Proyecto de Grado

Creación de Ambientes Virtuales Seguros para la Compilación, Ejecución y Evaluación de Códigos de Programación

Por

Br. Marianne Lucette Aymard Cuello

Tutor: Prof. MSc. Rodolfo Sumoza

Mayo 2017

Creación de Ambientes Virtuales Seguros para la Compilación, Ejecución y Evaluación de Códigos de Programación

Br. Marianne Lucette Aymard Cuello

Proyecto de Grado — Sistemas Computacionales, 57 páginas

Resumen: El uso y los beneficios de la virtualización son indiscutibles en el mundo informático de hoy. El hecho de poder simular Máquinas Virtuales en una máquina física, representa un avance de importancia para el desarrollo de *software* y para la seguridad de las redes y equipos. Este proyecto se concentra en la creación de entornos virtuales para la ejecución y compilación de código de programación, que serán utilizados para virtualizar las prácticas de algunas asignaturas de la carrera Ingeniería de Sistemas. Se permite la creación de múltiples Máquinas Virtuales y se realizaron pruebas en donde se comprobó que la compilación y ejecución de código realizada de forma paralela en cada Máquina Virtual no afecta el funcionamiento del entorno ni de ninguna otra Máquina Virtual, garantizando así la seguridad del ambiente de trabajo.

Palabras clave: Virtualización, Máquina Virtual, Seguridad

A mis padres, a mi familia y a Lucas.

Índice

Índice de Figuras

Agradecimientos

En primer lugar, agradezco a mis padres por su apoyo constante e incondicional.

A mi primo Germán Corredor por su ayuda e incondicional apoyo.

A mi amigo Luis Fernando Angulo, por su amistad y apoyo durante todo este camino. Por sus enseñanzas y gran ayuda a lo largo de la carrera.

A mi Tutor el Profesor Rodolfo Sumoza, por su paciencia, dedicación y ayuda durante la realización de este Proyecto de Grado.

A mis amigos Jesús Mazzei, Miguel Martínez y Omar Díaz, por su amistad y apoyo durante mi carrera universitaria.

A mis primos Ricardo Kowalski Corredor y Belkys Mora de Kowalski, por todo su apoyo durante esta etapa de mi vida.

A todos mis amigos y compañeros de estudio, que de una manera u otra estuvieron presentes y me brindaron su apoyo.

Finalmente, quiero agradecer a todos los profesores de la Ilustre Universidad de Los Andes que directa e indirectamente ayudaron a formarme y a convertirme en Ingeniero de Sistemas.

Capítulo 1

Introducción

La virtualización es la creación a través de *software* de versiones virtuales de algún recurso tecnológico, como puede ser una plataforma de *hardware*, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red [ETV08]. Se refiere a la abstracción de los recursos de un computador, utilizando un hipervisor o Monitor de Máquina Virtual (*Virtual Machine Monitor*) que crea una capa que abstrae el *hardware* de la máquina física (*host*) para ser utilizado por el Sistema Operativo de la Máquina Virtual (*guest*), dividiéndose el recurso en uno o más entornos de ejecución. Las soluciones de virtualización son cada vez más utilizadas, y entre sus beneficios está la posibilidad de tener a disposición diferentes plataformas, ya sea para desarrollo, pruebas, servidores, seguridad, entre otros, sin tener que incurrir en realizar inversiones muy costosas para adquirir nuevo *hardware*.

En la actualidad, la virtualización es un concepto en evolución y una tendencia que tiende a disminuir los costos. El hecho de transformar máquinas físicas en Máquinas Virtuales, permitiendo tener varias de ellas dentro de un mismo servidor físico que funcionen concurrentemente, es una opción cada vez más atractiva para las organizaciones; ya que les permite ahorrar costos de inversión en adquisición y mantenimiento de *hardware*.

Con un entorno virtual, la creación, duplicación, traslado y restauración de Máquinas Virtuales es relativamente fácil. En los casos de los problemas de seguridad en las Tecnologías de Información y Comunicación, la prevención y recuperación implican

tareas que toman mucho menos tiempo que en los escenarios normales, lo cual implica una mejora sustancial en el tiempo de funcionamiento de las organizaciones.

Hoy en día, la virtualización se encuentra presente en casi todos los aspectos de nuestras vidas, desde el uso de nuestros teléfonos personales hasta la forma en que las nuevas generaciones de personas estudian. La meta de la virtualización es llegar a la mayor cantidad de personas, por lo cual podría convertirse en el futuro de la educación.

Cada día se incrementa el uso de laboratorios virtuales para la enseñanza y aprendizaje en todos los niveles de la educación, y en particular en varias universidades del mundo. En la Universidad de Los Andes, núcleo Mérida, se tiene la iniciativa de crear laboratorios virtuales para el desarrollo de actividades académicas en diferentes asignaturas de la carrera de Ingeniería de Sistemas. Para ello se requiere desarrollar un entorno virtual que permita la compilación y ejecución de código fuente, que garantice la seguridad de los equipos físicos y de la red.

En este proyecto de grado de desarrolló un entorno virtual que permite ejecutar múltiples instancias de un Sistema Operativo simultáneamente. Cada entorno está formado por una Máquina Virtual, dentro de la cual se pueden compilar y ejecutar códigos de programación escritos en un lenguaje determinado.

En el capítulo 2, se muestra el Diseño del Entorno Virtual, en el capítulo 3 se habla sobre las Herramientas de Virtualización utilizadas para la creación del mismo, en el capítulo 4 se menciona cómo se implementaron las aplicaciones para virtualización seleccionadas para la construcción del entorno, y finalmente las conclusiones y recomendaciones se ubican en el capítulo 5.

1.1 Marco Teórico

1.1.1 Virtualización

La manera en que los servicios computacionales son provistos ha evolucionado progresiva y considerablemente durante las últimas 5 décadas, dando como resultado lo que conocemos hoy día. Desde el procesamiento en super computadores en los años 60 y 70, pasando por los computadores portátiles y la tecnología cliente-servidor en los años 80 y 90, hasta el Internet con todas sus ventajas de las que disfrutamos actualmente,

son una muestra de esto.

Actualmente nos encontramos en medio de una era donde la virtualización ha adquirido liderazgo y es uno de los paradigmas tecnológicos más utilizados. Desde sus inicios en los años 60, cuando su intención era la de "fraccionar lógicamente los super computadores, para así poder realizar múltiples tareas simultáneamente", hasta la computación en nube de la actualidad, su concepto ha evolucionado pero sigue manteniendo su esencia.

Anteriormente los computadores centrales o *mainframes* solo podían realizar una tarea a la vez. A diferencia de su bajo rendimiento, sus costos eran bastantes altos lo cual no justificaba el gasto económico que representaban para las empresas. Esto cambió cuando IBM desarrolló un método para crear múltiples particiones lógicas, o también conocido como *VM mainframe*, el cual permitía que estas particiones trabajaran independientemente unas de las otras y de forma simultánea, utilizando cada una de ellas los recursos suministrados por el *mainframe*. Ese fue el inicio de lo que hoy conocemos como Máquinas Virtuales.

Para finales de la década de los 80 y principios de los 90, la llegada de los sistemas basados en arquitectura x86 y la adopción de los Sistemas Operativos Windows y Linux, desviaron la atención de las aplicaciones basadas en *mainframe* y surgió la era de los micro computadores, orientada a aplicaciones cliente-servidor y a la computación distribuida. Los sistemas podían realizar múltiples tareas simultáneamente, lo que llevó a que los enormes computadores centrales que solo ejecutaban una función a la vez, empezaran a ser cambiados por pequeños servidores y computadores personales de arquitectura x86 que eran mucho más potentes y livianos, convirtiéndose rápidamente en el estándar de la industria. A causa de esto, el tema de la virtualización quedó temporalmente desplazado.

A medida que los centros de datos empezaron a crecer, también crecieron las cifras que costaba mantenerlos. Especialmente porque un servidor era dedicado enteramente a una aplicación, para evitar conflictos con otras aplicaciones. Esto produjo que solo se utilizase entre un 10% y 15% de los recursos computacionales de cada equipo, ocasionado grandes desperdicios de *hardware* y altos costos en energía, espacio y mantenimiento [Gra11]. Esta situación dejó de ser rentable para las empresas y fue así

como a finales de los años 90 y principios del 2000, al igual que en la década de los 60, las compañías empezaron a pensar nuevamente en la virtualización.

Gracias al incremento de la eficiencia del *hardware*, la utilización de un computador para efectuar una sola tarea significó un desperdicio de recursos, espacio, energía y dinero. No es conveniente asignar múltiples usos o instalar varias aplicaciones en un servidor convencional; ya que estas aplicaciones podrían ser conflictivas entre sí, podrían necesitar diferentes configuraciones o incluso distintos Sistemas Operativos, así como tener diferentes requerimientos de seguridad o causar problemas al ejecutarse simultáneamente. Éstas son parte de las razones por la que vuelve a la luz la idea de dividir el *hardware* de tal manera que funcione como varios servidores independientes pero compartiendo los recursos de una misma máquina física.

Desafortunadamente los sistemas de arquitectura x86 actuales no fueron diseñados pensando en virtualización, lo que causó grandes retos al intentar ejecutar instrucciones privilegiadas en este tipo de sistemas.

Durante la década que inició en el año 2000, múltiples empresas desarrollaron varias soluciones y también la comunidad de *software* libre presentó algunas, las cuales permitieron que de una manera u otra, varios Sistemas Operativos pudiesen ejecutarse simultáneamente en sistemas de arquitectura x86. Lo cual generó un cambio en la percepción de virtualización que se tenía previamente y produjo el concepto que conocemos hoy día.

La definición de virtualización ha sido usada muchas veces en el área de la computación para representar ideas diferentes pero con bases muy similares. En este proyecto de grado se seguirá la definición usada por Singh [Sin04]:

"La Virtualización es un *framework* o metodología para dividir los recursos de un computador en múltiples entornos de ejecución, aplicando uno o más conceptos o tecnologías tales como particionamiento de *hardware* y *software*, tiempo compartido, simulación parcial o completa de la máquina, emulación, calidad de servicio, y muchos otros."

En tal sentido, la virtualización puede ser usada para dividir una máquina física, llamada *host*, en múltiples entornos. Donde cada uno de ellos es usado para ejecutar una Máquina Virtual o *guest*, con su propio Sistema Operativo [And10]. Utiliza una capa de abstracción que podría integrarse en el *hardware*, un hipervisor o una aplicación que "engaña" al Sistema Operativo haciéndole creer que está corriendo de manera nativa en el *hardware*.

1.1.2 Tipos de Virtualización

Virtualización Parcial

La Virtualización Parcial, como su nombre lo indica, simula parcialmente un entorno. Este tipo de virtualización incluye virtualización del espacio de direcciones (*address space virtualization*) y la mayoría de las instancias del *hardware* subyacente. Lo que significa que un Sistema Operativo completo no podrá correr en la Máquina Virtual. Como resultado de esto, algunos programas podrían necesitar modificaciones para poderse ejecutar en ese tipo de ambientes virtuales.

La Virtualización Parcial representó un avance histórico en el camino hacia la Virtualización Completa. Fue usada en la primera generación de sistemas de tiempo compartido CTSS (*Compatible Time-Sharing System)* [WV11], en el sistema paginado experimental IBM M44/44X, también en sistemas como MVS y el Commodore 64. El término también podía usarse para describir cualquier Sistema Operativo que proporcionara espacios de direcciones separados para usuarios o procesos individuales.

La experiencia y limitaciones vividas con la Virtualización Parcial, llevaron a la creación del primer sistema con Virtualización Completa, el CP-40 de IMB, que fue la primera iteración del sistema CP/CMS.

Virtualización Completa

Virtualización Completa o *Full Virtualization* es una emulación total del *hardware*, a través de cierto *hardware* y *software* [And10]. Ésta permite que el Sistema Operativo pueda ejecutarse sin modificaciones. En este caso, el Sistema Operativo no está al tanto de que está corriendo en un entorno virtual, por lo tanto el *hardware* es virtualizado por

un Sistema Operativo del *host* (el Sistema Operativo del *hardware*) para que el Sistema Operativo *guest* (Sistema Operativo que está ejecutándose en el entorno virtual) pueda ejecutar comandos en lo que "piensa" que es el *hardware* real, pero que en realidad son equipos de *hardware* simulados por el *host*.

Paravirtualización

La Paravirtualización se implementa al definir una nueva arquitectura híbrida de *hardware* y *software*, llamada hipervisor o Monitor de Máquina Virtual, la cual es parecida al *hardware* subyacente, pero no es exactamente igual [And10]. La Máquina Virtual no simula el *hardware*, pero el Sistema Operativo *guest* se ejecuta aisladamente en su propio dominio, como si estuviese corriendo en un entorno separado. Todo esto es posible gracias a una API (Interfaz de Programación de Aplicaciones) que se proporciona para este fin, la cual solo puede ser usada modificando el Sistema Operativo *guest*.

La Paravirtualización se refiere a la comunicación entre el Sistema Operativo *guest* y el hipervisor, para mejorar de esta manera el desempeño y la eficiencia. A diferencia de la Virtualización Completa, en la Paravirtualización el Sistema Operativo *guest* está "consciente" de que se está ejecutando sobre un entorno virtual, en donde hace llamadas al hipervisor, *hypercalls*, mientras las instrucciones no privilegiadas permanecen igual. Esto reduce significativamente el *overhead* (tiempo adicional de procesamiento), y consume menos recursos del *hardware* [And10, VMw08].

