




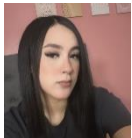

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Sistemas Operativos

Actividad Fundamental 2

Docente: Dra. Norma Edith Marín Martínez
Grupo: 002

Nombre	Matricula	Carrera	
Francisco Gael Reyes Cantú	1995983	IAS	
Frida Jaziry Juarez Fuentes	2028420	IAS	
Isac Alfredo Almaguer Espinosa	2049903	IAS	
Fátima Arizpe Sánchez	2025106	IAS	
Jose Angel Cardenas Contreras	1935156	IAS	

Índice

<i>Introducción.</i>	3
<i>¿Qué es la Concurrencia en Sistemas Operativos?</i>	4
<i>Tipos de Concurrencia en Sistemas Operativos</i>	6
<i>Programación Concurrente</i>	11
<i>Ventajas de la programación concurrente</i>	14
<i>Desventajas de la programación concurrente</i>	15
<i>Tipos de interacciones entre los procesos dentro de la concurrencia</i>	16
<i>Cuadro Sinóptico</i>	17
<i>Conclusion</i>	18
<i>Conclusiones:</i>	19
<i>Bibliografías:</i>	21

Introducción.

En el corazón de los sistemas operativos modernos yace un principio clave: la concurrencia. Esta capacidad permite a un sistema ejecutar múltiples procesos o tareas de manera simultánea, maximizando el aprovechamiento de los recursos y mejorando la eficiencia del sistema. Sin concurrencia, las aplicaciones serían lentas, ineficientes y estarían limitadas a una ejecución estrictamente secuencial.

A lo largo de la evolución tecnológica, **la concurrencia** ha dado origen a diversas estrategias y modelos, como la multiprogramación, el multiprocesamiento y el procesamiento distribuido. Estas técnicas han revolucionado la forma en que los sistemas manejan múltiples tareas, permitiendo que un solo procesador administre eficientemente varios procesos o que múltiples procesadores trabajen en conjunto para mejorar el rendimiento.

En el ámbito del desarrollo de software, la programación concurrente se ha convertido en una habilidad esencial. Aplicaciones modernas, desde sistemas operativos hasta servicios en la nube y videojuegos, dependen de la ejecución simultánea de múltiples tareas para ofrecer una experiencia fluida y ágil.

Sin embargo, la concurrencia no está exenta de desafíos. La sincronización de procesos, la gestión de recursos compartidos y la prevención de problemas como condiciones de carrera y bloqueos son aspectos críticos que deben abordarse con precisión. Un diseño deficiente puede derivar en errores difíciles de detectar y corregir, afectando la estabilidad del sistema.

Dominar la concurrencia es clave para desarrollar software robusto, eficiente y escalable. En este contexto, comprender sus principios, técnicas y mejores prácticas es fundamental para cualquier ingeniero de software que busque optimizar el rendimiento y la confiabilidad de sus aplicaciones.

¿Qué es la Concurrency en Sistemas Operativos?

En el ámbito de los sistemas operativos, **la concurrencia** se refiere a la capacidad de ejecutar múltiples procesos o hilos de manera simultánea o en un mismo período de tiempo. Esto permite aprovechar mejor los recursos del sistema y mejorar el rendimiento de las aplicaciones.

Aunque pueda parecer que todas las tareas se ejecutan al mismo tiempo, en realidad, esto depende del número de procesadores disponibles. En un sistema con un solo procesador, la concurrencia se logra mediante la intercalación de la ejecución de procesos, donde el sistema operativo asigna pequeños fragmentos de tiempo a cada proceso (esto se conoce como multiprogramación o multitarea). En cambio, en sistemas con múltiples procesadores o núcleos, es posible ejecutar procesos realmente en paralelo.

-Ejemplo de Concurrency vs. Paralelismo.

Para entender mejor, imaginemos una cafetería con un solo barista (un solo procesador). El barista puede atender a varios clientes de manera concurrente cambiando entre ellos rápidamente, preparando un café mientras toma la orden de otro cliente. Sin embargo, como solo hay un barista, no puede hacer dos cafés a la vez, solo alterna entre tareas.

Si la cafetería contrata más baristas (múltiples procesadores), ahora sí pueden trabajar en paralelo, preparando varias órdenes al mismo tiempo.

¿Cómo Logran los Sistemas Operativos la Concurrency?

