Actividad 2

Sistemas Operativos

Jessica Alexandra Magaña Salcedo

215616229

Centro Universitario de ciencias exactas e ingenierías

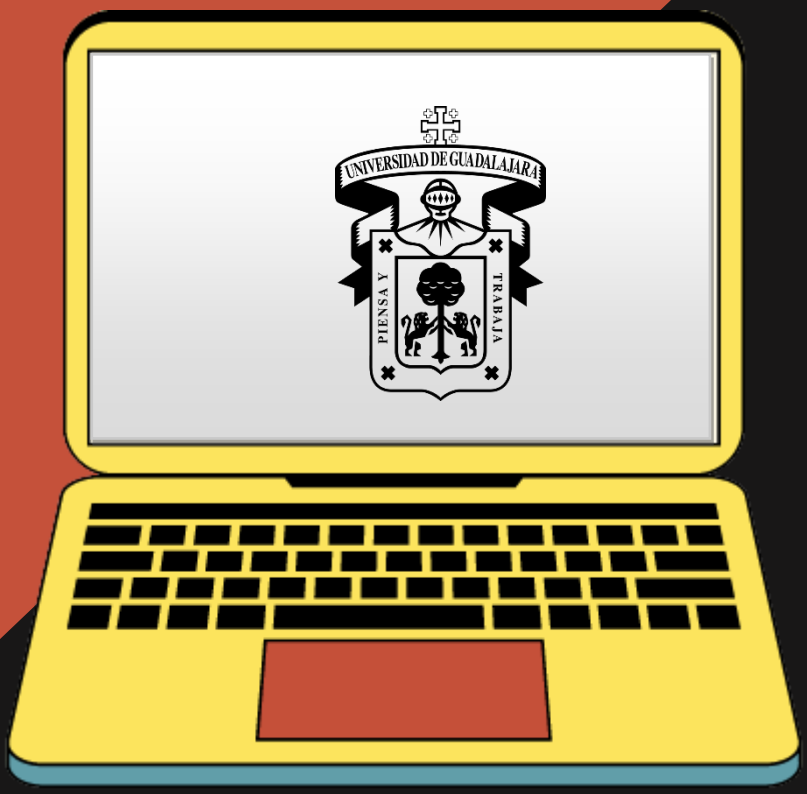
Departamento de ciencias computacionales

2024A

Maestra: Violeta Del Rocio Becerra Velazquez

D04

Ingeniería en computación



Índice

INTRODUCCIÓN

En el vasto panorama de la informática, los sistemas operativos desempeñan un papel esencial al actuar como intermediarios cruciales entre el hardware y el software. A medida que la tecnología ha evolucionado, han surgido diversos modelos de sistemas operativos, cada uno diseñado para abordar necesidades específicas en el complejo ecosistema informático. En esta investigación, nos sumergiremos en la exploración de cinco modelos fundamentales de sistemas operativos: el monolítico, cliente-servidor, máquina virtual, capas e híbrido. Además, profundizaremos en el análisis de un sistema operativo específico para comprender su historia, servicios, objetivos, funciones y estructura. Este estudio proporcionará una visión integral de la importancia y diversidad de los sistemas operativos en el mundo de la computación.

Modelos de sistemas operativos

Monolíticos

En el paradigma de diseño descrito, se aborda el concepto de un sistema operativo monolítico, una organización que ha sido considerada como la más común en la actualidad. En este enfoque, todo el sistema operativo se ejecuta como un único programa en modo kernel, donde cada procedimiento del sistema se encuentra vinculado en un único programa binario ejecutable extenso. Este programa se compone de miles de procedimientos que tienen la libertad de invocarse entre sí sin restricciones, formando así una estructura que, si bien funcional, a menudo resulta poco manejable y difícil de comprender.

La construcción del programa objeto del sistema operativo en este diseño implica la compilación de todos los procedimientos individuales, o los archivos que los contienen, seguida por la vinculación de estos procedimientos para formar un único archivo ejecutable. Este proceso se realiza mediante el enlazador del sistema. En términos de ocultamiento de información, este diseño presenta un enfoque directo, ya que todos los procedimientos son visibles para cualquier otro procedimiento en el sistema, a diferencia de estructuras más modularizadas donde se oculta información dentro de módulos.

