Centro Universitario de Ciencias Exactas e Ingenierías.

Departamento para la integración ciber humana.

Sistemas operativos.

Becerra Velázquez Violeta Rocío.

SAUL EMMANUEL YAÑEZ SALAZAR
ENRIQUE RODRIGUEZ SALAZAR
JONATHAN BRANDON JIMENEZ ALMAGUER
MUÑOZ LOPEZ MAURILIO ISRAEL

Ingeniería en computación.

D04

Preguntas sobre tema asignado.

Domingo 16 de marzo del 2025.

La concurrencia en los sistemas operativos es la capacidad de gestionar múltiples tareas que se ejecutan de manera simultánea o solapada, permitiendo un uso más eficiente de los recursos del sistema. Este concepto es clave en la informática moderna, ya que facilita la ejecución paralela de procesos e hilos, optimizando el rendimiento y la respuesta del sistema ante múltiples solicitudes de usuarios.

Uno de los principales beneficios de la concurrencia es la maximización del uso de la CPU y otros recursos, lo que reduce los tiempos de inactividad y mejora la eficiencia general. Además, permite que los sistemas interactivos respondan más rápido a los usuarios, lo que es crucial en entornos donde múltiples solicitudes deben ser gestionadas simultáneamente. Sin embargo, la concurrencia también introduce desafíos significativos. Las condiciones de carrera ocurren cuando varios procesos acceden simultáneamente a recursos compartidos sin la debida sincronización, lo que puede generar resultados inconsistentes. Otro problema común es el interbloqueo o deadlock, donde dos o más procesos quedan detenidos indefinidamente esperando recursos que están siendo utilizados por otros procesos. Asimismo, la inanición o starvation puede ocurrir cuando un proceso no logra acceder a los recursos que necesita debido a que otros procesos tienen prioridad y acaparan dichos recursos.

Para manejar estos problemas, los sistemas operativos implementan mecanismos de sincronización como los semáforos, que controlan el acceso a recursos compartidos y evitan condiciones de carrera. También se emplean monitores, estructuras de alto nivel que garantizan que solo un proceso pueda acceder a un recurso compartido a la vez, y bloqueos o locks, que restringen el acceso concurrente a los recursos para mantener la integridad de los datos. La correcta aplicación de estos mecanismos es fundamental para garantizar la estabilidad y confiabilidad del sistema en entornos multitarea. (GeeksForGeeks, 2024)

Tipos de procesos concurrentes.

En un sistema operativo, los procesos concurrentes pueden clasificarse en diferentes tipos según su grado de independencia, interacción y la forma en que comparten recursos. Estos tipos de procesos juegan un papel crucial en la administración eficiente del sistema y en la optimización del rendimiento.

Uno de los principales tipos de procesos concurrentes son los procesos independientes**,** que no comparten recursos ni interactúan entre sí. Cada uno opera de manera autónoma sin afectar a los demás, lo que los hace relativamente más sencillos de gestionar. Ejemplos de este tipo de procesos incluyen tareas como la ejecución de programas individuales que no dependen de datos externos o de otros procesos en ejecución.

Por otro lado, existen los procesos cooperativos, que interactúan entre sí y comparten recursos como memoria, archivos o dispositivos de entrada y salida. Estos procesos pueden beneficiarse del intercambio de información, pero también requieren mecanismos adecuados de sincronización para evitar problemas como condiciones de carrera o inconsistencias en los datos. Un ejemplo típico de procesos cooperativos son aquellos que

trabajan bajo el modelo de productor-consumidor, donde un proceso genera datos y otro los consume. (GeeksForGeeks, 2022)

Exclusión mutua.

La exclusión mutua es un mecanismo fundamental en la concurrencia que garantiza que solo un proceso o hilo pueda acceder a un recurso compartido en un momento dado, evitando así problemas como las condiciones de carrera y la inconsistencia de datos. Su correcta implementación es crucial para la estabilidad y eficiencia del sistema operativo, asegurando que los procesos no interfieran entre sí al manipular información compartida.

Existen diversas soluciones para lograr la exclusión mutua. Entre las soluciones basadas en software se encuentran los algoritmos de Dekker y Peterson, los cuales permiten la sincronización de dos procesos mediante variables de control, aunque no son ideales para sistemas modernos debido a la falta de atomicidad en sus operaciones. Por otro lado, las soluciones basadas en hardware incluyen instrucciones atómicas como Test-and-Set, Swap y Compare-and-Swap, que garantizan que una operación se complete sin interrupciones, aunque pueden llevar a problemas de espera activa. En cuanto a las soluciones proporcionadas por el sistema operativo, los semáforos, los monitores y los locks son ampliamente utilizados. Los semáforos pueden ser binarios o de conteo, asegurando el control sobre los accesos a recursos compartidos. Los monitores encapsulan variables y procedimientos para regular la concurrencia, mientras que los locks restringen el acceso a la sección crítica de un proceso para evitar conflictos.

