# Taller 2do corte: Administración de procesos desde el sistema operativo

**Juan David Piracon Guauque**

**universidad del trópico americano**

**Facultad de ingeniería en sistemas**

**2024**

**Resumen**

Un proceso en un sistema operativo es un ejemplo de un programa en ejecución, como cuando abre un navegador web. La gestión de procesos es importante en entornos multitarea para ejecutar de manera eficiente múltiples procesos al mismo tiempo, como un editor de texto y un reproductor de música. Los sistemas operativos han evolucionado desde la ejecución de un solo proceso como en MS-DOS hasta sistemas multitarea y multiprocesamiento como Windows y Linux. El kernel del sistema operativo gestiona procesos, asigna recursos y coordina su ejecución, al igual que el temporizador de procesos en Linux. Los procesos pueden tener diferentes estados: nuevo, listo, en ejecución, bloqueado y completado. Por ejemplo, se bloqueará un proceso que espera E/S. Un proceso pasa de Listo a En ejecución cuando un temporizador de CPU le asigna tiempo de CPU, como en el modelo de operación por turnos, y de En ejecución a Bloqueado cuando accede a un recurso, como una tarea, debe esperar. La programación de trabajos tiene como objetivo maximizar la utilización de la CPU, minimizar los tiempos de respuesta y garantizar la equidad. Algoritmos como el primero en llegar, primero en ser atendido (FCFS) asignan CPU al primer proceso en la cola, mientras que el algoritmo Shortest Job First (SJF) minimiza el tiempo de espera general, pero puede privar a los procesos de larga duración. Round Robin divide el tiempo de la CPU en niveles iguales, lo cual es ideal para sistemas de tiempo compartido. La programación de prioridades asigna CPU según la prioridad del proceso, mientras que los algoritmos en tiempo real como RMS y EDF son importantes en los sistemas de control de tráfico aéreo. Un hilo es una unidad de ejecución de proceso que permite la concurrencia y la concurrencia, como un servidor web con múltiples conexiones. Los subprocesos pueden ser subprocesos de usuario o de kernel, administrados por bibliotecas o el sistema operativo respectivamente. Los modelos de Las condiciones de carrera ocurren cuando se accede a varios subprocesos y algunos procesos se comparten sin una sincronización adecuada. Conducen a puntos muertos cuando dos o más procesos esperan indefinidamente lecciones que cada uno pueda utilizar. Los daños se pueden evitar mediante el uso de algoritmos de programación, como mecanismos de sincronización por turnos, como semáforos y mutex, que coordinan el acceso a cursos compartidos, mientras que los monitores y las variables de condición ayudan a sincronizar los subprocesos del monitor. Los métodos de comunicación entre procesos (IPC) incluyen canalizaciones, canalizaciones de mensajes, memoria compartida y sockets. El almacenamiento compartido ofrece un mejor rendimiento pero es más completo. El sistema operativo gestiona eficientemente los recursos del sistema, como la CPU, la memoria y las E/S. Dispositivos. Las técnicas de asignación de cursos incluyen programación de CPU y administración de memoria, como paginación y fragmentación. Los algoritmos de asignación correctos evitan problemas de contención de recursos. Las herramientas de monitoreo de procesos, como Tasks Administrator en Windows y la más nueva en Unix/Linux, le permiten ver y administrar procesos. Las técnicas de control de procesos incluyen priorización, prevención, reinicio y parada de procesos. La gestión avanzada de procesos incluye la gestión de servicios y la definición de políticas de programas de procesador. Ejemplos concretos muestran cómo se manejan los procesos en diferentes sistemas operativos. En Windows, puede utilizar el Administrador de tareas para eliminar procesos que no responden. En Linux, el temporizador CFS gestiona la asignación de CPU; y en macOS, Activity Monitor monitorea los procesos. Los problemas de sincronización se pueden resolver con semáforos, por ejemplo en el dominio productor-consumidor. La memoria compartida se utiliza para una comunicación eficiente en los procesos y se pueden detectar y resolver interbloqueos, por ejemplo, en sistemas de bases de datos. La programación con subprocesos mejora el rendimiento de las aplicaciones, por ejemplo en servidores web, y la gestión de la memoria permite que los procesos utilicen más memoria de la que está físicamente disponible, por ejemplo en Windows con memoria virtual. implementación de subprocesos varían, como el modelo uno a uno de Windows.

**Abstract**

A process in an operating system is an instance of a running program, such as when you open a web browser. Process management is important in multitasking environments to efficiently run multiple processes simultaneously, like a text editor and a music player. Operating systems have evolved from running a single process, like in MS-DOS, to multitasking and multiprocessing systems like Windows and Linux. The operating system's kernel manages processes, assigns resources, and coordinates their execution, similar to the process scheduler in Linux.

Processes can have different states: new, ready, running, blocked, and completed. For example, a process waiting for I/O will be blocked. A process transitions from Ready to Running when a CPU scheduler assigns CPU time to it, as in the Round Robin scheduling model, and from Running to Blocked when it accesses a resource it must wait for, such as a task.

Job scheduling aims to maximize CPU utilization, minimize response times, and ensure fairness. Algorithms like First Come, First Served (FCFS) assign the CPU to the first process in the queue, while the Shortest Job First (SJF) algorithm minimizes overall wait time but can starve long-running processes. Round Robin divides CPU time into equal slices, ideal for time-sharing systems. Priority scheduling assigns CPU based on process priority, while real-time algorithms like RMS and EDF are important in air traffic control systems.

A thread is a unit of process execution that allows for concurrency and parallelism, like a web server handling multiple connections. Threads can be user threads or kernel threads, managed by libraries or the operating system respectively. Race conditions occur when multiple threads access and modify shared data without proper synchronization, leading to deadlocks where two or more processes wait indefinitely for resources held by each other. Starvation can be avoided by using fair scheduling algorithms, such as Round Robin.

Synchronization mechanisms like semaphores and mutexes coordinate access to shared resources, while monitors and condition variables help synchronize threads within a monitor. Inter-process communication (IPC) methods include pipes, message queues, shared memory, and sockets. Shared memory offers better performance but is more complex to implement.

The operating system efficiently manages system resources such as CPU, memory, and I/O devices. Resource allocation techniques include CPU scheduling and memory management like paging and segmentation. Fair allocation algorithms prevent resource contention issues. Process monitoring tools, like Task Manager in Windows and top in Unix/Linux, allow you to view and manage processes. Process control techniques include prioritization, suspension, resumption, and termination of processes. Advanced process management involves managing services and defining CPU scheduling policies.

Concrete examples show how processes are handled in different operating systems. In Windows, Task Manager can be used to terminate unresponsive processes; in Linux, the CFS scheduler manages CPU allocation; and in macOS, Activity Monitor monitors processes. Synchronization issues can be resolved with semaphores, such as in the producer-consumer problem. Shared memory is used for efficient inter-process communication, and deadlocks can be detected and resolved, as in database systems. Threading improves application performance, as seen in web servers, and memory management allows processes to use more memory than physically available, such as in Windows with virtual memory. Thread implementation models vary, such as the one-to-one model in Windows.

