AOS Assignment-1 Part-A

Bratin Mondal 21CS10016

Datta Ksheeraj 21CS30037

Objective:

To configure, build, and install a customized Linux kernel from the source, focusing on specific modifications such as the removal of the IPv6 protocol and integrating the Kernel-based Virtual Machine (KVM) as an inbuilt feature, rather than as a loadable kernel module.

IPv6:

IPv6, the latest version of the Internet Protocol (IP), significantly improves upon IPv4 by expanding the address space from 32-bit to 128-bit, accommodating the growing number of internet-connected devices. It features a simplified header structure, enhancing routing efficiency and speeding up packet processing. Unlike IPv4, IPv6 includes mandatory support for IPsec, providing built-in security through packet authentication and encryption. Additionally, IPv6 offers better support for mobile devices with Mobile IPv6, and improved data transmission efficiency through multicast and anycast capabilities.

Specifically turn off the IPV6 support in the kernel configuration by navigating to the following path in the menuconfig

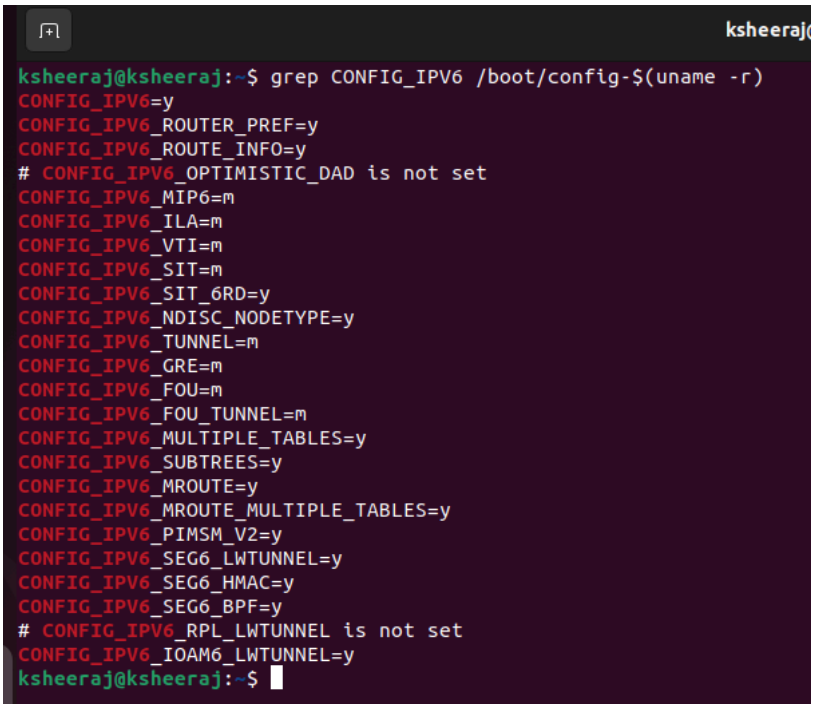
Networking support --->
Networking options --->
<> The IPv6 protocol

Before Installation of kernel:

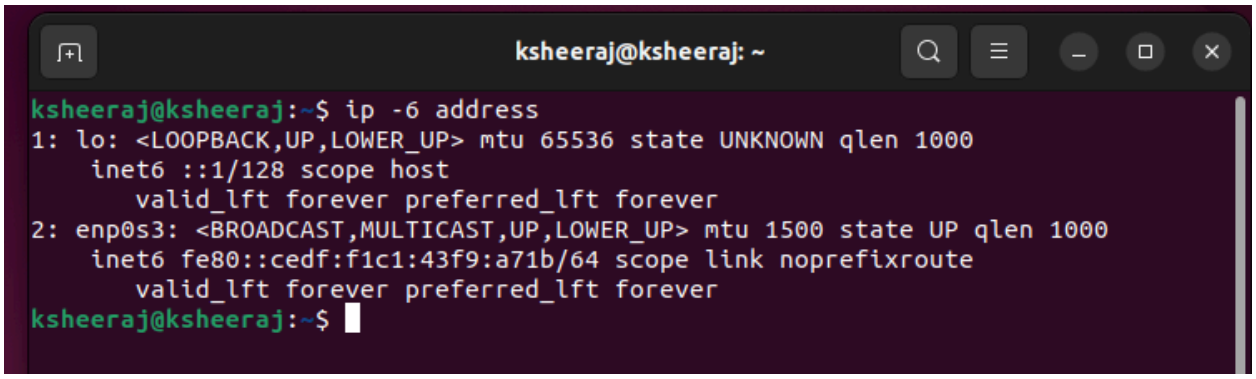
1. The config file had different options enabled for configuring IPv6.
    a. **CONFIG_IPV6=y**: Enables basic support for the IPv6 protocol in the kernel, allowing the system to handle IPv6 traffic.
    b. **CONFIG_IPV6_ROUTER_PREF=y**: Allows the kernel to set router preferences in Router Advertisement messages, enabling hosts on the network select the best router to use.
    c. **CONFIG_IPV6_ROUTE_INFO=y**: Enables the kernel to include route information options in Router Advertisement messages allowing routers to advertise more specific routes to hosts.
    d. **CONFIG_IPV6_OPTIMISTIC_DAD is not set**: The "Optimistic Duplicate Address Detection" (DAD) feature is disabled.

e. **CONFIG_IPV6_MIP6=m**: Compiles support for Mobile IPv6 as a module. It allows a mobile device to maintain the same IP address while moving between different networks.
f. **CONFIG_IPV6_ILA=m**: Compiles support for Identifier-Locator Addressing (ILA) as a module. ILA allows separating the network identity and location in the address space, facilitating network virtualization and mobility.
g. **CONFIG_IPV6_VTI=m**: Compiles support for Virtual Tunnel Interfaces (VTI) for IPv6 as a module. VTI is used for tunneling IPv6 packets over an IPv4 or IPv6 network, often for VPN connections.
h. **CONFIG_IPV6_SIT=m**: Compiles support for the Simple Internet Transition (SIT) tunneling protocol as a module. SIT is used to encapsulate IPv6 packets inside IPv4 packets, enabling IPv6 traffic to traverse IPv4 networks.
i. **CONFIG_IPV6_SIT_6RD=y**: Enables support for 6rd (IPv6 Rapid Deployment), an extension of the SIT tunneling protocol. 6rd allows an ISP to rapidly deploy IPv6 over its existing IPv4 infrastructure by providing a method to encapsulate IPv6 packets within IPv4.
j. **CONFIG_IPV6_NDISC_NODETYPE=y**: Enables support for determining the node type (host or router) during IPv6 Neighbor Discovery.
k. **CONFIG_IPV6_TUNNEL=m**: Compiles support for generic IPv6 tunneling as a module. This allows the encapsulation of IPv6 packets within another IPv6 header.
l. **CONFIG_IPV6_GRE=m**: Compiles support for GRE (Generic Routing Encapsulation) tunnels over IPv6 as a module. GRE is a protocol that encapsulates packets in order to tunnel them over different networks.
m. **CONFIG_IPV6_FOU=m**: Compiles support for the Foo-over-UDP tunneling protocol as a module. FOU allows encapsulation of IPv6 packets in UDP.
n. **CONFIG_IPV6_FOU_TUNNEL=m**: Compiles support for FOU tunneling specifically for IPv6 as a module. This enables the creation of UDP-based tunnels for IPv6 traffic.
o. **CONFIG_IPV6_MULTIPLE_TABLES=y**: Enables support for multiple routing tables for IPv6, allowing the system to maintain different routing policies for different traffic types or network interfaces.
p. **CONFIG_IPV6_SUBTREES=y**: Enables support for hierarchical routing tables in IPv6.
q. **CONFIG_IPV6_MROUTE=y**: Enables support for IPv6 multicast routing. Multicast allows the delivery of packets to multiple destinations in a single transmission.
r. **CONFIG_IPV6_MROUTE_MULTIPLE_TABLES=y**: Enables support for using multiple routing tables specifically for IPv6 multicast routes.
s. **CONFIG_IPV6_PIMSM_V2=y**: Enables support for Protocol Independent Multicast - Sparse Mode version 2 (PIM-SM v2) for IPv6. PIM-SM is a protocol used to efficiently route multicast traffic to multiple receivers across a network.
t. **CONFIG_IPV6_SEG6_LWTUNNEL=y**: Enables support for Segment Routing over IPv6 (SRv6) with lightweight tunnels. SRv6 allows the source of a packet to specify a list of segments (nodes or paths) that the packet should traverse.
u. **CONFIG_IPV6_SEG6_HMAC=y**: Enables support for HMAC (Hash-Based Message Authentication Code) in SRv6. This adds an authentication mechanism to segment routing.
v. **CONFIG_IPV6_SEG6_BPF=y**: Enables support for integrating eBPF (Extended Berkeley Packet Filter) with SRv6. This allows for the execution of custom programs in the kernel that can modify or analyze segment routing headers.
w. **CONFIG_IPV6_RPL_LWTUNNEL is not set**: The option for Routing Protocol for Low-Power and Lossy Networks (RPL) over IPv6 with lightweight tunnels is disabled. RPL is used in IoT networks to provide efficient routing.