Virtualización de Aplicaciones

Así como los servidores y sistemas de escritorio, las aplicaciones también pueden ser virtualizadas. Esto lleva a la virtualización a un nivel superior, haciendo que las aplicaciones sean completamente independientes del Sistema Operativo subyacente [Gra11]. Existen dos razones principales por las cuales es necesario virtualizar aplicaciones, la primera es la facilidad del uso. Por lo general, en un computador se tiene gran cantidad de programas y aplicaciones y actualizarlas puede resultar una tarea tediosa. Varias compañías tienen cientos o miles de computadores, y realizar una actualización de aplicaciones en cada una de ellas les llevaría demasiado

trabajo y tiempo, lo cual implicaría un incremento significativo de los costos. Por esta razón, es que se empieza a virtualizar las aplicaciones, para ahorrar tiempo y optimizar el trabajo. La segunda razón está relacionada con la manera en que las aplicaciones pueden interactuar entre ellas. En muchas ocasiones hay aplicaciones que no son compatibles con versiones de otras aplicaciones, y esta incompatibilidad genera conflictos en la ejecución, ocasionando que el computador no funcione correctamente. La Virtualización de Aplicaciones evita estos problemas, permitiendo tener todas las aplicaciones necesarias en el mismo computador físico, pero en entornos de trabajo diferentes [Por16].

Virtualización de Sistemas de Escritorio

El uso de computadores de escritorio se ha vuelto costoso e ineficiente para las compañías, ya que se necesita mucho personal para realizar soporte y actualizaciones, lo que eleva los costos y puede poner en riesgo la factibilidad económica de las organizaciones en general.

La Virtualización de Sistemas de Escritorio permite que el Sistema Operativo completo de un computador sea virtualizado dentro del centro de datos y sea presentado al usuario final a través de un cliente ligero.

Los escritorios virtuales corren en servidores dentro de los centros de datos, los cuales son mucho más potentes que un computador de escritorio normal.

El acceso a los escritorios virtuales se hace a través de clientes ligeros, de los cuales la mayoría son más confiables y menos costosos que computadores tradicionales. Estos clientes ligeros tienen una vida útil entre 7 y 10 años y consumen entre un 5% y 10% de la electricidad. Si llegasen a fallar o a dañarse, pueden ser cambiados fácilmente por el usuario sin necesidad de un especialista [Por16].

Los usuarios pueden interactuar con un escritorio virtual de la misma manera en que lo hacen con un computador tradicional, pueden personalizarlo a su gusto al igual que en un computador físico.

A diferencia de los computadores tradicionales, los escritorios virtuales pueden ser fácilmente accedidos desde cualquier parte vía Internet [Gra11].

Virtualización Asistida por Hardware

En la Virtualización Asistida por Hardware, la arquitectura del *hardware* ayuda a facilitar la construcción de un Monitor de Máquina Virtual y permite que los Sistemas Operativos *guest* se ejecuten aisladamente. Este tipo de virtualización fue introducida por primera vez en el Sistema 370 de IBM en el año 1972, para ser usada con el Sistema VM/370, el primer Sistema Operativo de Máquina Virtual [RU05].

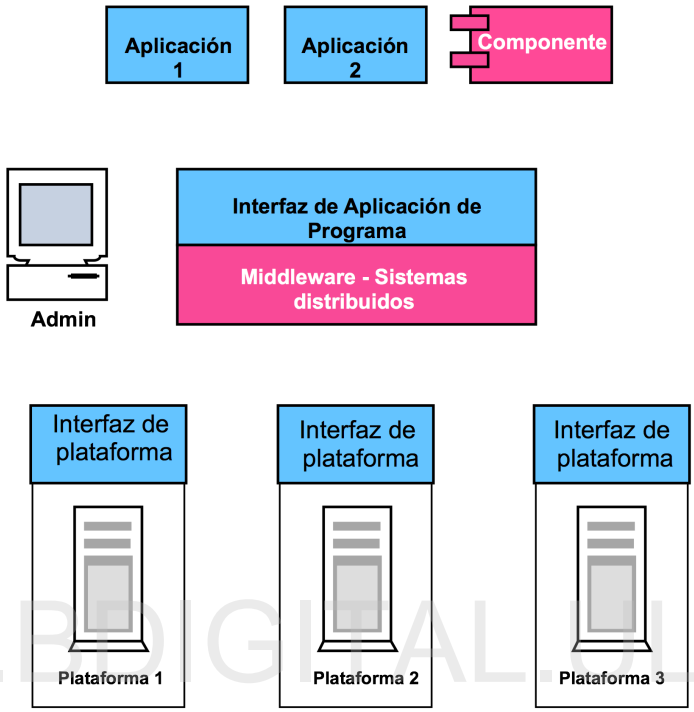
En el 2005 y 2006, Intel y AMD proporcionaron *hardware* adicional para soportar la virtualización.

Virtualización del Sistema Operativo

La Virtualización de Sistemas Operativos permite que múltiples instancias de Sistemas Operativos puedan ejecutarse simultáneamente en una misma máquina física (ver Figura 1.1). Desde el punto de vista del usuario final, estas instancias pueden llegar a verse y "sentirse" como un servidor real.

Este tipo de virtualización es la base de la Virtualización de Sistemas de Escritorio, pero originalmente fue diseñada para virtualizar servidores en los centros de datos e incrementar el uso del *hardware*.

Figura 1.1: Virtualización de Sistema Operativo. Tres instancias ejecutándose simultáneamente en un solo *hardware*



1.1.3 Componentes de la Virtualización de Sistema Operativo

Hipervisor

Un hipervisor o también llamado Monitor de Máquina Virtual (MMV), es la capa de *software* que permite que la virtualización sea posible. El hipervisor es el árbitro de los recursos, se ubica entre los recursos físicos del servidor y las Máquinas Virtuales que se ejecutan en ese servidor. Además de asignar y distribuir recursos, el hipervisor es el responsable de crear el ambiente virtual donde se ejecutan los Sistemas Operativos *guest* (el ambiente donde se realiza la virtualización). También hace posible la comunicación de los ambientes virtuales con el mundo real a través de la virtualización de redes y ofrece diversas formas de *clustering* [Por16].

El primer Monitor de Máquina Virtual fue creado para resolver un problema específico, el cual era la asignación de recursos tratando de utilizar áreas de memoria

que normalmente eran de difícil acceso para los programadores. Con el paso de los años han evolucionado desviándose un poco de su propósito original, por eso el término *Manager* de Máquina Virtual (*Virtual Machine Manager*) fue sustituido por hipervisor.

Los hipervisores de hoy día permiten hacer un mejor uso de los nuevos procesadores que hay en el mercado, ayudando a usar de una manera más eficiente la cantidad de memoria que ofrecen.

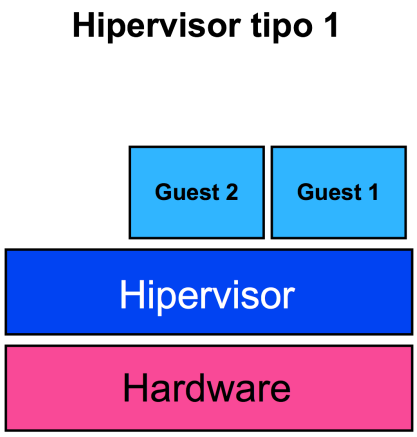
Sin un hipervisor, el Sistema Operativo se comunica directamente con el *hardware* subyacente, lo que ocasionaría que las operaciones de disco vayan directamente al disco y que las llamadas de memoria sean recibidas directamente por la memoria física. Así mismo, los Sistemas Operativos presentes en las múltiples Máquinas Virtuales pudiesen tener el control simultáneo del *hardware*, lo que resultaría en caos. El hipervisor se encarga de solucionar este problema, controlando las interacciones entre las Máquinas Virtuales y el *hardware* compartido por todas.

Los hipervisores son clasificados en dos tipos.

Hipervisor Tipo 1

El Hipervisor Tipo 1 se ejecuta directamente sobre el *hardware*, sin la necesidad de un Sistema Operativo de por medio (ver Figura 1.2). Por esta razón se le denomina nativo o *bare-metal*. Al no tener intermediaros, el hipervisor se comunica directamente con los recursos del *hardware*, siendo así más eficiente que un Hipervisor Tipo 2. Este tipo de hipervisor es el responsable de la asignación de todos los recursos (CPU, memoria, disco, etc.). Es también considerado más seguro que su homólogo, ya que al no tener contacto directo con un Sistema Operativo deja de ser vulnerable a ataques o códigos maliciosos. Adicionalmente, el *overhead* es menor, lo que significa que en cada servidor se puede ejecutar un mayor número de Máquinas Virtuales [Por16].

Figura 1.2: Hipervisor Tipo 1



Hipervisor Tipo 2

El Hipervisor Tipo 2 es una aplicación que se ejecuta directamente sobre el Sistema Operativo del *hardware* (ver Figura 1.3). Es mucho más fácil de instalar y ejecutar porque muchas de las configuraciones necesarias están cubiertas por el Sistema Operativo.

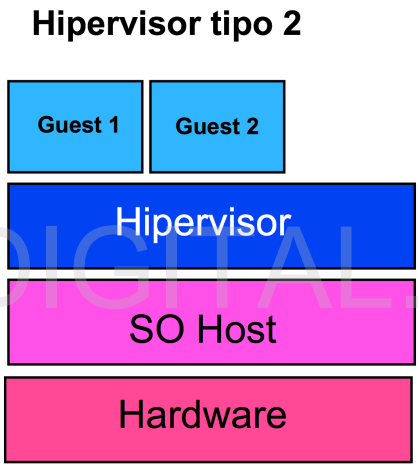
Los hipervisores Tipo 2 no son tan eficientes como los Tipo 1, debido a la capa extra de *software* que tienen entre el *hardware* y ellos. Esto ocasiona más *overhead* ya que cada vez que una Máquina Virtual haga cualquier interacción con el *hardware*, esta petición se entrega al hipervisor. El hipervisor la entrega al Sistema Operativo, que es el encargado de las operaciones de E/S, luego de procesarla, éste la envía de vuelta al hipervisor y a su vez, el hipervisor la entrega a la Máquina Virtual. Todo este proceso agrega dos pasos adicionales a cada transacción.

A diferencia de los hipervisores Tipo 1, los hipervisores Tipo 2 son menos seguros y confiables. Cualquier falla o problema que afecte al Sistema Operativo subyacente afectará al hipervisor y a las Máquinas Virtuales que ahí se ejecuten. Por ejemplo, si el Sistema Operativo del *host* necesitara ser reiniciado, todas las Máquinas Virtuales también se reiniciarían, lo cual interrumpiría cualquier operación importante que se

estuviese realizando en ese momento.

Los hipervisores Tipo 2 consumen mucho más recursos que los Tipo 1, ya que aparte de los recursos que utiliza el hipervisor se le adicionan los recursos que utiliza el Sistema Operativo donde éste está corriendo.
Los hipervisores Tipo 2 son usados generalmente en ambientes de desarrollo de aplicaciones, en donde un solo desarrollador trabaja con una o varias Máquinas Virtuales mientras perfecciona el producto final.

Figura 1.3: Hipervisor Tipo 2



Máquina Virtual

Las Máquinas Virtuales o MV son el componente fundamental de la virtualización. Son las "cajas" que contienen los Sistemas Operativos tradicionales que se ejecutan sobre un hipervisor. Internamente son muy parecidas a un servidor físico, pueden soportar un Sistema Operativo y las aplicaciones que en él se ejecuten. A diferencia de un servidor físico, múltiples Máquinas Virtuales pueden ejecutarse simultáneamente dentro de un servidor físico, como también estas MV pueden ser capaces de ejecutar diferentes Sistemas Operativos y soportar diferentes aplicaciones.

Las Máquinas Virtuales emulan a un computador real, y comparten con él sus recursos físicos como RAM, disco, tarjeta gráfica, entre otros.

En el artículo *"Formal Requirements for Virtualizable Third Generation Architectures"* Gerald J. Popek y Robert P. Goldberg [PG74] definen una Máquina Virtual de la siguiente manera:

"Una Máquina Virtual puede virtualizar todos los recursos del *hardware*, incluyendo procesadores, memoria, almacenamiento y conectividad. Un Monitor de Máquina Virtual (MMV), lo que es conocido como hipervisor hoy en día, es el *software* que provee el ambiente en el cual la Máquina Virtual opera."

Según Popek y Goldberg, un Monitor de Máquina Virtual necesita cumplir las siguientes tres propiedades, para satisfacer correctamente su definición, en el siguiente orden:

- **Fidelidad**. El ambiente que crea la MV tiene que ser exactamente igual a la máquina física.
- **Aislamiento o Seguridad**. El MMV debe tener un control absoluto sobre los recursos del sistema.
- **Desempeño**. No deben existir diferencias de desempeño entre la Máquina Virtual y la máquina física equivalente.

Las Máquinas Virtuales se dividen en dos categorías, Máquinas Virtuales de Proceso y Máquinas Virtuales de Sistema.

