¿QUÉ PASA DENTRO DE TU COMPUTADORA CUANDO HACES CLIC?

Daniel Maldonado Alcalá[a], Rogelio Ernesto García Chávez[b],
Víctor Alexis Hernández Castellanos[a], Jessica Marcelino López[c],
Ramón Silva Ortigoza[d], Magdalena Marciano Melchor[d]

[a] Instituto Politécnico Nacional, UPIITA.
[b] Colegio de Postgraduados, Campus Montecillo, PSEI-Cómputo Aplicado.
[c] Instituto Politécnico Nacional, ESCOM.
[d] Instituto Politécnico Nacional, CIDETEC, Laboratorio de Mecatrónica y Energía Renovable.

dmaldonadoa1900@alumno.ipn.mx, garcia.rogelio@colpos.mx
vhernandezc2200@alumno.ipn.mx, jmarcelinol2000@alumno.ipn.mx
rsilvao@ipn.mx, mmarciano@ipn.mx

RESUMEN

El artículo explica el proceso mediante el cual se traduce el lenguaje del usuario a uno comprensible para la computadora, desarrollando los conceptos de jerarquía de memoria, la cual prioriza la rapidez de cada tarea, la BIOS que revisa el estado del hardware, el kernel como núcleo de un sistema operativo, qué es y qué hace un sistema operativo, y como última parte del recorrido, cómo es posible la comunicación usuario-máquina y viceversa, con el objetivo de que la computadora deje de ser una caja negra para el público.

ABSTRACT

The article explains the process by which user language is translated into a form that the computer can understand. It discusses the concept of the memory hierarchy, which prioritizes tasks based on their speed; the BIOS, which checks the status of the hardware; the kernel, the core of an operating system; what an operating system is and what it does; and, finally, how communication between the user and the machine is made possible in both directions. The overall goal is to help readers understand how a computer works, so that it no longer remains a "black box" to the general public.

Introducción

En la vida contemporánea, el ordenador se ha consolidado como una herramienta fundamental, lo empleamos para llevar a cabo tareas que sin el ordenador serían laboriosas, costosas o imposibles: redactar un documento, procesar datos o ejecutar una aplicación. Sin embargo, rara vez nos detenemos a cuestionar cómo funciona, y termina siendo una caja negra indispensable en nuestras vidas.
Por ello, en el presente artículo se plantea analizar los subprocesos que se llevan a cabo en esta caja que llamamos computadora, que hacen posible su funcionamiento como una máquina amigable y segura para el usuario.
Antes de abordar los aspectos técnicos, conviene establecer una premisa fundamental: el lenguaje que empleamos no es directamente comprensible para la máquina. Debe realizar subprocesos que permiten la traducción entre el usuario y el ordenador.

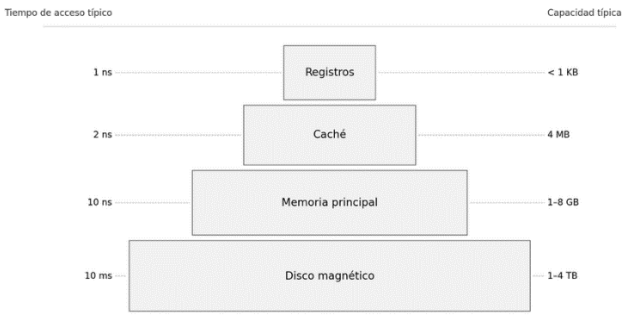


Figura 1: Jerarquía de memoria: relación entre tiempo de acceso y capacidad típica.
Adaptado de Tanenbaum (2015) [1].

A nivel físico, la computadora opera con voltajes; más específicamente, con señales binarias: la ausencia de pulso se interpreta como 0, y su presencia como 1. Sobre esta base, los subprocesos descritos a lo largo del artículo hacen posible la conversión entre el lenguaje natural y el binario, la manipulación de información, la ejecución de operaciones de alto nivel, tan rápido que parece instantáneo (milisegundos).
Para lograr el objetivo del artículo, el mismo se delimita en los siguientes temas: jerarquía de memoria, los componentes que hacen posible la computación; la BIOS, el primer intérprete; el kernel, el núcleo del Sistema Operativo (SO); como la interfaz del mundo abstracto; y, en última instancia el proceso de traducción mediante el cual una instrucción se convierte de lenguaje natural a señales eléctricas.

La jerarquía de memoria: los componentes que hacen posible la computación

En el caso ideal la memoria debería ser muy rápida, abundante, y barata[1]. Pero cumplir estas especificaciones no es posible con la tecnología actual, por lo que se opta por otra metodología para manipular los datos.

Las computadoras rigen la manipulación de los datos por una jerarquía de memoria (Figura 1), en la cual los componentes que son capaces de manipular más rápido la información se encuentran en la parte superior, aunque su coste de operación es el mayor y su capacidad de almacenamiento es reducida. A su vez, los que se encuentran más abajo cuentan con un bajo coste y gran capacidad de almacenamiento, pero su velocidad al transmitir datos es pobre al compararlos con los primeros.