x. **CONFIG_IPV6_IOAM6_LWTUNNEL=y**: Enables support for In-situ Operations, Administration, and Maintenance (IOAM) over IPv6 with lightweight tunnels. IOAM allows the embedding of operational data within IPv6 packets, such as performance metrics, for real-time monitoring and troubleshooting.

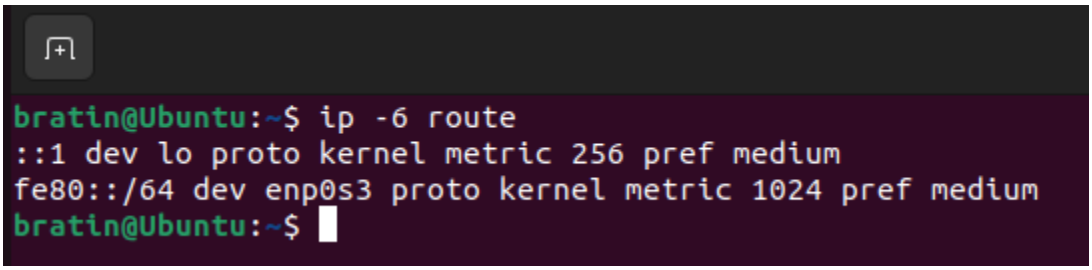


2. On using the `ip -6 address` command, the output showed that the linux had a valid IPv6 address. The screenshot shows two IPv6 addresses on the system. The `lo` (loopback) interface has an IPv6 address `::1/128`, which is the loopback address for local communication within the host. The `enp0s3` interface has a link-local IPv6 address starting with `fe80::`, which is used for communication within the local network segment. Both addresses are marked with `valid_lft` and `preferred_lft` as "forever," indicating they are valid and preferred for use indefinitely. The `noprefixroute` flag on the `enp0s3` interface means that the address will not automatically generate a route based on the prefix.



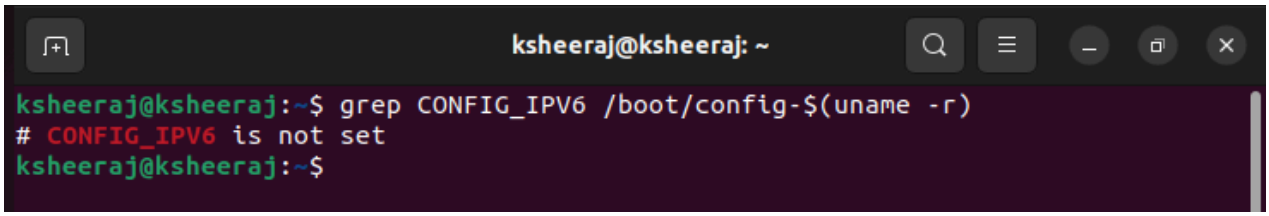
3. The `ip -6 route` shows the IPv6 routing table on the system.