A pesar de sus ventajas, una mala implementación de la exclusión mutua puede generar problemas como el interbloqueo, donde dos o más procesos quedan atrapados esperando recursos que nunca serán liberados, la inanición, en la que procesos de baja prioridad no logran acceder a los recursos porque otros de mayor prioridad los acaparan, y la espera activa, que ocurre cuando un proceso mantiene la CPU ocupada mientras espera su turno en la sección crítica en lugar de ceder su tiempo de ejecución. Para evitar estos problemas, se aplican técnicas como la detección y prevención de interbloqueos, la asignación justa de prioridades y el uso eficiente de los mecanismos de sincronización disponibles. (Saxena, 2024) (GeeksForGeeks, 2024)

Navegando profundamente en los problemas de la concurrencia.

La concurrencia en los sistemas operativos introduce una seria de. Problemas que pueden afectar a la integridad y atomicidad de los datos, el rendimiento del sistema. Y la correcta ejecución de los procesos. Estos problemas surgen cuando múltiples procesos o hilos intentan acceder simultáneamente a recursos compartidos sin una adecuada sincronización.

Condiciones de carrera.

Las condiciones de carrera ocurren cuando el resultado de la ejecución de un programa depende del orden en que se ejecutan los procesos o hilos concurrentes. Este problema surge cuando dos o más procesos acceden y modifican un recurso compartido al mismo tiempo sin mecanismos adecuados de sincronización.

Interbloqueo.

Un interbloqueo ocurre cuando un grupo de procesos queda en un estado de espera indefinida porque cada uno está esperando un recurso que otro proceso del grupo posee. En esta situación, ninguno de los procesos puede continuar su ejecución, lo que genera un bloqueo total del sistema.

Inanición.

La inanición ocurre cuando un proceso no puede acceder a los recursos necesarios para su ejecución porque otros procesos con mayor prioridad acaparan dichos recursos indefinidamente. Esto puede suceder en sistemas donde la planificación de procesos está basada en prioridades y ciertos procesos de baja prioridad quedan constantemente relegados. (Tpoint Tech, s.f.)

Métodos de sincronización.

Los métodos de sincronización son técnicas utilizadas en los sistemas operativos para coordinar la ejecución de procesos o hilos concurrentes, asegurando que accedan de manera controlada a los recursos compartidos y evitando problemas como condiciones de carrera, interbloqueos e inconsistencias de datos. La correcta sincronización es esencial para garantizar la integridad de la información y el correcto funcionamiento de los sistemas multitarea.

Uno de los métodos más utilizados es el uso de semáforos, estructuras de datos que permiten controlar el acceso a recursos compartidos. Los semáforos pueden ser binarios, cuando solo permiten valores de 0 y 1 y funcionan como un mecanismo de bloqueo, o de conteo, cuando permiten que múltiples procesos accedan simultáneamente hasta un límite predefinido. Los semáforos evitan la espera activa al suspender procesos cuando los recursos no están disponibles, permitiendo una gestión más eficiente del CPU. Otro método ampliamente utilizado es el uso de monitores, una abstracción de alto nivel que encapsula variables compartidas y los procedimientos que las manipulan, garantizando que solo un proceso pueda acceder a la sección crítica dentro del monitor en un momento dado. Los monitores emplean variables de condición para gestionar la sincronización entre procesos, permitiendo que los hilos esperen de manera segura hasta que se cumpla una determinada condición.

Los locks o mecanismos de bloqueo también son empleados para la sincronización. Un lock funciona como un candado que protege los recursos compartidos, permitiendo que solo un proceso tenga acceso a la vez. Existen diferentes tipos de locks, como los spinlocks, que

mantienen el proceso en un bucle de espera activa hasta que el recurso esté disponible, y los mutex, que permiten la suspensión del proceso si el recurso está ocupado, evitando el desperdicio de CPU. Otro enfoque de sincronización es el modelo de productor-consumidor, donde los procesos productores generan datos y los consumidores los procesan, utilizando estructuras como colas y buffers para gestionar la comunicación de manera eficiente. Este modelo evita la sobrecarga o pérdida de datos mediante el uso de técnicas como el búfer circular y los semáforos.

Un método más avanzado es el uso de regiones críticas, donde se definen secciones del código que solo pueden ser ejecutadas por un proceso a la vez. Estas regiones se implementan con instrucciones atómicas de hardware como Test-and-Set o Compare-and-Swap, que garantizan que las operaciones críticas se completen sin interrupciones. Asimismo, existen algoritmos de software como el de Peterson, que permite la sincronización de dos procesos sin necesidad de asistencia del hardware, aunque no es eficiente en arquitecturas modernas debido a la falta de atomicidad en las operaciones de memoria. (GeekForGeeks, 2025)

Algoritmo de Peterson y Algoritmo de Dekker.

