# **ANALYSIS AND CUSTOMIZATION OF HYPERVISOR**

VIRTUALIZATION (CSE4011)

PROJECT COMPONENT

FINAL REPORT

Submitted by

**Suruchi Parashar (17BCE2154)**

**Simran Koul (17BCE2210)**

Submitted To

**Dr. Siva Shanmugam G**

# Associate Professor

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**



Fall Semester 2019-20

November 2019

# ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our Professor, Dr. Siva Shanmugam G for his valuable guidance, consistent encouragement, timely help and providing us with an excellent atmosphere for doing our project. He has guided us throughout the project and we are highly obliged to receive her precious suggestions and advice.

We would like to extend our gratitude to everyone who has helped us with the project in the slightest way possible.

We also thank Vellore Institute of Technology for giving us the required resources and the opportunity to make this project. Many people, especially our classmates have made valuable suggestions on our project which gave us an inspiration to improve the quality of the same.

**Suruchi Parashar**

**Simran Koul**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.no.** | **Topic** | **Page no.** |
| 1 | Abstract | 4 |
| 2 | Introduction | 5 |
| 3 | Installation and tools used | 6 |
| 4 | Software Requirements | 15 |
| 5 | Methodology | 19 |
| 6 | System Implementation | 23 |
| 7 | Result | 26 |
| 8 | Conclusion | 27 |
| 9 | References | 29 |

1. **ABSTRACT**

The main aim of this project is to study the small build hypervisors and test them on different working environments. The focus will be on the testing of different hypervisors on different operating systems. Some hypervisors only work on windows environment and some are kernel based virtual machines. So the project focuses on the use of these hypervisors on different distributions and subsequently notes their reactions. K-Hypervisor is an Open Source light-weighted Nested-Virtual Machine Monitor in Windows x64 platform, which has been derived from other virtual platforms like DdiMon, which is a hypervisor performing inline hooking that is invisible to a guest.

# INTRODUCTION

The concept of virtualization is always deep and confusing. We aim to build an analysis which makes Hypervisors work easily logic wise. After the completion of this project we will be able to differentiate and distinguish between different working environments and we will be able to make changes in the working of these hypervisor orthodox kernels.

The idea of customizing a disc image after completing the course of operating systems was successfully established. The Hypervisor used was Oracle Virtual Box. The new idea and motivation now is to develop a customizing environment for Hypervisors as well. This will be a research based journey and the end results will be occupied on the basis of testing and different distribution analysis.

1. **INSTALLATION AND TOOLS USED**

**Hypervisor 1:**

* 1. **VMware ESX And ESXi**

VMware ESX: ESX (Elastic Sky X) is the VMware’s enterprise server virtualization platform. In ESX, VM kernel is the virtualization kernel which is managed by a console operating system which is also called as Service console which is linux based and its main purpose is it to provide a Management interface for the host and lot of management agents and other third party software agents are installed on the service console to provide the functionalists like hardware management and monitoring of ESX hypervisor.

VMware ESXi: ESXi (Elastic sky X Integrated) is also the VMware’s enterprise server virtualization platform. In ESXi, Service console is removed. All the VMware related agents and third party agents such as management and monitoring agents can also run directly on the VM kernel. ESXi is ultra-thin architecture which is highly reliable and its small code-base allows it to be more secure with less codes to patch. ESXi uses Direct Console User Interface (DCUI) instead of a service console to perform management of ESXi server. ESXi installation will happen very quickly as compared to ESX installation.

* 1. **Microsoft Hyper V**

Hyper-V is Microsoft's hardware virtualization product. It lets you create and run a software version of a computer, called a virtual machine. Each virtual machine acts like a complete computer, running an operating system and programs. When you need computing resources, virtual machines give you more flexibility, help save time and money, and are a more efficient way to use hardware than just running one operating system on physical hardware.

Hyper-V runs each virtual machine in its own isolated space, which means you can run more than one virtual machine on the same hardware at the same time. You might want to do this to avoid problems such as a crash affecting the other workloads, or to give different people, groups or services access to different systems.

* 1. **Citrix XenServer**

Citirix XenServer is a hypervisor platform that enables the creation and management of virtualized server infrastructure. It is developed by Citrix Systems and is built over the Xen virtual machine hypervisor. XenServer provides server virtualization and monitoring services. It is available in a 64-bit hypervisor platform and can be executed on the entire x86 series of processors.

Cirtix XenServer is among the virtualization solutions provided by Citrix Systems, which consolidates a physical server's computing power into multiple virtual machines, all emulating as a standard server. Citrix XenServer is built to provide the operational requirements of a standard server and supports most server operating systems, such as Linux and Windows Server, on guest server machines.

Through its virtual machine monitoring component, Citrix XenServer manages the allocation and distribution of physical server computing resources among virtual machines and administers their performance and use.

1. **Oracle VM Server**

Oracle VM Server for x86 is the server virtualization offering from Oracle Corporation. Oracle VM Server for x86 incorporates the free and open-source Xen hypervisor technology, supports Windows, Linux, and Solaris guests and includes an integrated Web based management console. Oracle VM Server for x86 features fully tested and certified Oracle Applications stack in an enterprise virtualization environment.

**Hypervisor 2:**

1. **VMware Server**

VMware Server (formerly VMware GSX Server) is a discontinued free-of-charge virtualization-software server suite developed and supplied by VMware, Inc.

VMware Server has fewer features than VMware ESX, software available for purchase, but can create, edit, and play virtual machines. It uses a client–server model, allowing remote access to virtual machines, at the cost of some graphical performance (and 3D support). It can run virtual machines created by other VMware products and by Microsoft Virtual PC.

VMware Server can preserve and revert to a single snapshot copy of each separate virtual machine within the VMware Server environment. The software does not have a specific interface for cloning virtual machines, unlike VMware Workstation.

VMware Server has largely been replaced by the "Shared Virtual Machines" feature, introduced in VMware Workstation 8.0 and onwards.

1. **Microsoft Virtual PC**

On July 12, 2006, Microsoft released Virtual PC 2004 SP1 for Windows free of charge, but the Mac version was not made free. The equivalent version for Mac, version 7, was the final version of Virtual PC for Mac. It ran on Mac OS X 10.2.8 or later for PowerPC and was a proprietary commercial software product.

Virtual PC 2007 was released only for the Windows platform, with public beta testing beginning October 11, 2006, and production release on February 19, 2007. It added support for hardware virtualization, viewing virtual machines on multiple monitors and support for Windows Vista as both host and guest. (The Windows Aero interface is disabled on Windows Vista guests due to limitations of the emulated video hardware; however, Aero effects can be rendered by connecting to the guest via Remote Desktop Services from an Aero-enabled Windows Vista host, provided that the guest is running Windows Vista Business or a higher edition.)

On May 15, 2008, Microsoft released Virtual PC 2007 Service Pack 1, which added support for both Windows XP SP3 and Windows Vista SP1 as guest and host OSes, as well as Windows Server 2008 Standard as a guest OS. A hotfix rollup for Virtual PC 2007 SP1, released February 20, 2009, solved networking issues and enhanced the maximum screen resolution to 2048×1920 (32-bit), enabling 16:9 resolutions such as 1920×1080. A security update was released on July 14, 2009 to address an elevation of privilege vulnerability in guest operating systems.

Microsoft Virtual PC (2004 and 2007) does not work at all on Windows 10 64-bit, and even on 32-bit platforms lack internet connectivity due to the lack of the VPC driver. This also impacts Windows Mobile emulators.

1. **Oracle VM virtual box**

Virtual Box is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is Virtual Box an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2.

Presently, Virtual Box runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and Open Solaris, OS/2, and Open BSD.