Los sistemas operativos emplean varias estrategias para gestionar la concurrencia de manera eficiente:

1. Multiprogramación: Alternan rápidamente entre diferentes procesos en un solo procesador.
2. Multitarea (time-sharing): Utilizan planificación de procesos para dividir el tiempo del CPU entre varios procesos activos.
3. Multiprocesamiento: Distribuyen la carga de trabajo entre varios procesadores o núcleos físicos.
4. Hilos (threads): Permiten dividir un proceso en subprocesos más pequeños que pueden ejecutarse en paralelo o intercaladamente.

-Desafíos de la Concurrency.

La concurrencia no solo mejora el rendimiento, sino que también introduce problemas complejos, como:

- Condiciones de carrera: Cuando dos o más procesos acceden a un recurso compartido sin la debida sincronización, lo que puede causar resultados inesperados.
- Interbloqueo (Deadlock): Ocurre cuando dos o más procesos quedan esperando indefinidamente porque cada uno tiene un recurso que el otro necesita.
- Starvation: Algunos procesos pueden quedar sin ejecución si otros acaparan los recursos por mucho tiempo.

Para solucionar estos problemas, los sistemas operativos utilizan mecanismos como semáforos, monitores y exclusión mutua (mutex) para controlar el acceso a los recursos compartidos.

Podemos llegar a la conclusión de que la concurrencia es una característica esencial en los sistemas operativos modernos, ya que permite optimizar el uso de recursos y mejorar la eficiencia del sistema. Sin embargo, su implementación requiere estrategias cuidadosas para evitar problemas de sincronización y garantizar que todos los procesos trabajen de manera estable y segura.



Tipos de Concurrency en Sistemas Operativos

La concurrencia en sistemas operativos se puede clasificar según el nivel en el que se implementa y cómo se gestiona la ejecución de múltiples procesos o hilos. Existen varios modelos de concurrencia, cada uno con características y aplicaciones específicas.

A continuación, se presentan los principales tipos de concurrencia:

1. Concurrency a Nivel de Procesos

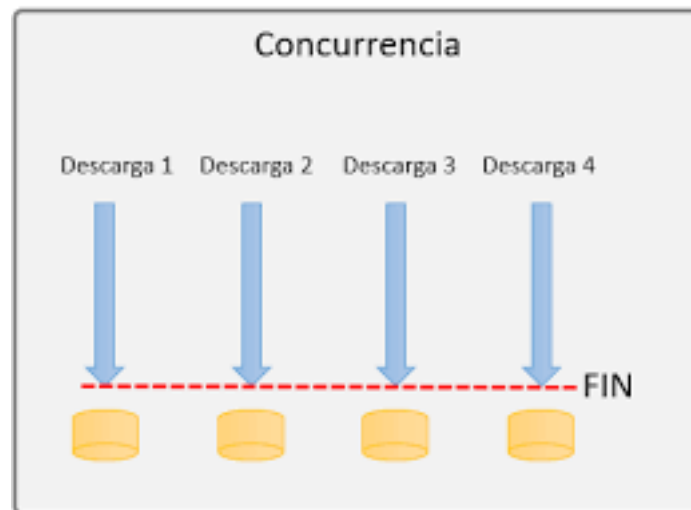
Este tipo de concurrencia ocurre cuando varios procesos independientes se ejecutan en un mismo sistema operativo, compartiendo recursos como el procesador, la memoria y los dispositivos de entrada/salida.

◆ Características:

- ✓ Cada proceso tiene su propio espacio de memoria.
- ✓ La comunicación entre procesos se realiza mediante mecanismos como pipes, colas de mensajes, memoria compartida o sockets.
- ✓ La planificación del procesador es administrada por el sistema operativo.

📌 Ejemplo:

En una computadora con un solo procesador, un usuario puede ejecutar simultáneamente un navegador web, un reproductor de música y un editor de texto. Aunque solo un proceso puede ejecutarse en un instante dado, el sistema operativo intercala su ejecución rápidamente, dando la ilusión de simultaneidad.



2. Concurrency a Nivel de Hilos (Threads)

A diferencia de los procesos, los **hilos (threads)** son unidades más pequeñas de ejecución dentro de un proceso. Todos los hilos de un mismo proceso comparten el mismo espacio de memoria y recursos, lo que permite una ejecución más eficiente.

