Sistemas Operativos

Exposición: Emulación

Grupo: 06

Alumno: Esquivel Santana Christian

Profesor: Ing. Gunnar Eyal Wolf Iszaevich

Fecha de exposición: 01/10/2024

(01 de Octubre de 2024)

Índice

1. Introducción

La emulación es un concepto fundamental en el ámbito de la informática, que permite replicar el comportamiento de un sistema de hardware o software en otro distinto. A través de la emulación, es posible que una computadora o dispositivo ejecute programas o simule el funcionamiento de sistemas que, en teoría, no están diseñados para él. Este proceso se logra mediante la traducción de instrucciones y comportamientos de un sistema emulado a la arquitectura anfitriona, abriendo una amplia gama de posibilidades tanto en el ámbito del desarrollo de software como en la conservación digital.

El propósito de este trabajo es proporcionar una comprensión clara y concisa de qué es la emulación, cómo funciona, y qué papel juega en la tecnología moderna. Además, se explorarán las diferencias entre emulación y otras tecnologías relacionadas, como la virtualización, y se analizarán los retos asociados con la emulación de sistemas completos.

La importancia de la emulación en nuestra vida cotidiana no puede subestimarse. Desde permitir la ejecución de software antiguo en hardware moderno, hasta la posibilidad de desarrollar y probar aplicaciones en múltiples plataformas sin necesidad de contar con hardware especializado, la emulación desempeña un papel clave en la innovación tecnológica. Asimismo, es crucial en la preservación digital, facilitando el acceso a sistemas antiguos que de otro modo quedarían obsoletos.

2. Emulación vs Simulación

A menudo se cree que ambos términos significan lo mismo, ya sea por su similitud en la pronunciación o porque, en términos generales, su funcionamiento parece parecido. Ambos implican que un sistema imita las operaciones de otro. Sin embargo, al profundizar en los significados de cada uno, es cuando descubrimos que existe una diferencia en cuanto a su propósito y forma de operar.

2.1. Emulación

Aunque ya se mencionó en la introducción qué es la emulación, es importante ofrecer otra definición para una comprensión más clara del tema. La emulación es un método que permite a una plataforma host ejecutar software o utilizar hardware periférico que fue originalmente diseñado para otro sistema. Su objetivo principal es reemplazar componentes de hardware o software y simular las funcionalidades del dispositivo original.

2.1.1. Ventajas

El uso de emuladores permite ahorrar espacio al eliminar la necesidad de hardware adicional. Además, contribuye a la preservación tanto del hardware como del software antiguos, ya que permite ejecutar sistemas más antiguos en hardware moderno, asegurando así su funcionamiento en la actualidad.

2.1.2. Desventajas

La depuración en los emuladores es especialmente compleja, ya que, a diferencia de otras técnicas, estos intentan replicar un sistema completo. Como resultado, el proceso de depuración puede ser bastante complicado y, en consecuencia, el rendimiento tiende a ser más lento.

2.2. Simulación

La simulación es un proceso mediante el cual se crea un modelo ideal de un sistema con el objetivo de analizar su comportamiento bajo condiciones específicas. Para lograrlo, es necesario utilizar fórmulas matemáticas y algoritmos que permitan recrear con precisión dicho modelo.

2.2.1. Ventajas

En las áreas donde se utiliza habitualmente, la principal ventaja es proporcionar mayor seguridad, evitando así riesgos en situaciones de la vida real. Además, permite un importante ahorro

económico, ya que no es necesario gastar recursos que, en la práctica, podrían resultar muy costosos.

2.2.2. Desventajas

Una de las mayores desventajas a considerar es la gran capacidad de procesamiento informático que se requerirá. Además, las simulaciones no reflejarán situaciones reales de manera exacta, por lo que no deberían tomarse como la única fuente para analizar un modelo en su totalidad..

2.3. Ejemplo

Una vez explicados los conceptos, es posible que aún queden dudas sobre la diferencia entre ambos. Por eso, un ejemplo de cada uno puede ayudar a aclararlos mejor.

Como mencionamos anteriormente, la emulación consiste en intentar reproducir la experiencia de un sistema en otro. Un ejemplo claro de esto se encuentra en los videojuegos, donde descargamos un emulador de una consola en nuestra computadora para replicar exactamente el funcionamiento del sistema original y poder jugar los juegos de esa plataforma.