Máquina Virtual de Proceso

Las Máquinas Virtuales de Proceso son también conocidas como Máquinas Virtuales de Aplicación. Su principal característica es que solo pueden ejecutar un proceso a la vez, para lo cual agregan una capa adicional de Sistema Operativo que simula el ambiente de programación para la ejecución de ese proceso. Se pueden crear múltiples instancias de Máquinas Virtuales de Proceso para permitir la ejecución de varias aplicaciones asociadas a múltiples procesos. Este tipo de MV se inicia automáticamente cuando el proceso comienza a ejecutarse y se detiene cuando éste termina. Su objetivo principal

es el de proporcionar un ambiente de ejecución independiente de la plataforma y del Sistema Operativo, lo que significa que permita que las aplicaciones se ejecuten siempre de la misma manera en el *hardware* subyacente y en diversas plataformas.

Las Máquinas Virtuales de Aplicación abstraen los lenguajes de programación de alto nivel y son implementadas usando un intérprete [Bis12].

La Máquina Virtual de Proceso más conocida es la Máquina Virtual de Java (JVM) y también la llamada *Common Language Runtime*, usada para virtualizar el entorno de programación de Microsoft .NET.

Máquina Virtual de Sistema

Las Máquinas Virtuales de Sistema emulan en su totalidad un computador y soportan la ejecución de un Sistema Operativo completo. Estas MV abstraen una Arquitectura de Conjunto de Instrucciones (ISA por sus siglas en inglés) la cual es usada como *hardware* subyacente por el Sistema Operativo *guest*. Las principales ventajas de las Máquinas Virtuales de Sistema son la consolidación (permite la coexistencia aislada de múltiples Sistemas Operativos en un solo computador), bajo mantenimiento, alto rendimiento y la recuperación de fallas. Además, otras ventajas son el *sandboxing* (método computacional utilizado para ejecutar programas de manera segura y separada) que provee mayor rapidez para reiniciar y para la depuración [Bis12].

Guest

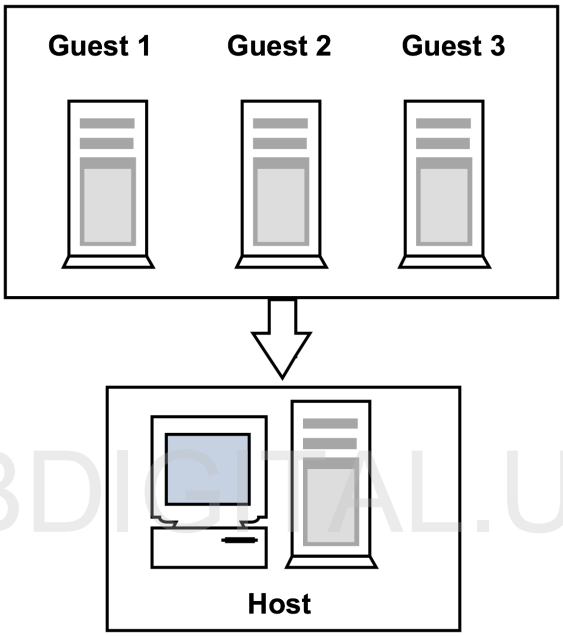
Guest o huésped, en su traducción al español, bien sea una aplicación o un Sistema Operativo, es lo que va a ser virtualizado a través del hipervisor. Por lo general lo que se virtualiza es un Sistema Operativo completo sin modificaciones, pero dependiendo del tipo de hipervisor, pueden llegar a necesitarse controladores de software (*driver*) o realizar modificaciones en el código fuente del Sistema Operativo.

Host

Host o anfitrión, es el *hardware* (Hipervisor Tipo 1) o *software* (Hipervisor Tipo 2) que alberga al Monitor de Máquina Virtual para hacer posible la virtualización. El

host simula por completo el entorno computacional necesario para que el *guest* pueda ejecutarse (ver Figura 1.4).

Figura 1.4: Ilustración del funcionamiento del *host* y *guest* en la Virtualización de Sistema Operativo



1.1.4 Ventajas de la Virtualización

La principal ventaja de la virtualización es el aprovechamiento del *hardware*. Lo cual ha permitido que varios sistemas se consoliden en un mismo servidor físico. Esto produjo que millones de servidores subutilizados fuesen virtualizados, generando un ahorro de energía, espacio, capacidad de refrigeración y administración.

Según VMware, la virtualización puede reducir hasta en un 50% los costos operativos y de *hardware*, y hasta en un 80% los costos de energía. Ahorrando más de US$3000 al año por cada espacio de trabajo virtualizado [VMw08].

Otra ventaja que ofrece la virtualización es el *sandboxing*, que permite probar distintas aplicaciones no confiables. Esto incrementa la seguridad, ya que si algo falla, el daño se limita a la Máquina Virtual. Permite el aislamiento y recuperación rápida

en caso de fallas, ya que es mucho más fácil restaurar o recuperar una Máquina Virtual que una física.

La consolidación es un aspecto muy importante de la virtualización, en particular para fines educativos. Por lo general, las instituciones no cuentan con presupuestos muy elevados para crear laboratorios de computación. La consolidación permite ejecutar varios Sistemas Operativos en un mismo computador, brindando a los estudiantes mayores experiencias en el área, sin la necesidad de adquirir equipos costosos [And10].

La virtualización proporciona flexibilidad, agilidad y portabilidad. Las Máquinas Virtuales pueden ser personalizadas con las características que el usuario desee otorgarles, su creación es un proceso muy rápido, lo cual agiliza cualquier operación, y toda su configuración reside en uno o varios archivos, lo cual facilita su clonación y/o traslado a otro servidor.

1.1.5 Desventajas de la Virtualización

La desventaja más obvia es la creación de un punto único de falla (el Sistema Operativo *host*), ya que todos los Sistemas Operativos *guest* pueden fallar, si el *host* falla. Cualquier falla en el Sistema Operativo, redes y/o *hardware* del *host*, puede afectar a múltiples sistemas.

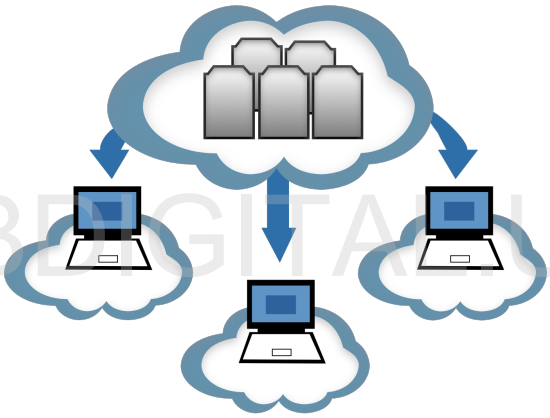
Otro problema con la virtualización es una reducción en la capacidad de manejar picos en el uso del CPU o de la red. Si el número de Máquinas Virtuales que se ejecutan en el *host* está determinado por la carga de CPU promedio esperada, los picos de uso pueden requerir más recursos que los disponibles, lo que resulta en sistemas *guest* que pudiesen tener un servicio degradado [And10].

1.1.6 Computación en la Nube

La virtualización es el motor que impulsa la Computación en la Nube mediante la transformación de los centros de datos en recursos consumibles, fáciles de gestionar, escalar y de alta disponibilidad. Antes de la virtualización, los administradores de sistemas pasaban 70% o más del tiempo, realizando actividades de rutinas y solucionando problemas, lo que no dejaba tiempo para el crecimiento y la innovación.

La virtualización, y su extensión, la Computación en la Nube, ofrecen mayores posibilidades de automatización, como el respaldo y recuperación automática de datos, reduciendo los costos administrativos y aumentando la capacidad de las organizaciones de implementar soluciones dinámicamente. Siendo capaz de abstraer la capa física del *hardware*, la Computación en la Nube crea el concepto de un centro de datos virtual, una estructura que contiene todo lo que un centro de datos físico tendría (ver 1.5). Este centro de datos virtual, desplegado en la nube, ofrece recursos según sean necesarios [Por16].

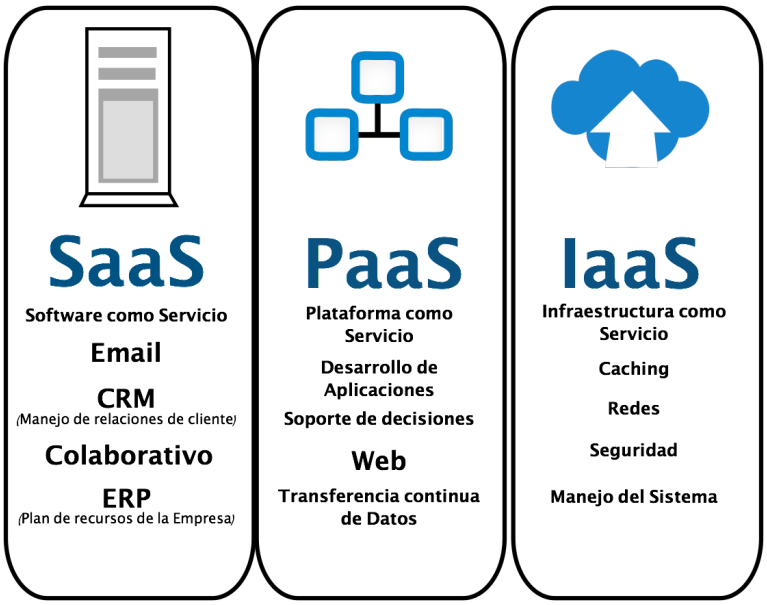
Figura 1.5: Computación en la Nube



La Computación en la Nube ofrece sus servicios a través de diferentes modelos, los cuales según el Instituto Nacional de Estándares y Tecnología (NIST por sus siglas en inglés) son los siguientes: **Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS).**

Estos modelos son abstracciones representadas a menudo como capas en una pila. Infraestructura, plataforma y *software* como servicio, no tienen que estar necesariamente relacionados entre sí (ver 1.6).

Figura 1.6: Servicios que ofrece la Computación en la Nube



Estos modelos simplifican drásticamente el uso de nuevas aplicaciones y permiten a las empresas acelerar sus implementaciones sin sacrificar la escalabilidad, la resiliencia o la disponibilidad [Sos11].

Infraestructura como Servicio

Infraestructura como Servicio o IaaS por sus siglas en inglés, es la última capa de Computación en la Nube, y provee recursos de *hardware* virtualizados a través de Internet. Es un servicio en línea que abstrae al usuario de los detalles de infraestructura (CPU, RAM, discos, red, etc.). Permite a los clientes utilizar un espacio en la nube y así evitar los costos de construcción de sus propios centros de datos [Sos11].

Para su utilización, se requiere de un hipervisor desplegado en la nube que se ejecuta como *guest*. Además se requieren de un grupo de hipervisores dentro del Sistema Operativo de la nube para poder soportar un gran número de Máquinas Virtuales y tienen la habilidad de ajustar sus servicios de acuerdo a los requisitos del cliente [TEP13].

El cliente no controla la nube, pero sí puede controlar el Sistema Operativo, el almacenamiento, las aplicaciones implementadas y posiblemente tenga control limitado

de algunos componentes de red.

La IaaS presenta algunas desventajas en el área de seguridad y privacidad. Para poder hacer uso del servicio, las compañías deben almacenar sus datos en la nube o hacerlos accesibles a través de ella. Esto significa que los mismos deben salir de la organización, lo que los hace más vulnerables [Sos11].

Plataforma como Servicio

Plataforma como Servicio o PaaS por sus siglas en inglés, provee un ambiente para crear y desplegar aplicaciones a través de Internet. Normalmente incluye un Sistema Operativo, ambiente de compilación y ejecución de lenguajes de programación, bases de datos y servidores web [MBK10]. Representa un alto nivel de abstracción, ya que los desarrolladores de aplicaciones pueden crear y ejecutar sus productos en la nube, sin preocuparse por los costos y la complejidad del *hardware* y del *software* [TEP13].

Algunos proveedores de PaaS como Microsoft Azure y Google App Engine, proveen el servicio de ofrecer automáticamente en la nube los recursos de almacenamiento del computador subyacente, para que el usuario no tenga que lidiar con la asignación de recursos de manera manual [Sos11].

El usuario no tiene control de la nube ni del Sistema Operativo, tampoco de las redes y almacenamiento. Pero si puede tener el control de las aplicaciones desplegadas y posibles ajustes y configuración de las mismas.

Software como Servicio

Software como Servicio o SaaS por sus siglas en inglés, es la primera capa de Computación en la Nube. En este modelo el *hardware* y el *software* son provistos por un proveedor como un servicio completo. Los proveedores administran la infraestructura y la plataforma donde se ejecutan las aplicaciones [Sos11]. También instalan y operan el *software* de la aplicación en la nube y los usuarios acceden a éste como clientes. Esto elimina la necesidad de instalar y ejecutar la aplicación en el computador del cliente, facilitando el mantenimiento y servicio. Las aplicaciones están disponibles en todo momento a través de Internet [TEP13].

El usuario no controla la nube, ni el Sistema Operativo, tampoco el almacenamiento

y las redes, ni siquiera algunas capacidades de la aplicación; con una posible excepción de ciertos ajustes específicos de la misma.

1.2 Justificación

La virtualización puede ser utilizada para la creación de laboratorios virtuales con fines educativos, haciendo un uso más eficiente de los beneficios de los recursos físicos. Además, se ha demostrado que el uso de laboratorios virtuales produce en los estudiantes el mismo resultado que el uso de laboratorios físicos, con la diferencia de que en ocasiones los alumnos disfrutan más el entorno virtual porque les permite trabajar desde sus hogares [LS06].