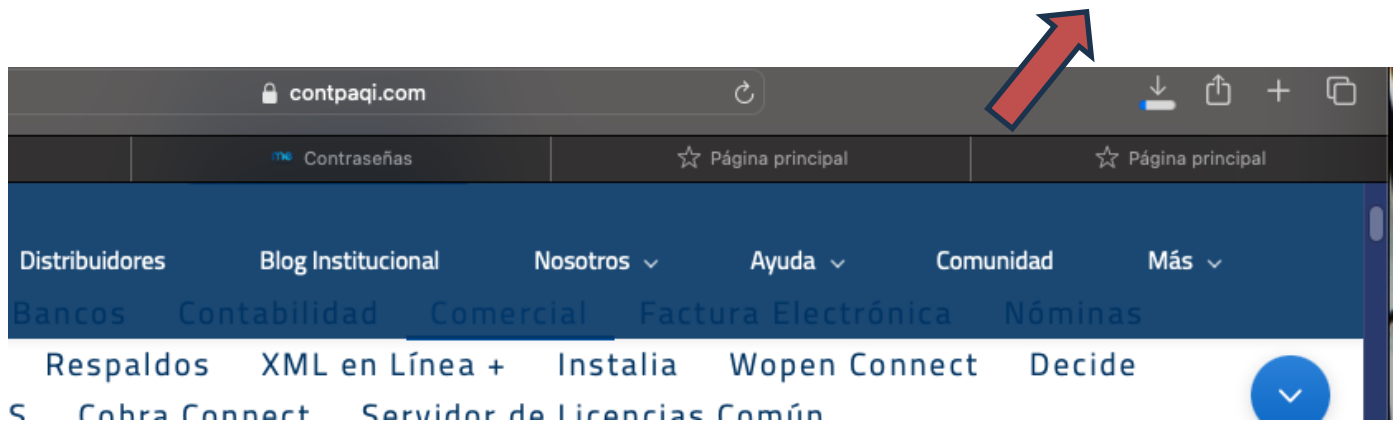
◆ Características:

- ✓ Los hilos dentro de un proceso pueden ejecutar tareas en paralelo sin necesidad de una comunicación compleja.
- ✓ Se pueden usar para dividir una tarea grande en subtareas más pequeñas y ejecutarlas de manera concurrente.
- ✓ El sistema operativo puede planificar los hilos individualmente o permitir que la propia aplicación lo haga.

📌 Ejemplo:

Un navegador web usa múltiples hilos:

- Un hilo maneja la interfaz de usuario.
- Otro hilo descarga archivos en segundo plano.
- Otro hilo procesa scripts en una página web.



3. Concurrency en Sistemas de Multiprocesamiento

Este tipo de concurrencia ocurre en sistemas con múltiples procesadores o núcleos, donde las tareas realmente se ejecutan en paralelo, no solo intercaladas.

◆ Características:

- ✓ Permite la ejecución simultánea de múltiples procesos o hilos en diferentes CPUs.
- ✓ Mejora el rendimiento del sistema, ya que varias tareas se ejecutan realmente al mismo tiempo.
- ✓ Se requiere un sistema operativo capaz de manejar múltiples procesadores y distribuir la carga de trabajo de manera eficiente.

📌 Ejemplo:

Los servidores modernos utilizan sistemas **multiprocesador** para gestionar miles de solicitudes web simultáneamente, distribuyendo la carga entre varios núcleos para mejorar la velocidad y la eficiencia.



4. Concurrency en Sistemas Distribuidos

Aquí la concurrencia ocurre en un conjunto de computadoras conectadas en red, en lugar de ejecutarse en un solo sistema operativo.

◆ Características:

- ✓ Los procesos concurrentes se ejecutan en diferentes dispositivos que se comunican a través de la red.
- ✓ La comunicación se realiza mediante protocolos de red como TCP/IP o RPC (Remote Procedure Call).

✓ Se usa en sistemas de computación en la nube, bases de datos distribuidas y arquitecturas microservicio.

📌 Ejemplo:

Plataformas como Netflix o YouTube distribuyen el procesamiento de video en múltiples servidores alrededor del mundo para servir contenido a millones de usuarios sin saturar un solo sistema.



5. Concurrency en Sistemas de Entrada/Salida

En este caso, la concurrencia se da cuando múltiples dispositivos de entrada/salida operan simultáneamente mientras el procesador sigue ejecutando otros procesos.

◆ Características:

✓ Permite realizar operaciones de lectura/escritura sin bloquear la ejecución de otros procesos.

✓ Se utiliza técnicas como buffering, caching e interrupciones para mejorar la eficiencia.

📌 Ejemplo:

Un usuario puede copiar archivos a una memoria USB mientras escucha música y navega por internet en la misma computadora sin que ninguna de estas tareas interfiera con la otra.