# 1. Introducción

La gestión de procesos es un componente esencial de los sistemas operativos modernos y permite realizar múltiples tareas de manera simultánea y eficiente. Un proceso es un programa que se ejecuta, por ejemplo, cuando abre un navegador web. Su gestión es esencial en entornos multitarea para mantener la eficiencia y funcionalidad del sistema. Desde los primeros sistemas operativos como MS-DOS, que sólo podían ejecutar un proceso a la vez, hasta sistemas avanzados de multitarea y multiprocesamiento como Windows y Linux, la evolución ha sido notable. El núcleo del sistema operativo juega un papel fundamental en la gestión de procesos, asignación de recursos y coordinación de su ejecución. Los procesos pueden tener diferentes estados, incluidos Nuevo, Listo, En ejecución, Bloqueado y Completado. La transición entre estos estados se gestiona cuidadosamente para optimizar el rendimiento del sistema. La programación de tareas utilizando varios algoritmos maximiza la utilización de la CPU y garantiza la equidad, mientras que la sincronización y la comunicación entre procesos garantiza operaciones coordinadas y eficientes.

***1. ¿Cómo definirías un proceso en el contexto de un sistema operativo? Proporciona un ejemplo.***

La gestión de procesos es un componente esencial de los sistemas operativos modernos y permite lograr múltiples efectos de forma simultánea y eficiente. Un proceso es un programa que se ejecuta, por ejemplo, para que puedas navegar por la web. La gestión es esencial en entornos multipropósito para mantener la eficiencia y funcionalidad del sistema. Tras los primeros sistemas operativos como MS-DOS, ya no podemos ejecutar el mismo proceso que antes, sino sólo sistemas avanzados multitarea y multiprocesamiento como Windows y Linux, cuya evolución es notable. El núcleo del sistema operativo juega un papel fundamental en la gestión de procesos, asignación de recursos y coordinación de su ejecución. Los procesos pueden tener diferentes estados, incluidos Nuevo, Listo, En ejecución, Bloqueado y Completado. La transición entre estas etapas se gestiona cuidadosamente para optimizar el rendimiento del sistema. La programación de tareas utilizando diferentes algoritmos maximizó la utilización de la CPU y garantizó la equidad, asegurando que la sincronización y la comunicación entre procesos aseguraran operaciones coordinadas y eficientes. (Silberschatz, (2018))

***2. Explica la importancia de la gestión de procesos en un entorno multitarea con un ejemplo.***

Un ejemplo que ilustra la importancia de la gestión de procesos en un entorno multitarea es el caso de un servidor web que procesa múltiples conexiones simultáneamente. En este escenario, el servidor web debe poder gestionar de manera eficiente cada solicitud de cliente y determinar procesos individuales que requieren recurrir al sistema operativo.

La gestión de procesos es esencial en este contexto para garantizar que el servidor web pueda responder a todas las solicitudes para funcionar de manera eficiente y sin interrupciones. El sistema operativo asigna recursos como tiempo de CPU, memoria y ancho de banda rojo a cada proceso del servidor web, lo que permite que se ejecuten múltiples conexiones simultáneamente.

Por ejemplo, cuando varios usuarios acceden al servidor web, es hora de cargar páginas web o descargar archivos. Cada una de estas interacciones se convierte en un proceso individual que el sistema operativo debe manejar. La asignación de solicitudes y la programación de trabajos adecuadas permiten que el servidor web responda de manera eficiente a todas las solicitudes, lo que garantiza un rendimiento óptimo y una experiencia de usuario fluida.

En este escenario, la gestión de procesos garantiza que el servidor web pueda gestionar de manera eficiente la concurrencia de múltiples conexiones, optimizando así el uso de los recursos del sistema y garantizando un servicio continuo e ininterrumpido a los usuarios. (Silberschatz, Operating System Concepts (10th ed.). Wiley., 2018)

***3. Describe la evolución histórica de los sistemas operativos desde sistemas de un solo proceso hasta sistemas multitarea y multiproceso. Menciona ejemplos de sistemas operativos para cada etapa.***

Los sistemas operativos han evolucionado desde la ejecución de un solo proceso como en MS-DOS hasta sistemas multitarea y multiprocesador como Windows y Linux.

Al principio, los sistemas operativos ejecutaban programas de forma secuencial y uno tras otro. Luego aparecieron los sistemas por lotes que agrupaban programas para procesarlos de manera eficiente, como por ejemplo: B. E/S GM-NAA.

Más tarde, los sistemas de tiempo compartido permitieron realizar múltiples tareas y compartir recursos. Un ejemplo es Unix, que introdujo el tiempo compartido y el multiusuario.

Con las computadoras personales surgieron sistemas como MS-DOS, Windows y Mac OS, que fueron diseñados para PC individuales.

Después de todo, los sistemas operativos modernos como Windows, Linux y macOS son capaces de realizar múltiples tareas y multiprocesamiento, ejecutar varios procesos al mismo tiempo, distribuir recursos de manera óptima y coordinar tareas que se ejecutan al mismo tiempo.

Esta evolución refleja cómo los sistemas operativos se han adaptado a las necesidades cambiantes de la informática, desde simples sistemas de un solo proceso hasta complejos sistemas multitarea y multiproceso que son esenciales en la actualidad. (Informàtica, 2023)

***4. ¿Cuál es el rol del kernel del sistema operativo en la gestión de procesos? Proporciona un ejemplo de cómo el kernel maneja los procesos.***

Un ejemplo de cómo el kernel del sistema operativo procesa los procesos es el procesamiento de interrupciones. Cuando un proceso solicita acceso a un dispositivo de E/S (entrada/salida), como un disco duro o una impresora, el núcleo coordina la interacción entre el proceso y el dispositivo mediante interrupciones de procesamiento. El kernel maneja las interrupciones generadas por los dispositivos de E/S, lo que permite una ejecución de procesos eficiente y sin fallas. (explicación, 2024)

***5. Detalla cada uno de los estados posibles de un proceso. Proporciona un ejemplo de un proceso en cada estado.***

Posibles estados de un proceso y ejemplos:

Listo: el proceso está listo para ejecutarse y está esperando que el programador de procesos le asigne CPU. Por ejemplo, un proceso de reproducción de música en un reproductor multimedia.

En ejecución: el proceso se está ejecutando actualmente en el procesador. Por ejemplo, un proceso de representación de gráficos en un software de diseño.

Bloqueado: el proceso está temporalmente en pausa porque está esperando un recurso externo, como una operación de entrada/salida. Por ejemplo, una operación que guarda un documento mientras espera que se complete una operación de escritura en el disco. (Virtual, 2024)

***6. ¿Qué eventos pueden provocar que un proceso pase del estado de 'listo' a 'ejecución'? Da un ejemplo de tal evento.***

Eventos que pueden hacer que un proceso pase de Listo a Listo:

Un proceso puede pasar del estado Listo al estado En ejecución cuando ocurre una interrupción de hardware, como por ejemplo: una solicitud de E/S completada, una señal de temporizador o una interrupción de hardware específica que requiere atención inmediata.