En la Escuela de Ingeniería de Sistemas de la Universidad de Los Andes, se tiene una iniciativa de permitirle a los alumnos poder realizar sus prácticas, tareas y proyectos de laboratorio vía remota y de forma más efectiva. Para esto se requiere de varios esfuerzos, que incluyan, entre otras cosas, la seguridad del entorno de compilación y ejecución de código. Basándose en el hecho que la mayoría de las actividades de los laboratorios en la mencionada carrera conllevan la necesidad de compilar y ejecutar código escrito en algún lenguaje de programación, se requiere de algún entorno seguro que permita su compilación y ejecución de forma particular sin que afecte o comprometa el funcionamiento del sistema general, ni la compilación y ejecución del resto de códigos pertenecientes al mismo alumno o a otros participantes.

Uno de los motivos que lleva a la realización de este proyecto es la ausencia de este tipo de herramientas en la Institución. Considerando que este tipo de recurso ya ha sido creado y utilizado anteriormente en distintas instituciones alrededor del mundo, nunca se ha desarrollado en la Universidad de Los Andes.

1.3 Planteamiento del Problema

Se requiere de un entorno computacional seguro para la compilación y ejecución de código fuente, para esto se necesita tener un control preventivo que evite que se comprometa el funcionamiento adecuado del computador durante el tiempo de

compilación y el tiempo de ejecución. Este entorno debe permitir la compilación y ejecución de código en múltiples instancias y de forma simultánea con el objetivo de realizar pruebas seguras en actividades como prácticas de laboratorio, proyectos, etc. que requieren programación, vinculadas a algunas asignaturas de la carrera de Ingeniería de Sistemas.

1.4 Objetivos

1.4.1 Objetivo General

Crear Ambientes Virtuales Seguros para la Compilación y Ejecución de Código.

1.4.2 Objetivos Específicos

1. Seleccionar las herramientas de virtualización.
2. Implementar la Máquina Virtual.
3. Instalar una imagen reducida de kernel para la compilación y ejecución de código.

1.5 Actividades

- Buscar distintas herramientas de virtualización existentes.
- Estudiar detalladamente cada herramienta encontrada.
- Instalar y probar las herramientas encontradas.
- Seleccionar la herramienta a utilizar.
- Configurar el ambiente de trabajo.
- Instalar y configurar la Máquina Virtual a utilizar.
- Instalar en la Máquina Virtual una imagen reducida de kernel para la compilación y ejecución de código.

- Realizar pruebas de varias instancias en el ambiente de trabajo.

1.6 Alcance

Este proyecto tiene como fin la creación de un entorno virtual seguro para la compilación y ejecución de código fuente, que será utilizado en algunas materias de la Carrera Ingeniería de Sistemas de la Universidad de Los Andes. Con este objetivo, el entorno permitirá ejecutar varias instancias del entorno de compilación y ejecución de código al mismo tiempo, lo cual permitirá atender a varios estudiantes a la vez. Se realizarán pruebas del entorno virtual en donde se garantice la seguridad del ambiente físico (computadores del laboratorio de computación) y de las otras Máquinas Virtuales.

No se incluye el manejo de colas para la atención de las solicitudes no atendidas. La forma de interacción es vía consola, y no gráfica, ya que se prevé que esta solución se comunicará con otras aplicaciones, utilizando APIs, y no directamente con los usuarios.

1.7 Ancedentes

Existen numerosos proyectos que han desarrollado laboratorios virtuales y ambientes de programación similares.

1.7.1 Xen Worlds

El proyecto Xen Worlds utiliza el hipervisor Xen para crear un laboratorio virtual que provee a los estudiantes una red personal de Máquinas Virtuales. El ambiente de trabajo de este proyecto consume una cantidad mínima de *hardware* y *software*, utiliza una licencia de *software* libre, lo que hace que sea de bajo costo y bastante accesible para instituciones educativas. El *hardware* actual de Xen Worlds consiste de cinco servidores que son capaces de proveer hasta 470 Máquinas Virtuales.

Cada Xen World (Máquina Virtual personal del estudiante) puede aislarse entre sí, y los estudiantes pueden obtener acceso como superusuario a sus Máquinas Virtuales a través de Internet, sin los problemas de privacidad y seguridad que están presentes en un laboratorio físico normal. Además, para apoyar a los estudiantes que están fuera

de la institución, Xen Worlds tiene varias características que procuran asegurar que el sistema sea igualmente accesible y fácil de usar, incluso si el estudiante tiene acceso limitado a recursos informáticos o de red [And10].

1.7.2 SEED: SEcurity EDucation

El proyecto SEED, es un laboratorio virtual diseñado para enseñar seguridad informática [WDW07]. La principal motivación detrás de su creación es la de tener un ambiente común en donde puedan realizarse actividades en distintas áreas de seguridad informática [DW08]. En este proyecto, los estudiantes utilizan VMware (como hipervisor) para ejecutar sus Máquinas Virtuales con un Sistema Operativo Linux o Minix. La ventaja de esto es que en laboratorios que requieran una revisión profunda de los módulos del kernel, el uso de Minix lo hace más sencillo; al igual que los laboratorios que requieran utilizar un Sistema Operativo completo, utilizan Linux. Uno de los aspectos más importantes de SEED es el hecho de utilizar *software* libre, lo que lo hace más accesible y permite a los estudiantes descargar e instalar en sus Máquinas Virtuales todo lo necesario para realizar sus actividades.

Además de proveer un entorno virtual, SEED apunta a crear una biblioteca de actividades, donde se cubren una variedad de áreas, y pueda ser utilizada por otras instituciones.

1.7.3 SOFTICE: Scalable, Open source, Fully Transparent and Inexpensive Clustering for Education

El proyecto SOFTICE fue creado para proveer un ambiente virtual donde el estudiante tuviese acceso a su propio sistema y pudiese crear redes de cualquier tamaño, en cualquier topología [WAR07b]. Para crear este entorno, SOFTICE se enfocó en tres áreas [WAR07a]:

- Virtualización
- Acceso Remoto
- Adaptabilidad

Para la Virtualización, SOFTICE utiliza User Mode Linux para proveer a los estudiantes un gran número de Máquinas Virtuales. Para el Acceso Remoto, los estudiantes usan SSH o X-Windows para conectarse a un *"master-node"* (nodo principal), que actúa como *gateway* en el ambiente virtual. Finalmente, la Adaptabilidad se maneja utilizando un *cluster* de nodos computacionales, creado a partir de computadores antiguos a través de la herramienta Warewulf Clustering Toolkit [WAR07a].

La meta pedagógica de SOFTICE es la de minimizar la necesidad que tienen los estudiantes de "aprender la plataforma del laboratorio", y maximizar su tiempo con el entorno [WAR07a].

1.7.4 V-NetLab: Virtual Network Laboratory

Fue desarrollado para proveer a estudiantes de computación y de redes un ambiente virtual, seguro y aislado [KKS05]. Además, fue diseñado para abordar los problemas de aislamiento con la realización de investigaciones sobre el código malicioso de propagación automática y otras aplicaciones potencialmente dañinas [WSS08]. Teniendo una finalidad educativa, se identificaron tres desafíos [KKS05]:

- Los cambios de configuración pueden dañar el Sistema Operativo completo.
- Los estudiantes pueden utilizar incorrectamente los permisos de root o de administrador.
- Los costos relacionados con la creación de un laboratorio físico.

Para lidiar con estos problemas, V-NetLab utiliza VMware para proveer Máquinas Virtuales con Sistemas Operativos Linux o Windows. Estas MV balancean la carga de trabajo en un *cluster* conformado por estaciones de trabajo, con un único servidor NFS (*Network File System*) que proporciona el almacenamiento de las imágenes de las MV [WSS08].

Capítulo 2

Diseño del Entorno Virtual para Compilación y Ejecución de Código de Programación

En este capítulo se describe la solución teórica al problema, se explica cómo a través de la Virtualización de Sistemas Operativos, se dió respuesta a los requerimientos planteados.

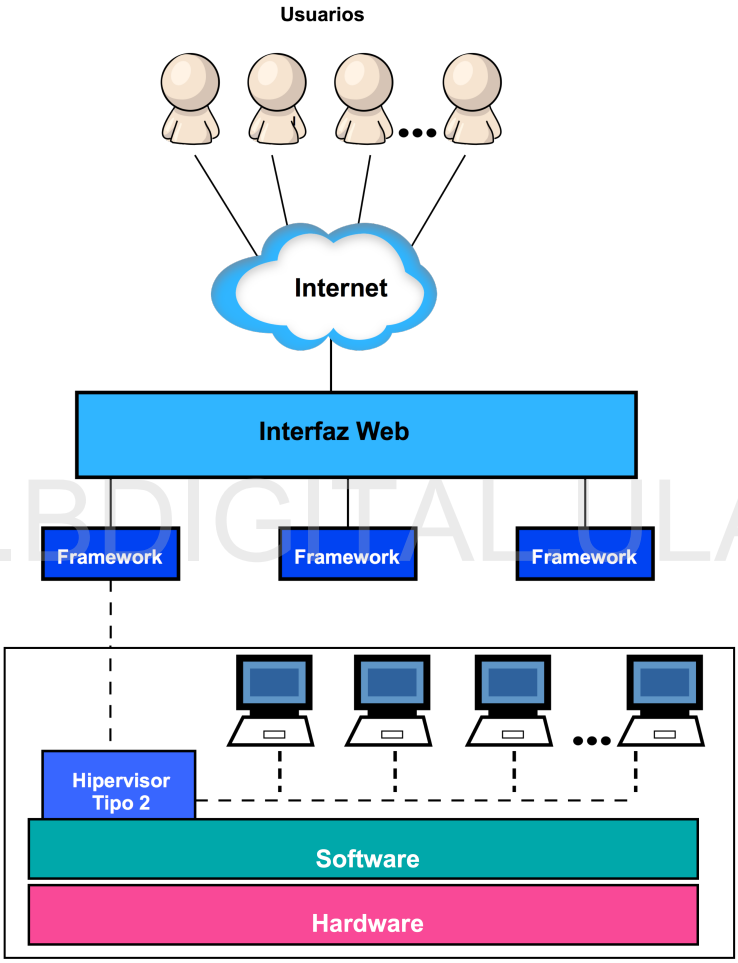
Tal como se mencionó en el capítulo anterior, el problema a resolver es que se requiere de un entorno computacional seguro para la compilación y ejecución de código fuente, para esto se necesita tener un control preventivo que evite que se comprometa el funcionamiento adecuado del computador durante el tiempo de compilación y el tiempo de ejecución. Este entorno debe permitir la compilación y ejecución de código en múltiples instancias y de forma simultánea con el objetivo de realizar pruebas seguras en actividades como prácticas de laboratorio, proyectos, etc. que requieren programación, vinculadas a algunas asignaturas de la carrera de Ingeniería de Sistemas.

En esencia, lo que se realizó fue la culminación de una fase necesaria para la creación de un laboratorio virtual con fines educativos para la Escuela de Ingeniería de Sistemas de la Universidad de Los Andes, Núcleo Mérida, el cual permita resolver las limitaciones que ofrecen los entornos físicos, incluidas las deficiencias de los equipos actuales que afecta el desarrollo de las sesiones prácticas de la mayoría de los cursos vinculados a la

programación de sistemas.

En la Figura 2.1 se ilustra el diseño del laboratorio virtual, en donde el entorno de compilación y ejecución de código es una parte fundamental.

Figura 2.1: Ilustración del diseño del laboratorio virtual



Para desarrollar un entorno seguro para la compilación y ejecución de código fuente con estas características, se propuso construir un entorno de Virtualización de Sistemas Operativos de manera que puedan compilarse y ejecutarse simultáneamente múltiples instancias de código, de forma segura, lo que implica aplicar una solución de bajo costo y efectiva para resolver este problema, ya que no es necesario tener un equipo físico por estudiante, sino proporcionarle un equipo virtual donde éste pueda realizar las actividades que desee sin necesidad de estar en el aula de clases.

Este equipo virtual, también llamado entorno de compilación y ejecución de código, debe permitir a sus usuarios compilar y probar sus programas desde cualquier lugar, requiriendo para esto una conexión a Internet que le provea el acceso remoto, sin importar las características y herramientas del equipo físico que esté utilizando.

El entorno de compilación y ejecución por ser un componente de un laboratorio virtual, se estructuró de tal forma que cumple las siguientes condiciones:

- **Seguridad**. Se procura garantizar la seguridad del equipo físico desde donde se accede, y a su vez se procura permitir que todo tipo de código de programación sea ejecutado sin la preocupación del daño que pueda ocasionar.

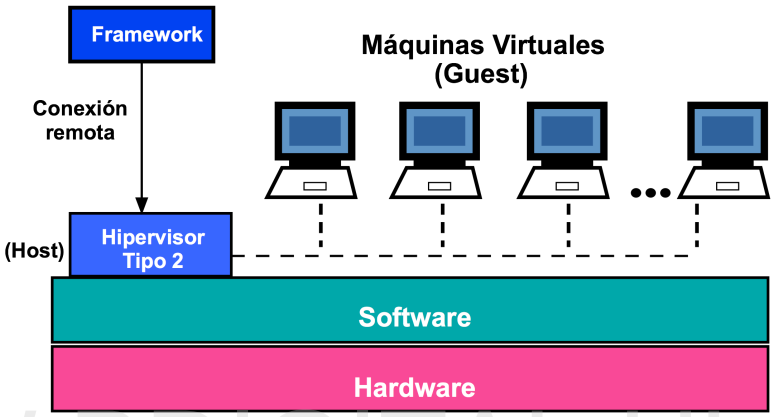
- **Aislamiento**. Cada entorno debe ser completamente independiente de otros. La ejecución de programas solo debe afectar el funcionamiento del entorno en donde se esté ejecutando. También cada uno debe estar aislado del *host* que lo ejecuta, para que pueda ser creado y eliminado cualquier cantidad de veces sin ningún inconveniente.