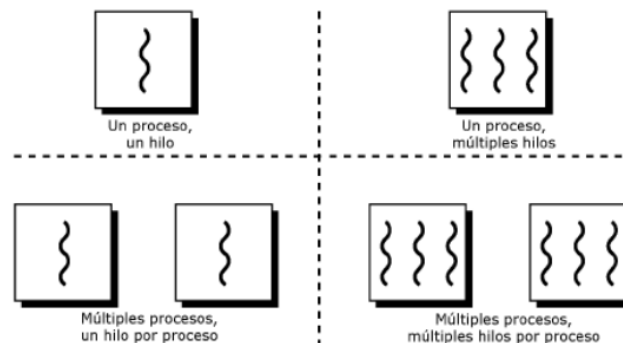
Programación Concurrente

La **programación concurrente** es como organizar una fiesta en la que todos quieren hablar al mismo tiempo. Imagina que tienes un grupo de amigos (llamémoslos "procesos" o "hilos") que quieren hacer cosas simultáneamente. Algunos quieren poner música, otros quieren servir comida, y otros simplemente quieren bailar. El problema es que, si todos hablan o actúan al mismo tiempo sin coordinación, el caos está garantizado. Aquí es donde entra la programación concurrente: es la forma en que le dices a tus amigos (o a tu programa) cómo organizarse para que todo funcione sin que se pisen los unos a los otros.

Hilos y Procesos: Los Protagonistas

En la programación concurrente, los "hilos" (threads) y los "procesos" son los actores principales. Un proceso es como un programa independiente que tiene su propio espacio en la memoria, mientras que un hilo es una parte de un proceso que puede ejecutarse de manera concurrente con otros hilos. Los hilos comparten memoria y recursos, lo que los hace más ligeros que los procesos, pero también más propensos a meterse en problemas si no se manejan bien.

Por ejemplo, imagina que tienes un programa que descarga varias imágenes de Internet. Podrías crear un hilo para cada descarga, de modo que todas las imágenes se descarguen al mismo tiempo. Pero si los hilos intentan acceder al mismo recurso (como escribir en el mismo archivo), podría terminar en un gran problema. Aquí es donde entran los mecanismos de sincronización.



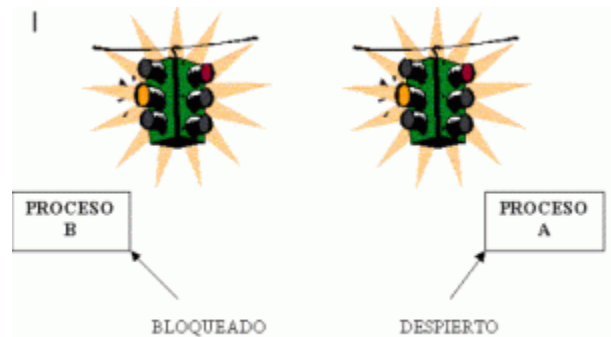
Sincronización

La sincronización es como poner reglas en tu fiesta para que no todos hablen al mismo tiempo. En programación concurrente, esto se logra con herramientas como **semáforos**, **mutexes** y **monitores**. Estas herramientas aseguran que solo un hilo acceda a un recurso compartido en un momento dado.

- Un **mutex** (abreviatura de "exclusión mutua") es como un micrófono en una reunión: solo una persona puede hablar a la vez. Si un hilo tiene el mutex, los demás tienen que esperar su turno.
- Un **semáforo** solo deja entrar a un número limitado de personas a la vez. En

programación, un semáforo limita cuántos hilos pueden acceder a un recurso simultáneamente.

- Un **monitor** es como un moderador que supervisa las interacciones entre los hilos para evitar conflictos.



Sin estas herramientas, podrías enfrentarte a problemas como las **condiciones de carrera**, donde el resultado del programa depende del orden en que los hilos acceden a los recursos. Esto es como si dos amigos intentaran servir comida al mismo tiempo y terminarían derramando todo.

Memoria compartida

Los módulos concurrentes interactúan **leyendo y escribiendo objetos compartidos en la memoria**:

- Dos procesadores o núcleos CPU en el mismo computador que comparten la misma memoria física.
- Dos programas que se están ejecutando en el mismo ordenador que comparten el sistema de ficheros y que están leyendo y escribiendo en distintos archivos.
- Dos hilos (*threads*) de un programa codificado en Java que comparten los mismos objetos.