Loopback Address Route (`::1 dev lo`): This route directs IPv6 traffic destined for the loopback address `::1` through the loopback interface (`lo`).
Link-Local Address Route (`fe80::/64 dev enpos3`): This route manages IPv6 traffic for the link-local address range `fe80::/64`, directing it through the `enpos3` network interface.

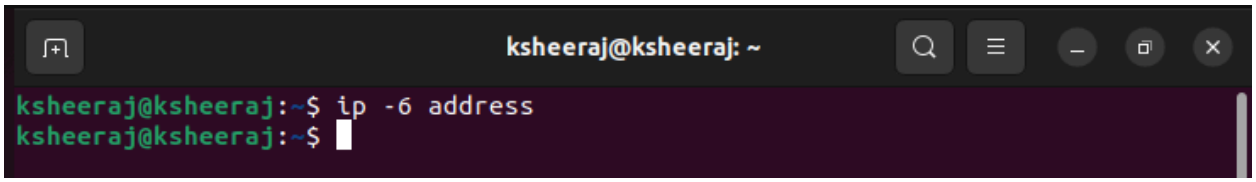


After Installation of kernel:

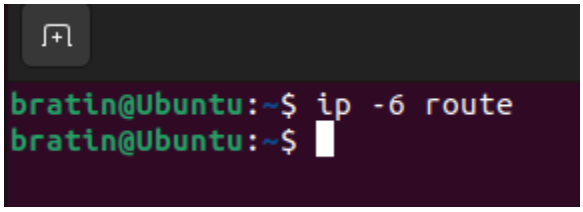
1. The config file has no options set for IPv6, only one comment stating that Configuration option for IPv6 is not set



2. On using the `ip -6 address` command, no output was shown meaning that the device doesn't have a valid IPv6 address.



3. The `ip -6 route` shows the empty IPv6 routing table on the system.



KVM (Kernel Virtual Machine):

Kernel-based Virtual Machine (KVM) is a vital component of Linux's virtualization capabilities, turning the Linux kernel into a type-1 (bare-metal) hypervisor. This allows multiple virtual machines (VMs) to run on a single physical host, each with its own isolated environment, including virtualized hardware components like CPU, memory, and network interfaces.

KVM is tightly integrated with the Linux kernel, which means it benefits from all the performance, scalability, and security features that Linux offers. It supports both hardware-assisted virtualization (leveraging CPU extensions like Intel VT-x and AMD-V) and paravirtualization, where the guest operating system is aware of the virtualization environment and can interact with the hypervisor more efficiently.

One of the strengths of KVM is its ability to manage resources dynamically. It can allocate CPU, memory, and I/O resources to VMs based on their needs, ensuring optimal performance across all VMs running on the host. KVM also supports live migration, allowing VMs to be moved between hosts without downtime, which is crucial for maintaining service availability in production environments.

For developers and administrators, KVM provides extensive tooling and APIs, such as libvirt, for managing and automating virtualized environments. Additionally, KVM's integration with the Linux kernel ensures it receives regular updates and improvements, making it a robust and reliable solution for virtualization.

To make KVM an inbuilt feature instead of a kernel module, navigate to the following path in the menuconfig

Virtualization ---->
<*> Kernel-based Virtual Machine (KVM) support
[*] Compile KVM with -Werror
<*> KVM for Intel (and compatible) processors support
<*> KVM for AMD processors support
[*] Audit KVM MMU

The first option, "Kernel-based Virtual Machine (KVM) support," enables the core KVM functionality, allowing the kernel to manage virtual machines. The "Compile KVM with -Werror" option ensures that any warnings during the compilation of KVM code are treated as errors, helping to maintain code quality by preventing potentially problematic code from being compiled.