Ejemplo de un evento que provoca este cambio de estado:

Si un proceso en estado Listo está esperando una operación de E/S, como leer datos de un disco duro, se genera una interrupción de hardware al final de la operación de E/S para informar al sistema explotador que continúe. En este punto, el proceso cambia de Listo a En ejecución para procesar los datos que acaba de leer. (Perplexity., 2024)

***7. ¿Qué sucede en un sistema operativo cuando un proceso pasa de 'ejecución' a 'bloqueado'? Proporciona un ejemplo de esta transición.***

En un sistema operativo, un proceso en estado de ejecución puede quedar "bloqueado" cuando realiza una solicitud de entrada/salida a un dispositivo y tiene que esperar a que se complete la operación de procesamiento. Por ejemplo, un proceso en un sistema de gestión de inventario que necesita leer datos de un archivo en el disco puede entrar en un estado bloqueado mientras espera que el archivo se lea por completo. (Risk, No especificada)

***8. Ilustra con un diagrama las transiciones entre los estados de un proceso y proporciona ejemplos de situaciones que provocan estas transiciones.***

Texto

Descripción generada automáticamente

9. ¿Cuáles son los objetivos principales de la planificación de procesos en un sistema operativo? Proporciona ejemplos de cómo se alcanzan estos objetivos.

Además de optimizar el rendimiento del sistema y brindar un buen servicio a todos los procesos, la programación de procesos también tiene como objetivo minimizar los tiempos de respuesta en entornos interactivos y cumplir con los plazos de ejecución de los sistemas en tiempo real.

Ejemplo del mismo color de objetos:

Utilice un algoritmo de programación adecuado para priorizar procesos interactivos y en tiempo real y garantizar tiempos de respuesta adecuados.

Aplicar técnicas de multiprogramación para mantener múltiples procesos en la memoria, aumentando las posibilidades de utilización de la CPU en caso de falla del proceso.

Ajusta dinámicamente las prioridades de los procesos en función de su comportamiento, favoreciendo aquellos que consumen menos recursos y liberan CPU rápidamente.

Utilice mecanismos de cuotas y límites para garantizar una distribución justa del tiempo de CPU entre usuarios y clases de procesos. (especificado, Planificación de procesos - Sistemas Operativos, No especificado)

***10. Explica el algoritmo de planificación First Come, First Served (FCFS) y sus características. Da un ejemplo de su aplicación.***

El algoritmo de programación FCFS (First Come, First Served) es un método de programación de procesos que asigna procesadores a los procesos en el orden en que llegan al sistema. Este algoritmo no es preventivo, lo que significa que una vez que se inicia un proceso, no se detendrá hasta que se complete.

Características del algoritmo FCFS:

Simplicidad: el algoritmo FCFS es fácil de entender e implementar.

No preventivo: El proceso iniciado no se detiene hasta su finalización.

FIFO: El algoritmo FCFS sigue el principio FIFO (First In, First Out), donde el proceso que llegue primero se ejecutará primero.

Ejemplo de aplicación del algoritmo FCFS:

Un ejemplo común de la aplicación del algoritmo FCFS es una cola de procesos en un sistema operativo. Supongamos que tenemos cuatro procesos (P1, P2, P3 y P4) con tiempos de ráfaga (tiempos de ejecución) de 6, 8, 3 y 4 segundos respectivamente, llegando al sistema en momentos diferentes. El algoritmo FCFS asignaría CPU a los procesos en el orden en que llegan, es decir, h. P1, P2, P3 y P4. Primero se inicia el proceso P1, seguido de P2, P3 y finalmente P4. (especificado, Virtual Pro, No especificado)

***11. ¿Qué ventajas y desventajas presenta el algoritmo Shortest Job First (SJF)? Proporciona un ejemplo donde se utilice SJF.***

En un sistema de procesamiento de transacciones bancarias, el algoritmo SJF se utiliza para priorizar transacciones más cortas, como consultas de saldo o transferencias de pequeñas cantidades, asegurando una respuesta rápida a los clientes. De esta manera, las transacciones más complejas y que requieren más tiempo, como abrir una cuenta o solicitar un préstamo, se procesan después de transacciones más simples, optimizando así el tiempo de respuesta de todo el sistema. (Santander, No especificada)

***12. ¿Cómo funciona el algoritmo de planificación Round Robin y en qué escenarios es más efectivo? Da un ejemplo de su uso.***

Un ejemplo de aplicación del algoritmo de todos contra todos es la programación de torneos deportivos, donde se utiliza para organizar comparaciones justas entre equipos. El algoritmo garantiza que cada equipo juegue contra todos los demás con la misma frecuencia para evitar posibles desventajas y garantizar el equilibrio de la competición. (Castillo Duarte, 2009)

***13. Compara los algoritmos de planificación de CPU por prioridades con otros algoritmos como FCFS y SJF. Proporciona ejemplos.***

Un ejemplo de aplicación del algoritmo round robin es la planificación de vuelos de una aerolínea. En este contexto, el algoritmo se utiliza para asignar franjas horarias a diferentes vuelos y garantizar que cada aerolínea tenga las mismas posibilidades de despegar y aterrizar en los horarios más deseados. Esto evita que una sola aerolínea monopolice los mejores horarios y garantiza una distribución justa de los recursos aeroportuarios. (Silberschatz, Operating system concepts (10th ed.). Wiley., 2018)

***14. Describe los algoritmos de planificación en tiempo real, como Rate Monotonic Scheduling (RMS) y Earliest Deadline First (EDF). Proporciona ejemplos de su aplicación.***

Los algoritmos de planificación en tiempo real como Rate Monotonic Scheduling (RMS) y Earliest Deadline First (EDF) se utilizan para asignar procesos a la CPU de manera eficiente y garantizar que se cumplan los plazos de ejecución críticos.

Rate Monotonic Scheduling (RMS)

RMS es un algoritmo de prioridades fijas donde las prioridades se asignan inversamente proporcionales a los períodos de las tareas. Las tareas con períodos más cortos tienen mayor prioridad. RMS es óptimo entre los algoritmos de prioridades fijas cuando los plazos de finalización son iguales a los períodos.

Ejemplo: En un sistema de control de un reactor nuclear, las tareas de monitoreo y control tienen períodos cortos y alta prioridad, mientras que las tareas de registro de datos tienen períodos más largos y menor prioridad. RMS garantiza que las tareas críticas se ejecuten a tiempo.

Earliest Deadline First (EDF)

EDF es un algoritmo de prioridades dinámicas donde la prioridad de una tarea se basa en su plazo de finalización. La tarea con el plazo más cercano tiene la mayor prioridad. EDF es óptimo en un solo procesador cuando los plazos son menores o iguales a los períodos.