Virtual Box is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. Virtual Box is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.

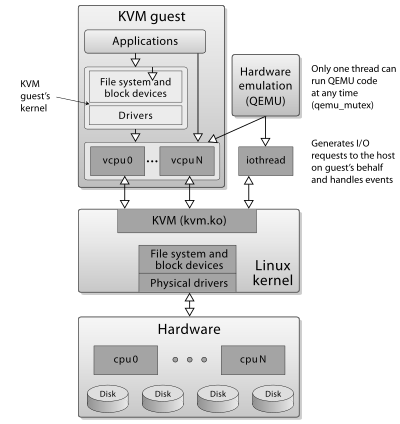
1. **Kernel based VM**

Kernel-based Virtual Machine (KVM) is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor. It was merged into the Linux kernel mainline in kernel version 2.6.20, which was released on February 5, 2007. KVM requires a processor with hardware virtualization extensions, such as Intel VT or AMD-V. KVM has also been ported to other operating systems such as FreeBSD and illumos in the form of loadable kernel modules.

KVM was originally designed for x86 processors and has since been ported to S/390, PowerPC, IA-64, and ARM.

KVM provides hardware-assisted virtualization for a wide variety of guest operating systems including Linux, BSD, Solaris, Windows, Haiku, ReactOS, Plan 9, AROS Research Operating System and macOS. In addition, Android 2.2, GNU/Hurd (Debian K16), Minix 3.1.2a, Solaris 10 U3 and Darwin 8.0.1, together with other operating systems and some newer versions of these listed, are known to work with certain limitations.

Additionally, KVM provides para-virtualization support for Linux, OpenBSD, FreeBSD, NetBSD, Plan 9 and Windows guests using the VirtIO API. This includes a para-virtual Ethernet card, disk I/O controller, balloon device, and a VGA graphics interface using SPICE or VMware drivers.



**Small build hypervisors:**

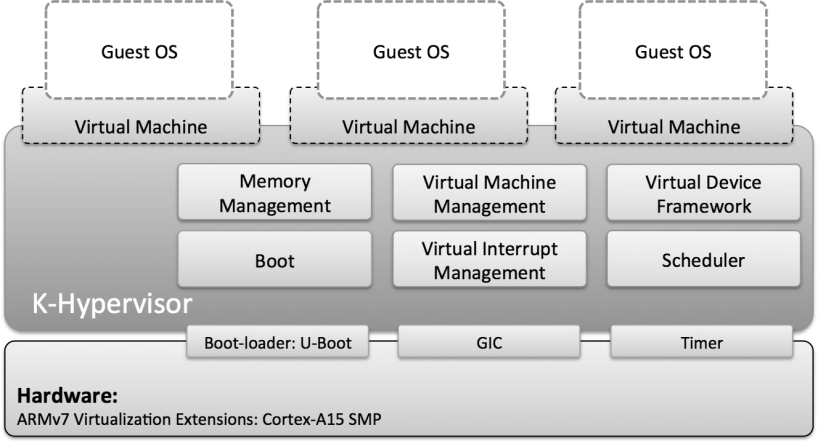
**K- hypervisor:**

K-hypervisor is an open-source type-1 hypervisor, which aims at providing a monolithic, light-weight, portable and flexible virtualization solution. It provides a high performance for ARMv7 Virtualization Extensions.

K-hypervisor primarily supports Full virtualization hence, supports a wide range of unmodified Guest operating systems such as Linux and RTOS, and the Bare Metal Application.

It has most features expected from a modern hypervisor, such as: IO device emulation framework, pass through hardware access and many more.

K-hypervisor is Hardware Accelerated Hypervisor, so it supports ARM GICv2, ARM Generic Timer and 2-Stage Address Translation.



The hypervisor consists of a few sub-systems and device drivers. On a target hardware to run a virtualized software system it needs other software components such as boot-loader, software images for guest software such as Linux and RTOS, and the hypervisor image. A boot-loader is responsible for loading the software images of the hypervisor and guest software to designated address in the memory and triggering the very first execution of the hypervisor image in the memory. It is up to the boot-loader use on the target hardware whether to load the software images from the Flash Memory, an SD Card or any storage media it prefers.

**Hyper Platform:**

Hyper Platform is an Intel VT-x based hypervisor (a.k.a. virtual machine monitor) aiming to provide a thin platform for research on Windows. Hyper Platform is capable of monitoring a wide range of events, including but not limited to, access to virtual/physical memory and system registers, occurrences of interrupts and execution of certain instructions.

Researchers are free to selectively enable and/or disable any of those event monitoring and implement their own logic on the top of Hyper Platform. Some potential applications are:

• Analyzing kernel mode rootkit

• Implementing virtual-machine-based intrusion prevention system (VIPS)

• Reverse-engineering the Windows kernel

**MemoryMon:**

MemoryMon is able to detect execution of kernel memory where is not backed by any image file (kernel modules) using extended page table (EPT). It can help researchers analyze kernel mode code installs and runs code outside of an image file.

**EopMon:**

EopMon is a hypervisor-based elevation of privilege (EoP) detector. It can spots a process with a stolen system token and terminate it by utilizing hypervisor’s ability to monitor process context-switching. While EopMon is tested against multiple EoP exploits carried out by in the wild malware, it is rather meant to be an educational tool to demonstrate a potential use case of a hypervisor for security research and not aimed for comprehensive exploit prevention. EopMon is meant to be an educational tool and not robust, production quality software which is able to handle various edge cases. For this reason, researchers are encouraged to use this project only as a reference to examine and develop ideas of using a hypervisor.

**DdiMon :**

DdiMon is a hypervisor performing inline hooking that is invisible to a guest (ie, any code other than DdiMon) by using extended page table (EPT).

DdiMon is meant to be an educational tool for understanding how to use EPT from a programming perspective for research. To demonstrate it, DdiMon installs the invisible inline hooks on the following device driver interfaces (DDIs) to monitor activities of the Windows built-in kernel patch protection, a.k.a. PatchGuard, and hide certain processes without being detected by PatchGuard.

• ExQueueWorkItem

• ExAllocatePoolWithTag

• ExFreePool

• ExFreePoolWithTag

• NtQuerySystemInformation

Those stealth shadow hooks are hidden from guest's read and write memory operations and exposed only on execution of the memory. Therefore, they are neither visible nor over writable from a guest, while they function as ordinal hooks. It is accomplished by making use of EPT enforcing a guest to see different memory contents from what it would see if EPT is not in use. This technique is often called memory shadowing.

**GuardMon:**

GuardMon is a hypervisor based tool for monitoring system register accesses. GuardMon is capable of logging read and write activities on CR0, CR4, debug registers, GDT, IDT and MSRs from kernel memory not backed by any images.

This tool is particularly useful for analyzing the Windows built-in kernel patch protection, a.k.a. PatchGuard as it runs on non-image regions most of time.

**Virtdbg :**

The purpose of Virtdbg is to implement a kernel debugger using the hardware virtualization technology provided by Intel (VT-x). This project is born because when it comes to Windows 7 x64, the available kernel debuggers tend to be very limited.

We have WinDbg which is very good but need cooperation of the OS. We can't use it in order to debugging protected parts of the operating system like PatchGuard for example. Microsoft doesn't allow it.

The other kernel debuggers are local debuggers like Soft ICE, Syser or HyperDbg (which uses the same approach). I made the choice of not using a local debugger because I find that they are difficult to extend and to script.

VirtDbg is in very alpha state. So I decline all responsibilities if your computer blue-screened, however I will be happy if you give it a try. It is under heavy development so expect a lot of changes quickly.