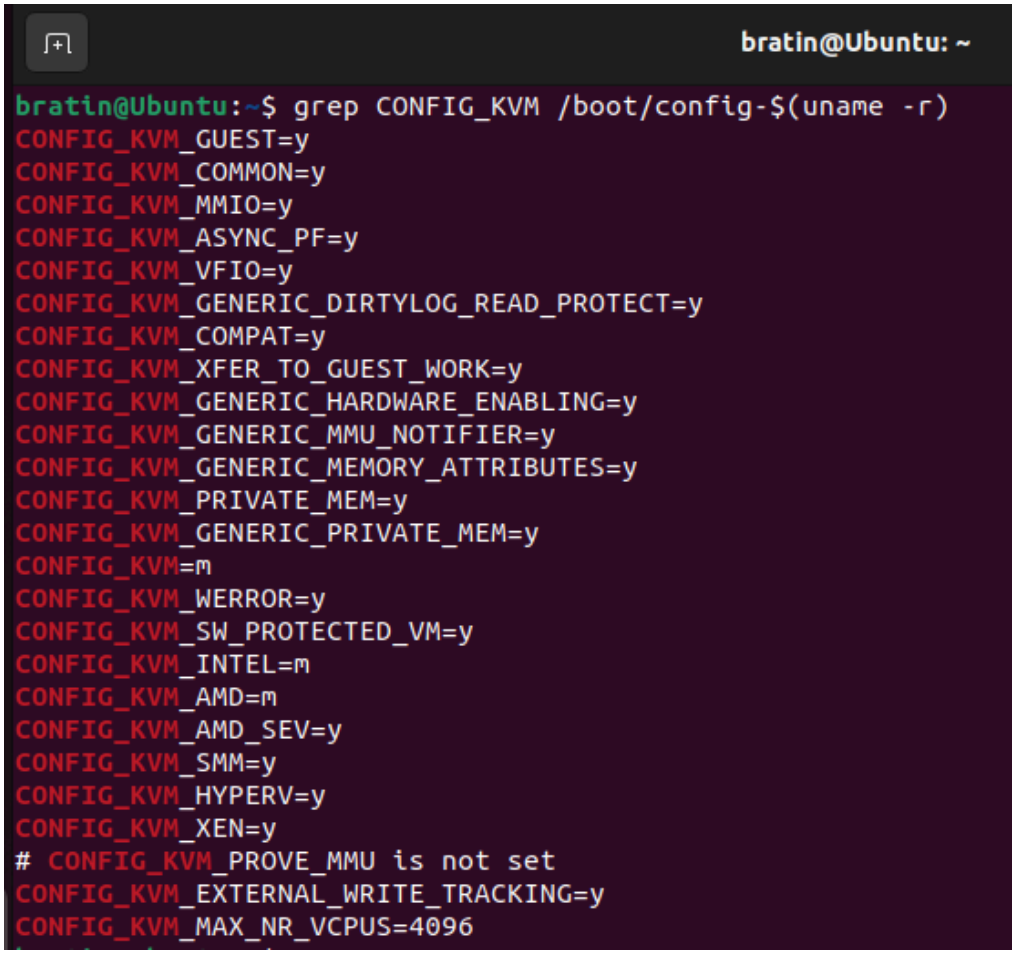
The options for "KVM for Intel (and compatible) processors support" and "KVM for AMD processors support" are specific to the two major CPU architectures. Enabling these allows KVM to utilize hardware virtualization extensions provided by Intel's VT-x or AMD's AMD-V technologies, improving the performance and efficiency of virtual machines running on these processors.

Finally, the "Audit KVM MMU" option enables auditing for the KVM Memory Management Unit (MMU), which is responsible for handling memory-related tasks in virtual machines. This auditing feature helps track and monitor the behavior of the KVM MMU, which can be useful for debugging and ensuring the reliability of memory management in virtualized environments.