Ejemplo: En un sistema de control de tráfico aéreo, las tareas de control de vuelo tienen plazos muy ajustados y se ejecutan con alta prioridad, mientras que las tareas de mantenimiento de registros tienen plazos más holgados. EDF garantiza que las tareas críticas se completen a tiempo.

Ambos algoritmos son efectivos en sistemas de tiempo real donde es crucial cumplir con los plazos de ejecución de las tareas. RMS es más simple de implementar, mientras que EDF es más flexible al adaptarse a las prioridades dinámicas (Cabalar, s.f.)

***15. Define lo que es un hilo y diferencia este concepto del de proceso. Proporciona un ejemplo de cada uno.***

Un hilo es una unidad de ejecución dentro de un proceso que comparte el mismo espacio de memoria que otros hilos del mismo proceso. Los hilos son livianos, es decir, requieren menos recursos y tiempo para su creación y cambio de contexto en comparación con los procesos. Esto los hace más eficientes en situaciones que requieren una gran cantidad de hilos. Los hilos pueden comunicarse de manera más eficiente que los procesos y pueden compartir recursos, como variables y objetos.

Por otro lado, un proceso es una instancia de un programa en ejecución que puede ser creado por el sistema operativo o por otro proceso. Cada proceso tiene su propio espacio de memoria y no puede acceder a la memoria de otros procesos. Los procesos pueden ser independientes entre sí y pueden comunicarse a través de mecanismos de comunicación como pipes, sockets, memoria compartida, entre otros.

Ejemplo de proceso: Un navegador web es un proceso que se ejecuta en un sistema operativo. Dentro de este proceso, hay varios hilos que se encargan de diferentes tareas, como cargar páginas web, manejar entradas del usuario, y realizar operaciones de fondo.

Ejemplo de hilo: En un juego de video, cada personaje puede ser un hilo que se ejecuta dentro del proceso del juego. Cada hilo puede tener su propia lógica de comportamiento y compartir recursos con otros hilos, como la posición en el mapa o el estado del juego.

En resumen, los procesos son instancias de programas en ejecución que tienen su propio espacio de memoria, mientras que los hilos son unidades de ejecución dentro de un proceso que comparten el mismo espacio de memoria y recursos

***16. Explica las diferencias entre hilos de usuario y hilos de kernel. Da ejemplos de situaciones donde se usan cada tipo.***

La distinción entre hilos de usuario y hilos de kernel se basa en su gestión y el nivel de control que el sistema operativo ejerce sobre ellos. Los hilos de usuario son manejados por la aplicación sin la intervención del kernel, mientras que los hilos de kernel son creados, planificados y gestionados directamente por el sistema operativo.

Los hilos de usuario son implementados en librerías y gestionados sin el soporte del sistema operativo. Su cambio de contexto es más sencillo que el de los hilos de kernel. Un ejemplo de esto es la librería POSIX Pthreads.

Por otro lado, los hilos de kernel son creados, planificados y gestionados por el sistema operativo. El kernel reconoce tantos hilos como se hayan creado y permite aprovechar mejor las arquitecturas multiprocesadores. Ejemplos de esto son los sistemas operativos Windows NT/2000 y Linux.

En cuanto a situaciones donde se usan cada tipo de hilo, los hilos de usuario son útiles cuando se necesita una gestión más ligera y rápida de los hilos, como en aplicaciones donde la complejidad de la gestión de hilos no es crítica. Por otro lado, los hilos de kernel se utilizan en entornos donde se requiere un mayor control por parte del sistema operativo, como en sistemas operativos que necesitan una gestión detallada de los recursos y la concurrencia.

En resumen, la elección entre hilos de usuario y hilos de kernel depende de las necesidades específicas de la aplicación y del sistema operativo en el que se ejecuta. (Perplexity, 2024)

***17. Describe los modelos de implementación de hilos en sistemas operativos: uno a uno, muchos a uno, y muchos a muchos. Proporciona ejemplos para cada modelo.***

Según la literatura, en los sistemas operativos se utilizan diferentes modelos de implementación de subprocesos para gestionar el paralelismo en las aplicaciones. Estos patrones incluyen uno a uno, muchos a uno y muchos a muchos. En el modelo uno a uno, cada subproceso de usuario está asociado con un único subproceso del núcleo, lo que permite una gestión de subprocesos independiente y eficiente. Por ejemplo, este enfoque se adopta en sistemas como Windows NT/2000 para garantizar un control preciso de cada hilo de usuario por parte del hilo del núcleo correspondiente.

Sin embargo, en el modelo de muchos a uno, varios subprocesos de usuario comparten un único subproceso del núcleo. Aunque este enfoque es más eficiente en cuanto a recursos, puede causar cuellos de botella si un subproceso de usuario bloquea el subproceso del núcleo compartido. Se puede encontrar un ejemplo de este patrón en versiones anteriores de Solaris, donde varios subprocesos de usuario comparten un único subproceso del núcleo para su ejecución.

Finalmente, en el modelo de muchos a muchos, varios subprocesos de usuario se asignan dinámicamente a múltiples subprocesos del núcleo, lo que proporciona mayor flexibilidad y eficiencia en el manejo de la concurrencia. Por ejemplo, este modelo se implementa en sistemas como Linux para optimizar la distribución de subprocesos de usuario en función de la carga de trabajo y la disponibilidad de recursos, lo que permite un escalado más eficiente en diferentes entornos de ejecución.

***18. ¿Cuáles son las ventajas de la programación con hilos en términos de concurrencia y paralelismo? Proporciona un ejemplo que ilustre estas ventajas.***

Las ventajas de la programación con hilos en términos de concurrencia y paralelismo son significativas. En cuanto a la concurrencia, la programación con hilos permite la ejecución simultánea de múltiples tareas, mejorando la eficiencia y el rendimiento de las aplicaciones. Por otro lado, en términos de paralelismo, la programación con hilos facilita la ejecución de tareas en paralelo, aprovechando al máximo los recursos de hardware disponibles y acelerando el procesamiento de datos.

Un ejemplo que ilustra estas ventajas es el desarrollo de una aplicación de procesamiento de imágenes. Al utilizar hilos, es posible dividir el proceso de carga, procesamiento y visualización de imágenes en tareas independientes que se ejecutan simultáneamente. Mientras un hilo se encarga de cargar una imagen, otro hilo puede procesarla y un tercero puede mostrarla en pantalla. Esta división de tareas en hilos permite un procesamiento más rápido y eficiente de las imágenes, mejorando la experiencia del usuario y optimizando el rendimiento de la aplicación. (Hilos, 2021) (paralelismo, s.f.)