Paso de mensajes

Los módulos concurrentes interactúan **enviándose mensajes entre sí** a través de un canal de comunicación. Los módulos mandan mensajes y los mensajes entrantes a cada módulo se ponen en cola para su procesamiento.

- Dos equipos que se están comunicando mediante una conexión de red.
- Un navegador y un servidor web que se están comunicando para mostrar una página web solicitada.
- Un cliente y un servidor de mensajería instantánea.

- Dos programas que se están ejecutando en el mismo computador cuya entrada y salida han sido conectadas a través de una tubería, p. ej. en Linux el comando `/s / grep` escrito en el terminal.

Los modelos de paso de mensajes y de memoria compartida tratan sobre cómo se comunican los módulos concurrentes. **Los módulos concurrentes en sí son de dos tipos:**

- **Proceso**
- **Hilo o *thread***

Lenguajes y Herramientas para la Concurrencia

Diferentes lenguajes de programación abordan la concurrencia de maneras distintas. Algunos, como **Java** y **C#**, tienen soporte nativo para hilos y sincronización. Otros, como **Python**, tienen limitaciones debido al **GIL** (Global Interpreter Lock), que impide que múltiples hilos ejecuten código Python al mismo tiempo. Sin embargo, Python ofrece alternativas como el módulo `multiprocessing` para lograr paralelismo.

Lenguajes como **Go** y **Rust** han ganado popularidad por su manejo moderno de la concurrencia. Go, por ejemplo, usa **goroutines**, que son hilos ligeros gestionados por el runtime del lenguaje. Rust, por otro lado, se enfoca en la seguridad de la memoria y evita muchos de los problemas comunes de la programación concurrente mediante su sistema de propiedad.

Problemas de la Programación Concurrente

La programación concurrente no es fácil. Además de las condiciones de carrera, hay otros problemas como el **deadlock** (bloqueo mutuo), donde dos hilos se quedan esperando el uno al otro indefinidamente. Imagina que dos amigos están sosteniendo un plato cada uno y necesitan el plato del otro para servir la comida. Si ninguno suelta su plato, ambos se quedan atascados.

Otro problema es el **starvation** (inanición), donde un hilo no puede acceder a los recursos que necesita porque otros hilos siempre los están usando. Es como si en tu fiesta, alguien siempre se quedara con el micrófono y no dejara hablar a los demás.

Un sistema operativo que funcione de manera secuencial y determinista es impensable. Con la enorme cantidad de actividades que debe de gestionar simultáneamente, como asignar memoria, atender eventos I/O de distintos dispositivos periféricos (teclado, mouse, monitor, impresora, etc.), ejecutar procesos y encargarse de funciones de red, resulta patente la necesidad de una solución recurrente y que además sea eficiente, puesto que de su correcto funcionamiento depende el funcionamiento de todas las demás aplicaciones que se ejecuten en el sistema.

La programación concurrente es una herramienta poderosa, pero también es un campo lleno de desafíos. Requiere pensar de manera diferente y estar atento a los posibles problemas que surgen cuando múltiples tareas compiten por los mismos recursos.



Ventajas de la programación concurrente

- **Mejor Utilización de los Recursos del Sistema** La programación concurrente permite que varias tareas se ejecuten al mismo tiempo, maximizando la utilización de los recursos disponibles, como los procesadores y la memoria. En sistemas con múltiples núcleos o procesadores, las tareas pueden ejecutarse simultáneamente, lo que mejora la eficiencia del sistema. Según Tanenbaum y Bos (2014), la concurrencia optimiza el uso de los recursos del sistema, permitiendo que los procesos de larga duración no bloqueen la ejecución de otros procesos.
- **Reducción del Tiempo de Ejecución** La programación concurrente puede reducir el tiempo total de ejecución de un programa al dividir una tarea grande en partes más pequeñas que pueden ejecutarse simultáneamente. Esto es especialmente útil en aplicaciones de alto rendimiento, como procesamiento de datos o simulaciones científicas. Como se menciona en el trabajo de Culler y Singh (1999), la capacidad de ejecutar tareas en paralelo reduce el tiempo necesario para completar procesos complejos.
- **Mejora de la Escalabilidad** En sistemas distribuidos o en aplicaciones que requieren una alta escalabilidad, la programación concurrente permite que el sistema maneje un número mayor de usuarios o procesos de manera más eficiente. Al permitir que múltiples procesos se ejecuten en paralelo, la escalabilidad mejora, ya que se pueden agregar más recursos sin que el rendimiento se degrade. Esto se observa en el análisis de sistemas en paralelo descrito por Hennessy y Patterson (2011).
- **Mayor Responsividad en Aplicaciones Interactivas** La programación concurrente es esencial en aplicaciones interactivas, como las interfaces de usuario (UI) de los sistemas operativos o aplicaciones web. Permite que los programas sigan siendo interactivos, incluso mientras se están realizando operaciones en segundo plano. Según Herlihy y Shavit (2012), la concurrencia en aplicaciones interactivas mejora la experiencia del usuario al permitir que las tareas de la interfaz de usuario y las tareas de fondo se ejecuten sin que una bloquee a la otra.
- **Mejor Manejo de Tareas Independientes** La programación concurrente es ideal para dividir tareas que son independientes entre sí y pueden ejecutarse sin interferencias. Esto mejora el rendimiento de aplicaciones que involucran múltiples tareas paralelas que no dependen unas de otras. Como se discute en el trabajo de Lee et al. (2010), la independencia de tareas permite un manejo eficiente y efectivo de los recursos.