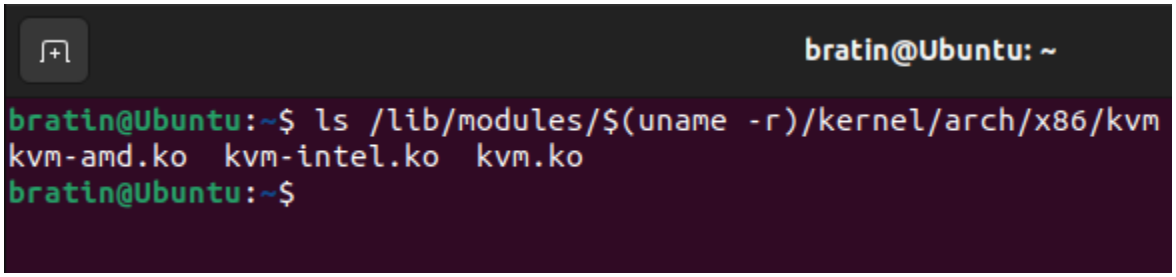
Before Installation of kernel

1. The config file had KVM set as kernel module earlier with some other options regarding it.
    a. **CONFIG_KVM_GUEST=y**: Enables support for running the Linux kernel as a guest inside a KVM virtual machine.
    b. **CONFIG_KVM_COMMON=y**: Ensures common KVM infrastructure is compiled into the kernel, shared between different architectures.

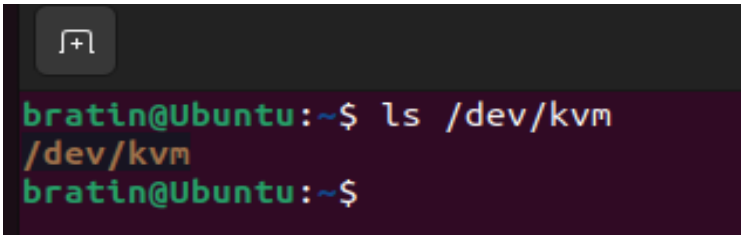
c. **CONFIG_KVM_MMIO=y**: Enables support for handling Memory-Mapped I/O (MMIO) in KVM guests.
d. **CONFIG_KVM_ASYNC_PF=y**: Enables support for asynchronous page faults, allowing the guest to handle page faults more efficiently.
e. **CONFIG_KVM_VFIO=y:** Enables support for VFIO (Virtual Function I/O), allowing direct device assignment to KVM guests.
f. **CONFIG_KVM_GENERIC_DIRTYLOG_READ_PROTECT=y**: Adds support for tracking dirty pages in KVM guests.
g. **CONFIG_KVM_COMPAT=y**: Enables compatibility code for older or legacy KVM guests.
h. **CONFIG_KVM_XFER_TO_GUEST_WORK=y**: Supports transferring work to the KVM guest during context switches.
i. **CONFIG_KVM_GENERIC_HARDWARE_ENABLING=y**: Enables generic hardware features required for KVM operation across different architectures.
j. **CONFIG_KVM_GENERIC_MMU_NOTIFIER=y:**: Adds support for notifying the KVM guest about changes in the host's Memory Management Unit (MMU) state.
k. **CONFIG_KVM_GENERIC_MEMORY_ATTRIBUTES=y**: Enables support for generic memory attributes in KVM.
l. **CONFIG_KVM_PRIVATE_MEM=y**: Allows KVM guests to manage memory that is private to the guest and not shared with the host.
m. **CONFIG_KVM_GENERIC_PRIVATE_MEM=y**: Supports generic handling of private memory in KVM guests.
n. **CONFIG_KVM=m**: Compiles the KVM core as a loadable module, allowing it to be loaded and unloaded dynamically.
o. **CONFIG_KVM_WERROR=y**: Treats all warnings as errors during KVM compilation.
p. **CONFIG_KVM_SW_PROTECTED_VM=y**: Enables support for software-protected virtual machines, adding a layer of security to the guest.
q. **CONFIG_KVM_INTEL=m**: Compiles Intel-specific KVM support as a loadable module, enabling Intel hardware virtualization extensions (VT-x).
r. **CONFIG_KVM_AMD=m**: Compiles AMD-specific KVM support as a loadable module, enabling AMD hardware virtualization extensions (AMD-V).
s. **CONFIG_KVM_AMD_SEV=m**: Enables AMD Secure Encrypted Virtualization (SEV), providing memory encryption for guest VMs on AMD hardware.
t. **CONFIG_KVM_SMM=m**: Enables support for System Management Mode (SMM) in KVM guests.
u. **CONFIG_KVM_HYPERV=m**: Enables support for running Hyper-V guests on a KVM hypervisor.
v. **CONFIG_KVM_XEN=m**: Enables support for running Xen guests under a KVM hypervisor.
w. **CONFIG_KVM_PROVE_MMU is not set**: This option, when enabled, would add extra checks in the KVM MMU code for debugging purposes.
x. **CONFIG_KVM_EXTERNAL_WRITE_TRACKING=y**: Supports tracking external writes to guest memory.
y. **CONFIG_KVM_MAX_NR_VCPUS=4096**: Sets the maximum number of virtual CPUs (vCPUs) that can be assigned to a KVM guest, here set to 4096.