A pesar de la aparente falta de ocultamiento de información, incluso en sistemas monolíticos es posible introducir cierta estructura. Para solicitar servicios del sistema operativo, como las llamadas al sistema, se colocan parámetros en un lugar definido (por ejemplo, en la pila), y se ejecuta una instrucción de trap. Esta instrucción cambia la máquina del modo usuario al modo kernel, transfiriendo el control al sistema operativo. Luego, el sistema operativo obtiene los parámetros y determina la llamada al sistema que se llevará a cabo, indexándola en una tabla que contiene apuntadores a los procedimientos correspondientes.

Este modelo de organización sugiere una estructura básica para el sistema operativo, que se compone de un programa principal que invoca procedimientos de servicio solicitados, un conjunto de procedimientos de servicio que ejecutan las llamadas al sistema, y un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio. Esta división en tres niveles ofrece una cierta organización dentro del sistema monolítico, permitiendo una gestión más estructurada de las diversas funcionalidades y servicios ofrecidos por el sistema operativo.

Es importante destacar que, además del núcleo del sistema operativo que se carga al arrancar la computadora, este modelo permite la incorporación de extensiones como drivers de dispositivos de E/S y sistemas de archivos, que pueden cargarse por demanda según las necesidades específicas del usuario o del sistema en un momento dado. Esto proporciona flexibilidad y eficiencia al sistema operativo al no cargar componentes innecesarios en la memoria hasta que realmente se requieran. En resumen, este enfoque monolítico, a pesar de sus desafíos en términos de complejidad, ofrece una estructura robusta para el diseño y la implementación de sistemas operativos.

Cliente-Servidor

En la variante del diseño conocida como el modelo cliente-servidor, se introduce una distinción fundamental entre dos clases de procesos: los servidores, encargados de proporcionar servicios específicos, y los clientes, que utilizan dichos servicios. Aunque comúnmente la capa inferior en este modelo es un microkernel, esta característica no es esencial, siendo lo crucial la presencia de procesos cliente y servidor. La comunicación entre estas entidades se realiza típicamente mediante el paso de mensajes, donde un proceso cliente construye un mensaje que indica la acción deseada y lo envía al servicio correspondiente. Posteriormente, el servidor realiza el trabajo y devuelve la respuesta al cliente. Esta estructura, aunque a menudo incluye un microkernel en la capa inferior, destaca por su enfoque modular y la clara separación de responsabilidades entre clientes y servidores.

La generalización natural de este concepto lleva a que los clientes y servidores no necesariamente se ejecuten en la misma computadora, sino que pueden estar distribuidos en distintas máquinas interconectadas por una red. La comunicación entre ellos sigue basándose en el intercambio de mensajes, proporcionando una abstracción uniforme para los procesos cliente, independientemente de si se ejecutan localmente o en una máquina remota. Este modelo ha demostrado ser altamente efectivo en entornos de red, donde los clientes se comunican con servidores a través de mensajes, facilitando la flexibilidad y el escalado de sistemas distribuidos.

El modelo cliente-servidor presenta una serie de ventajas notables, especialmente en entornos que requieren un alto grado de fiabilidad. Entre estas ventajas se encuentran la centralización de recursos, donde el servidor administra recursos comunes a todos los usuarios, como una base de datos centralizada; una mejora en la seguridad, ya que se reducen los puntos de entrada que permiten el acceso a los datos; una administración más eficiente a nivel del servidor, ya que los clientes requieren menos intervención y, por ende, menos administración; y la escalabilidad de la red, permitiendo la adición o eliminación de clientes sin afectar significativamente el funcionamiento global.

Este modelo se ha convertido en una elección popular para sistemas que involucran a usuarios en sus PCs domésticas como clientes, mientras que equipos más grandes operan como servidores en algún lugar remoto. Un ejemplo común de su aplicación es la arquitectura de la World Wide Web (WWW), donde las solicitudes de páginas web son enviadas desde PCs a servidores remotos, utilizando la abstracción del modelo cliente-servidor en una red.