***19. ¿Qué son las condiciones de carrera y cómo pueden afectar a los procesos en un sistema operativo? Proporciona un***

Las condiciones de carrera son situaciones en los sistemas operativos y en la programación concurrente donde el comportamiento del software depende del orden o la temporización específica de las operaciones múltiples que se están ejecutando simultáneamente. En otras palabras, es una condición en la cual dos o más procesos acceden y manipulan datos compartidos y el resultado final depende del orden en que se ejecutan las instrucciones.

Ejemplo

Para ilustrar una condición de carrera, consideremos un contador compartido entre dos procesos, Proceso A y Proceso B. Ambos procesos incrementan el valor del contador en 1.

Supongamos que el contador inicialmente tiene el valor 0.

Proceso A lee el valor del contador: 0.

Proceso B lee el valor del contador: 0.

Proceso A incrementa el valor leído: 0 + 1 = 1.

Proceso B incrementa el valor leído: 0 + 1 = 1.

Proceso A escribe el valor incrementado en el contador: contador = 1.

Proceso B escribe el valor incrementado en el contador: contador = 1.

Problema

Aunque ambos procesos han incrementado el contador, el resultado final es 1 en lugar de 2. Esto ocurre porque ambos procesos han leído el mismo valor antes de que cualquiera de ellos haya tenido la oportunidad de escribir el nuevo valor. Este es un claro ejemplo de una condición de carrera, donde la salida depende del orden en que se ejecutan las operaciones.

Efectos en los Procesos y el Sistema Operativo

Las condiciones de carrera pueden tener varios efectos negativos:

Resultados Incorrectos: Como en el ejemplo anterior, pueden llevar a resultados incorrectos o inesperados, lo que puede causar errores en el comportamiento del software.

Inconsistencia de Datos: Los datos compartidos pueden volverse inconsistentes, lo cual es crítico en aplicaciones que dependen de la integridad de los datos, como bases de datos y sistemas de archivos.

Dificultad en la Depuración: Los errores causados por condiciones de carrera pueden ser difíciles de reproducir y depurar, ya que pueden depender de la temporización específica de los procesos.

Seguridad: En algunos casos, las condiciones de carrera pueden ser explotadas por atacantes para causar comportamientos no deseados o comprometer la seguridad del sistema.

Prevención y Soluciones

Para evitar las condiciones de carrera, se pueden emplear varias técnicas de sincronización:

Bloqueos (Locks): Utilizar mecanismos de bloqueo como mutexes para asegurar que solo un proceso acceda a los recursos compartidos a la vez.

Semáforos: Utilizar semáforos para controlar el acceso a los recursos compartidos.

Variables de Condición: Utilizar variables de condición para permitir que los procesos esperen hasta que ciertas condiciones se cumplan antes de proceder.

Secciones Críticas: Definir secciones críticas en el código donde solo un proceso puede ejecutarse a la vez.

Monitores: Utilizar monitores que proporcionan una estructura de alto nivel para manejar la sincronización de manera más segura y sencilla.

Ejemplo con Solución

Modifiquemos el ejemplo anterior utilizando un mutex para evitar la condición de carrera.

Proceso A adquiere el mutex.

Proceso A lee el valor del contador: 0.

Proceso A incrementa el valor leído: 0 + 1 = 1.

Proceso A escribe el valor incrementado en el contador: contador = 1.

Proceso A libera el mutex.

Proceso B adquiere el mutex.

Proceso B lee el valor del contador: 1.

Proceso B incrementa el valor leído: 1 + 1 = 2.

Proceso B escribe el valor incrementado en el contador: contador = 2.

Proceso B libera el mutex.

Al usar un mutex, aseguramos que solo un proceso a la vez puede modificar el contador, lo que elimina la condición de carrera y garantiza que el contador se incremente correctamente.

En resumen, las condiciones de carrera son un problema crítico en la programación concurrente que puede afectar la precisión y la integridad de los datos en un sistema operativo. Utilizar técnicas de sincronización adecuadas es esencial para evitar estos problemas y asegurar el correcto funcionamiento del software. (Condiciones de carrera en sistemas operativos y programación concurrente., 2023)

***20. Interbloqueo (Deadlock)***

Un interbloqueo (deadlock) es una situación en un sistema donde dos o más procesos se bloquean indefinidamente esperando que un recurso sea liberado por uno de los otros procesos. Ocurre cuando cada proceso en el conjunto está esperando un recurso que está siendo retenido por otro proceso en el mismo conjunto.21. Define la inanición en el contexto de la gestión de procesos y cómo puede prevenirse. Proporciona un ejemplo.

Ejemplo:

Proceso A tiene el recurso 1 y espera el recurso 2.

Proceso B tiene el recurso 2 y espera el recurso 1.

Ambos procesos quedan bloqueados indefinidamente.

***22. Describe los mecanismos de sincronización como semáforos y mutex, y proporciona ejemplos de su uso.***

Los mecanismos de sincronización son fundamentales en la programación concurrente para evitar condiciones de carrera y asegurar el acceso ordenado a recursos compartidos. Dos de los mecanismos más utilizados son los semáforos y los mutex.

Semáforos: Un semáforo es una variable entera que se utiliza para controlar el acceso a recursos compartidos. Puede tomar cualquier valor entero y permite operaciones de incremento (signal) y decremento (wait). Por ejemplo, en un sistema de impresión con múltiples impresoras, un semáforo puede gestionar el número de impresoras disponibles, permitiendo a varios procesos solicitar impresoras de manera segura.

Mutex: Un mutex (mutual exclusion) es un mecanismo que asegura que solo un hilo puede acceder a un recurso crítico a la vez. A diferencia de los semáforos, un mutex solo puede tener dos estados: bloqueado o desbloqueado. Por ejemplo, en una aplicación de banco, un mutex puede proteger el acceso a una cuenta bancaria para que solo un hilo a la vez pueda realizar transacciones.23. ¿Qué son los monitores y las variables de condición, y cómo se utilizan para resolver problemas de sincronización? Proporciona ejemplos de su aplicación. (Tanenbaum, 2015) (Stallings, 2018)

***24. Enumera y describe los métodos de comunicación entre procesos (IPC) disponibles en los sistemas operativos. Proporciona ejemplos de su uso.***

Los monitores son estructuras de alto nivel que combinan la sincronización y la protección de datos compartidos. Un monitor contiene variables de condición que permiten a los hilos suspender su ejecución hasta que una condición particular sea verdadera.

Monitores: Un monitor encapsula variables compartidas, las funciones que operan sobre ellas, y las variables de condición. Solo un hilo puede estar ejecutando una función dentro del monitor a la vez, garantizando exclusividad. Por ejemplo, en un sistema de productor-consumidor, el monitor puede asegurar que los productores no añadan datos a un buffer lleno y que los consumidores no intenten retirar datos de un buffer vacío.

Variables de condición: Las variables de condición permiten que un hilo espere hasta que una condición específica sea señalada (signaled). Por ejemplo, en el sistema de productor-consumidor mencionado, una variable de condición puede hacer que un consumidor espere hasta que el buffer tenga al menos un ítem.