Por otro lado, la simulación se refiere al modelado de un sistema. Un ejemplo evidente son los simuladores de aviación, donde un programa imita lo más fielmente posible el comportamiento de un avión en la realidad, permitiendo a los usuarios experimentar condiciones similares a las que enfrentarían en un vuelo real.

3. Estructura básica de un emulador

Hoy en día, todas las computadoras están basadas en la arquitectura de von Neumann, la cual se fundamenta en un bus al que están conectados la CPU, la memoria y los dispositivos de entrada/salida.

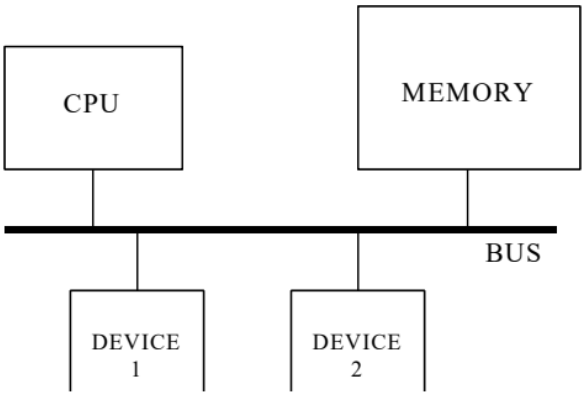

Figura 1: Arquitectura von Neuman tomado del libro Study of the techniques for emulation programming

La CPU es el núcleo de la computadora y, por lo tanto, en un emulador, la emulación de la CPU será también el núcleo del emulador. La simulación de la CPU es una de las tareas más importantes, y en la mayoría de los casos será la parte donde necesitaremos mayor potencia de la CPU objetivo. Por lo tanto, optimizar su emulación se vuelve fundamental.

Es por eso que los aspectos de la CPU que se emularán se centran principalmente en el cálculo que se ejecuta y su forma de funcionamiento. En la mayoría de los casos, solo queremos emular la ejecución de las instrucciones de programa por parte de la CPU.

3.1. Intérprete

Un intérprete es la forma más sencilla de emular una CPU. Solo reproduce cómo funciona una CPU básica, la cual lee un byte (o el número de bytes que forman una instrucción de la CPU emulada) de la dirección de la memoria apuntada por un registro especial. Luego, decide la operación a realizar y, finalmente, ejecuta las funciones correspondientes a esa instrucción. Una de las implementaciones más fáciles es utilizar una instrucción switch/case para cada uno de los diferentes bytes que definen

una instrucción.

```
switch(memory[PC++])
{
        case OPCODE1:
                opcode1();
                break;
        case OPCODE2:
                opcode2();
                break;

        ....

        case OPCODEn:
                opcodeN();
                break;
}
```

Figura 2: Estructura de un intérprete usando switch/case tomado del libro Study of the techniques for emulation programming

Ahora bien, ¿cuál es el mayor problema de esta forma de emulación? Principalmente, es el tiempo; el intérprete es lento en comparación con otras formas de emulación. La sobrecarga se debe a la programación de la decodificación y las instrucciones. Por ejemplo, los cálculos con banderas son muy costosos y, por lo tanto, es difícil aprovechar al máximo las capacidades de la CPU.

3.2. Traducción binaria

A diferencia del intérprete, que ejecuta instrucción por instrucción, la traducción binaria se encarga de traducir el código de una arquitectura al código de la arquitectura en la CPU de destino. ¿Cómo se realiza esto? La traducción binaria trabaja mediante bloques de código, y hay varias razones para hacerlo de esta manera. Muchas veces, no se conoce el código completo desde el inicio, y en

otras ocasiones, es necesario establecer puntos para detener la emulación y facilitar el proceso de traducción.

Existen dos formas de traducción binaria: estática y dinámica.

3.2.1. Estática

No es muy común utilizarla en los emuladores. Esto se debe a que, por lo general, los emuladores tienden a ser dinámicos, por lo que, en la mayoría de los casos, realizar este tipo de traducción es muy difícil de implementar e incluso imposible. La idea de traducir anticipadamente los miles de programas posibles que una computadora puede ejecutar representa un gran problema.

3.2.2. Dinámica

Es más similar a la idea principal de la traducción binaria y es de las más útiles para emular CPUs. El código se traduce en bloques mientras se está ejecutando y solo se traduce cuando realmente es necesario. Una vez que un bloque se traduce por primera vez, se guarda en un caché de traducción para poder ser utilizado en cualquier momento.