En términos de diseño, el sistema Cliente/Servidor se basa en la provisión de servicios, unidades básicas de diseño, que son ofrecidos por servidores y utilizados por clientes. La idea de recursos compartidos es fundamental, ya que muchos clientes pueden utilizar los mismos servidores, compartiendo tanto recursos lógicos como físicos. La asimetría en los protocolos, donde los clientes inician las "conversaciones" mientras los servidores esperan pasivamente, es otra característica distintiva.

Además, la transparencia de la localización física de los servidores y clientes permite a los clientes acceder a recursos sin conocer su ubicación exacta.

La independencia de la plataforma hardware y software es esencial, al igual que la implementación de sistemas débilmente acoplados basados en la interacción a través de mensajes. La encapsulación de servicios asegura que los detalles de implementación sean transparentes para el cliente. La escalabilidad, tanto horizontal (agregando clientes) como vertical (ampliando la potencia de los servidores), es inherente a este diseño. Además, la integridad de datos y programas se facilita mediante la centralización en servidores, simplificando el mantenimiento y asegurando la consistencia de la información.

En el contexto del modelo cliente-servidor, un servidor, también conocido como "daemon", se activa y espera las solicitudes de los clientes. Los servicios de un servidor son utilizados comúnmente por múltiples clientes distintos, y tanto los programas cliente como los servidores pueden ser parte de aplicaciones más grandes. El cliente, por su parte, es el proceso que permite al usuario formular requerimientos y pasarlos al servidor. Puede clasificarse como basado en aplicación de usuario o basado en lógica de negocio, según la interacción y manipulación de datos que realiza. La funcionalidad del cliente se centra en la administración de la interfaz de usuario, la interacción con el usuario, el procesamiento de la lógica de la aplicación, la generación de requerimientos de bases de datos y la recepción y formateo de resultados del servidor.

Máquina Virtual

En primer lugar, es crucial entender que existen dos tipos de máquinas virtuales, diferenciadas por su funcionalidad: las de sistema y las de proceso, siendo las de sistema las más comunes y mencionadas. Enfocándonos en las máquinas virtuales de sistema, estas emulan un ordenador completo, es decir, son un software que puede simular otro dispositivo, como un PC, permitiendo la ejecución de otro sistema operativo dentro de su entorno virtual.

Una máquina virtual de sistema tiene sus propios componentes virtuales, como disco duro, memoria, tarjeta gráfica, entre otros, a pesar de que todos ellos son virtuales. Esto no implica necesariamente que estos recursos no existan físicamente; por ejemplo, una máquina virtual puede reservar 2 GB de RAM y 20 GB de disco duro, recursos que provienen del PC en el cual está instalada la máquina virtual, denominado a veces como hipervisor, host o anfitrión. Algunos dispositivos pueden ser inexistentes físicamente, como un CD-ROM que en realidad es el contenido de una imagen ISO en lugar de un lector de CD físico. Para el sistema operativo que se ejecuta dentro de la máquina virtual, toda esta emulación es transparente e invisible, funcionando de manera similar a un PC convencional, sin ser consciente de que está operando dentro de un entorno virtual.

Dentro de su burbuja, la máquina virtual está aislada y no puede acceder al resto de los datos de la máquina anfitrión, a pesar de estar físicamente en el mismo

dispositivo. Sin embargo, las aplicaciones principales de máquinas virtuales, como VirtualBox o VMWare, proporcionan herramientas para facilitar la transferencia de archivos entre la máquina virtual y la máquina anfitrión.

Para operar, una máquina virtual realiza el mapeo de los dispositivos virtuales que ofrece a su sistema invitado con los dispositivos reales presentes en la máquina física. Por ejemplo, la máquina virtual puede emular una tarjeta de sonido Sound Blaster de 16 bits, aunque en realidad esté conectada a la tarjeta de sonido interna de la placa base del PC, que podría ser de la marca Realtek.