2. The ls /lib/modules/$(uname -r)/kernel/arch/x86/kvm command shows three .ko files kvm-amd.ko, kvm-intel.ko and kvm.ko which means these were compiled as KVM modules and not as inbuilt feature.

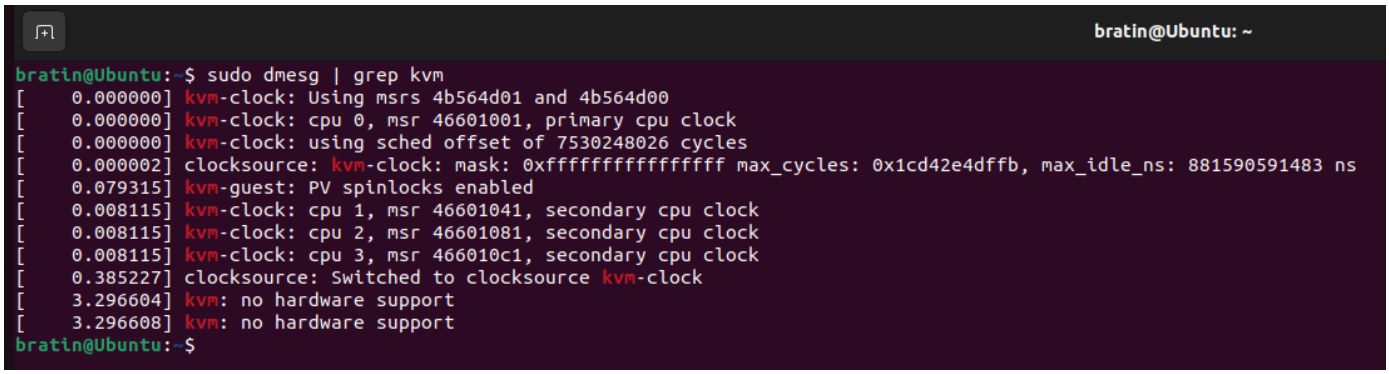


3. After having the nested virtualization on, the ls /dev/kvm shows the existence of the file signifying that KVM is enabled.



After Installation of kernel:

The initial rebooting in the Virtual Machine with the new kernel was unable to enable KVM.
Upon inspecting the following error was found:



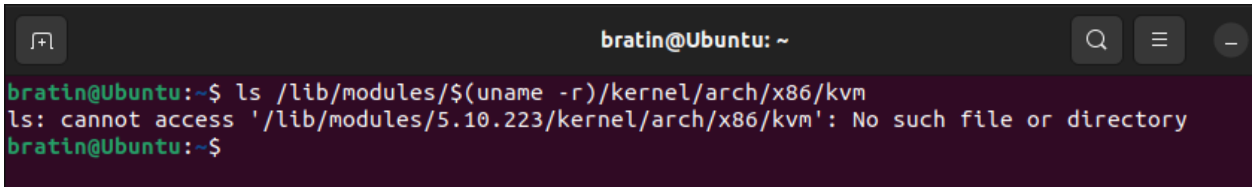
Although the Host Windows machine on which the Oracle VM was installed had virtualization support on, nested hardware virtualization for the VM was disabled.