En la cima están los **registros**, ubicados dentro del CPU y fabricados con la misma tecnología que este, por lo que operan a su misma velocidad. Le sigue la caché, una memoria intermediaria de naturaleza volátil que evita que el procesador tenga que buscar constantemente en niveles inferiores. Después viene la DRAM o memoria principal, donde residen los programas en ejecución y los datos que se están manipulando; aunque más lenta que la caché, ofrece varios GB de capacidad, suficientes para múltiples programas simultáneos. Finalmente, el disco duro es el único nivel no volátil: aquí residen los datos de forma permanente, aunque su velocidad es la más baja de la jerarquía.

Ya explicados los componentes que procesan la memoria, podemos hablar de lo primero que ocurre al encender la máquina antes de usar dichos componentes.

BIOS/UEFI: El primer intérprete

Al encender la computadora, el primer programa en ejecutarse no es el sistema operativo, sino un firmware grabado de fábrica en la tarjeta madre: la **BIOS**[2] o su sucesor moderno, la **UEFI**[3]. Su función es inicializar el CPU y la RAM, verificar que los componentes críticos del hardware estén en condiciones óptimas mediante una rutina diagnóstica conocida como POST, y, si todo es correcto, localizar en el disco un gestor de arranque válido para cargarlo y cederle el control. Sin este proceso, la

[1] en términos de costo por unidad de almacenamiento, de modo que pueda estar disponible en grandes cantidades sin representar un costo prohibitivo.
[2] Basic Input/Output System.
[3] Unified Extensible Firmware Interface.

jerarquía de memoria no podría operar, pues los registros, la caché y la DRAM se encuentran completamente vacíos en cada nuevo encendido [2].

Kernel: Núcleo del SO

El Kernel es el programa que funciona como intermediario entre el hardware y los programas de usuario, gracias a este se presenta una interfaz amable y segura al usuario, por esto último se ejecuta permanentemente en el equipo. Este se carga al inicializar el dispositivo, se carga en la RAM y comienza su ejecución. A partir de este momento toma control absoluto, pues es el que inicializa los drivers[4].

A este también se le atribuyen dos funciones primordiales para la operación de la computadora:

1. Máquina extendida: la construcción del hardware es rudimentaria y difícil de programar. Es el kernel el que oculta esa complejidad intrínseca del hardware a los usuarios y ofrece una "abstracción". Por ejemplo: en la realidad física del hardware, el disco es una colección de sectores[5], los cuales el Kernel organiza para poder presentarlos en una jerarquía de directorios y nombres de archivos, lo que representa un modelo de recursos simples y limpios para el programador.

2. Administrador de recursos: es el encargado de repartir el tiempo de CPU, la memoria y dispositivos de entrada/salida a los programas que compiten por estos constantemente. Esto se logra mediante dos mecanismos:

- **Multiplexación en el tiempo**: decide quién emplea el recurso en el momento y por cuánto tiempo (ej. le da a la CPU permiso de procesar el sonido de la música por unos milisegundos, se hace un cambio de prioridad para el CPU, y le dice que procese un par de fotogramas para reproducir un video en internet).
- **Multiplexación en el espacio**: decide qué parte del espacio le toca a cada quién (ej. divide la memoria RAM para que dos programas no entren en conflicto y se sobrescriban).

El Kernel también crea una barrera de protección entre el usuario y el hardware, conocida como la distinción entre modo usuario y modo kernel. Cuando un programa necesita realizar una tarea real —como escribir en el disco o enviar un paquete de red—, debe cruzar esta barrera mediante una llamada al sistema (System Call), la cual es una petición formal que el programa hace al kernel para cruzar la barrera de privilegios. Esta danza constante entre privilegios y restricciones es lo que nos permite entender qué es un Sistema Operativo [1].

6. El sistema operativo: la interfaz del mundo abstracto

El sistema operativo es un entorno en donde se gestionan privilegios y restricciones. También se administran las competencias por los recursos y se ofrece una máquina extendida donde el software coexiste de forma segura y eficiente a costa del hardware.

Con base en esto podemos hacer la distinción de tres niveles en un sistema operativo, pues es una entidad más grande que el modo Kernel, también abarca todos los programas de sistema que corren en el modo usuario:

1. Hardware: El nivel físico (CPU, Memoria).
2. Modo Kernel: El sistema operativo propiamente.
3. Modo Usuario: Programas de sistema (APIs[6], shells y GUIs[7], compiladores y editores) y aplicaciones de usuario (Chrome, juegos, reproductores de multimedia).

Una de las funciones más importantes del SO es la capacidad de cargar y cerrar programas, mediante una serie de 5 pasos:

1. Localización: El SO busca el archivo ejecutable en el sistema de archivos.
2. Creación del Espacio de Direccionamiento: El kernel reserva un pedazo de RAM para el nuevo proceso (segmentos de texto, datos y stack).