- **Eficacia**. Cada entorno deberá cumplir su función de compilar y ejecutar código fuente utilizando la menor cantidad de recursos posibles.

Este entorno se implementó a través Máquinas Virtuales ligeras que utilizan muy pocos recursos del computador físico, independientes entre si, controlables por el usuario (ver Fase de Eliminación) y seguras, que garantizan que el estado del equipo físico donde se ejecutan no sea afectado por códigos de programación mal diseñados o maliciosos. Para su construcción, se propuso la utilización de Máquinas Virtuales controladas por un Hipervisor Tipo 2, el cuál depende enteramente del Sistema Operativo que controla el *hardware* subyacente. Se utilizó este tipo de hipervisor porque es compatible con cualquier arquitectura de *hardware*, ya que no depende de la misma. También tiene un mejor soporte de controladores en *software (drivers)*, lo que permite aprovechar en su totalidad el Sistema Operativo donde se ejecuta. La función de este hipervisor es la de crear todas las Máquinas Virtuales que sean requeridas y hacer posible la Virtualización de Sistemas Operativos.

Se ejecutan Máquinas Virtuales de Sistema que emulan computadores físicos y que permiten a los usuarios realizar cualquier operación necesaria relacionada a la escritura y compilación de código. En la Figura 2.2 se ilustra el diseño del entorno.

Figura 2.2: Ilustración del diseño del Entorno de Compilación y Ejecución



El entorno está constituido en fases que representan su creación, funcionamiento y destrucción. Cada equipo virtual se crea solo una vez, y al cumplir con su objetivo, se destruye de forma permanente, sin posibilidad de ser recuperado.

2.1 Configuración Requerida para la Ejecución del Entorno

En necesario habilitar un ambiente de trabajo en donde se cree y ejecute el entorno.

1. Se ubica un equipo físico o virtual, con suficientes recursos para poder ejecutar el número requerido de Máquinas Virtuales [1].

2. Se instala el Hipervisor Tipo 2 en esta máquina física o virtual.

3. Se configura el entorno para que sea capaz de establecer comunicaciones vía comandos de consola.

[1] En las pruebas realizadas, con una máquina con 6GB de RAM, un CPU con 1.70GHz, se lograron levantar 16 Máquinas Virtuales.

2.2 Fases de la solución

2.2.1 Fase de Creación: Mecanismos de Generación de Máquinas Virtuales

- El Hipervisor Tipo 2 genera las Máquinas Virtuales por medio de comandos de consola.
- A través de un archivo de configuración de parámetros, se modifican los comandos de consola, y se especifican las características que se desea que tenga cada Máquina Virtual, tales como la cantidad de memoria RAM, el puerto de conexión asociado a la Máquina Virtual y la imagen de Sistema Operativo que va a ejecutar.
- Las Máquinas Virtuales pueden crearse y modificarse tantas veces como se requiera.
- Mediante un protocolo seguro de comunicación, el Sistema Operativo *host* se comunica con las Máquinas Virtuales existentes de forma remota, para que se puedan realizar operaciones y transferencia de archivos entre ambas partes.

2.2.2 Fase de Funcionamiento

El objetivo de este entorno es compilar y ejecutar código proporcionado por los usuarios. Una vez creadas las Máquinas Virtuales, y establecida la comunicación con el *host*, el entorno estará listo para ser usado. El funcionamiento está divido en dos procesos, compilación y ejecución, los cuales serán explicados a continuación:

Compilación de Código

La compilación del código se realiza en la Máquina Virtual, ya que es parte de las tareas de programación, y para que de esta manera el *hardware* no pueda ser afectado en el caso de que existan códigos maliciosos o mal diseñados, que puedan generar inconvenientes en el tiempo de compilación.

- Cada Máquina Virtual tiene incorporado un compilador necesario para ejecutar su tarea.

- Si el código compilado está libre de errores, tendrá una compilación exitosa y se pasará al proceso de ejecución.
- En caso de haber errores en el código, el compilador emitirá un mensaje que será notificado al usuario.

Ejecución de Código

La ejecución de código es una de las funciones más importantes a realizar en el entorno. Su éxito dependerá que se tenga previamente una compilación completa.
El diseño planteado supone interacción con otros sistemas y no con el usuario final, por ende, la evaluación de código debe diseñarse de tal forma que se adapte a esta condición.

Cualquier programa diseñado para que cumpla su funcionamiento de Entrada/Proceso/Salida (E/P/S) sin intervención humana, podrá ser ejecutado sin ningún inconveniente.

El entorno también provee la opción de ejecutar un código en donde la entrada sea un archivo en formato texto.

A continuación se lista la secuencia de instrucciones que describen el funcionamiento del entorno:

1. *Host* se comunica mediante un protocolo seguro con *guest*.
2. *Guest* recibe el programa proporcionado por usuario.
3. *Guest* compila programa.
4. *Guest* ejecuta programa.

2.2.3 Fase de Eliminación

Al culminar la Fase de Funcionamiento el entorno pasará a la Fase de Eliminación, en la cual, como su nombre lo indica, será eliminado permanentemente.

Las Máquinas Virtuales serán eliminadas si ocurren tres posibles escenarios:

1. Al terminar la ejecución de un programa.

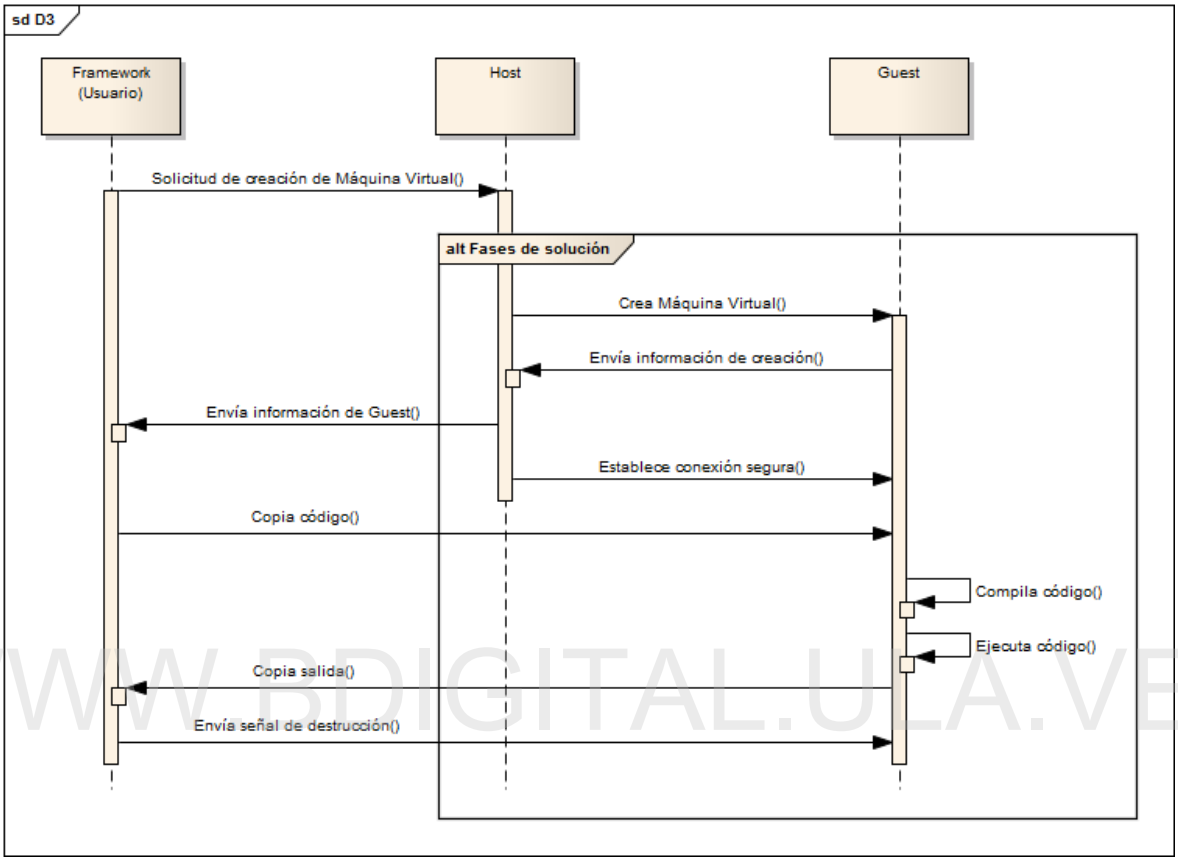
2. Cuando la compilación o ejecución de un programa dure un tiempo mayor de lo configurado.

3. De forma manual cuando el usuario lo desee.

Al ser eliminada la Máquina Virtual, el entorno devuelve al usuario un archivo en donde se indica el estado de la salida que ha producido el programa ejecutado. En este archivo se copia el resultado del código compilado y ejecutado. Si la compilación y ejecución son exitosas, en el archivo se puede ver la salida del programa. En caso de ocurrir un fallo en la compilación (errores de programación), se copia en el archivo el mensaje otorgado por el compilador.

La Figura 2.3 es un diagrama UML que representa cada una de las fases mencionadas. Se puede apreciar el proceso inicial de solicitud del usuario final (en este caso es una aplicación, no un humano) al *host*, la interacción del *host* con el *guest*, hasta su culminación con la eliminación de la Máquina Virtual.

Figura 2.3: Fases de la Solución



Las Figuras 2.4 y 2.5 ilustran los casos en que existen errores en los procesos de compilación y ejecución respectivamente. Se puede observar que al ocurrir uno de estos casos, la Máquina Virtual se destruye luego de informarle al usuario lo ocurrido.

Figura 2.4: Error en el Proceso de Compilación

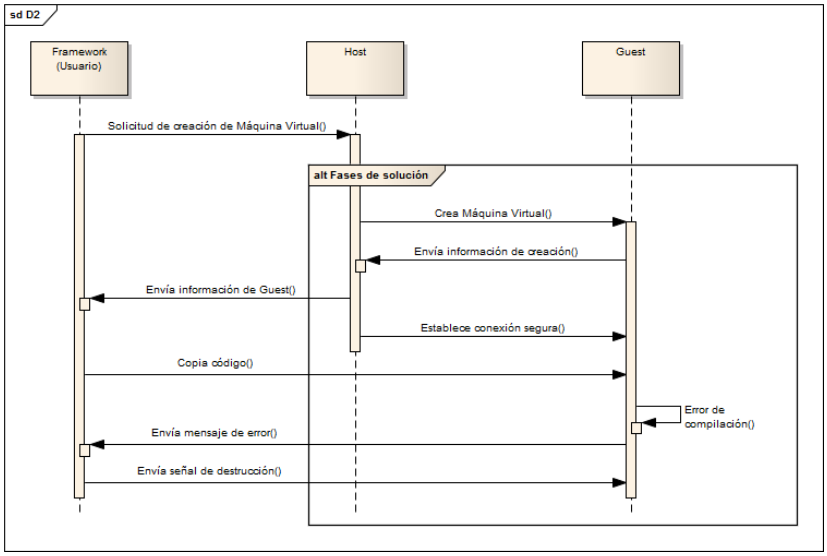
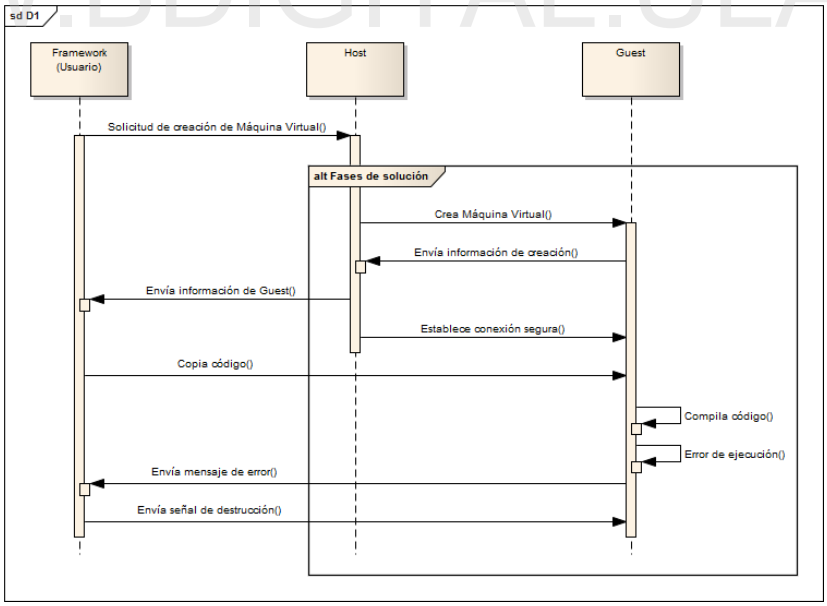


Figura 2.5: Error en el Proceso de Ejecución



Capítulo 3

Herramientas de Virtualización

3.1 Herramientas de Virtualización Analizadas

En este capítulo se muestran las herramientas tecnológicas que se seleccionaron para implementar la solución planteada en el capítulo anterior. Es importante volver a mencionar que esta solución requiere los siguientes componentes:

- Hipervisor Tipo 2 que permita Virtualización Completa.
- Imagen reducida de Sistema Operativo

Esta selección se realizó después de hacer una comparación entre las siguientes aplicaciones de virtualización. Esta lista se obtuvo al explorar diferentes sitios web[2] donde se indican su vigencia y uso actual.