**HyperDbg :**

HyperDbg is a kernel debugger that leverages hardware-assisted virtualization. More precisely, HyperDbg is based on a minimalistic hypervisor that is installed while the system runs. Compared to traditional kernel debuggers (e.g., WinDbg, SoftIce, Rasta R0 Debugger) HyperDbg is completely transparent to the kernel and can be used to debug kernel code without the need of serial (or USB) cables. For example, HyperDbg allows to single step the execution of the kernel, even when the kernel is executing exception and

interrupt handlers. Compared to traditional virtual machine based debuggers (e.g., the VMware builtin debugger), HyperDbg does not require the kernel to be run as a guest of a virtual machine, although it is as powerful.

Once loaded, the debugger will sits in background and will pop up the GUI when the F12 hot-key is pressed or when a debug event occurs.

The current version of HyperDbg is a prototype and will evolve drastically in the future. Currently the debugger allows to set breakpoints, to single step the execution of the kernel, to inspect the memory of the operating system and of a particular process. However, the debugger currently does not distinguish between threads, has limited support for kernel's symbols, and has does not clone shared pages before setting a breakpoint. Future version of the debugger will be based on an enhanced version of the hypervisor that guarantees complete isolation and transparency.

HyperDbg currently only supports:

• systems with PS/2 keyboards

• systems with Intel VT-x

• systems running 32-bit OSes and applications (no PAE)

• Windows XP (SP2).

HyperDbg renders the GUI by writing directly to the memory of the video card. Some video cards are known to give problems. The debugger does not work correctly when the accelerated drivers for the following cards are loaded:

• Intel 82915g

• nvidia GeForce 9800GT

• nvidia GeForce GT 130

If you have any of the aforementioned cards (and you are using the accelerated driver) or if the interface is not correctly rendered on the screen, you have to disable the driver in order to be able to use HyperDbg. The driver used by default by Windows XP does not give any problem.

**4.SOFTWARE REQUIREMENTS**

**HVPP:**

Hvpp is a lightweight Intel x64/VT-x hypervisor. Its essential codes are written in C++ concentrated principally on virtualization of previously running operating systems. A few open-source look into hypervisors went for straightforwardness as of now exist, as I would like to think this field is still fairly unexplored and needs progressively open-source ventures. This can particularly enable the individuals who to have recently begun investigating virtualization advancements and are searching for little/reference ventures. In the event that you're one of them, my wagers are that you're truly frustrated at this moment, since all you have are scarcely dozen of ventures and colossal heap of Intel Manual pages.

C++ has been picked as a language for this task in light of two reasons:

* The virtualization design can be better communicated in OOP ideas (with so much articles as VCPU, EPT).
* We didn't discover different undertakings which would utilize present day C++17 highlights, aside from bareflank. Despite the fact that bareflank is compilable under Windows, Linux and UEFI, on Windows, it utilizes cygwin to cross-gather the hypervisor. Since the hypervisor is an independent Mythical person twofold, the Windows portion is feeling the loss of the investigate images for the hypervisor, which averts simple troubleshooting with WinDbg.

Hvpp is compiled as a static library, which is linked with the hvppdrv project.

* Bootstrap of the driver (hvpp, driver.cpp)
* Allocation of memory beforehand and initialization of the hvpp memory manager
* initialization the logger
* Bootstrap of the hypervisor (hvppdrv, main.cpp)
* Create- vmexit\_handler instance
* Start the hypervisor with provided VM-exit handler: hypervisor::start(vmexit\_handler& handler)
* Initializing a virtual cpu (VCPU) for every logical processor
* Assigning the provided vmexit\_handler instance to every VCPU
* launch all VCPUs through IPI (inter-processor interrupt): vcpu\_t::start()
* setup VMXON region and then further VMCS: vcpu\_t::vmx\_enter()
* vmexit\_handler::setup() is called, which allows the initialization of the VM-exit handler and/or modify the VMCS before the launch.
* Hypervisor is now performing and also handles VM-exits through provided VM-exit handler
* Stop the hypervisor: hypervisor::stop()
* destroy each VCPU through IPI: vcpu\_t::stop()
* vmexit\_handler::teardown() is called and it further switches into VMX mode.
* in VMX mode, vcpu\_t::vmx\_leave() is called and it leaves VMX mode with VMXOFF instruction.

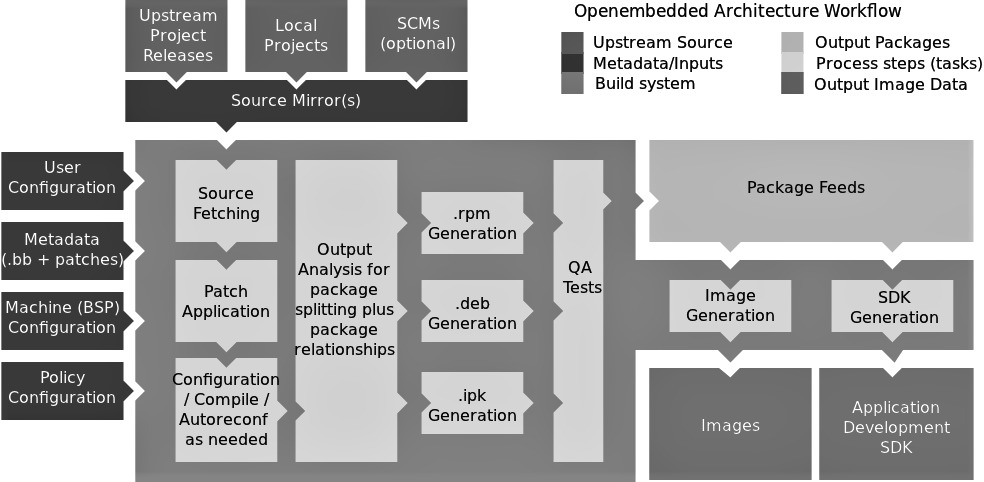
**Description of "stealth hooking" process:**

* finds ZwClose work in ntdll.dll
  + dismantles initial 16 bytes of this capacity and prints them
* printed guidelines ought to demonstrate that this capacity isn't snared at this point
  + calls this capacity (with NULL parameter, this capacity call will doubtlessly fizzle with some NTSTATUS blunder code, which it overlooks)
  + prints estimation of HookCallCount and it's normal worth.
* hooks ntdll!ZwClose function utilizing Temporary re-routes
* dismantles initial 16 bytes of this capacity and prints them
* printed guidelines should now demonstrate that the capacity IS snared (by jmp being first guidance)
* calls this capacity (with NULL parameter)
* rather than unique capacity, the snare capacity will be called - on each call, it increases variable HookCallCount
* prints estimation of HookCallCount and it\'s normal worth - it ought to be 1 now, as the hooked capacity has been required the first run through at this point
* calls hvpp by VMCALL guidance and RCX = 0xc1 (self-assertively picked), RDX = AddressOfReadPage and R8 = AddressOfExecutePage - this teaches hvpp to shroud the hook
* dismantles initial 16 bytes of this capacity and prints them
* printed directions should now demonstrate that the capacity function IS hooked up (by indicating unique substance of memory - no jmp)
* calls this capacity (with Invalid parameter)
* regardless of what the memory returned when we read it, the snare capacity will be called again and the HookCallCount will be increased once more
* prints estimation of HookCallCount and it\'s normal worth - it ought to be 2
* calls hvpp by VMCALL guidance and RCX = 0xc2 (self-assertively picked) - this teaches hvpp to unhide the snare
* dismantles initial 16 bytes of this capacity and prints them
* printed guidelines should now demonstrate that the capacity snare isn't Covered up and it should show jmp as a first guidance once more
* calls this capacity (with Invalid parameter)
* since the capacity is as yet snared, the snare capacity will be called and HookCallCount will be augmented once more
* prints estimation of HookCallCount and it\'s normal worth - it ought to be 3
* unhooks ntdll!ZwClose work
* dismantles initial 16 bytes of this capacity and prints them
* printed directions should now demonstrate that the capacity isn't snared - it should show a similar substance as when this capacity wasn\'t snared
* calls this capacity (with NULL parameter)
* unique capacity will be called, in this way the HookCallCount ought not be increased at this point
* prints estimation of HookCallCount and it\'s normal worth - it ought to be as yet 3