This was fixed by running the command on Powershell VBoxManage modifyvm Ubuntu –nested-hw-virt on

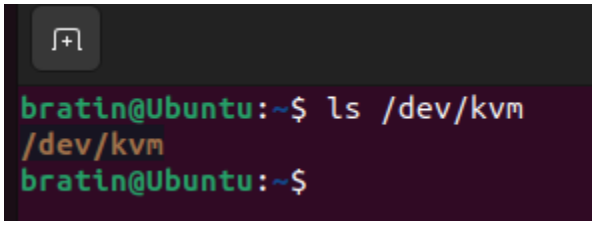
1. The config file now has KVM as an inbuilt feature
    a. **CONFIG_KVM_GUEST=y**: Enables support for KVM guest functionality.
    b. **CONFIG_KVM_MMIO=y**: Enables support for Memory-Mapped I/O (MMIO) in KVM. MMIO is used for devices that perform read/write operations directly to memory addresses.
    c. **CONFIG_KVM_ASYNC_PF=y**: Enables support for asynchronous page faults in KVM. This allows the guest to handle page faults asynchronously rather than synchronously.
    d. **CONFIG_KVM_VFIO=y**: Enables support for Virtual Function I/O (VFIO) with KVM. VFIO provides secure user-space access to PCI devices.
    e. **CONFIG_KVM_GENERIC_DIRTYLOG_READ_PROTECT=y**: Enables generic dirty log read protection in KVM. This helps manage and protect memory regions that have been modified (dirty pages) during VM execution.
    f. **CONFIG_KVM_COMPAT=y**: Enables compatibility features for KVM to ensure that older VMs and guests can run on newer KVM versions.
    g. **CONFIG_KVM_XFER_TO_GUEST_WORK=y**: Enables support for transferring work to the guest VM, allowing the guest to handle some tasks or operations that were previously managed by the host.
    h. **CONFIG_KVM=y**: Enables KVM support in the kernel. This is the core configuration option for KVM, allowing the kernel to act as a hypervisor.
    i. **CONFIG_KVM_WERROR=y**: Enables treating warnings as errors during the KVM build process.
    j. **CONFIG_KVM_INTEL=y**: Enables support for Intel hardware virtualization features in KVM. This is needed for running VMs on Intel processors.
    k. **CONFIG_KVM_AMD=y**: Enables support for AMD hardware virtualization features in KVM. This is needed for running VMs on AMD processors.

l. **CONFIG_KVM_MMU_AUDIT=y**: Enables MMU (Memory Management Unit) auditing in KVM. This allows auditing of memory management operations for debugging and monitoring purposes.

2. The ls /lib/modules/$(uname -r)/kernel/arch/x86/kvm command shows an error saying no such directory was found which means these were not compiled as KVM modules.



3. The ls /dev/kvm command shows the existence of the file signifying that KVM is enabled. Same as before.



Justification for some changes done:

1. System trusted keys and revocation keys are completely set to none

CONFIG_SYSTEM_TRUSTED_KEYS:

- This option specifies the file that contains keys to be added to the system keyring at boot. These keys are used to verify the signatures of kernel modules or firmware to ensure they are from trusted sources.
- Setting this option to **none** means that no keys are added to the system keyring. As a result, the kernel will not automatically trust any external modules or firmware unless explicitly configured to do so later.
- We used this to ensure that the system faces no security checks as we are customizing our kernel

CONFIG_SYSTEM_REVOCATION_KEYS:

- This option specifies the file that contains keys that have been revoked and should no longer be trusted by the kernel.
- Setting this to **none** means no revocation keys are provided, so the kernel will not revoke trust in any keys by default.
- As mentioned earlier too we used this to ensure that the system faces no security checks as we are customizing our kernel

2. We also changed grub timeout so as to ensure that we get to choose our own kernel while start or booting.