La virtualización puede ser realizada por software o con soporte a través del hardware, siendo esta última opción más eficiente en términos de rendimiento. Desde 2005, es común que los procesadores cuenten con tecnología de virtualización por hardware, aunque esta característica no siempre está activada por defecto en la BIOS.

En contraste, las máquinas virtuales de proceso son menos ambiciosas y ejecutan un proceso específico, como una aplicación, en su entorno de ejecución. Este enfoque es común en el desarrollo de aplicaciones multiplataforma, donde la máquina virtual se encarga de lidiar con el sistema operativo, facilitando la portabilidad de aplicaciones.

Las máquinas virtuales son ampliamente utilizadas en diversos contextos, tanto a nivel profesional como en el ámbito del consumidor final. Algunos de los usos más comunes incluyen:

1. **Pruebas de Sistemas Operativos:** Facilitan la prueba de nuevos sistemas operativos sin comprometer la instalación en el hardware principal. Esto permite probar, revertir y experimentar con sistemas operativos sin riesgos.
2. **Ejecución de Programas Antiguos:** Permiten la ejecución de programas y sistemas operativos antiguos en hardware moderno, evitando la obsolescencia de aplicaciones críticas para ciertos negocios.
3. **Compatibilidad con Aplicaciones de Otros Sistemas:** Posibilitan la ejecución de aplicaciones diseñadas para otros sistemas operativos, permitiendo una mayor flexibilidad y compatibilidad.
4. **Desarrollo y Pruebas de Software:** Son esenciales para los desarrolladores que necesitan probar sus aplicaciones en diferentes entornos sin necesidad de múltiples dispositivos físicos.
5. **Seguridad Adicional:** Proporcionan un entorno aislado para realizar tareas riesgosas, como analizar virus y malware, sin poner en peligro el sistema principal.
6. **Dinamismo y Flexibilidad:** Permiten guardar estados, ampliar, trasladar y mantener máquinas virtuales con gran dinamismo, siendo esenciales en entornos empresariales con servidores web que gestionan múltiples máquinas virtuales.

A pesar de sus diversas utilidades, las máquinas virtuales pueden afectar el rendimiento, ya que implican la ejecución de dos sistemas operativos simultáneamente. Aunque las aplicaciones de creación de máquinas virtuales son

cada vez más eficientes y el hardware más potente, la emulación siempre conlleva un esfuerzo adicional en comparación con la ejecución directa del software en el hardware.

Capas

La complejidad inherente de los sistemas operativos se debe a la variedad de servicios y funciones que implementan, tales como la gestión de entrada y salida, el control y seguimiento de los programas en ejecución, y la administración de la memoria, entre otros. Este conjunto de operaciones genera una intricada red de procesos internos que los diseñadores del sistema operativo deben gestionar. Sin embargo, para el usuario final, estas operaciones deben ser transparentes, lo que significa que los detalles internos deben permanecer ocultos. La estrategia para lograr esta transparencia consiste en ocultar los detalles de implementación y las estructuras de datos mediante la encapsulación de funciones en módulos.

Esta estrategia de ocultación se materializa a través de una jerarquía de niveles de abstracción, donde cada nivel ofrece un conjunto específico de funciones primitivas que son utilizadas por las capas superiores. En el contexto de esta jerarquía, se presenta un modelo general de sistema operativo por capas.

En la representación por capas, la capa más interna es el Núcleo o Kernel. Esta capa juega un papel crucial al gestionar todos los procesos del sistema. Se encarga de llevar un registro de los procesos activos y de planificar su ejecución, determinando cuál de ellos ocupará el tiempo del procesador. La eficiencia y rendimiento del sistema operativo están directamente influenciados por la calidad y diseño del núcleo. Un ejemplo destacado es Windows XP, que obtuvo un rendimiento sólido al ser construido sobre un núcleo UNIX adquirido a la compañía Santa Cruz Operations.

La segunda capa, denominada Entrada y Salida Básica, proporciona funciones primitivas para la gestión de la memoria secundaria, que implica la localización, escritura y lectura de bloques de datos en el disco duro. Es importante señalar que en esta capa, la información almacenada no se representa como archivos; esta representación se implementa en capas superiores.