In short:

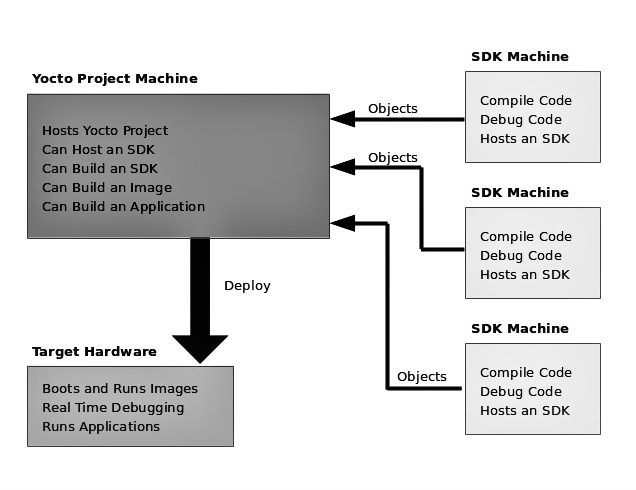
* hvpp is intended to virtualize previously running operating system - for example it's not equipped for running numerous visitors like VMWare or VirtualBox. It additionally needs support for any settled VMX operations.
* hvpp is intended to run distinctly on 64bit Intel processors, which bolster VT-x and EPT. This makes the code increasingly straightforward and less enlarged.
* hvpp is intended to run distinctly on Windows - future work may concentrate on Linux.
* hvpp as of now doesn't exit VMX mode on rest or sleep (S3 and S4 control states) - Intel Manual says we ought to do as such - this is known limitation.

**5.METHODOLOGY**



The Yocto Project is an open source collaboration project that helps developers create custom Linux-based systems that are designed for embedded products regardless of the product's hardware architecture. Yocto Project provides a flexible toolset and a development environment that allows embedded device developers across the world to collaborate through shared technologies, software stacks, configurations, and best practices used to create these tailored Linux images.

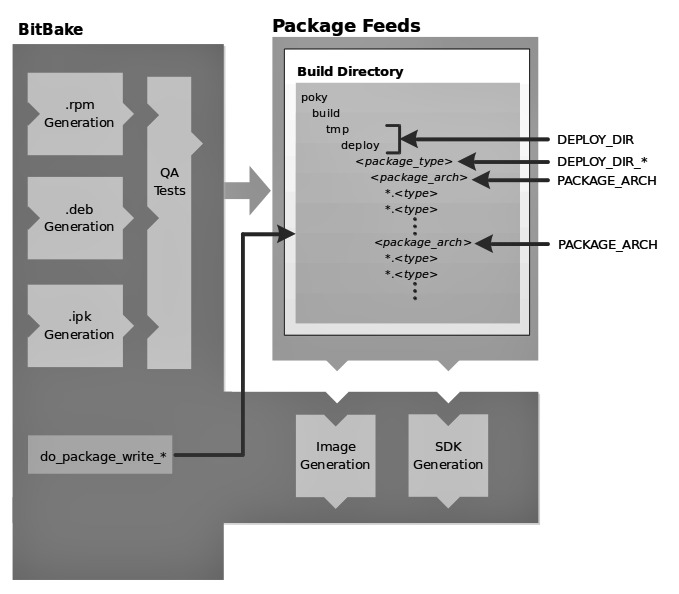
Thousands of developers worldwide have discovered that Yocto Project provides advantages in both systems and applications development, archival and management benefits, and customizations used for speed, footprint, and memory utilization. The project is a standard when it comes to delivering embedded software stacks. The project allows software customizations and build interchange for multiple hardware platforms as well as software stacks that can be maintained and scaled.



The Yocto Project's "Layer Model" is a development model for embedded and IoT Linux creation that distinguishes the Yocto Project from other simple build systems. The Layer Model simultaneously supports collaboration and customization. Layers are repositories that contain related sets of instructions that tell the Open Embedded build system what to do. You can collaborate, share, and reuse layers.

Layers can contain changes to previous instructions or settings at any time. This powerful override capability is what allows you to customize previously supplied collaborative or community layers to suit your product requirements.

You use different layers to logically separate information in your build. As an example, you could have BSP, GUI, distro configuration, middleware, or application layers. Putting your entire build into one layer limits and complicates future customization and reuse. Isolating information into layers, on the other hand, helps simplify future customizations and reuse. You might find it tempting to keep everything in one layer when working on a single project. However, the more modular your Metadata, the easier it is to cope with future changes.



The following list consists of components associated with the Open Embedded build system:

• BitBake: BitBake is a core component of the Yocto Project and is used by the Open Embedded build system to build images. While BitBake is key to the build system, BitBake is maintained separately from the Yocto Project.

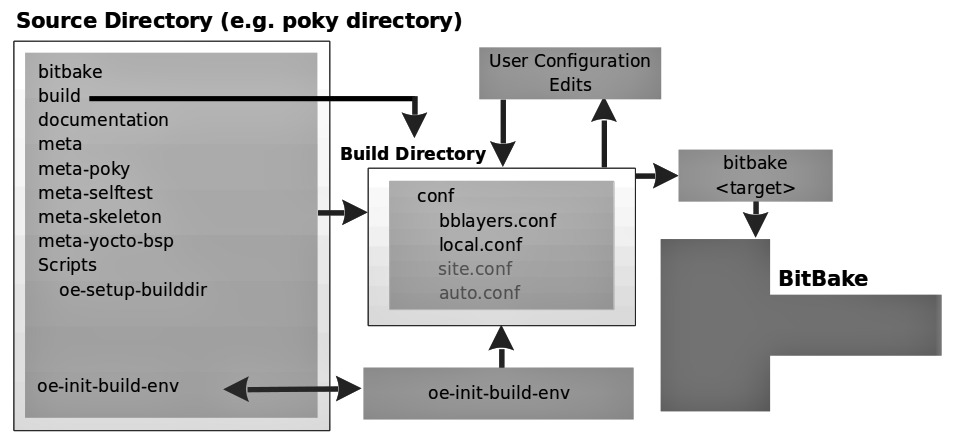
BitBake is a generic task execution engine that allows shell and Python tasks to be run efficiently and in parallel while working within complex inter-task dependency constraints. In short, BitBake is a build engine that works through recipes written in a specific format in order to perform sets of tasks.

You can learn more about BitBake in the BitBake User Manual.

• Open Embedded-Core: Open Embedded-Core (OE-Core) is a common layer of metadata (i.e. recipes, classes, and associated files) used by Open Embedded-derived systems, which includes the Yocto Project. The Yocto Project and the Open Embedded Project both maintain the Open Embedded-Core. You can find the OE-Core metadata in the Yocto Project Source Repositories.

Historically, the Yocto Project integrated the OE-Core metadata throughout the Yocto Project source repository reference system (Poky). After Yocto Project Version 1.0, the Yocto Project and Open Embedded agreed to work together and share a common core set of metadata (OE-Core), which contained much of the functionality previously found in Poky. This collaboration achieved a long-standing Open Embedded objective for having a more tightly controlled and quality-assured core. The results also fit well with the Yocto Project objective of achieving a smaller number of fully featured tools as compared to many different ones.