3.1.1 Oracle VM VirtualBox

Es un *software* de virtualización para arquitecturas x86/AMD64, que permite instalar dentro del Sistema Operativo denominado *host* del *hardware* diversos Sistemas Operativos adicionales, también conocidos como *guest*, cada uno con su propio ambiente virtual. Ofrece Virtualización Completa, Paravirtualización y Virtualización Asistida

[2] Se consultaron los sitios web http://opensourceforu.com/2016/03/the-top-open-source-hypervisor-technologies/ y http://lifehacker.com/5714966/five-best-virtual-machine-applications

por Hardware, además también permite la ejecución de Máquinas Virtuales de forma remota, por medio del *Remote Desktop Protocol* (RDP).

Una de las principales ventajas de este *software* es que a pesar de ser una aplicación comercial, ofrece una versión *open source*, la cual dispone de una excelente documentación técnica y de usuario[vboa] y está bajo la licencia *General Public License 2*. Otra virtud de VirtualBox es el soporte de asignación de dispositivos USB 3.0 entre el *host* y el *guest*; también permite el uso de USB sobre RDP (USB over RDP) haciendo posible la conexión de dispositivos USB desde máquinas distintas al *host* utilizando un cliente RDP.

VirtualBox se puede ejecutar en los Sistemas Operativos *host* GNU/Linux, Mac OS X, OS/2 Warp, Microsoft Windows y Solaris/OpenSolaris. Dentro de los cuales es posible virtualizar los Sistemas Operativos FreeBSD, GNU/Linux, OpenBSD, OS/2 Warp, Windows, Solaris, MS-DOS y muchos otros [vbob].

3.1.2 Xen

Es un Hipervisor Tipo 1 o *baremetal open-source* para arquitecturas x86, el cual hace posible ejecutar muchas instancias de uno o varios Sistemas Operativos diferentes en paralelo en una sola máquina física. Xen proporciona aislamiento seguro, control de recursos, garantías de calidad de servicio y migración de Máquinas Virtuales en línea; y también ofrece Virtualización Completa y Paravirtualización. Se utiliza como base para diversas aplicaciones comerciales y de código abierto, tales como la Virtualización de Servidores, implementación de IaaS (Infraestructura como Servicio), Virtualización de Escritorios, aplicaciones de seguridad, aplicaciones incorporadas en *hardware*.

Además del aislamiento y seguridad que ofrece, Xen también brinda otras funciones bastante ventajosas, como la eliminación del tiempo de inactividad de una Máquina Virtual (al hacerle mantenimiento o alguna otra tarea), alta disponibilidad ya que reinicia las Máquinas Virtuales en el hipervisor o a nivel de servidor en casos de fallos. También aprovecha las características del *hardware* para consumir menos energía y procura optimizar la memoria compartiéndola entre las Máquinas Virtuales, si no está siendo usada por el servidor. Xen es un *software* de código abierto, publicado bajo los términos de la licencia *GNU General Public License* [xen].

3.1.3 Kernel-based Virtual Machine (KVM)

Es un hipervisor integrado en el kernel de Linux. Es una solución para realizar Virtualización Completa en Linux, ya que permite ejecutar Máquinas Virtuales utilizando imágenes de disco que contienen Sistemas Operativos sin modificar. Cada Máquina Virtual tiene su propio *hardware* virtualizado: una tarjeta de red, discos duros, tarjeta gráfica, etc. Es similar a Xen con respecto a su propósito, pero mucho más simple de utilizar. Contiene soporte de Paravirtualización y soporta el conjunto de instrucciones de virtualización en procesadores Intel VTx y AMD-V. La principal ventaja de esta herramienta es que al hacer que el propio kernel GNU/Linux actúe como hipervisor es capaz de repartir los recursos de la máquina entre varios procesos y así utilizar menor cantidad de CPU.

A diferencia de QEMU, el cual utiliza como estrategia la emulación, KVM se puede entender como un modo de operación especial del anterior, que utiliza extensiones de CPU (Hardware Virtual Machine) para la virtualización a través de un módulo de kernel [kvm].

3.1.4 QEMU

Es un emulador y virtualizador de máquina de código abierto. Cuando se utiliza como emulador de máquinas, puede ejecutar Sistemas Operativos y programas realizados para una máquina (por ejemplo, una tarjeta ARM) en una máquina diferente (su propia PC). Al utilizar la traducción dinámica, logra un mejor rendimiento. Cuando se utiliza como un virtualizador, logra un rendimiento casi nativo ejecutando el código del *guest* directamente en la CPU del *host*.

QEMU admite la virtualización cuando se ejecuta bajo el hipervisor Xen o usa el módulo kernel KVM en Linux. Permite realizar Virtualización Completa, Paravirtualización y Virtualización Asistida por Hardware; y cuando se utiliza con KVM, QEMU puede virtualizar arquitecturas x86, PowerPC, POWER de 64 bits, S390, ARM de 32 bits y 64 bits y *guest* MIPS.

También tiene la ventaja de soportar diversos tipos de imágenes (archivos que representan la data en el disco duro) lo que le permite ofrecer un mejor servicio y

lo hace compatible con otros *software*. Actualmente QEMU soporta estos tipos de imágenes:

- **raw** (crudo): el formato raw (formato por defecto) es una imagen binaria simple de la imagen del disco, y es muy portátil. En sistemas de archivos que admiten archivos dispersos, las imágenes en este formato sólo utilizan el espacio realmente utilizado por los datos registrados en ellos.
- **cloop**: formato Compressed Loop (bucle comprimido), utilizado principalmente para leer formatos de imagen de CD directo de Knoppix (distribución GNU/Linux) y similares.
- **cow**: formato copy-on-write (de copia en escritura), soportado sólo por razones históricas y no disponible para QEMU en Windows.
- **qcow**: es el antiguo formato QEMU de copia en escritura, soportado por razones históricas y reemplazado por qcow2.
- **qcow2**: formato QEMU de copia en escritura con una gama de características especiales, incluyendo la capacidad de realizar *snapshots*, imágenes más pequeñas en sistemas de archivos que no admiten archivos dispersos. Admite encriptación AES (opcional) y compresión zlib (opcional).
- **vmdk**: formato de imagen VMware 3 y 4 o 6 para intercambiar imágenes con ese producto.
- **vdi**: formato de imagen compatible con VirtualBox 1.1, para intercambiar imágenes con VirtualBox.
- **vhdx**: formato de imagen compatible con Hyper-V, para intercambiar imágenes con Hyper-V 2012 o posterior.
- **vpc**: formato de imagen legado Hyper-V, para intercambiar imágenes con Virtual PC / Virtual Server / Hyper-V 2008.

QEMU es miembro de *Software Freedom Conservancy* [qem].

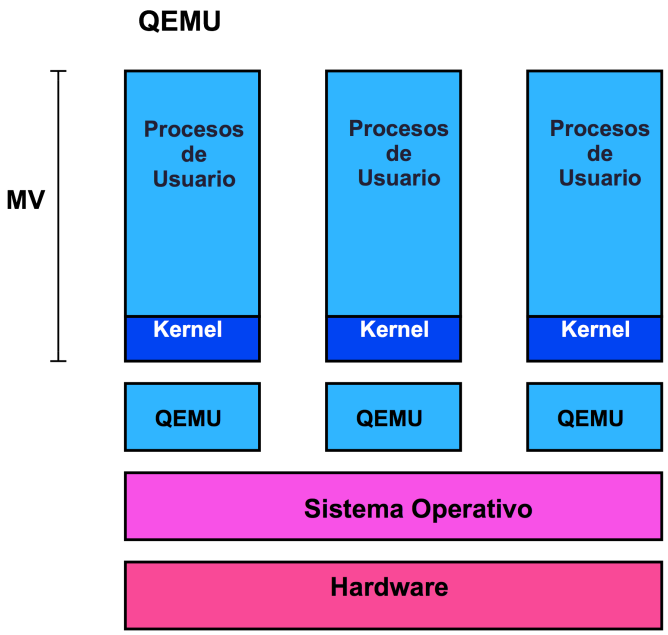
3.2 Herramientas de Virtualización Seleccionadas

Con el propósito de virtualizar los Sistemas Operativos y construir un entorno seguro para ejecutar y compilar código fuente para la construcción de un laboratorio virtual para las asignaturas de Ingeniería de Sistemas de la Universidad de Los Andes, se decidió utilizar la herramienta QEMU con el módulo KVM. QEMU es una aplicación que se ejecuta directamente sobre el Sistema Operativo del *hardware* y crea una Máquina Virtual totalmente independiente con su propio kernel.

Al usar QEMU con el módulo de virtualización KVM, el cual se ubica directamente sobre el kernel Linux, se transforma el kernel Linux en un hipervisor en donde se crean Máquinas Virtuales independientes. Al utilizar el módulo KVM la velocidad de arranque mejora significativamente, ya que usa el kernel Linux y no tiene que crear un kernel para cada Máquina Virtual.

Tal como se muestra en la Figura 3.1, se observa como QEMU crea Máquinas Virtuales independientes con su propio kernel.

Figura 3.1: Arquitectura de QEMU



Estas herramientas fueron seleccionadas luego de comparar varias aplicaciones de virtualización: Oracle VM VirtualBox, Xen y QEMU/KVM.

El diseño propuesto del entorno de ejecución y compilación de código requiere que esta solución pueda ser aplicada en un Sistema Operativo Linux, en donde el *hardware* puede variar.

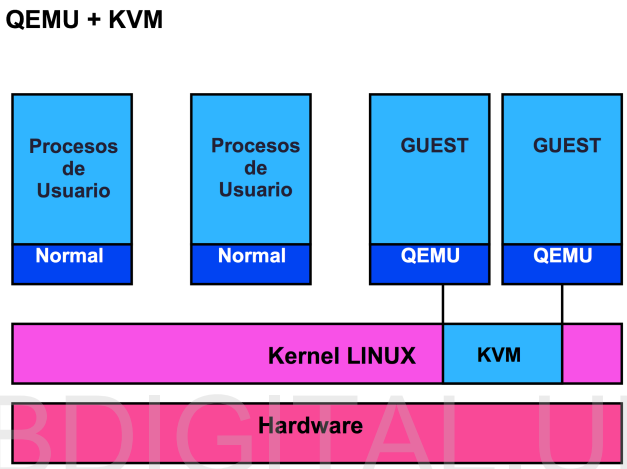
Xen fue automáticamente descartado por ser un Hipervisor Tipo 1 lo que implica que depende netamente de la arquitectura del *hardware*. Este tipo de hipervisor solo puede ser ejecutado en arquitecturas específicas, como condición para este proyecto, el *hardware* subyacente puede variar, y esto implica que Xen no garantiza que pueda ser instalado correctamente.

QEMU/KVM y Oracle VM VirtualBox son Hipervisores Tipo 2 y pueden ser ejecutados con múltiples Sistemas Operativos *host*. Ambas herramientas permiten realizar Virtualización Completa, Paravirtualización y Virtualización Asistida por Hardware, así como también permiten hacer *snapshots* (guardar el estado actual de una Máquina Virtual en un archivo de respaldo). QEMU resultó ser más ventajoso frente a la otra opción, por ser más liviano y requerir menos recursos del Sistema Operativo *host*. Además, ofrece un modo de interacción vía consola, lo cual permite que las Máquinas Virtuales sean generadas utilizando comandos por medio de los cuales el usuario puede especificar exactamente las características que requiera. Por el contrario, Oracle VM VirtualBox agrega funcionalidades por defecto que la mayoría de las veces el usuario no desea tener o simplemente no necesita. Oracle VM VirtualBox ofrece una versión no gratuita y otra que sí lo es pero con ciertas limitaciones en comparación a la anterior.

QEMU es *open source*, lo que permite reducir los gastos de adquisición. Al ejecutar QEMU con el módulo KVM se obtiene un rendimiento casi nativo, función que no provee Oracle VM VirtualBox. Por tener este comportamiento y consumir menos recursos del *hardware*, QEMU es capaz de crear más instancias de Sistemas Operativos en el *host* que Oracle VM VirtualBox.

En la Figura 3.2 se muestra cómo el módulo de virtualización KVM se integra en el kernel Linux, transformándolo en el hipervisor que crea múltiples Máquinas Virtuales independientes. Se observa cómo el kernel Linux es ahora el kernel de las Máquinas Virtuales.