Desventajas de la programación concurrente

- **Complejidad en el Desarrollo** La programación concurrente es más difícil de implementar y depurar que la programación secuencial debido a los problemas relacionados con la sincronización, la comunicación entre hilos y el manejo de recursos compartidos. Según Tanenbaum y Bos (2014), los errores de concurrencia, como las condiciones de carrera o los bloqueos mutuos (deadlocks), son difíciles de identificar y corregir, lo que aumenta la complejidad en el desarrollo de programas concurrentes.
- **Condiciones de Carrera y Problemas de Sincronización** En un entorno concurrente, las tareas que comparten recursos pueden entrar en conflicto, lo que puede llevar a condiciones de carrera y otros problemas relacionados con la sincronización. Estos problemas ocurren cuando varios hilos intentan acceder y modificar datos compartidos de manera no controlada. Herlihy y Shavit (2012) explican que, si no se manejan adecuadamente, las condiciones de carrera pueden resultar en comportamientos impredecibles y errores difíciles de reproducir.
- **Sobrecarga de Contexto y de Gestión de Hilos** La gestión de múltiples hilos de ejecución en un programa concurrente puede introducir una sobrecarga significativa. Cada cambio de contexto entre hilos consume recursos del sistema, y la creación y destrucción de hilos puede ser costosa en términos de tiempo y memoria. Como mencionan Culler y Singh (1999), la sobrecarga de la gestión de hilos puede reducir los beneficios de la concurrencia, especialmente si no se controla adecuadamente.
- **Dificultad para Depurar y Probar** La depuración de aplicaciones concurrentes es considerablemente más difícil que la depuración de aplicaciones secuenciales debido a la naturaleza no determinista de la ejecución concurrente. Los errores pueden no ocurrir siempre, lo que dificulta su identificación. De acuerdo con Hennessy y Patterson (2011), las pruebas de aplicaciones concurrentes son complicadas porque es difícil asegurar que todas las posibles interacciones entre hilos se hayan probado completamente.
- **Deadlocks y Livelocks** Uno de los mayores desafíos en la programación concurrente es evitar situaciones de bloqueo mutuo (deadlocks) y de bloqueo activo (livelocks). Un **deadlock** ocurre cuando dos o más hilos se bloquean mutuamente esperando por recursos que están siendo bloqueados por otros hilos, mientras que un **livelock** ocurre cuando los hilos siguen ejecutando pero sin hacer progreso. Estos problemas pueden llevar a una disminución del rendimiento o incluso a la detención total del sistema, lo que requiere una planificación cuidadosa para evitar que ocurran (Tanenbaum & Bos, 2014).



Procesos e hilos

Un proceso es un programa en ejecución. Es la unidad fundamental de trabajo que gestiona un sistema operativo. Cada proceso tiene su propio espacio de memoria y recursos asociados (como archivos abiertos y variables de entorno), lo que significa que es relativamente independiente de otros procesos.

- Instancia en un programa en ejecución
- Es cuando las instrucciones (líneas de código) de un programa son ejecutadas por el procesador
- Un proceso tiene asignados recursos como tiempo de CPU, archivos abiertos y dispositivos de entrada/salida




Un hilo (o subproceso) es la unidad más pequeña de ejecución dentro de un proceso. Un proceso puede contener uno o más hilos que comparten el mismo espacio de memoria y recursos, pero que pueden ejecutarse de manera independiente y simultánea

- Un proceso puede crear múltiples hilos
- Los hilos de un mismo proceso comparten ciertos recursos o contexto y tienen otros recursos propios

Los programas cuando se desarrollan se basan en programación concurrente o programación paralela.