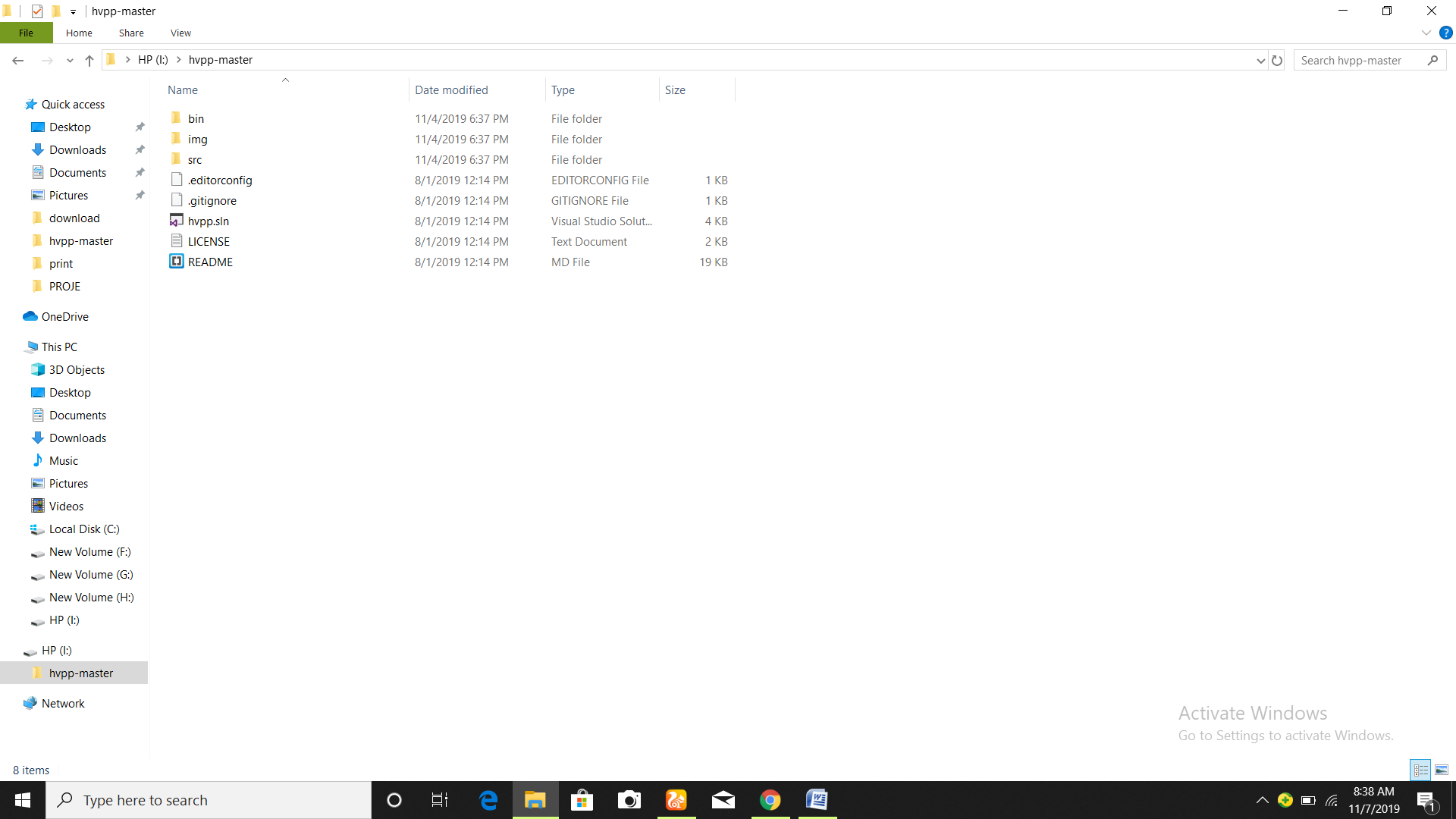
Sharing a core set of metadata results in Poky as an integration layer on top of OE-Core. You can see that in this figure. The Yocto Project combines various components such as BitBake, OE-Core, script “glue”, and documentation for its build system.