***25. Compara las ventajas y desventajas de la memoria compartida y las pipes en términos de rendimiento y complejidad. Da ejemplos de su aplicación.***

Memoria Compartida:

Ventajas: Alta velocidad de comunicación ya que los procesos acceden directamente a la memoria. Ideal para grandes volúmenes de datos.

Desventajas: Complejidad en la sincronización y potenciales problemas de coherencia de datos.

Ejemplo: En una aplicación de simulación científica, múltiples procesos pueden compartir grandes matrices de datos para cálculos concurrentes.

Pipes:

Ventajas: Sencillez de implementación y uso. Ideal para la comunicación de flujos de datos.

Desventajas: Menor rendimiento debido a la necesidad de copiar datos entre procesos.

Ejemplo: En una cadena de procesamiento de datos, una pipe puede pasar datos entre un proceso de lectura y uno de análisis.

***26. Explica cómo se implementan los mecanismos de comunicación entre procesos en un sistema operativo específico, como Linux o Windows. Proporciona ejemplos.***

En Linux, varios mecanismos de IPC están disponibles:

Pipes y Named Pipes (FIFOs): Se crean usando funciones como pipe() y mkfifo(). Por ejemplo, un proceso de shell puede usar pipes para encadenar comandos.

Memoria Compartida: Utiliza funciones como shmget(), shmat(), y shmdt(). Un ejemplo es un sistema de monitoreo de red que comparte estadísticas entre procesos.

Colas de Mensajes: Se manejan con msgget(), msgsnd(), y msgrcv(). Por ejemplo, un sistema de mensajería interno puede usar colas de mensajes para comunicación entre servicios.

Sockets: Implementados con socket(), bind(), listen(), y accept(). Por ejemplo, servidores web usan sockets para manejar conexiones de clientes. (UC3M., s.f.)

***27. ¿Cómo se gestionan los recursos del sistema como la CPU, la memoria y los dispositivos de E/S en un sistema operativo? Proporciona ejemplos.***

CPU: La planificación de CPU asigna tiempo de procesamiento a los procesos. Algoritmos como Round Robin o Prioridad pueden ser utilizados. Por ejemplo, en un servidor web, el sistema operativo puede priorizar procesos de respuesta rápida para mejorar la experiencia del usuario.

Memoria: La gestión de memoria incluye la asignación de espacios de memoria y la paginación. Por ejemplo, en un entorno de base de datos, la paginación permite a grandes bases de datos residir parcialmente en memoria.

Dispositivos de E/S: Se gestionan mediante controladores que medían el acceso y uso. Por ejemplo, en un sistema de impresión, el spooler de impresión gestiona las tareas de impresión enviadas a las impresoras disponibles.28. Describe las técnicas de asignación de recursos, incluyendo la planificación de CPU y la gestión de memoria (paginación y segmentación). Proporciona ejemplos. (Madrid, s.f.)

***28. Describe las técnicas de asignación de recursos, incluyendo la planificación de CPU y la gestión de memoria (paginación y segmentación). Proporciona ejemplos.***

Planificación de CPU: Algoritmos como Round Robin, SJF (Shortest Job First), y planificación por prioridad determinan cómo los procesos acceden a la CPU. Por ejemplo, en un sistema interactivo, Round Robin asegura que todos los procesos reciban un tiempo de CPU razonable.

Gestión de Memoria:

Paginación: Divide la memoria física en bloques de tamaño fijo llamados páginas. Por ejemplo, en Linux, la paginación permite que un proceso utilice más memoria de la que físicamente tiene disponible, utilizando memoria virtual.

Segmentación: Divide la memoria en segmentos de diferente tamaño, según las necesidades del proceso. Por ejemplo, sistemas operativos antiguos como Multics utilizaban segmentación para mejorar la gestión de memoria y protección. ((UNED)., s.f.)

***29. ¿Qué estrategias se pueden implementar para prevenir problemas de competencia por recursos, como los algoritmos de asignación justa? Da ejemplos.***

Asignación Justa: Algoritmos como el de Justicia Equitativa (Fairness) aseguran que todos los procesos reciban una parte equitativa de los recursos. Por ejemplo, en un sistema multiusuario, la planificación equitativa evita que un usuario monopolice la CPU.

Control de Concurrencia: Utilización de semáforos y mutex para sincronizar el acceso a recursos compartidos. Por ejemplo, en una base de datos, los bloqueos (locks) pueden prevenir que dos transacciones accedan y modifiquen el mismo registro simultáneamente.

Algoritmos de Evitación de Deadlock: Técnicas como el algoritmo del banquero pueden prevenir situaciones de deadlock al no permitir que los procesos entren en estados que pueden llevar a un bloqueo. Por ejemplo, en un sistema de reservación de vuelos, la gestión cuidadosa de los recursos puede evitar que las solicitudes se queden en un estado de espera permanente.30. Menciona y describe las herramientas y utilidades disponibles en diferentes sistemas operativos para monitorear procesos, como el Administrador de tareas en Windows y `top` en Unix/Linux. Proporciona ejemplos de su uso. (Madrid., s.f.)

***30. Menciona y describe las herramientas y utilidades disponibles en diferentes sistemas operativos para monitorear procesos, como el Administrador de tareas en Windows y top en Unix/Linux. Proporciona ejemplos de su uso.***

Administrador de tareas (Windows): Permite a los usuarios ver y gestionar procesos en ejecución, monitorear el rendimiento del sistema y finalizar procesos problemáticos. Por ejemplo, un usuario puede cerrar un programa que no responde.

top (Unix/Linux): Muestra una lista de procesos ordenados por uso de recursos en tiempo real. Por ejemplo, un administrador de sistemas puede usar top para identificar procesos que consumen demasiada CPU.

htop (Unix/Linux): Una versión más interactiva y visual de top, proporcionando una interfaz amigable. Por ejemplo, htop facilita la visualización del uso de recursos y permite finalizar procesos directamente desde la interfaz.

ps (Unix/Linux): Muestra información estática sobre los procesos en ejecución. Por ejemplo, ps aux proporciona una lista detallada de todos los procesos, que se puede filtrar y analizar. ((s.f.). G. , s.f.)

***31. Explica las técnicas de control de procesos, como el ajuste de prioridades, suspensión y reanudación, y terminación de procesos. Proporciona ejemplos.***

Ajuste de Prioridades: Cambiar la prioridad de un proceso para influir en su acceso a la CPU. Por ejemplo, en Windows, se puede usar el Administrador de tareas para cambiar la prioridad de un proceso.

Suspensión y Reanudación: Detener temporalmente un proceso y luego reanudar su ejecución. Por ejemplo, en Unix, se puede suspender un proceso con kill -STOP y reanudarlo con kill -CONT.

Terminación de Procesos: Finalizar la ejecución de un proceso. Por ejemplo, en Linux, se puede usar el comando kill para terminar un proceso específico.32. ¿Qué implica la administración avanzada de procesos en un sistema operativo? Proporciona ejemplos de gestión de servicios y configuración de políticas de planificación de CPU. (Madrid U. P., s.f.)