Figura 3.2: Arquitectura de QEMU+KVM



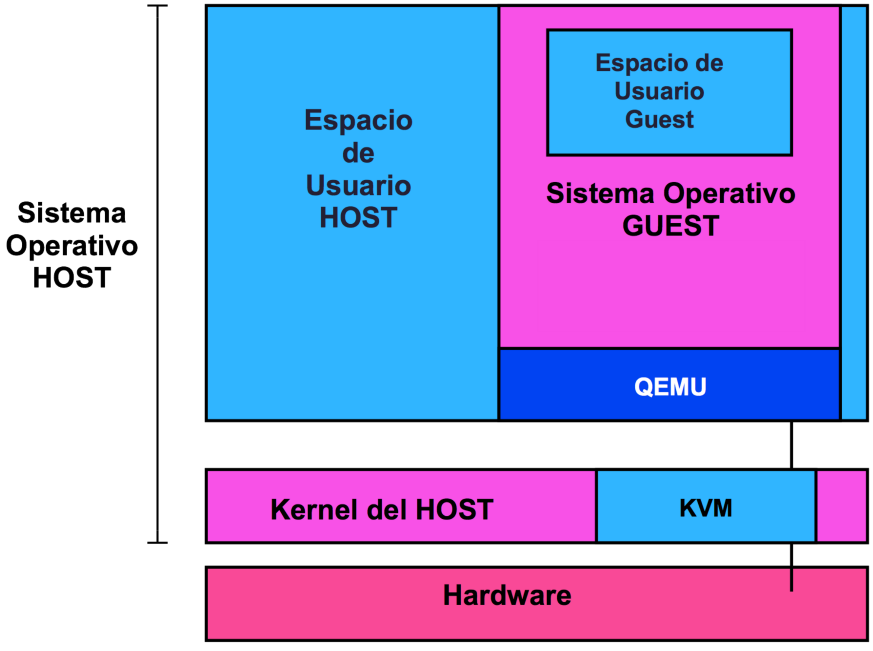
Capítulo 4

Implementación

En este capítulo se explica cómo se implementó la solución a través de las herramientas señaladas en el capítulo anterior. El procedimiento mostrado tiene el objetivo de fungir como manual para poder repetir la experiencia desde su inicio.

Según se muestra en la Figura 4.1, se pueden observar los componentes de la solución:

Figura 4.1: Esquema de solución para construcción del Entorno Virtual.



Esta configuración cumple con el requisito de poder crear entornos de compilación y ejecución de código fuente, ya que cumple con las especificaciones planteadas:

- Utiliza un Hipervisor Tipo 2 que permite Virtualización Completa.
- Ejecuta una imagen de Sistema Operativo en cada Máquina Virtual.

Las características propias de la solución son:

- QEMU/KVM como herramienta de virtualización para la creación de múltiples instancias del Sistema Operativo Linux.
- Como Sistema Operativo que se ejecuta en la Máquina Virtual se utilizó Micro Core Linux, el cual por ser un Sistema Operativo *open source*, está disponible gratuitamente en su sitio web [tc].
- Este Sistema Operativo es liviano al requerir una menor cantidad RAM, ya que el usuario puede seleccionar los módulos que se cargan en el sistema.
- Es capaz de soportar aplicaciones adicionales de la elección de los usuarios cumpliendo con el propósito de mantener los requerimientos relativamente bajos.

Se diseñó una versión de Micro Core Linux con los módulos requeridos para este proyecto, para permitir la compilación y ejecución de códigos de programación. El diseño de la nueva versión está formado por la meta-extensión de compilación **Compiletc** que está compuesta de las siguientes dependencias: binutils, base-dev, bison, diffutils, file, findutils, flex, gawk, gcc, gperf, grep, m4, make, patch, pkg-config, sed. Esta extensión es la que hace posible la compilación de código en el entorno.
También está formado por la aplicación **OpenSSH**, que permite realizar comunicaciones cifradas a través de una red, usando el protocolo SSH, permitiendo la comunicación entre el Sistema Operativo *host* y *guest*.

El entorno está compuesto por una Máquina Virtual, la cual utiliza un conjunto de parámetros para su ejecución. Estos parámetros definen las características de una Máquina Virtual, tales como:

- Sistema Operativo que se ejecutará en la Máquina Virtual: Cada Máquina Virtual ejecutará una imagen de Sistema Operativo donde existen módulos que hacen posible la compilación y ejecución de código.
- Cantidad de RAM: Se asigna una cantidad suficiente de memoria principal del *host* para cargar los módulos del Sistema Operativo *guest*.
- Puerto de conexión: Se asigna un puerto de conexión específico para permitir la comunicación entre *host* y *guest*.

Una configuración adecuada de la Máquina Virtual asegura un buen funcionamiento del entorno, es decir, que compile y ejecute el código correctamente y que pueda comunicarse con el *host* sin inconvenientes.

4.1 Instalación y Configuración

4.1.1 Instalación del Software de Virtualización

El *software* utilizado para la construcción del entorno fue QEMU con el módulo de virtualización KVM. Se obtuvo el paquete QEMU[3] en la versión 2.9.0-1 (versión estable) del repositorio oficial de Linux Arch. Se utilizó Linux Arch ya que como cualquier Sistema Operativo Linux tiene lo requerido para realizar este proyecto, y estaba instalado en la máquina de pruebas. La descarga se hizo utilizando el gestor de paquetes pacman, y con los siguientes parámetros:

```
pacman -S qemu
```

[3]QEMU headless para servidores

4.1.2 Creación y Configuración del Sistema Operativo de la Máquina Virtual

Cada Máquina Virtual necesita un Sistema Operativo para ser ejecutada. Para ello, se seleccionó y modificó Micro Core Linux. Para modificar Micro Core Linux se realizó lo siguiente:

1. Se crea una Máquina Virtual con el Sistema Operativo Tiny Core Linux (versión con interfaz gráfica de Micro Core Linux). Esto es necesario porque Tiny Core ofrece aplicaciones que permite remasterizar su propia Imagen de Sistema Operativo.

2. Se descarga el paquete **Ezremaster**. [4]

3. Se descarga la imagen de Micro Core Linux desde su sitio web.

4. Se inicia **Ezremaster** y se siguen los siguientes pasos:

   (a) Se selecciona "Use ISO image", para indicar la Imagen de Sistema Operativo que se va a crear.

   (b) En la primera barra se debe hacer click para seleccionar la ruta, que en este caso fue /home/tc/Core-current.iso.

   (c) Se vuelve a hacer click en OK para continuar con el proceso.

   (d) La barra de "Add Boot Codes" se deja vacía y hace click en "Next" en esa ventana y en la siguiente.

   (e) Se selecciona "Connect to App Repo", para que se carguen todos los paquetes disponibles de Tiny Core Linux que pueden ser descargados.

   (f) Se elige "compiletc.tcz", y se hace click en "Add inside initrd on Boot".

   Se utiliza la opción "Add inside initrd on Boot" para obtener como resultado un sistema que arranque completo desde la memoria.

[4] Ezremaster es una aplicación con GUI (Interfaz Gráfica de Usuario) que simplifica remasterizar Tiny Core o Micro Core Linux.

(g) Se selecciona “Connect to App Repo”, y se elige “openssh.tcz”, para después hacer click en “Add inside initrd on Boot”.

(h) Se espera que los dos paquetes aparezcan en la ventana de “Add inside initrd on Boot” y después se hace click en “Next”.

(i) Se hace click en “Create ISO” y se espera a que termine el proceso.

4.1.3 Configuración de Servidor SSH

Para la comunicación entre el *host* y el *guest* se utilizó el protocolo SSH y es necesario configurar el Sistema Operativo *guest* como un servidor SSH con la finalidad de transferir los archivos y ejecutar los comandos de forma remota en el *guest*.
Para permitir que el *host* se comunique con la Máquina Virtual que contiene la imagen Micro Core Linux, se debe modificar esta imagen siguiendo los pasos siguientes:

1. Se inicia el OpenSSH con el comando:

   ```
   sudo /usr/local/etc/init.d/openssh start
   ```

2. Se hace una copia de la versión original del archivo isolinux.cfg, y se cambia el tiempo de arranque de 300 (30 s) a 10 (1 s), a través de los siguientes comandos:

   ```
   sudo cp /tmp/isoTC/image/boot/isolinux/isolinux.cfg /tmp/isoTC/image/boot/
   isolinux/isolinux.cfg.backup

   sudo sed -i 's/timeout 300/timeout 10/' /tmp/isoTC/image/boot/
   isolinux/isolinux.cfg
   ```

   El tiempo de arranque se modifica para acelerar el inicio de las Máquinas Virtuales que ejecutan este Sistema Operativo.

3. Se copia la clave SSH generada al instalar la aplicación SSH en Micro Core Linux. En general esto no es necesario, pero sino se hace de esta forma, cada vez que arranque el Sistema Operativo del *host*, se genera una clave nueva.

```
sudo cp -f /usr/local/etc/ssh/ssh_host_*
 /tmp/isoTC/extract/usr/local/etc/ssh
```

4. Se cambia la configuración del servidor SSH, llevando a cabo los siguientes pasos:

   - Se habilita el proceso de autenticación del SSH, y se utiliza el usuario root:

     ```
     sudo sed -i 's/#PermitRootLogin/PermitRootLogin/'
     /tmp/isoTC/extract/usr/local/etc/ssh/sshd_config
     ```

   - Se habilita la conexión a través de puertos (TCP)

     ```
     sudo sed -i 's/#GatewayPorts no/GatewayPorts yes/'
     /tmp/isoTC/extract/usr/local/etc/ssh/sshd_config
     ```

5. Se debe asegurar que las claves SSH tengan los permisos correctos.

   - Se asegura que el archivo ssh_host* pertenezca al usuario root

     ```
     sudo chown root /tmp/isoTC/extract/usr/local/etc/ssh/ssh_host*
     ```

   - Se cambian los permisos de lectura y escritura de los archivos ssh_host*pub y ssh_host*key:

     ```
     sudo chmod 644 /tmp/isoTC/extract/usr/local/etc/ssh/ssh_host*pub
     sudo chmod 600 /tmp/isoTC/extract/usr/local/etc/ssh/ssh_host*key
     ```

6. Se configura el sistema para que inicie el servidor SSH desde su arranque.

   ```
   sudo cp /tmp/isoTC/extract/opt/bootlocal.sh /tmp/
   isoTC/extract/opt/bootlocal.sh.backup

   sudo echo "/usr/local/etc/init.d/openssh
   start" >> /tmp/isoTC/extract/opt/bootlocal.sh
   ```

7. Micro Core Linux genera dos usuarios por defecto, tc y root. Se sugiere que se cambien sus contraseñas para mayor seguridad y facilidad de acceso.

- Se le asigna la contraseña al usuario tc.

  ```
  passwd tc
  ```

- Se cambia la contraseña del usuario root.

  ```
  sudo passwd root
  ```

8. Se hacen copias de los archivos /etc/shadow y /etc/passwd (los cuales contienen las nuevas contraseñas de tc y root).

   ```
   sudo cp -f /etc/shadow /tmp/isoTC/extract/etc/shadow
   sudo cp -f /etc/passwd /tmp/isoTC/extract/etc/passwd
   ```

4.2 Ejecución

Para automatizar[5] la creación y destrucción de una Máquina Virtual y a la vez la compilación y ejecución de código dentro de ella, se creó un *script*[6] que ejecuta todos estos pasos y agiliza el proceso al no requerir la intervención humana.

4.2.1 Montaje de una Máquina Virtual con un Sistema Operativo específico usando QEMU

Para crear una Máquina Virtual con QEMU que ejecute el Sistema Operativo Micro Core Linux previamente modificado, es necesario configurar QEMU de la siguiente manera:

```
qemu-system-x86_64 -enable-kvm -boot d -cdrom image.iso -m 320
-net user,hostfwd=tcp::60000-:22 -net nic &
```

- **qemu-system-x86_64**: Comando de consola para ejecutar QEMU.
- **-enable-kvm**: Habilita el módulo de virtualización KVM.

[5]Este producto se desarrolló en un Sistema Operativo Linux Arch. Todos los paquetes utilizados y los pasos de instalación y ejecución están basados en esta distribución.

[6]El *script* fue escrito en el lenguaje de programación Python3, pero puede ser escrito en cualquier otro lenguaje.

- **-boot d -cdrom image.iso**: Inicia la Máquina Virtual usando la ISO modificada.
- **-m 320**: Asigna a la Máquina Virtual 320MB de RAM (del *host*).

  Se asigna esta cantidad de memoria después de realizar pruebas y concluir que 320MB de RAM es la mínima cantidad que permite ejecutar correctamente las aplicaciones instaladas previamente.
- **-net user,hostfwd=tcp::60000-:22 -net nic**: Mapea el puerto 60000 del *host* al puerto 22 del *guest*.

Se utilizan puertos a partir del 60000 porque están en el rango de los puertos dinámicos o privados, los cuales son usados para servicios personalizados o para fines temporales. Cada Máquina Virtual utiliza un puerto diferente dentro de este rango.