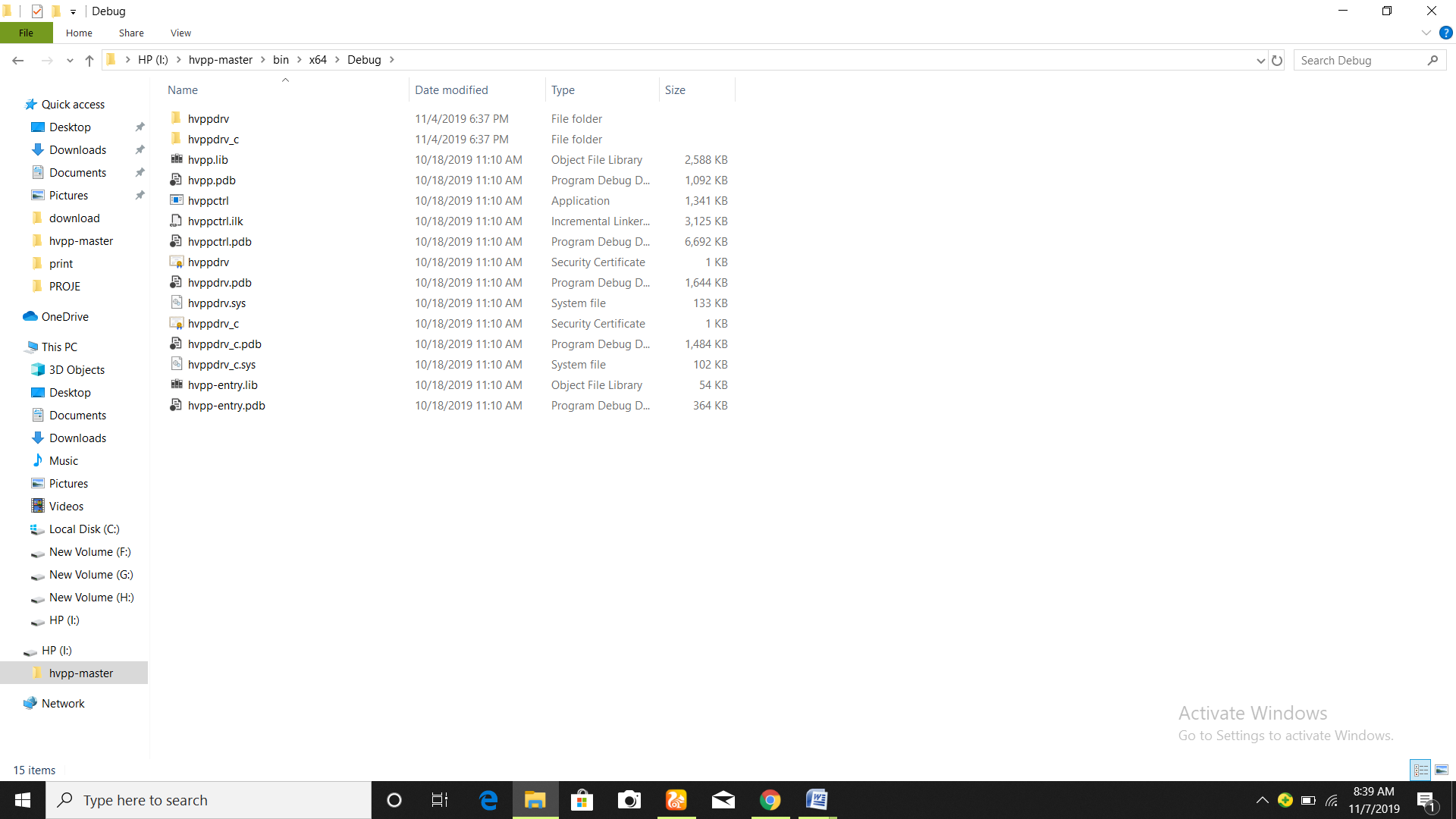


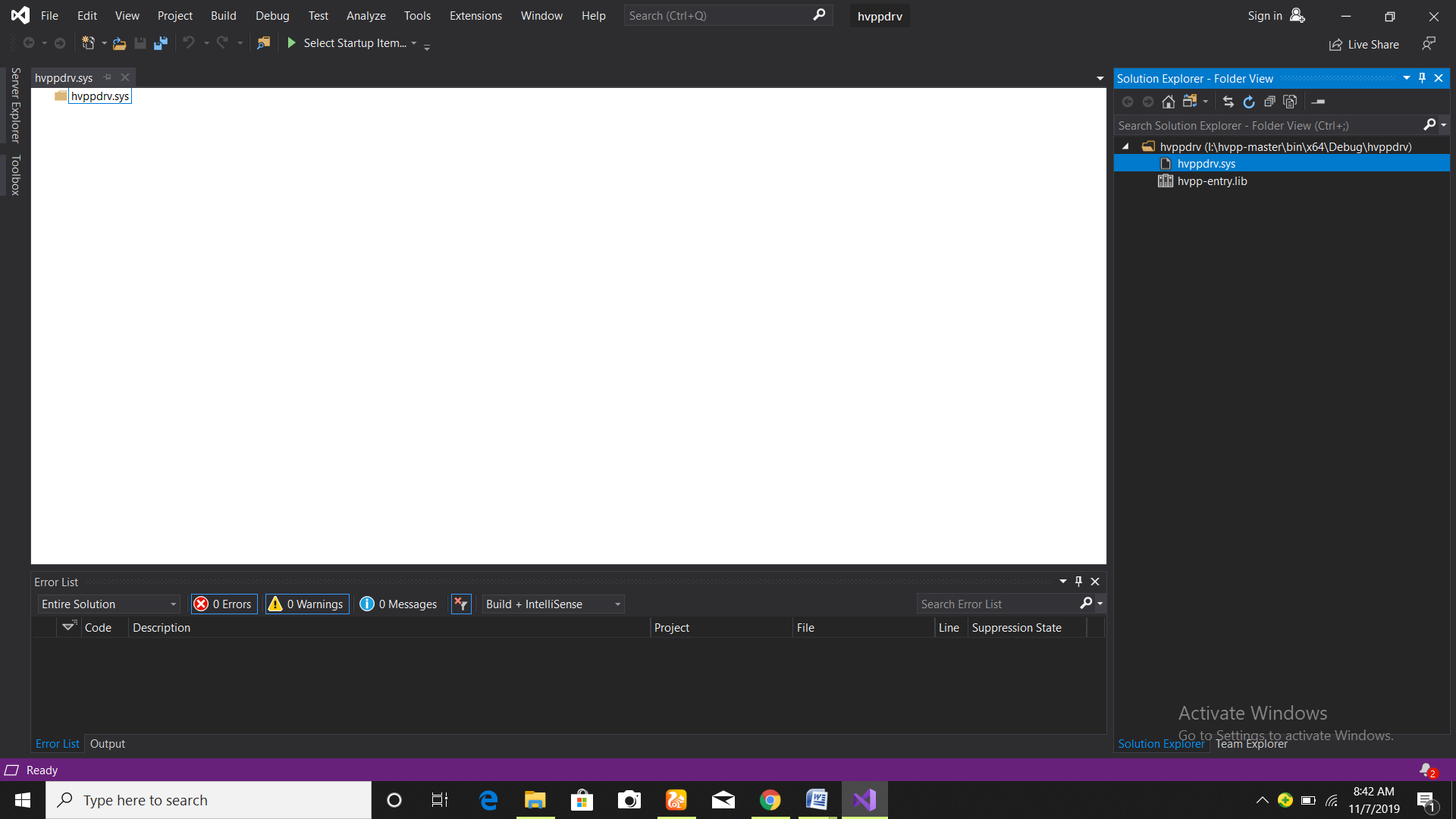
## Poky is the Yocto Project reference distribution. It contains the [Open-Embedded build system](http://www.yoctoproject.org/docs/2.5/ref-manual/ref-manual.html#build-system-term) (BitBake and OE-Core) as well as a set of metadata to get you started building your own distribution. To use the Yocto Project tools and components, you can download (clone) Poky and use it to bootstrap your own distribution.

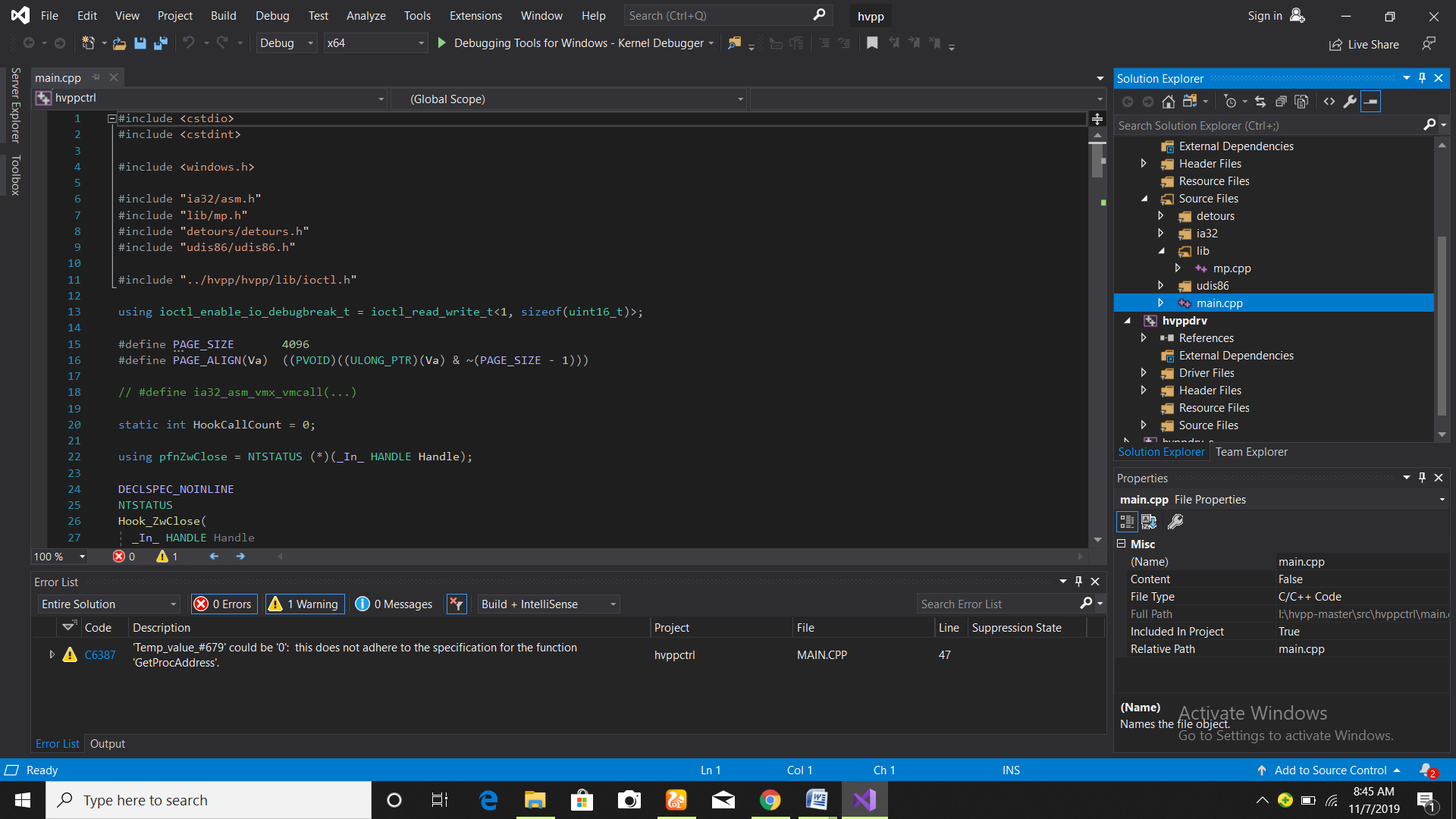
## Poky does not contain binary files. It is a working example of how to build your own custom Linux distribution from source.