La capa de Gestión de Memoria, en el tercer nivel, administra la memoria principal o RAM. Es responsable de asignar bloques de memoria a los procesos y liberarlos cuando dichos procesos han concluido. Además, esta capa gestiona la retirada de algunos procesos de la memoria, almacenando una imagen de ellos en el disco duro. Este proceso, conocido como memoria virtual, simula la existencia de más memoria de la que físicamente está disponible.

La cuarta capa, Sistema de Archivos, proporciona funciones esenciales para almacenar información en archivos. Se apoya en las primitivas de la capa de Entrada y Salida Básica, y toma decisiones sobre qué procesos utilizan la memoria.

Finalmente, en la quinta capa se encuentra el Interprete de Comandos, que constituye la interfaz visible para el usuario. Esta interfaz puede presentarse como una línea de comandos o como una Interfaz Gráfica de Usuario (GUI). La función principal

de esta capa es traducir las interacciones del usuario en comandos comprensibles para el conjunto de primitivas de las capas inferiores.

Hibrido

En el ámbito de la computación, un núcleo híbrido representa un tipo de núcleo en el diseño de un sistema operativo. En esencia, se trata de un micronúcleo con una porción de código "no esencial" ubicado en el espacio del núcleo, permitiendo así que este se ejecute más eficientemente en comparación con su ejecución en el espacio de usuario.

Este enfoque surge como un compromiso adoptado por muchos desarrolladores de los primeros sistemas operativos con arquitectura basada en micronúcleo, antes de que se demostrara que los micronúcleos también podían ofrecer un rendimiento sobresaliente. La característica distintiva de un núcleo híbrido radica en la inclusión de cierto código "no esencial" en el espacio de núcleo para optimizar su ejecución.

La mayoría de los sistemas operativos modernos se clasifican en la categoría de núcleos híbridos, siendo uno de los ejemplos más destacados Microsoft Windows. El núcleo de Mac OS X, conocido como XNU, también sigue un enfoque de micronúcleo no modificado, incorporando código del núcleo de FreeBSD en el núcleo basado en Mach. DragonFlyBSD es un ejemplo adicional y notable, siendo el primer sistema BSD que adopta una arquitectura de núcleo híbrido sin depender de Mach.

Es crucial no confundir erróneamente los núcleos híbridos con los núcleos monolíticos que permiten la carga dinámica de módulos después del arranque. La esencia del núcleo híbrido radica en la utilización de mecanismos y conceptos de arquitectura tanto del diseño monolítico como del micronúcleo. Esto incluye la implementación de características como el paso de mensajes y la migración de código "no esencial" hacia el espacio de usuario. A pesar de trasladar parte del código "no esencial" al espacio de usuario, aún retiene cierto código en el núcleo por razones de rendimiento.

Ejemplos destacados de sistemas operativos con núcleos híbridos incluyen:

1. **Microsoft Windows NT:** Utilizado en una amplia gama de sistemas que comparten el código base de Windows NT.
2. **XNU (utilizado en Mac OS X):** Incorpora elementos de micronúcleo no modificado, fusionando características del núcleo basado en Mach con código de FreeBSD.
3. **DragonFlyBSD:** Se destaca como el primer sistema BSD en adoptar una arquitectura de núcleo híbrido sin depender de Mach.
4. **ReactOS:** Un proyecto de sistema operativo de código abierto que también emplea un núcleo híbrido, ofreciendo compatibilidad con aplicaciones de Windows.

Estos ejemplos subrayan la versatilidad y eficiencia de los núcleos híbridos al combinar aspectos de diseños monolíticos y de micronúcleos para lograr un rendimiento óptimo y una funcionalidad avanzada.

Sistema Operativo “Linux”

Linux, conocido más formalmente como GNU/Linux, representa un sistema operativo con una historia fascinante y una evolución que ha dejado una huella significativa en el mundo de la informática. Este sistema, comparable a otros como MacOS, DOS o Windows, ha sido moldeado por la visión de Linus Torvalds y ha crecido desde sus modestos inicios a principios de la década de los noventa.