4. Arquitectura

Con base en lo anteriormente mencionado, hemos dicho que el objetivo de un emulador es la optimización de la CPU. En este sentido, las arquitecturas que serían más "fáciles"de emular son las RISC (como ARM o MIPS), a diferencia de las arquitecturas CISC (como x86). Esto se debe al conjunto de instrucciones más sencillo de las RISC, que consiste en un número reducido de instrucciones, lo que facilita tanto la traducción como la emulación, y, en consecuencia, hace que la emulación sea más estable. Sin embargo, esto no significa que sea imposible emular arquitecturas CISC, ya que, debido a su gran complejidad en las instrucciones, el método ideal será la traducción binaria dinámica.

5. Chipset

Ya habíamos mencionado anteriormente que la mayoría de las computadoras de hoy en día utilizan la arquitectura de von Neumann. Sin embargo, no solo existe la CPU, y por lo tanto, al momento de emular, no se trata únicamente de emular la CPU. También debemos considerar el chipset, que es un conjunto de chips que se extiende entre todos los componentes de una placa base, permitiendo gestionar la comunicación entre la CPU y otros componentes clave, como la memoria, los dispositivos de almacenamiento, las interfaces de red y los buses de entrada y salida.

El chipset se divide en dos partes: el "northbridge", que se encuentra en la parte superior y tiene una conexión directa con la CPU. Este se conecta a los componentes de mayor velocidad del sistema, así como a las tarjetas gráficas.

Por otro lado, tenemos la "southbridge", que se encarga de manejar componentes de baja velocidad, como el disco duro y los periféricos de entrada y salida. Además, gestiona algunos sistemas, como el sistema de energía y el reloj.

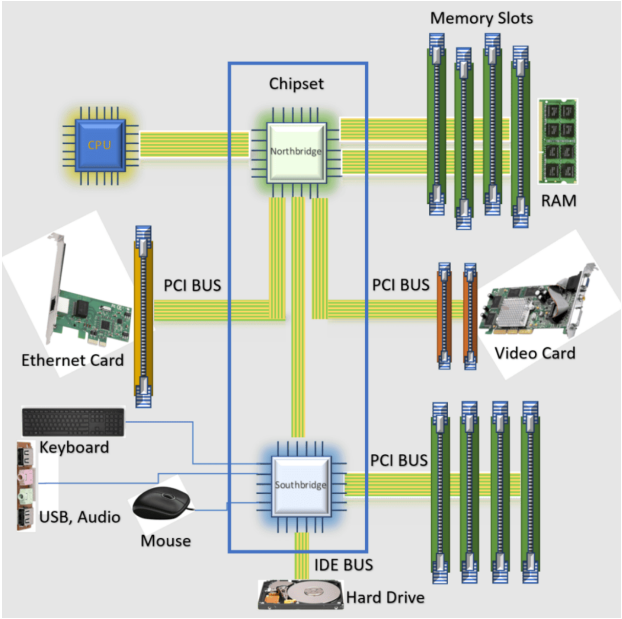

Figura 3: Estructura del chipset consultado del sitio web: https://www.baeldung.com/cs/chipset programming

Por supuesto, al emular la CPU, es importante tener en cuenta que no se limita solo a la CPU, sino que debemos involucrar todo el chipset, lo que implica identificar las formas de emular cada una de sus partes.

5.1. Problemas

Podríamos entrar en un detalle específico sobre el procedimiento para emular cada parte del chipset, pero debido a la complejidad de implementarlo, no se profundizará en cada una. Cada tipo de chipset tiene su propia forma de emularse y, además de los problemas individuales, la realidad es que, en conjunto, presentan ciertas complicaciones que afectan a todos los componentes por igual, las cuales se mencionarán a continuación.

5.1.1. Independencia de componentes

En la mayoría de los sistemas, los componentes del chipset están interconectados, por lo que al emularlos es importante recrear relaciones independientes y verificar su correcto funcionamiento. Si alguno de ellos falla, podría afectar el funcionamiento del emulador. Gracias a esta independencia, en aquellas arquitecturas que cuenten con múltiples núcleos de CPU, es posible que estas emulaciones se ejecuten en paralelo, lo que mejorará el rendimiento.

5.1.2. Interrupciones

Al estar encargado de las interrupciones, el chipset también debe considerar la gestión de las interrupciones que notifican a la CPU. Por lo tanto, el emulador debe manejar estas interrupciones de forma precisa. El problema radica en la diferencia en el esquema de gestión de interrupciones entre las arquitecturas emulada y anfitriona.