***32. ¿Qué implica la administración avanzada de procesos en un sistema operativo? Proporciona ejemplos de gestión de servicios y configuración de políticas de planificación de CPU.***

La administración avanzada de procesos incluye la gestión detallada y personalizada de los procesos del sistema para optimizar el rendimiento y la eficiencia.

Gestión de Servicios: Iniciar, detener y monitorear servicios del sistema. Por ejemplo, en Windows, la herramienta services.msc permite a los administradores gestionar los servicios del sistema, como servidores web o bases de datos.

Configuración de Políticas de Planificación de CPU: Ajustar las políticas de planificación para optimizar el uso de la CPU. Por ejemplo, en Linux, se puede usar chrt para establecer políticas de planificación en tiempo real para procesos críticos. (Salamanca., s.f.)

***33. Proporciona un ejemplo detallado de cómo se gestiona un proceso en Windows.***

En Windows, la gestión de procesos se puede realizar utilizando el Administrador de tareas y comandos de la línea de comandos como tasklist y taskkill.

Identificación del Proceso: Utilizando el Administrador de tareas, se puede ver una lista de procesos en ejecución, junto con información sobre el uso de CPU y memoria.

Ajuste de Prioridad: Haciendo clic derecho sobre un proceso en el Administrador de tareas y seleccionando "Establecer prioridad", se puede ajustar la prioridad del proceso para darle más o menos tiempo de CPU.

Terminación del Proceso: Para finalizar un proceso que no responde, se puede seleccionar el proceso y hacer clic en "Finalizar tarea".

***34. Describe cómo Linux maneja la planificación de procesos con un ejemplo específico.***

Linux utiliza varios algoritmos de planificación de procesos, incluyendo la planificación de tiempo real y de prioridad. El algoritmo más común es el de Completely Fair Scheduler (CFS).

CFS: Este algoritmo asegura que todos los procesos reciban una cantidad justa de tiempo de CPU. Utiliza un árbol rojo-negro para mantener los procesos ordenados por su tiempo de ejecución.

Ejemplo: Un administrador de sistemas puede usar el comando chrt para asignar un proceso a una política de planificación en tiempo real, asegurando que tareas críticas, como la gestión de una base de datos, reciban prioridad sobre otras tareas menos importantes. (Granada., s.f.)

35. Explica cómo macOS utiliza herramientas de monitoreo para controlar procesos con un ejemplo.

macOS proporciona varias herramientas para monitorear y controlar procesos, incluyendo Activity Monitor y comandos de la terminal como ps y kill.

Activity Monitor: Permite a los usuarios ver información detallada sobre los procesos, como el uso de CPU, memoria y energía. También se pueden finalizar procesos directamente desde esta interfaz.

Ejemplo: Un usuario puede utilizar Activity Monitor para identificar un proceso que está consumiendo demasiada CPU y finalizarlo para mejorar el rendimiento general del sistema. (IONOS, s.f.) (Inc, s.f.)

***36. Da un ejemplo de un problema de sincronización y cómo fue resuelto usando semáforos.***

Un ejemplo clásico de problema de sincronización es el del productor-consumidor. En este problema, los productores generan datos que los consumidores deben procesar, utilizando un buffer compartido.

Problema: Si los productores intentan añadir datos a un buffer lleno, o los consumidores intentan retirar datos de un buffer vacío, se produce un error.

Solución con Semáforos: Se utilizan dos semáforos: uno para contar los elementos llenos del buffer y otro para contar los espacios vacíos. El semáforo de elementos llenos se incrementa cuando un productor añade un ítem y se decrementa cuando un consumidor retira un ítem. El semáforo de espacios vacíos se incrementa cuando un consumidor retira un ítem y se decrementa cuando un productor añade un ítem. (Tanenbaum, Modern operating systems (4th ed.). Pearson., 2015)

***37. Describe un caso donde se utilizó memoria compartida para la comunicación entre procesos en un sistema operativo específico.***

En un sistema de vigilancia por video, se puede utilizar memoria compartida para que diferentes procesos accedan a los cuadros de video capturados sin necesidad de copiar datos.

Caso: Un proceso captura los cuadros de video y los almacena en una región de memoria compartida. Varios procesos de análisis de video acceden a esta memoria compartida para procesar los cuadros en tiempo real.

Sistema Operativo: En Linux, se utilizan las funciones shmget(), shmat(), y shmdt() para gestionar la memoria compartida.

***38. Proporciona un ejemplo de cómo un deadlock fue identificado y resuelto en un entorno de producción.***

En un sistema de gestión de bases de datos, los deadlocks pueden ocurrir cuando dos o más transacciones esperan indefinidamente por recursos bloqueados por las otras. Esto se debe a la contención de recursos entre transacciones concurrentes que necesitan acceder a los mismos datos.

Identificación: Utilizando herramientas de monitoreo como el Administrador de tareas en Windows o comandos como top en Unix/Linux, el administrador de la base de datos puede detectar si una transacción está esperando más del tiempo límite especificado para obtener un bloqueo. Esto puede indicar la presencia de un deadlock. El sistema de base de datos también puede emplear un mecanismo periódico de detección de deadlocks.

Resolución: Cuando se detecta un deadlock, el sistema de base de datos elige una de las transacciones como víctima del deadlock. La transacción de la víctima se termina, se deshace su transacción actual y se devuelve un error 1205 a la aplicación. Esto libera todos los bloqueos mantenidos por esa transacción, permitiendo que las otras transacciones involucradas en el deadlock puedan continuar.

Por ejemplo, en Apache Derby, cuando una transacción espera más de un tiempo específico para obtener un bloqueo (llamado tiempo de espera de deadlock), Derby puede detectar si la transacción está atrapada en un deadlock. Si es así, Derby deshace la transacción más apropiada para romper el deadlock.

En resumen, los deadlocks en bases de datos se identifican monitoreando transacciones bloqueadas y se resuelven abortando una transacción para liberar recursos y permitir que las demás continúen. Aunque no se pueden evitar por completo, siguiendo ciertas convenciones de codificación se pueden minimizar.

***39. Describe una situación donde la programación con hilos mejoró significativamente el rendimiento de una aplicación.***

En una aplicación de servidor web, la capacidad de manejar múltiples solicitudes concurrentes es crucial.

Situación: El servidor web originalmente manejaba cada solicitud de manera secuencial, lo que causaba tiempos de respuesta lentos durante picos de tráfico.

Mejora: Al implementar un modelo multihilo, cada solicitud se manejó en un hilo separado, permitiendo que el servidor procesara múltiples solicitudes simultáneamente. Esto resultó en una mejora significativa en el rendimiento y la capacidad de respuesta del servidor.

***40. Explica cómo la técnica de paginación se utiliza en la gestión de memoria con un ejemplo específico de un sistema operativo.***

La paginación es una técnica de gestión de memoria que divide el espacio de direcciones lógicas en páginas de tamaño fijo y la memoria física en marcos de página.