[4] programas especializados que actúan como 'traductores' o puentes entre el sistema operativo y el hardware.
[5] que tradicionalmente abarcan 512 bytes de datos, aunque en los discos modernos se agrupa la información en sectores más grandes de 4 kB.
[6] conjunto de llamadas al sistema que el SO expone para que los programadores puedan pedirle cosas al kernel.
[7] conjunto de componentes visuales como iconos, ventanas, menús y botones que permiten a una persona interactuar con un dispositivo electrónico [3].

3. Carga (Loading): El kernel lee el código del disco y lo copia a la RAM.
4. El Salto: Cambia simultáneamente el procesador de modo kernel a modo usuario.
5. Ejecución: En ese instante, el kernel suelta el CPU y deja que el programa corra, pero mantiene un temporizador activo para recuperar el control si el programa se queda atascado (multiplexación en el tiempo).

Hasta el momento se ha obviado un detalle importante, y es que la computadora y el usuario no hablan el mismo lenguaje. ¿Cómo es posible que la computadora que opera con 1 y 0 entienda lo que requerimos si empleamos ideas complejas?

7. Del lenguaje de usuario al lenguaje máquina: el proceso de traducción

La computación de la modernidad se basa en varias capas de abstracción que ocultan la complejidad de programar a nivel máquina, pues el proceso sigue este flujo:

Lenguaje de Alto Nivel → Lenguaje Ensamblador → Código Máquina (Binario) → Señales Eléctricas

Este proceso mediante el cual se traduce el lenguaje abstracto a señales eléctricas lo realiza un compilador, el cual se encargará de descomponer el lenguaje de alto nivel antes de que el SO pueda enviar las instrucciones a la computadora. Este proceso consta de tres etapas:

1. **Análisis Léxico (Scanner)**: En primera instancia el compilador se encarga de leer los caracteres del código fuente proporcionado en lenguaje de alto nivel, y agrupa los caracteres en tokens[8] [4]. Ejemplo: Transforma la línea *Fuerza = masa * aceleracion* en una secuencia de tokens: [Fuerza] [OP: =] [ID: masa] [OP: *] [ID: aceleracion].
2. **Análisis Sintáctico (Parser)**: Los tokens generados por el Scanner se organizan en una estructura jerárquica, la cual se suele llamar **Árbol de Sintaxis Abstracta**. Revisa que el código proporcionado siga las reglas gramaticales[9] del lenguaje de programación, y empieza a dar un orden lógico a lo que se convertirá en instrucciones para el CPU [4].
3. **Análisis Semántico**: El compilador revisa por última ocasión que la jerarquía tenga sentido lógico; en esta fase la tarea principal es la comprobación de tipos [4].

Una vez se ha generado este nuevo código, el linker se encarga de unirlo a bibliotecas externas, las cuales servirán para crear un ejecutable único [5]. Posteriormente se emplea el Loader, que asigna un espacio de direccionamiento en la RAM y carga las instrucciones. Sin embargo, el código por sí mismo es estático; no sirve de nada sin el SO [1].

En la última etapa el SO gestiona los buffers de salida[10], y los datos se escriben en el Frame Buffer[11]. Los circuitos convierten los bits en señales que controlan los píxeles del monitor, y de este modo es posible visualizar la respuesta del sistema [5].

Conclusiones

Lo que ocurre dentro de una computadora al hacer clic no es magia, sino una cadena de jerarquías y traducciones que operan en milisegundos. La jerarquía de memoria prioriza la velocidad de acceso a los datos; el firmware despierta el hardware y cede el control; el kernel administra los recursos y protege al usuario del hardware; y el compilador traduce las instrucciones en señales eléctricas que finalmente se convierten en píxeles en tu pantalla. Conocer este proceso convierte a la computadora de caja negra en algo comprensible, y esa es precisamente la intención de este artículo.

[8] palabras reservadas, identificadores, operadores .
[9] como el orden de los paréntesis y/o puntos y coma.
[10] espacios de memoria temporal en RAM que almacenan datos procesados antes de enviarlos a un dispositivo externo.
[11] porción de memoria RAM, generalmente en la tarjeta gráfica (VRAM), que almacena todos los píxeles de una imagen antes de enviarla a la pantalla.

Referencias

[1] Andrew S. Tanenbaum and Herbert Bos. Modern Operating Systems. Pearson Education, 4th edition, 2015. ISBN 978-0-13-359162-0.

[2] Unified EFI Forum. Unified Extensible Firmware Interface (UEFI) Specification, Version 2.10. https://uefi.org/specs/UEFI/2.10/, 2022. Consultado en abril de 2026.

[3] Lenovo. ¿Qué es una interfaz gráfica de usuario (GUI)?, 2024. URL https://www.lenovo.com/mx/es/glosario/que-es-una-interfaz-grafica-de-usuario/.

[4] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Boston, MA, 2nd edition, 2006. ISBN 978-0321486813.

[5] David A. Patterson and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 5th edition, 2014.