Tipos de interacciones entre los procesos dentro de la concurrencia

Los procesos de una aplicación concurrente pueden interactuar entre sí de acuerdo con los siguientes esquemas:

-  Independientes entre sí: Interfieren por compartir el procesador.
-  Cooperan entre sí: Uno genera una información o realiza algún servicio que el segundo necesita.
-  Compiten entre sí: Requieren usar recursos comunes en régimen exclusivo.

Cuadro Sinóptico



Conclusion

La concurrencia es uno de los pilares fundamentales en el diseño y funcionamiento de los sistemas operativos modernos. Gracias a ella, los sistemas pueden ejecutar múltiples procesos y tareas de manera simultánea o intercalada, optimizando el uso de los recursos disponibles y mejorando la experiencia del usuario. Sin la concurrencia, las computadoras y dispositivos actuales serían ineficientes, incapaces de realizar múltiples tareas a la vez, lo que limitaría drásticamente su rendimiento y utilidad.

A lo largo de la evolución de la computación, la concurrencia ha dado lugar a distintos enfoques y técnicas, como la multiprogramación, la multitarea, el multiprocesamiento y los sistemas distribuidos. Cada uno de estos modelos ha permitido mejorar la capacidad de los sistemas para manejar múltiples procesos o hilos, ya sea en un solo procesador, en múltiples núcleos o incluso en sistemas distribuidos a nivel global. Estos avances han sido cruciales en el desarrollo de tecnologías como la computación en la nube, los servidores web y las aplicaciones móviles, donde la capacidad de gestionar múltiples tareas simultáneamente es esencial.

No obstante, la concurrencia no está exenta de desafíos. Problemas como las condiciones de carrera, los interbloqueos (deadlocks) y la inanición (starvation) pueden afectar gravemente el desempeño y la estabilidad de un sistema si no se manejan adecuadamente. Para mitigar estos riesgos, los sistemas operativos implementan mecanismos de sincronización como semáforos, mutex, monitores y algoritmos de planificación de procesos, garantizando así que los procesos y hilos puedan compartir recursos de manera segura y eficiente.

Además, la importancia de la concurrencia no se limita solo al nivel de los sistemas operativos, sino que también se extiende al desarrollo de software en general. Los lenguajes de programación modernos incluyen bibliotecas y herramientas especializadas para la programación concurrente y paralela, lo que permite a los desarrolladores diseñar aplicaciones más rápidas y eficientes. Desde videojuegos hasta sistemas financieros y bases de datos, la concurrencia juega un papel clave en el rendimiento de las aplicaciones modernas.

En conclusión, la concurrencia es un concepto esencial en la informática actual, ya que permite mejorar la eficiencia y el rendimiento de los sistemas operativos y las aplicaciones. Sin embargo, su implementación requiere un diseño cuidadoso y estrategias adecuadas para evitar problemas de sincronización y garantizar la estabilidad del sistema. En un mundo donde la tecnología avanza rápidamente y la demanda de procesamiento eficiente sigue en aumento, comprender y aplicar los principios de la concurrencia es una habilidad indispensable para cualquier ingeniero en computación o desarrollador de software.

Link video

https://www.canva.com/design/DAGflxiTe6o/XRtsXm5eVcuPKrtxLYuz8g/watch?utm_content=DAGflxiTe6o&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=hc233e88703

Conclusiones:

Francisco Gael Reyes Cantú 1995983 IAS

La concurrencia es un concepto clave en la informática moderna, permitiendo que los sistemas operativos y las aplicaciones ejecuten múltiples tareas de manera eficiente. Sin embargo, su implementación no es fácil, ya que implica desafíos como la sincronización de procesos y la prevención de errores críticos como condiciones de carrera o interbloqueos.

Personalmente, considero que entender la concurrencia es fundamental para cualquier desarrollador o ingeniero en computación, ya que optimiza el rendimiento de los sistemas y permite crear software más rápido y escalable. En un mundo donde la computación en la nube y la inteligencia artificial dependen de la ejecución simultánea de múltiples procesos, la concurrencia no es solo una ventaja, sino una necesidad.