Técnica: Cuando un proceso necesita una página que no está en memoria física, se produce una falla de página y el sistema operativo carga la página necesaria desde el disco a un marco de página disponible.

Ejemplo en Windows: En Windows, la paginación permite que aplicaciones usen más memoria de la disponible físicamente mediante el uso de un archivo de paginación en el disco duro. Si una aplicación requiere más memoria, Windows mueve las páginas menos utilizadas al archivo de paginación, liberando memoria física para otras páginas.

# 2. Conclusiones

En el amplio campo de los sistemas operativos y la programación concurrente, la sincronización, la gestión de recursos y la comunicación entre procesos son elementos cruciales. Una comprensión profunda de estos conceptos es esencial para diseñar sistemas eficientes y confiables.

Los mecanismos de sincronización, semáforos y mutex le permiten controlar el acceso a algunos recursos y evitar condiciones de carrera. Por otro lado, los métodos de comunicación entre procesos, como tuberías, memoria compartida y sockets, facilitan el intercambio de datos y señales entre procesos.

La gestión eficiente de los recursos, incluidos la CPU, la memoria y los dispositivos de E/S, es esencial para optimizar el rendimiento del sistema y garantizar un funcionamiento fluido. Las técnicas muestran que la programación de procesos y la gestión de la memoria son esenciales en este sentido.

Las herramientas de monitoreo brindan información valiosa sobre el estado del sistema y permiten a los administradores del sistema identificar y resolver problemas de rendimiento. Estas herramientas son esenciales para mantener la estabilidad y la eficiencia del sistema en entornos de producción.

En resumen, programar y administrar sistemas operativos simultáneamente son áreas complejas que requieren una aplicación cuidadosa y un conocimiento profundo de los principios subyacentes. Con la correcta aplicación de técnicas y herramientas, es posible diseñar sistemas robustos y eficientes que satisfagan las necesidades del usuario y los requisitos del entorno operativo.

# Trabajos citados

(s.f.)., C. S. (s.f.). Obtenido de Realizzazione di apparecchiature a pressione: https://www.cital.it

(s.f.)., G. (s.f.). Obtenido de https://geekland.eu/usar-entender-monitor-de-recursos-top/

(UNED)., U. N. (s.f.). Obtenido de Repositorio Institucional de la UNED.: http://e-spacio.uned.es/fez/view.php?pid=bibliuned:207

Cabalar, P. (. (s.f.). *Sistemas Operativos. Universidad da Coruña*. Obtenido de https://www.dc.fi.udc.es/~so-grado/SO-Procesos-planif.pdf

Castillo Duarte, P. F. (2009). Obtenido de Sistema Web de Gestión Deportiva. Recuperado de enlace al PDF: https://dspace.unl.edu.ec/jspui/bitstream/123456789/14407/1/Castillo%20Duarte,%20Pablo%20Fernando,%20Figueroa%20Robles,%20Lucia%20Margarita.pdf

*Condiciones de carrera en sistemas operativos y programación concurrente.* (2023). Obtenido de https://www.example.com/condiciones-de-carrera

especificado, N. (No especificada de No especificada de No especificada). *Planificación de procesos - Sistemas Operativos*. Obtenido de Cursos Clavijero: https://cursos.clavijero.edu.mx/cursos/182\_so/modulo2/contenidos/tema2.4.2.html

especificado, N. (No especificado de No especificado de No especificado). Obtenido de Virtual Pro: https://www.virtualpro.co/noticias/que-es-la-planificacion-de-procesos-de-un-sistema-operativo

especificado, N. (No especificado de No especificado de No especificado). *Planificación de procesos - Sistemas Operativos*. Obtenido de Cursos Clavijero: https://cursos.clavijero.edu.mx/cursos/182\_so/modulo2/contenidos/tema2.4.2.html

explicación, N. h. (21 de mayo de 2024). *Rol del kernel del sistema operativo en la gestión de procesos*.

Granada., U. d. (s.f.). Obtenido de http://digibug.ugr.es/

Hilos, c. y.-J. (2021). Obtenido de https://javiblog.com/2021/hilos-threads-concurrencia-programacion-asincrona.html[1]

Inc, A. (s.f.). *Monitor de Actividad en la Mac*. Obtenido de https://support.apple.com/es-co/guide/activity-monitor/actmntr1001/mac

Informàtica, C. (27 de abril de 2023). Obtenido de Evolución de los sistemas operativos.: https://www.cisinformatica.cat/es/evolucion-de-los-sistemas-operativos/

IONOS. (s.f.). *Cómo abrir el administrador de tareas en Mac.* Obtenido de https://www.ionos.es/digitalguide/servidores/configuracion/administrador-de-tareas-en-mac/

Madrid, B. d. (s.f.). *E-Archivo.* Obtenido de http://hdl.handle.net/10016/18468

Madrid, U. P. (s.f.). Obtenido de http://oa.upm.es/

Madrid., U. C. (s.f.). *Repositorio Institucional de la Universidad Complutense de Madrid.* . Obtenido de http://eprints.ucm.es/

paralelismo, P. c. (s.f.). *[Modulo 1 - Threads]. (s.f.)*. Obtenido de https://platzi.com/tutoriales/2255-python-intermedio/11485-programacion-concurrente-y-paralelismo-modulo-1-threads-2-threads-y-procesos/[5

Perplexity, A. V. ( 2024). Obtenido de Diferencias entre hilos de usuario y hilos de kernel.

Perplexity., A. V. (21 de mayo de 2024). *Eventos que provocan el cambio de estado de un proceso.*

Risk, R. P. (No especificada de No especificada de No especificada). Obtenido de https://www.piranirisk.com/es/blog/descubra-como-hacer-el-registro-de-eventos-de-riesgo-operativo

Salamanca., U. d. (s.f.). Obtenido de http://gredos.usal.es/

Santander, R. B. (No especificada de No especificada de No especificada). Obtenido de Transacciones bancarias: tipos y características: https://www.santander.com/es/stories/transacciones-bancarias-tipos-y-caracteristicas

Silberschatz, A. G. ((2018)). *Operating System Concepts (10th ed.). Wiley.*

Silberschatz, A. G. (2018). *Operating system concepts (10th ed.). Wiley.*

Silberschatz, A. G. (2018). *Operating System Concepts (10th ed.). Wiley.*

Stallings, W. (2018). Obtenido de Operating systems: Internals and design principles (9th ed.). Pearson.

Tanenbaum, A. S. (2015). *Modern operating systems*.

Tanenbaum, A. S. (2015). *Modern operating systems (4th ed.). Pearson.*

UC3M., B. (s.f.). *Guía temática sobre citas bibliográficas*. Obtenido de https://uc3m.libguides.com/guias\_tematicas/citas\_bibliograficas/APA

Virtual, A. (21 de mayo de 2024). Obtenido de Estados posibles de un proceso y ejemplos.