La génesis de Linux se encuentra en Unix, desarrollado en los años sesenta por Dennis Ritchie y Ken Thompson en los Laboratorios Telefónicos Bell. Posteriormente, Andrew Tanenbaum creó Minix, un sistema operativo similar a Unix diseñado con fines educativos. Linus Torvalds, estudiante finlandés, observó la limitación de Minix para expansiones y optó por escribir su propio sistema operativo compatible con Unix.

En paralelo, el proyecto GNU (GNU's Not Unix) liderado por Richard Stallman avanzaba hacia la creación de un sistema operativo completo, con la excepción crucial del kernel o núcleo que controla el hardware. Torvalds decidió combinar el sistema GNU con su propio núcleo, al que llamó Linux (Linux Is Not UniX), dando origen a la fusión que hoy conocemos como GNU/Linux.

La aparición oficial de Linux se remonta a 1991, cuando Linus Torvalds inició el desarrollo como un proyecto personal mientras estudiaba informática en la Universidad de Helsinki. Inspirado en Minix, las primeras líneas de código fueron programadas sin anticipar el alcance que este proyecto alcanzaría. Las discusiones iniciales sobre Linux se llevaron a cabo en el grupo de noticias comp.os.minix, donde se exploraba la posibilidad de crear un sistema Unix más completo para usuarios de Minix.

La versión 0.01 de Linux, lanzada en agosto de 1991, no era ejecutable y solo contenía los principios del núcleo del sistema, escrito en lenguaje ensamblador y asumiendo el acceso a un sistema Minix para su compilación. La primera versión "oficial", la 0.02, fue anunciada el 5 de octubre de 1991, permitiendo la ejecución de Bash y gcc. A medida que más programadores se unieron al proyecto, Linus incrementó la numeración del sistema hasta la versión 0.95 en marzo de 1992.

El desarrollo continuó rápidamente, y para diciembre de 1993, Linux alcanzó la versión 0.99, mientras que la tan esperada versión 1.0 no se materializó hasta el 14 de marzo de 1994. Desde entonces, Linux ha experimentado un crecimiento constante, alcanzando la versión 2.2 y continuando su evolución con el objetivo de perfeccionar y mejorar el sistema día a día.

Linux no solo es un sistema operativo, sino un testimonio del poder de la colaboración global y la filosofía de código abierto. Desde su origen en la mente de

Linus Torvalds hasta su estatus actual en la vanguardia de la tecnología, Linux ha dejado una marca indeleble en la historia de la informática.

Servicios que presta: Linux ofrece una amplia gama de servicios, desde la gestión de archivos hasta la administración de redes. Funciona como servidor web a través de servicios como Apache o Nginx y proporciona servicios esenciales como gestión de impresión, correo electrónico y bases de datos.

Objetivos: Los objetivos de Linux se centran en ofrecer un sistema operativo robusto, seguro y de alto rendimiento de forma gratuita y accesible para todos. Promueve la colaboración a través del código abierto y busca proporcionar una alternativa confiable a los sistemas operativos propietarios.

Funciones: Linux desempeña diversas funciones, desde la gestión de recursos en servidores hasta su uso en sistemas embebidos en dispositivos como routers y televisores inteligentes. Como sistema operativo de escritorio, proporciona una interfaz gráfica de usuario y una amplia gama de aplicaciones.

Estructura: La estructura de Linux se basa en un núcleo central que gestiona recursos y actúa como intermediario entre hardware y software. El núcleo es monolítico, con controladores esenciales incorporados. La capa de espacio de usuario incluye bibliotecas, utilidades y aplicaciones, permitiendo una fácil adición de funciones y controladores. Esta estructura modular contribuye a su flexibilidad y adaptabilidad.

