Teorica 1 - SisOps

1. Introducción a los Sistemas Operativos

- **Definición General:**

  Un Sistema Operativo (SO) es la capa de software que gestiona los recursos físicos de una computadora (procesador, memoria, dispositivos de entrada/salida, etc.) y ofrece servicios a las aplicaciones. En otras palabras, el SO abstrae el hardware para que los programas puedan ejecutarse sin preocuparse por los detalles físicos del sistema.

- **Objetivo Principal:**

  Facilitar la ejecución de múltiples programas simultáneamente y proporcionar una interfaz uniforme para el acceso a recursos, lo que se logra principalmente mediante la **virtualización**: transformar recursos físicos en “recursos virtuales” más generales y manejables.

---

2. Ciclo de Ejecución y Arquitectura

- **Ciclo Fetch-Decode-Execute:**
  Cuando un programa se ejecuta, el procesador realiza en secuencia:
  - **Fetch:** Buscar la instrucción en memoria.
  - **Decode:** Decodificar la instrucción.
  - **Execute:** Ejecutar la instrucción.
- **Arquitectura de Von Neumann:**
  Modelo clásico donde se unen las áreas de datos e instrucciones en la memoria. Se ilustra con ejemplos históricos (ENIAC, IBM 360, etc.) y se contrasta con las tendencias actuales que incluyen arquitecturas distribuidas y el uso de GPUs y Cloud Computing.

---

3. Tendencias Actuales

- **GPUs:**

  Se utilizan para computación en paralelo, cálculos científicos y entrenamiento de redes neuronales.

- **Cloud Computing:**

  Consiste en construir “supercomputadoras” usando hardware común (commodity hardware) y tecnologías de virtualización y contenedorización, lo que permite gestionar sistemas distribuidos a gran escala.

- **Convergencia en Linux:**

  Tanto en arquitecturas modernas como en la nube, Linux y otros sistemas Unix-like son la base de muchos servicios actuales.

---

4. Funciones y Metáforas del Sistema Operativo

El SO cumple funciones esenciales que pueden entenderse a través de tres metáforas:

- **Referee (Árbitro):**
  - **Gestión de Recursos:** Distribuye y asigna recursos físicos entre aplicaciones.
  - **Aislamiento:** Separa procesos para evitar que los errores de uno afecten a otros.
  - **Seguridad:** Protege el sistema y los programas de comportamientos erróneos o maliciosos.
- **Illusionist (Ilusionista):**
  - **Abstracción del Hardware:** Provee la ilusión de tener recursos “infinitos” (como memoria) y de disponer de un procesador exclusivo, facilitando el desarrollo sin preocuparse por la configuración física.
- **Glue (Conector):**
  - **Servicios Comunes:** Permite la interacción y el intercambio de datos entre programas (por ejemplo, mediante el portapapeles o acceso a archivos) y mantiene una interfaz homogénea entre aplicaciones y dispositivos.

---

5. Caso de Estudio: Unix y Linux

- **Unix:**
  - Nació en 1969 y sentó las bases del diseño de sistemas operativos modernos.
  - Se caracteriza por una interfaz reducida pero poderosa, donde “casi todo es un archivo”.
  - Su filosofía enfatiza la simplicidad, la modularidad, y el uso de herramientas especializadas que se combinan para lograr tareas complejas.
- **Linux y el Kernel:**
  - Inspirado en Unix, Linux introduce el concepto de *kernel* que actúa como puente entre las aplicaciones (user-land) y el hardware.
  - El kernel gestiona la ejecución de procesos, el manejo de la memoria y las llamadas al sistema (system calls).

---

6. Abstracciones Fundamentales en Unix

- **Archivos y File Descriptors:**
  - **Concepto “Todo es un Archivo”:** En Unix, no solo los archivos físicos, sino dispositivos, sockets, pipes y otros recursos se manejan mediante la misma interfaz de lectura y escritura.
  - **File Descriptors:** Son números enteros que identifican de forma abstracta a estos “archivos”. Por convención, un proceso inicia con tres file descriptors estándar:
    - **0:** Entrada estándar (stdin)
    - **1:** Salida estándar (stdout)
    - **2:** Error estándar (stderr)
  - Los procesos pueden redirigir estos descriptores, lo que permite, por ejemplo, que la salida de un programa se guarde en un archivo.
- **Procesos:**

- Un proceso es una instancia de un programa en ejecución, que posee su propio espacio de memoria (código, datos, stack, heap) y una tabla de file descriptors.
- El **Process Table** es una estructura interna del kernel que almacena información (como el PID) de cada proceso.

---

7. Llamadas al Sistema (System Calls) Importantes

Los SO basados en Unix proveen un conjunto de llamadas al sistema que permiten la gestión de procesos y recursos:

- **fork():**

  Crea un nuevo proceso (hijo) a partir del proceso actual (padre). El proceso hijo es una copia exacta del padre, salvo por el PID.

  - **Ejemplo:** Tras un fork, ambos procesos continúan su ejecución de forma independiente.
  - *Punto clave:* El padre recibe el PID del hijo y el hijo recibe 0.

- **wait():**

  Permite al proceso padre esperar la finalización de uno de sus hijos, obteniendo información sobre el estado de finalización.

- **getpid() y getppid():**

  Devuelven el ID del proceso actual y el del proceso padre, respectivamente.

- **execve():**

  Reemplaza el contenido del proceso actual por un nuevo programa.

  - **Nota:** No crea un nuevo proceso; mantiene el mismo PID y los file descriptors abiertos, lo que permite redirigir entradas/salidas sin necesidad de reconfigurar.

- **exit():**

  Termina el proceso actual, devolviendo un código de salida al sistema.

- **dup() y dup2():**

  Duplican un file descriptor, permitiendo que dos descriptores hagan referencia al mismo “archivo” (recurso).

- **pipe():**

  Crea un canal de comunicación unidireccional entre procesos.

  - Se genera un par de file descriptors: uno para escribir y otro para leer, permitiendo que un proceso envíe datos a otro, por ejemplo, para conectar la salida de un proceso con la entrada de otro (como en la utilización de pipes en la terminal).

---

8. Comentarios Adicionales

- **Virtualización en Profundidad:**

  La virtualización es el mecanismo central que permite al SO ocultar la complejidad del hardware. Esto no solo simplifica la programación, sino que también permite técnicas avanzadas como la multitarea y la concurrencia.

- **Diseño Modular de Unix:**

  La filosofía Unix insiste en que cada programa debe hacer una tarea única y bien definida. Esta modularidad facilita la interoperabilidad, ya que la salida de un programa puede ser fácilmente utilizada como entrada para otro.

Resumen Final

Los Sistemas Operativos actúan como mediadores entre el hardware y las aplicaciones, proporcionando una abstracción que permite la ejecución concurrente, segura y eficiente de múltiples procesos. Mediante la virtualización, el SO transforma recursos físicos en interfaces virtuales, facilitando el desarrollo y la ejecución de programas. Unix, como caso de estudio, ejemplifica estos principios a través de conceptos como “todo es un archivo”, el uso de file descriptors y una serie de llamadas al sistema que permiten la creación, gestión y comunicación entre procesos. Estos fundamentos son esenciales para comprender tanto el funcionamiento interno de un SO como para el desarrollo de aplicaciones robustas y eficientes.