El algoritmo de Peterson y el algoritmo de Dekker son dos soluciones clásicas para la exclusión mutua en entornos de concurrencia, diseñadas para permitir que dos procesos accedan a una sección crítica de manera controlada sin interferencias. Ambos son algoritmos de software que no requieren soporte de hardware especial, aunque en arquitecturas modernas han sido reemplazados por mecanismos más eficientes debido a la falta de atomicidad en sus operaciones.

Algoritmo de Peterson

El algoritmo de Peterson es una solución simple y efectiva para la exclusión mutua entre dos procesos. Se basa en el uso de dos variables compartidas:

- flag[i]: Indica si el proceso i quiere acceder a la sección crítica.
- turn: Indica a quién se le da preferencia en caso de conflicto (GeeksForGeeks, 2025).

El funcionamiento del algoritmo es el siguiente:

1. Cada proceso establece su flag[i] = true para indicar su interés en entrar a la sección crítica.
2. Luego, asigna la variable turn al otro proceso, cediendo la preferencia de acceso.
3. Si el otro proceso también quiere entrar (flag[j] = true) y tiene el turno, el proceso actual esperará hasta que el otro termine.
4. Una vez dentro de la sección crítica, el proceso realiza sus operaciones.
5. Al salir de la sección crítica, establece flag[i] = false, permitiendo que el otro proceso acceda.

El algoritmo de Peterson garantiza la exclusión mutua y evita la inanición, ya que siempre cede el turno al otro proceso cuando ambos compiten por el recurso. Sin embargo, su implementación en arquitecturas modernas con múltiples núcleos y optimización de memoria puede fallar debido a la falta de instrucciones atómicas.

Algoritmo de Dekker

El algoritmo de Dekker es una solución más antigua y compleja que el de Peterson, también diseñada para la sincronización de dos procesos. Se basa en la combinación de variables de estado y un mecanismo de turnos para garantizar la exclusión mutua. Su funcionamiento es el siguiente:

- Se utilizan dos variables flag[i] para indicar si un proceso quiere entrar a la sección crítica.
- Se usa una variable turn para gestionar la preferencia entre procesos.
- Cuando un proceso quiere entrar a la sección crítica, marca su flag[i] = true.
- Si el otro proceso también quiere entrar (flag[j] = true), se verifica el turno. Si no es su turno, el proceso espera hasta que el otro lo libere.
- Una vez dentro de la sección crítica, realiza sus operaciones y luego establece flag[i] = false para permitir el acceso al otro proceso.

El algoritmo de Dekker es pionero en garantizar la exclusión mutua sin depender de soporte de hardware, pero es menos eficiente que el de Peterson debido a su estructura más compleja y la dependencia de múltiples condiciones de verificación. (GeeksForGeeks, 2024)

Dinamica.

https://create.kahoot.it/share/kahoot-sistemas-operativos/eabb4264-3b95-484f-8685-31dbe1eccc40

Bibliografía

GeekForGeeks. (10 de Junio de 2025). *introduction of process syncronization*. Obtenido de GeeksForGeeks: https://www.geeksforgeeks.org/introduction-of-process-synchronization/

GeeksForGeeks. (16 de Junio de 2022). *Concurrent Processes in Operating System*. Obtenido de GeeksForGeeks: https://www.geeksforgeeks.org/concurrent-processes-in-operating-system/

GeeksForGeeks. (28 de Diciembre de 2024). *Concurrency in Operating System*. Obtenido de Geeks For Geeks: https://www.geeksforgeeks.org/concurrency-in-operating-system/

GeeksForGeeks. (20 de Marzo de 2024). *Dekker's algorithm in Process Synchronization* . Obtenido de GeeksForGeeks: https://www.geeksforgeeks.org/dekkers-algorithm-in-process-synchronization/

GeeksForGeeks. (22 de Julio de 2024). *Mutual Exclusion*. Obtenido de GeeksForGeeks: https://www.geeksforgeeks.org/mutual-exclusion-in-synchronization/

GeeksForGeeks. (10 de Junio de 2025). *Peterson's Algorithm in Process Synchronization*. Obtenido de GeeksForGeeks: https://www.geeksforgeeks.org/petersons-algorithm-in-process-synchronization/

Saxena, A. (16 de Junio de 2024). *Mutual Exclusion in OS*. Obtenido de Scaler Topics: https://www.scaler.com/topics/mutual-exclusion-in-os/

Tpoint Tech. (s.f.). *Concurrency in Operating System*. Obtenido de TpointTech: https://www.tpointtech.com/concurrency-in-operating-system