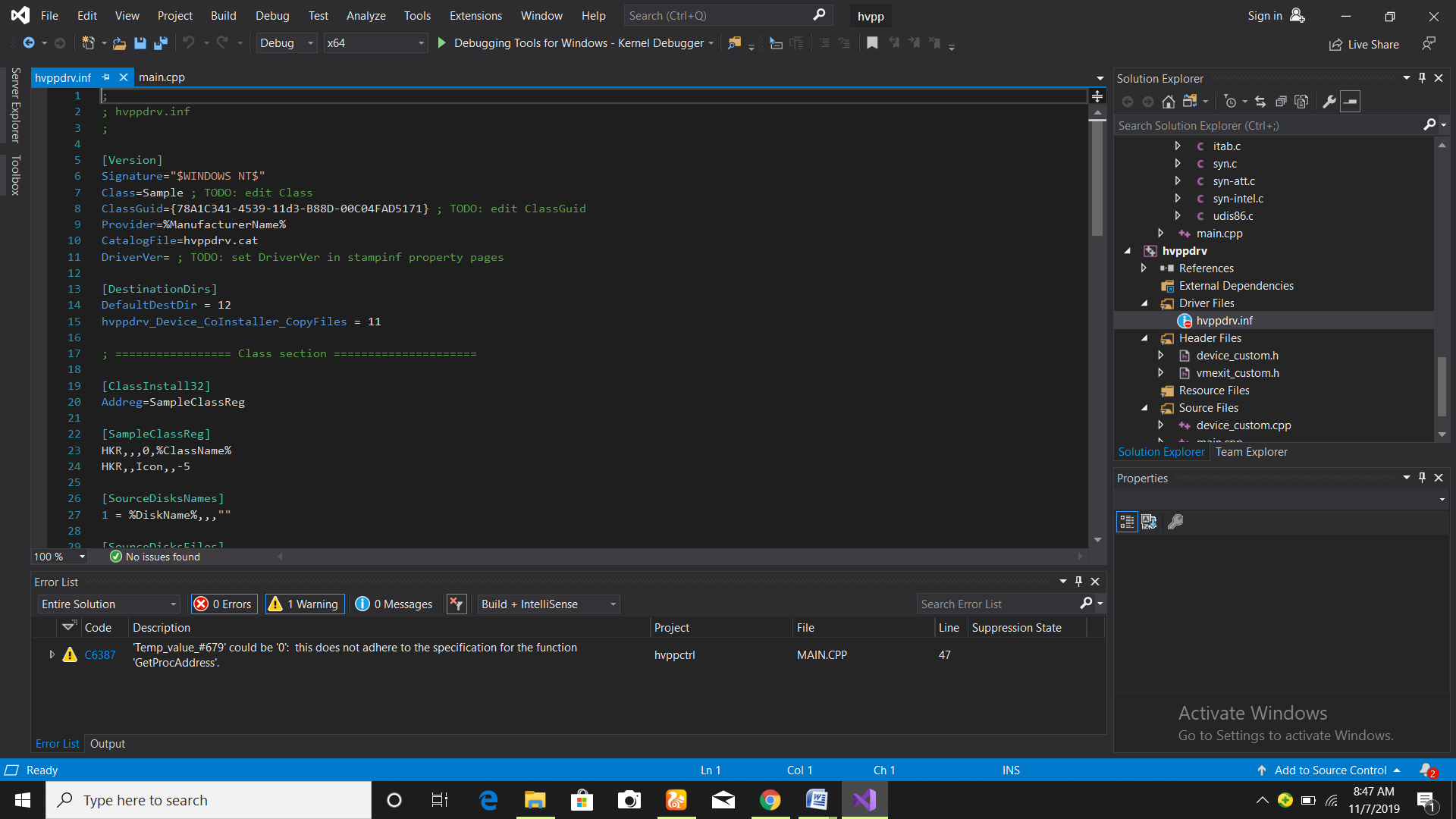
**6.SYSTEM IMPLEMENTATION**





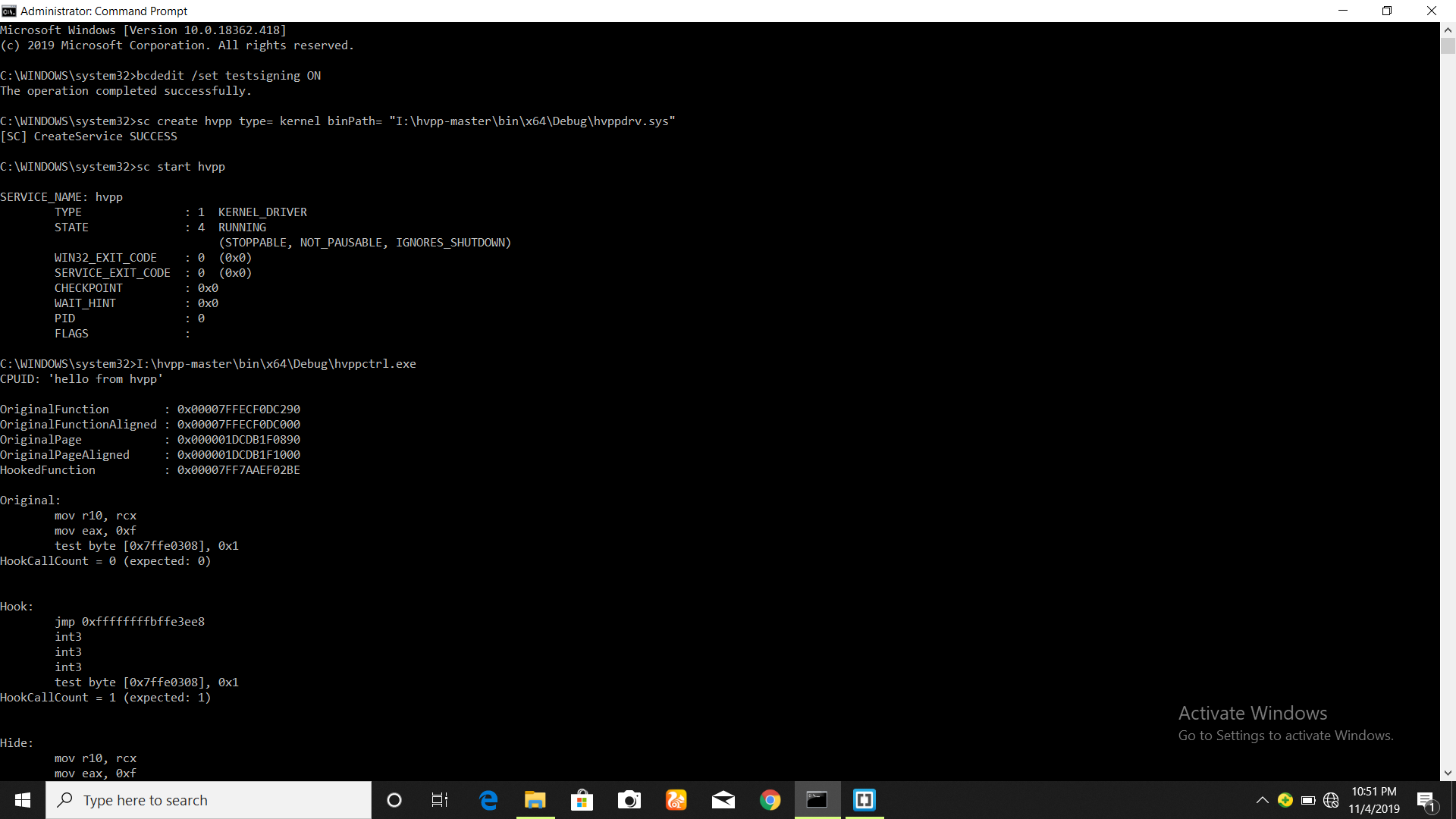






**7.RESULT**

The final output will be depicting different working of Hypervisors on different Operating Systems. Also, if an Operating system works totally on the basis of kernel, we will try and customize it to work on different environments and also, if it doesn’t work we will have the outcome of WHY and HOW.



**8.CONCLUSION**

The clear leader in VM execution is ESXi. The shockingly low scores of Hyper-V 2012 R2 might be brought about by higher framework prerequisites, and the low execution of the test seat framework. Tests likewise plainly demonstrate that in a greater part of cases Type 1 hypervisors have extraordinary focal points over Type 2 hypervisors, on account of direct access to the framework's assets. A huge distinction between these outcomes and the consequences of J. Hwang, S. Zeng, F. y Wu, and T. Wood from George Washington College and IBM is perceptible. For instance, XenServer has a lot higher scores, much of the time coming to ESXi results. Moreover, despite the fact that Hyper-V 2008 outcomes are like the past ones, 2012 shows an astonishing misfortune in performance. As apparent from the directed benchmarks and analysis, the fundamental contender for use in big business conditions are the arrangements made by Microsoft and VMWare. The vast majority of the rest of the frameworks don't have a greater part of highlights important in huge scale virtualization. It must be, nonetheless, noticed that XenServer has accomplished great scores, and is an entirely moderate arrangement, which settles on it a decent decision for littler scale tasks, eg. a little ofﬁce. Due to complex conﬁguration and its generally little list of capabilities, Prophet's item might be suggested for the most part for organizations that as of now use numerous arrangements made by the organization, eg. Prophet Database. The downsides have less effect at that point, on account of the coordination highlights OVM offers, just as great outcomes in benchmarks. Hyper-V and VMWare offer comparative, wide abilities. In earlier years, the permit cost for VMWare's product was a lot higher, anyway as of late, with the arrival of Windows Server 2012 R2, Microsoft has raised the cost of their solution, causing the value focuses to get comparative. Considering the all out unit cost for Windows Server 2012 R2 Server farm and Framework Center 2012 R2 Server farm, approx. 7000 USD, and the permit cost for vSphere, approx. 5000 USD, VMWare's item has all the earmarks of being the better decision in value/quality. How-ever, Microsoft additionally offers complimentary working framework licenses, which takes into account sparing 800 US Dollars for every virtual machine, which isn't a possibility for VMWare, accepting, as is normal in big business, the utilization of Microsoft arrangements. Hyper-V likewise takes into consideration simpler usage, on account of speedy combination in a previous Windows foundation. Nonetheless, ESXi has indicated better outcomes in benchmarks, besting Hyper-V in about each classification. VirtualBox has scored shockingly well in benchmarks in certain classes. In any case, this doesn't expel the enormous loss of execution saw in other tests. Ultimately, the decision must be made with thought for benchmark results, yet in addition highlights and cost. Each organization must choose an answer dependent on its own needs and budget. Due to the benchmark brings about certain cases indicating huge errors from the normal results, apparently a second round of tests on an all the more dominant test seat would be useful. This would permit to dispense with the factor of low equipment execution, which may improve the aftereffects of eg. Hyper-V 2012 R2, just as considering establishment of a more up to date form of ESXi.