Preguntas

1. **¿Qué significa JCL?**

JCL, o "Job Control Language" (Lenguaje de Control de Trabajos), es un componente esencial en entornos de sistemas mainframe. Se trata de un lenguaje de scripting que se utiliza para definir, controlar y ejecutar trabajos en sistemas operativos mainframe, como IBM z/OS. A través de instrucciones en JCL, los usuarios especifican los recursos necesarios para sus programas, definen la secuencia de ejecución y establecen parámetros de configuración. JCL es crucial en entornos de procesamiento por lotes, donde los trabajos se ejecutan de manera secuencial sin intervención directa del usuario.

2. **Diferencia entre procesamiento por lotes y procesamiento por lotes con multiprogramación.**

El procesamiento por lotes implica la ejecución secuencial de un solo programa sin intervención directa del usuario. En cambio, el procesamiento por lotes con multiprogramación permite la ejecución simultánea de varios programas dentro de un lote. La multiprogramación mejora la eficiencia del sistema al aprovechar los períodos de inactividad de un programa para ejecutar otros, optimizando así la utilización de los recursos del sistema.

3. **Utilidades de la interrupción int86 en C.**

La interrupción **int86** en C es una función que facilita la comunicación con el hardware de la computadora, especialmente en entornos DOS. Una utilidad común es su capacidad para realizar llamadas a interrupciones del sistema operativo para acceder a funciones de bajo nivel, como el control de interrupciones, la manipulación de registros del procesador y la interfaz con dispositivos de hardware específicos.

4. **¿Para qué sirve la función Kbhit?**

La función **Kbhit** se utiliza en programación en C para detectar si se ha presionado una tecla en el teclado sin bloquear la ejecución del programa. Permite la creación de programas interactivos donde la ejecución puede responder inmediatamente a la entrada del usuario sin detenerse para esperar la entrada.

5. **Equivalentes de Kbhit en otros dos lenguajes de programación.**

En Python, la función **msvcrt.kbhit()** de la biblioteca **msvcrt** proporciona funcionalidad similar. En Java, la clase **java.awt.event.KeyEvent** junto con un objeto **KeyListener** se puede utilizar para lograr la detección de pulsaciones de teclas de manera no bloqueante. Estos equivalentes permiten lograr interactividad en programas escritos en Python y Java, respectivamente.

Conclusión

En esta investigación, se ha explorado a fondo diferentes modelos de sistemas operativos, desde el monolítico hasta el cliente-servidor, la máquina virtual, las capas y el híbrido. Cada modelo presenta sus propias ventajas y desafíos, abordando distintas necesidades y contextos de implementación.

El modelo monolítico destaca por su estructura robusta, aunque compleja, proporcionando una organización dentro del sistema operativo a través de un programa único. A pesar de la falta aparente de ocultamiento de información, se pueden introducir ciertos niveles de estructura, lo que permite una gestión más ordenada de funcionalidades y servicios.

El modelo cliente-servidor, por otro lado, introduce una clara separación entre procesos de servidor y cliente, fomentando la centralización de recursos y mejorando la seguridad y la eficiencia de la administración. Este enfoque se ha vuelto especialmente popular en entornos de red, como la arquitectura de la World Wide Web.

Las máquinas virtuales, ya sean de sistema o de proceso, ofrecen flexibilidad y aislamiento, permitiendo la ejecución de sistemas operativos adicionales en un entorno virtual. Se utilizan en una variedad de aplicaciones, desde pruebas de sistemas operativos hasta el desarrollo y pruebas de software.

La estructura por capas aborda la complejidad inherente de los sistemas operativos mediante la encapsulación de funciones en módulos, proporcionando niveles de abstracción que ocultan detalles internos al usuario final. Cada capa desempeña un papel específico en la gestión del sistema.

El modelo híbrido, representado por núcleos híbridos, combina aspectos de los diseños monolíticos y de micronúcleos para lograr un rendimiento óptimo y una funcionalidad avanzada. Ejemplos notables incluyen Microsoft Windows y Mac OS X.

Finalmente, se exploró el caso específico del sistema operativo Linux, que ha evolucionado desde sus modestos inicios en la década de 1990 hasta convertirse en un sistema operativo robusto y altamente colaborativo. Linux, basado en la filosofía de código abierto, ha dejado una huella significativa en la historia de la informática, demostrando el poder de la colaboración global.