Automatización del Proceso de Creación del entorno de compilación y ejecución

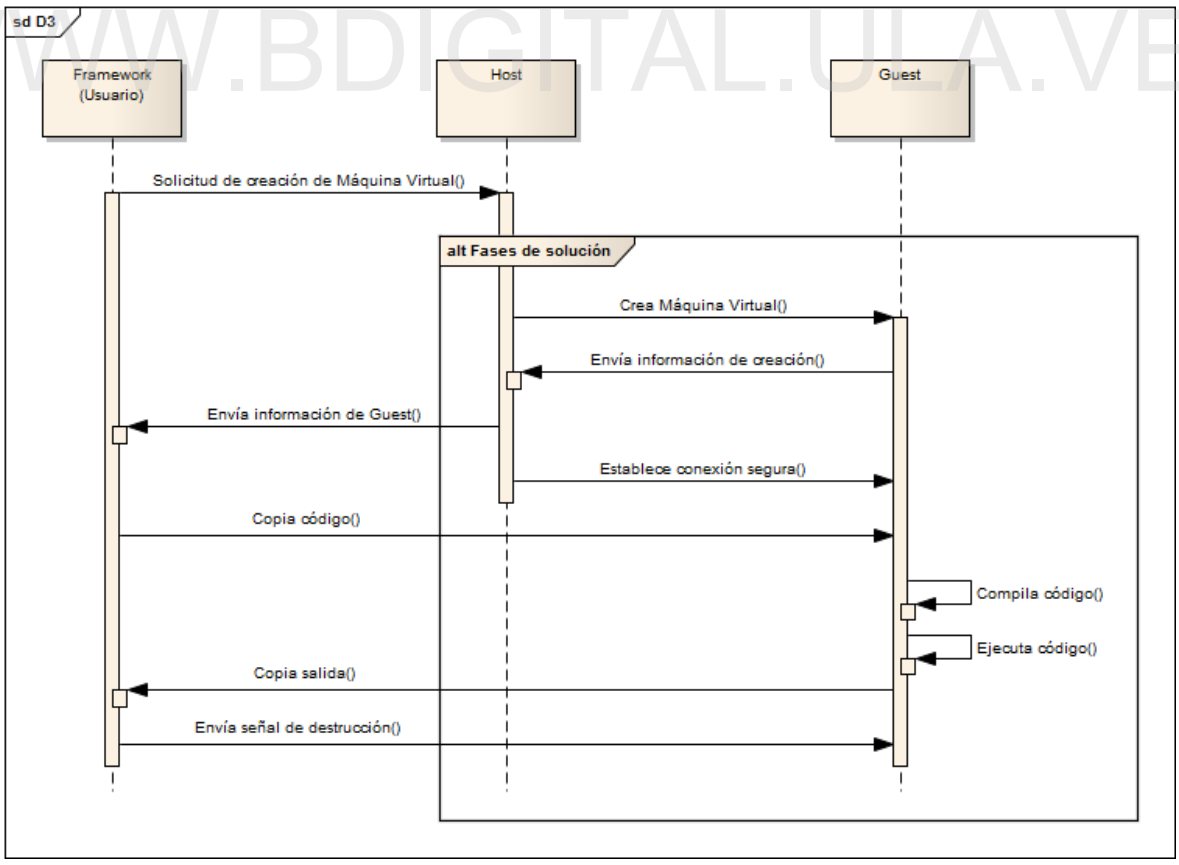
El Sistema Operativo *host* utiliza un puerto TCP (Protocolo de Control de Transmisión) diferente para comunicarse con cada entorno virtual de forma independiente. En un escenario ideal es necesario tener más de un entorno a la vez, por lo que se requiere llevar un control de los puertos asociados a cada uno. Para este fin, se creó un archivo que registra la cantidad de Máquinas Virtuales creadas con sus respectivas características. Este archivo garantiza que no hayan inconvenientes en el momento de crear más de una Máquina Virtual porque permite revisar los puertos que están ocupados para poder asignarles a las Máquinas Virtuales que se crean puertos desocupados.

Las instrucciones del *script* que automatiza el proceso de creación y destrucción de Máquinas Virtuales junto con la compilación y ejecución de códigos de programación dentro de ellas, son las siguientes y se ilustran en el diagrama (ver Figura 4.2):

1. Se verifica en el archivo de control de puertos de conexión si es posible crear la Máquina Virtual.

2. Se crea la Máquina Virtual invocando el comando de QEMU.
3. Se copia el programa y los archivos necesarios en el *guest*.
4. Se compila el programa en la Máquina Virtual.
5. Se ejecuta el programa con los archivos de entrada previamente copiados en la Máquina Virtual.
6. Se guarda la salida del programa en la Máquina Virtual.
7. Se devuelve la salida del programa al usuario.
8. Se destruye la Máquina Virtual.

Figura 4.2: Ilustración de las instrucciones del script de automatización



4.3 Pruebas Realizadas

Durante la construcción del entorno de compilación y ejecución de código fuente, se llevaron a cabo múltiples pruebas en donde se pudo comprobar el correcto funcionamiento del entorno y la cantidad de Máquinas Virtuales simultáneas que se ejecutaban.

En un *hardware* con 6GB de RAM y un CPU de 1.70GHz, se lograron levantar 16 Máquinas Virtuales a la vez. Cada una de estas MV compilaba y ejecutaba un código de programación escrito en lenguaje C.

Los programas escritos en C consistían en leer datos de un archivo en formato texto y utilizarlos para realizar una tarea simple, como dibujar figuras geométricas o resolver una serie de Fibonacci.

También se probó el funcionamiento del entorno con programas que tenían errores de sintaxis y de diseño, como la ausencia de delimitadores y la presencia de ciclos infinitos respectivamente. Con este tipo de programas se verificó la respuesta del entorno ante situaciones no ideales y se comprobó que los mensajes de error emitidos por el compilador si estaban siendo transmitidos al usuario.

Mediante estas pruebas también se confirmó que el entorno puede ser eliminado de manera manual cuando el usuario lo desee, y que si la ejecución de un programa dura más del tiempo de vida configurado para la Máquina Virtual, el entorno se destruirá aunque el proceso de ejecución no haya finalizado.

Capítulo 5

Conclusiones y Recomendaciones

5.1 Conclusiones

Se implementó un entorno virtual utilizando una herramienta que crea simultáneamente múltiples Máquinas Virtuales. La selección de esa herramienta se realizó después de comparar distintos *software* de virtualización y elegir el que permitiese crear un mayor número de Máquinas Virtuales. Estas Máquinas Virtuales son creadas con características específicas que permiten compilar y ejecutar código fuente escrito en un determinado lenguaje de programación, utilizando una cantidad reducida de recursos físicos. El entorno está diseñado para comunicarse a través del protocolo SSH con el Sistema Operativo *host* en donde se ejecuta, lo que hace que funcione sin que el usuario se percate de su funcionamiento.

Las Máquinas Virtuales se implementaron con un Sistema Operativo especialmente diseñado para ellas, en donde tiene instalado por defecto varios compiladores y una aplicación que hace posible la comunicación mediante el protocolo SSH. Se pudo comprobar a través de múltiples pruebas que el entorno funciona según lo establecido: creando el mayor número de Máquinas Virtuales con un Sistema Operativo específico, creando satisfactoriamente una comunicación con el Sistema Operativo del *hardware* subyacente, necesaria para la transferencia de los programas a compilar y ejecutar, y eliminando correctamente las Máquinas Virtuales al finalizar la ejecución del código.

Las pruebas indicaron que el número posible de instancias virtuales simultáneas

depende solamente de la capacidad del *hardware* disponible, y no de la solución virtual propuesta.

5.2 Recomendaciones

Se puede mejorar esta implementación probando otros Sistemas Operativos *open source* que sean aún más livianos que el utilizado durante la ejecución de este proyecto. Es importante resaltar que para este proyecto el Sistema Operativo utilizado fue el más liviano que se encontró.

Se sugiere implementar un método de gestión de peticiones (manejo de colas) que permita manejar las peticiones que no pueden ser atendidas cuando se rechacen por no disponer de los recursos de *hardware* suficientes.

Se recomienda seguir utilizando herramientas *open source* y Sistemas Operativos Linux, ya que los mismos cuentan con características que hacen factible su implementación en instituciones como la Universidad de Los Andes: bajo costo, buena documentación, actualizaciones constantes, y soporte a través de las comunidades de *software* libre.

Bibliografía

[And10] Benjamin Anderson. Xen worlds: Creating a virtual laboratory environment for use in education. Master's thesis, Iowa University, 2010.

[Bis12] Karan Bisht. System virtual machine vs. process virtual machine. http://androiddeveloperindelhi.blogspot.com/2012/10/system-virtual-machine-vs-process.html, 2012. Accedido 17-02-2017.

[DW08] Wenliang Du and Ronghua Wang. Seed: A suite of instructional laboratories for computer security education. *Journal on Educational Resources in Computing (JERIC)*, 8(3):1–24, March 2008.

[ETV08] Jae Lee Judy McKay Efraim Turbam, Dave King and Dennis Viehland. *Electronic Commerce A Managerial Perspective*. Prentice Hall, Upper Saddle River, New Jersey, 2008.

[Gra11] Charles Graziano. A performance analysis of xen and kvm hypervisors for hosting the xen worlds project. Master's thesis, Iowa University, 2011.

[KKS05] Pratik Rana Tianning Li Kumar Krishna, Weiqing Sun and R. Sekar. V-netlab: A cost-effective platform to support course projects in computer security. In *CISSE*, 2005.

[kvm] Kernel virtual machine official site. https://www.linux-kvm.org/page/Main_Page. Accedido 02-03-2017.

[LS06] Edith Larson and William Stackpole. Does a virtual networking laboratory

result in similar student achievement and satisfaction? In *SIGITE*, pages 105–114, New York, New York, 2006.

[MBK10] Juri Papay Stephen Phillips Arturo Servin Xiaoyu Yang Zlatko Zlatev Spyridon Gogouvitis Gregory Katsaros Kleopatra Konstanteli George Kousiouris Andreas Menychtas Michael Boniface, Bassem Nasser and Dimosthenis Kyriazis. Platform-as-a-service architecture for real-time quality of service management in clouds. In *Internet and Web Applications and Services (ICIW)*, Barcelona, Spain, May 2010.

[PG74] Gerald Popek and Robert Goldberg. Formal requirements for virtualizable third generation architectures. http://dl.acm.org/citation.cfm?doid=361011.361073, 1974. Accedido 04-02-2017.

[Por16] Matthew Portnoy. *Virtualization Essentials*. Sybex, Indianapolis, Indiana, second edition, 2016.

[qem] Qemu official site. http://www.qemu.org. Accedido 02-03-2017.

[RU05] Dion Rodgers Amy Santoni Fernando Martins Andrew Anderson Steven Bennett Alain Kägi Felix Leung Larry Smith Rich Uhlig, Gil Neiger. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005.

[Sin04] Amit Singh. An introduction to virtualization. urlhttp://www.kernelthread.com/ publications/virtualization/, January 2004. Accedido 04-02-2017.

[Sos11] Barrie Sosinsky. *Cloud Computing Bible*. Wiley Publishing, Indianapolis, Indiana, 2011.

[tc] Tiny core linux official site. http://distro.ibiblio.org/tinycorelinux/downloads.html. Accedido 02-03-2017.

[TEP13] Zaigham Mahmood Thomas Erl and Ricardo Putini. *Cloud Computing Concepts, Technology & Architecture*. Prentice Hall, Westford, Massachussets, 2013.

[vboa] Oracle vm virtualbox official documentation. https://www.virtualbox.org/wiki/Documentation. Accedido 02-03-2017.

[vbob] Oracle vm virtualbox official site. https://www.virtualbox.org. Accedido 02-03-2017.

[VMw08] VMware Inc. *Understanding full virtualization, paravirtualization, and hardware assist.*, 2008.

[WAR07a] Alessio Gaspar William Armitage and Matthew Rideout. Remotely accessible sandboxed environment with application to a laboratory course in networking. In *SIGITE*, pages 83–90, New York, New York, 2007.

[WAR07b] Alessio Gaspar William Armitage and Matthew Rideout. A uml and mln based approach to implementing a networking laboratory on a scalable linux cluster. *Journal of Computing Sciences in Colleges*, 23(2):112–119, December 2007.

[WDW07] Zhouxuan Teng Wenliang Du and Ronghua Wang. Seed: a suite of instructional laboratories for computer security education. In *SIGCSE*, pages 486–490, New York, New York, 2007.

[WSS08] Kumar Krishna Weiqing Sun, Varun Katta and R. Sekar. V-netlab: an approach for realizing logically isolated networks for security experiments. In *CSET*, pages 1–6, Berkeley, California, 2008.

[WV11] David Walden and Tom Van Vleck. The compatible time sharing system (1961-1973).fiftieth anniversary commemorative overview. urlhttp://www.computer.org/portal/web/csh, 2011. Accedido 15-02-2017.

[xen] The xen project official site. https://www.xenproject.org. Accedido 02-03-2017.

Glosario

host Software o Hardware donde se ejecuta una Máquina Virtual.. 5

guest Sistema Operativo que se ejecuta en una Máquina Virtual.. 5

cliente ligero Computador o Software cliente en una arquitectura cliente-servidor diseñado para tener acceso remoto de un servidor (normalmente ambientes de virtualización). Depende al 100% del servidor central para realizar sus funciones computacionales. Su tarea principal es el transporte de datos entrada y salida entre el usuario y el servidor remoto.. 7

clustering Técnica de agrupamiento que forma un cluster.. 9

overhead Tiempo de procesamiento empleado por el software del sistema. 10

intérprete Programa que permite que un programa fuente escrito en un determinado lenguaje vaya traduciéndose y ejecutándose directamente, por el computador, sin generar ningún código equivalente.. 14

ISA (Infrastructure Set Architecture) Instrucciones que un CPU puede entender y ejecutar. Aspectos del procesador generalmente visibles para el programador.. 14

driver Software que permite al Sistema Operativo interactuar con los dispositivos conectados a la Unidad Central de Procesamiento de un computador, haciendo una abstracción del hardware y proporcionando una interfaz que permite usar el dispositivo.. 14

sandboxing Método computacional utilizado para ejecutar programas de manera segura y separada.. 15

gateway Dispositivo que actúa de interfaz de conexión entre aparatos o dispositivos, y también posibilita compartir recursos entre dos o más computadoras.. 23

cluster Conjunto de computadores unidos entre sí por una red de alta velocidad donde se comportan como si fuesen un solo computador.. 23

NFS (Network File System) Protocolo de nivel de aplicación utilizado para sistemas de archivos distribuidos en un entorno de red de computadores de área local.. 24

RDP (Remote Desktop Protocol) Protocolo propietario desarrollado por Microsoft que permite la comunicación en la ejecución de una aplicación entre un terminal y un servidor Windows.. 34