**9.REFERENCES**

[1]. 2018, A Study On Virtualization And Hypervisor In Cloud Computing By Bohar Singh(Computer Science and Application, bohar2@gmail.com), Gursewak Singh(Computer Science and Application, [mgursewak@gmail.com](mailto:mgursewak@gmail.com))

[2]. 2018, Evaluation Of Type-1 Hypervisors On Desktop-Class Virtualization Hosts By Duarte Pousa(Polytechnic Institute of Bragança, 5300-253 Bragança, Portugal) and José Rufino(Polytechnic Institute of Bragança, 5300-253 Bragança, Portugal. Laboratory of Instrumentation and Experimental Particle Physics, University of Minho, 4710-057 Braga, Portugal)

[3]. 2017, Data Center Server Virtualization Solution Using Microsoft Hyper-V By Sujitha Dandu (St Cloud State University, [dandusujitha@gmail.com](mailto:dandusujitha@gmail.com))

[4]. 2017, Survey On Hypervisors By Naveed Alam, School Of Informatics and Computing, Indiana University Bloomington, [nalam@indiana.edu](mailto:nalam@indiana.edu)

[5]. 2017, Multi-Hypervisor Virtual Machines: Enabling an Ecosystem of Hypervisor-level Services By Kartik Gopalan, Rohit Kugve, Hardik Bagdi, and Yaohui Hu, Binghamton University; Daniel Williams and Nilton Bila, IBM T.J. Watson Research Center

<https://www.usenix.org/conference/atc17/technical-sessions/presentation/gopalan>

[6]. 2018, Hypervisor From Scratch – Part 1: Basic Concepts & Configure Testing Environment By Sina Karvandi

[7]. 2018, Security Recommendations for Server-based Hypervisor Platforms By Ramaswamy Chandramouli

[8]. 2015, Hypervisor For Virtualization In Private Cloud By Pavol ZÁVACKÝ, Andrej ELIÁŠ, Maximilián STRÉMY

Ing. Pavol Závacký, PhD.( Slovak University of Technology in Bratislava, Faculty of Materials Science and Technology in Trnava, Division of Communication and Information Systems, Paulínska 16, 917 24 Trnava, Slovakia), Ing. Andrej Eliáš, PhD., doc. Ing. Maximilián Stémy, PhD.( Slovak University of Technology in Bratislava, Faculty of Materials Science and Technology in Trnava, Advanced Technologies Research Institute, Bottova 25, 917 24 Trnava, Slovakia, e-mail: pavol.zavacky@stuba.sk, andrej.elias@stuba.sk, [maximilian.stremy@stuba.sk](mailto:maximilian.stremy@stuba.sk))

[8]. 2014, A Study On Virtualization Techniques And Challenges In Cloud Computing By [Durairaj](https://www.researchgate.net/profile/Durairaj) M. and P. Kannan, [Bharathidasan University](https://www.researchgate.net/institution/Bharathidasan_University)

[9]. 2014, Virtualization In Cloud Computing By T.Swathi(Asst Prof, CSE Dept, GPREC, KURNOOL, swathitudi@gmail.com), K.Srikanth(Asst Prof, CSE Dept, GPREC, KURNOOL, gprec.srikanth@gmail.com), S. Raghunath Reddy(Asst Prof, CSE Dept, GPREC, KURNOOL, [raghu10091d2505@gmail.com](mailto:raghu10091d2505@gmail.com))

[10]. 2016, Virtualization Basics: Understanding Techniques and Fundamentals By Hyungro Lee, School of Informatics and Computing, Indiana University, 815 E 10th St., Bloomington, IN 47408, lee212@indiana.edu