5.1.3. Sincronización de tiempos

Los componentes del chipset funcionan a partir de la sincronización precisa de los ciclos de reloj. Por lo tanto, al emular estos componentes, también se debe garantizar el tiempo de sincronización de cada uno de ellos. El problema radica en que cada tipo de arquitectura puede tener distintas frecuencias de reloj, por lo que cualquier error puede afectar el funcionamiento del sistema a emular.

5.1.4. Controladores

El chipset, al tener todas las conexiones con la CPU, depende de controladores específicos del sistema operativo. Por lo tanto, al crear un emulador, también debemos considerar los controladores específicos; de no hacerlo, nuestro emulador podría intentar comunicarse con un controlador inexistente o incompatible, lo que podría generar errores críticos e incluso causar que el emulador no pueda ejecutarse.

5.1.5. Plataformas cerradas

Normalmente, este problema se presenta en la emulación de consolas de videojuegos, donde evidentemente no solo replicamos la CPU, sino que también se incluyen chipsets de sonido y gráficos, los cuales son propiedad de la misma empresa y están optimizados para su funcionamiento. La falta de documentación puede causar que nuestro emulador tenga problemas de compatibilidad al estar incompleto o presentar errores imprecisos, lo que puede dificultar la identificación del origen del error.

6. Virtualización

Por último, otro término que a menudo se confunde con emulación es "virtualización". Aunque son muy parecidos (incluso más que simulación), sí existen diferencias que son más fáciles de entender una vez que se comprende la emulación. La mayor diferencia es que la virtualización permite tener varios sistemas operativos simultáneamente, pero solo funcionará cuando las arquitecturas sean las mismas, lo que permite un rendimiento casi nativo debido a la compatibilidad existente (el anfitrión y el invitado comparten la misma arquitectura). En cambio, como ya vimos a lo largo del documento, un emulador no está limitado en cuanto a arquitecturas, pero esto puede conllevar problemas de rendimiento que ya fueron mencionados anteriormente.

7. Conclusión

La emulación es una herramienta sumamente poderosa y útil. Aunque su fama y mayor uso se concentran en el sector de los videojuegos, sigue siendo fundamental para la preservación de arquitecturas que, de otro modo, se considerarían perdidas. De esta manera, la emulación actúa como una especie de "máquina del tiempo.[en] el ámbito de la computación, permitiéndonos apreciar la evolución de un sector que ha experimentado un desarrollo acelerado.

Sin embargo, llevar a cabo la emulación no es una tarea sencilla, ya que implica el desafío constante de preservar el conocimiento sobre cada arquitectura y asegurar la compatibilidad. La

historia ha demostrado que la incompatibilidad es uno de los factores que puede llevar al olvido de un sistema. En última instancia, somos nosotros quienes decidimos qué tecnologías perdurarán en el tiempo y cuáles quedarán en el pasado.

8. Referencias

- Del Barrio, V. M., Jiménez, A. F. (2001). Study of the Techniques for Emulation Programing.
- GeeksforGeeks. (2024, 23 septiembre). Difference between Emulation and Simulation. GeeksforGeeks. Consultado 27 de septiembre de 2024, de `https://www.geeksforgeeks.org/difference-\between-emulation-and-simulation/`
- Jiang, X., Chen, X., Wang, H., Chen, H. (2013). A Parallel Full-System Emulator for Risc Architure Host.
- GeeksforGeeks. (2024b, septiembre 27). RISC and CISC in Computer Organization. GeeksforGeeks. Consultado 27 de septiembre de 2024, de `https://www.geeksforgeeks.org/computer-\organization-risc-and-cisc/`
- Cottino, Damián. 1a ed. - Banfield - Lomas de Zamora : Gradi, 2009.
- Villar, E. y Gómez, J. (s.f.). Introducción a la virtualización. Consultado de: `http://www.adminso.es/images/6/6d/Eugenio_cap1.pdf`
- GeeksforGeeks. (2022, 16 febrero). North Bridge and its functions. GeeksforGeeks. Consultado 27 de septiembre de: `https://www.geeksforgeeks.org/north-bridge-and-its-functions/?ref=gcse_ind`
- GeeksforGeeks. (2020, 30 diciembre). South Bridge and its Functions. GeeksforGeeks. Recuperado 27 de septiembre de: `https://www.geeksforgeeks.org/south-bridge-and-its-functions/?ref=gcse_ind`