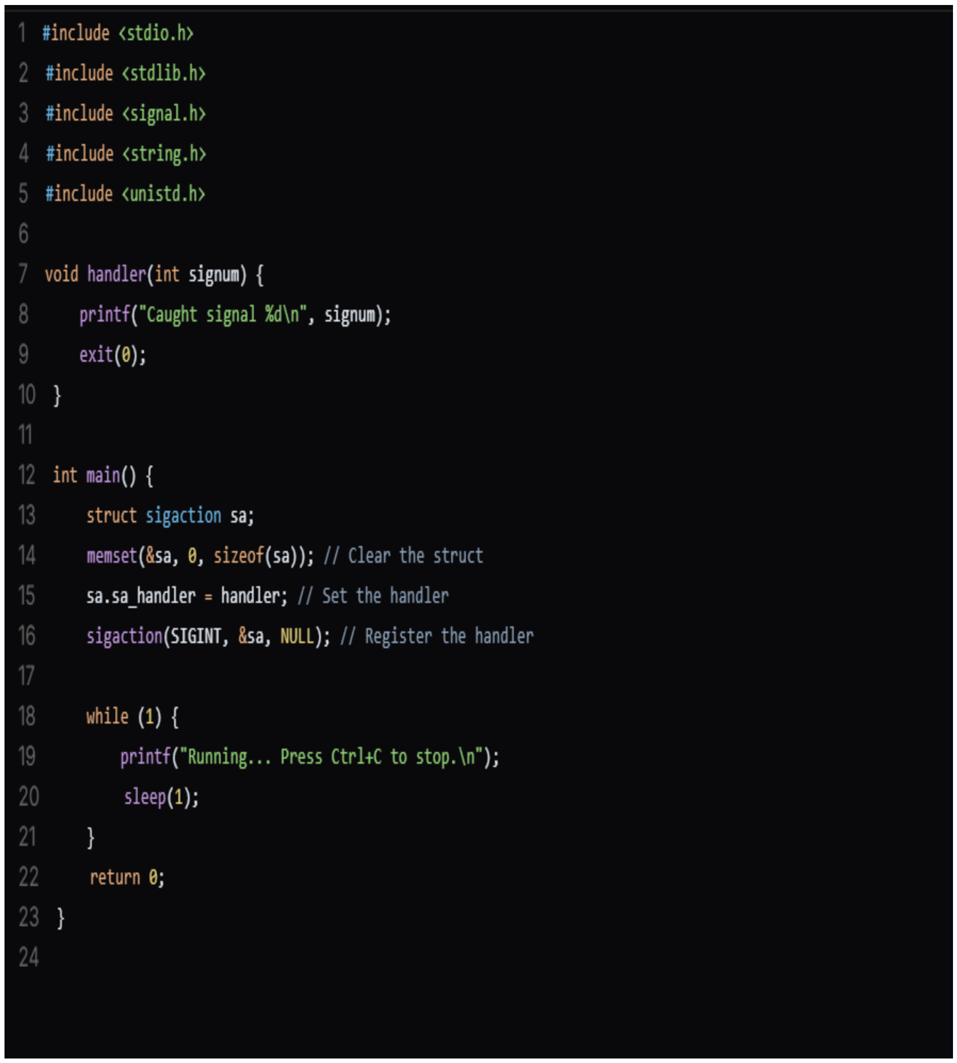
Conclusion

Virtualization in modern operating systems is a foundational technology that enables efficient, flexible, and cost-effective computing. It allows multiple operating systems to run on the same hardware while maintaining isolation and independence. Through hypervisors and virtual machines, it enhances resource utilization, simplifies system management, and supports modern IT infrastructure needs.

IMPLEMENT SYSTEM CALL sigaction()

The sigaction() system call in Unix-like operating systems is used to change the action taken by a process on receipt of a specific signal. It allows you to specify a new signal handler for a signal, as well as to retrieve the current action for that signal.This system call is a fundamental part of the POSIX standard for signal handling in Unix-like operating systems including Linux,macOS,BSD ,etc. It allows as to examine and/or modify the action associated with a specific signal. Also it is much more powerful and flexible than the older signal() function.

- sigaction() provides a more flexible and reliable way to handle signals compared to signal().
- it allows for the specification of additional options and the retrieval of the previous signal action .
-proper use of sigaction() can help manage complex signal handling scenarios in application.



```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

void handler(int signum) {
    printf("Caught signal %d\n", signum);
    exit(0);
}

int main() {
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa)); // Clear the struct
    sa.sa_handler = handler; // Set the handler
    sigaction(SIGINT, &sa, NULL); // Register the handler

    while (1) {
        printf("Running... Press Ctrl+C to stop.\n");
        sleep(1);
    }
    return 0;
}
```

Signal Handler: The handle_sigint function is defined to handle the SIGINT signal. When the signal is received, it prints a message and exits the program.

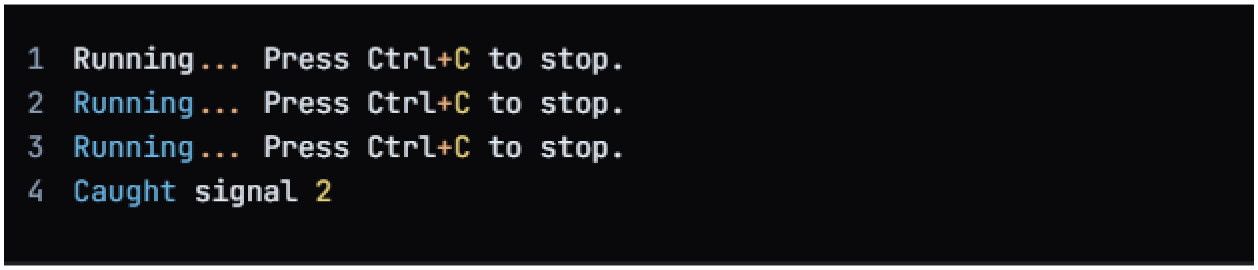
sigaction Structure: A struct sigaction is created and initialized. The sa_handler field is set to point to the signal handler function. The sa_mask is initialized to an empty set, meaning no additional signals will be blocked while the handler is executing. The sa_flags field is set to 0, indicating no special behavior.

Setting the Signal Action: The sigaction() function is called to set the action for SIGINT. If it fails, an error message is printed, and the program exits.

Main Loop: The program enters an infinite loop, calling pause() to wait for signals. When SIGINT is received (by pressing Ctrl+C), the signal handler is invoked.

When you run the program, it will wait for you to press Ctrl+C. Upon doing so, it will execute the signal handler and exit gracefully.

- The program will look like this if ctrl+c is pressed after a few iterations .



```
Running... Press Ctrl+C to stop.
Running... Press Ctrl+C to stop.
Running... Press Ctrl+C to stop.
Caught signal 2
```