Frida Jaziry Juarez Fuentes 2028420 IAS

La programación concurrente es una herramienta poderosa que mejora el rendimiento de las aplicaciones al permitirles ejecutar múltiples tareas al mismo tiempo. Aunque ofrece grandes ventajas, como la optimización de recursos y la mayor escalabilidad, también presenta desafíos, como la complejidad en su implementación y los problemas de sincronización. Sin embargo, con práctica y el uso adecuado de herramientas, estos problemas pueden manejarse. En un mundo cada vez más demandante de eficiencia, aprender y dominar la programación concurrente se vuelve esencial para crear aplicaciones rápidas y robustas.

Isac Alfredo Almaguer Espinosa 2049903 IAS

En esta actividad aprendí mucho acerca de la programación concurrente, aprendí la diferencia entre paralelismo y concurrencia, sus diferencias y sobre todo de que se compone la programación concurrente, los procesos que conlleva la concurrencia y se me hizo muy curioso que una de las cosas que vemos diariamente este hecha con programación concurrente como lo son los semáforos, siempre había pensado el cómo se lograba que se sincronizarán todos los semáforos de la ciudad, que es justamente uno de los temas de la concurrencia la sincronización. En general aprendí sobre los hilos, procesos, los desafíos que conlleva la programación concurrente y los lenguajes en los que se aplica está.

FATIMA ARIZPE SANCHEZ 2025106

La programación concurrente es un enfoque que permite ejecutar múltiples tareas o procesos de manera simultánea, mejorando la eficiencia y el rendimiento de las aplicaciones, especialmente en sistemas con múltiples núcleos o procesadores. La clave está en dividir un problema en partes que se puedan resolver de forma independiente o en paralelo, lo que facilita la optimización de recursos y reduce el tiempo de ejecución. Sin embargo, también presenta desafíos, como la sincronización de los procesos para evitar conflictos y garantizar la correcta ejecución. En resumen, la programación concurrente es fundamental para aprovechar al máximo la potencia de las máquinas modernas, pero requiere una buena

gestión de recursos y un diseño cuidadoso para evitar errores.

Jose Angel Cardenas Contreras 1935156

Los procesos en Linux están diseñados para ser altamente manejables, permitiendo una administración detallada. la arquitectura basada en procesos de Linux fomenta la multitarea y la seguridad mediante espacios de memoria aislados, lo que evita que un proceso afecte directamente a otro. Sin embargo, una mala gestión de procesos, como la ejecución de demasiados procesos intensivos sin una asignación de prioridades adecuada, puede provocar problemas de rendimiento y consumo excesivo de recursos. la gestión de procesos en Linux es una de sus fortalezas, ya que permite un control detallado y eficiente, pero requiere conocimientos adecuados para optimizar su uso y evitar conflictos o desperdicio de recursos del sistema.

Bibliografías:

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10ª ed.). Wiley. Disponible en: <https://archive.org/details/operating-system-concepts-10th>

Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4ª ed.). Pearson. Disponible en: <https://github.com/lighthouseand/books/blob/master/Modern%20Operating%20Systems%204th%20Edition-Andrew%20Tanenbaum.pdf>

Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9ª ed.). Pearson. Más información en: <https://williamstallings.com/OperatingSystems/>

Bacon, J. (2003). *Concurrent Systems: Operating Systems, Database and Distributed Systems* (2ª ed.). Addison-Wesley. Más información en: <https://www.pearson.com/us/higher-education/program/Bacon-Concurrent-Systems-2nd-Edition/PGM63987.html>

Andrews, G. R. (2000). *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley. Más información en: <https://www.pearson.com/us/higher-education/program/Andrews-Foundations-of-Multithreaded-Parallel-and-Distributed-Programming/PGM63980.html>

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed Systems: Concepts and Design* (5ª ed.). Pearson. Más información en: <https://www.pearson.com/us/higher-education/program/Coulouris-Distributed-Systems-Concepts-and-Design-5th-Edition/PGM79164.html>

Goetz, B. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional.

Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.

Rust Community. (2023). *The Rust Programming Language*. Recuperado de <https://doc.rust-lang.org/book/>

Python Software Foundation. (2023). *Python Documentation: threading — Thread-based parallelism*. Recuperado de <https://docs.python.org/3/library/threading.html>

Go Documentation. (2023). *Effective Go: Concurrency*. Recuperado de https://golang.org/doc/effective_go#concurrency

Programación concurrente UNAL. (n.d.). Github.io. Retrieved February 8, 2025, from https://ferestrepoca.github.io/paradigmas-de-programacion/progconcurrente/concurrente_teor%C3%ADa/index.html