**1.1. Enumere y defina brevemente los cuatro elementos principales de un computador.**

Al más alto nivel, un computador consta del procesador, la memoria y los componentes de E/S, incluyendo uno o más módulos de cada tipo. Estos componentes se interconectan de manera que se pueda lograr la función principal del computador, que es ejecutar programas. Por tanto, hay cuatro elementos estructurales principales:

* **Procesador.** Controla el funcionamiento del computador y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, se denomina usualmente **unidad central de proceso** (*Central Processing Unit*, CPU).
* **Memoria principal.** Almacena datos y programas. Esta memoria es habitualmente volátil; es decir, cuando se apaga el computador, se pierde su contenido. En contraste, el contenido de la memoria del disco se mantiene incluso cuando se apaga el computador. A la memoria principal se le denomina también *memoria real* o *memoria primaria*.
* **Módulos de E/S.** Transfieren los datos entre el computador y su entorno externo. El entorno externo está formado por diversos dispositivos, incluyendo dispositivos de memoria secundaria (por ejemplo, discos), equipos de comunicaciones y terminales.
* **Bus del sistema.** Proporciona comunicación entre los procesadores, la memoria principal y los módulos de E/S.

**1.2. Defina las dos categorías principales de los registros del procesador.**

Un procesador incluye un conjunto de registros que proporcionan un tipo de memoria que es más rápida y de menor capacidad que la memoria principal. Los registros del procesador sirven para dos funciones:

* **Registros visibles para el usuario.** Permiten al programador en lenguaje máquina o en ensamblador minimizar las referencias a memoria principal optimizando el uso de registros. Para lenguajes de alto nivel, un compilador que realice optimización intentará tomar decisiones inteligentes sobre qué variables se asignan a registros y cuáles a posiciones de memoria principal. Algunos lenguajes de alto nivel, tales como C, permiten al programador sugerir al compilador qué variables deberían almacenarse en registros.
* **Registros de control y estado.** Usados por el procesador para controlar su operación y por rutinas privilegiadas del sistema operativo para controlar la ejecución de programas.

No hay una clasificación nítida de los registros entre estas dos categorías. Por ejemplo, en algunas máquinas el contador de programa es visible para el usuario, pero en muchas otras no lo es. Sin embargo, para el estudio que se presenta a continuación, es conveniente utilizar estas categorías.

**1.3. En términos generales, ¿cuáles son las cuatro acciones distintas que puede especificar una instrucción de máquina?**

Un programa que va a ejecutarse en un procesador consta de un conjunto de instrucciones almacenado en memoria. En su forma más simple, el procesamiento de una instrucción consta de dos pasos: el procesador lee (*busca*) instrucciones de la memoria, una cada vez, y ejecuta cada una de ellas. La ejecución del programa consiste en repetir el proceso de búsqueda y ejecución de instrucciones. La ejecución de la instrucción puede involucrar varias operaciones dependiendo de la naturaleza de la misma.

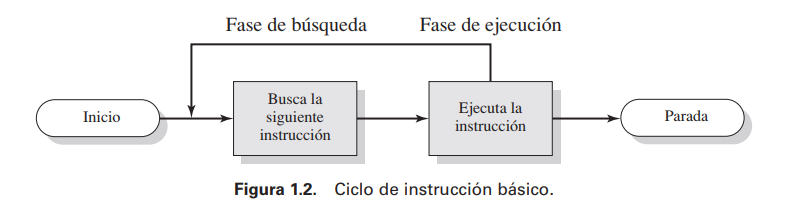
Se denomina *ciclo de instrucción* al procesamiento requerido por una única instrucción.

Al principio de cada ciclo de instrucción, el procesador lee una instrucción de la memoria. En un procesador típico, el contador del programa (PC) almacena la dirección de la siguiente instrucción que se va a leer. A menos que se le indique otra cosa, el procesador siempre incrementa el PC después de cada instrucción ejecutada, de manera que se leerá la siguiente instrucción en orden secuencial (es decir, la instrucción situada en la siguiente dirección de memoria más alta).

La instrucción leída se carga dentro de un registro del procesador conocido como registro de instrucción (IR). La instrucción contiene bits que especifican la acción que debe realizar el procesador. El procesador interpreta la instrucción y lleva a cabo la acción requerida. En general, estas acciones se dividen en cuatro categorías:

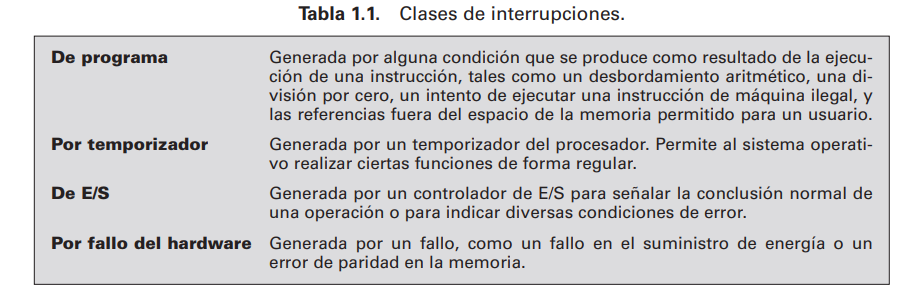
* **Procesador-memoria.** Se pueden transferir datos desde el procesador a la memoria o viceversa.
* **Procesador-E/S.** Se pueden enviar datos a un dispositivo periférico o recibirlos desde el mismo, transfiriéndolos entre el procesador y un módulo de E/S.
* **Procesamiento de datos.** El procesador puede realizar algunas operaciones aritméticas o lógicas sobre los datos.
* **Control.** Una instrucción puede especificar que se va a alterar la secuencia de ejecución. Por ejemplo, el procesador puede leer una instrucción de la posición 149, que especifica que la siguiente instrucción estará en la posición 182. El procesador almacenará en el contador del programa un valor de 182. Como consecuencia, en la siguiente fase de búsqueda, se leerá la instrucción de la posición 182 en vez de la 150.

Una ejecución de una instrucción puede involucrar una combinación de estas acciones.



**1.4. ¿Qué es una interrupción?**

Una interrupción es un mecanismo esencial en los computadores que permite a otros módulos, como la memoria y los dispositivos de entrada/salida (E/S), interrumpir la secuencia normal de ejecución del procesador. Este sistema se implementa para mejorar la eficiencia del procesador y optimizar su tiempo de uso. Por ejemplo, dado que la velocidad de procesamiento del procesador supera con creces la velocidad de algunos dispositivos de E/S, como las impresoras, el procesador puede quedarse inactivo durante períodos prolongados esperando a que se completen las operaciones de E/S. Las interrupciones permiten que el procesador atienda otras tareas mientras espera la finalización de estas operaciones más lentas, lo que maximiza la utilización de sus recursos y minimiza los tiempos de inactividad.



**1.5. ¿Cómo se tratan múltiples interrupciones?**

El estudio realizado hasta el momento ha tratado solamente el caso de que se produzca una única interrupción. Supóngase, sin embargo, que se producen múltiples interrupciones. Por ejemplo, un programa puede estar recibiendo datos de una línea de comunicación e imprimiendo resultados al mismo tiempo. La impresora generará una interrupción cada vez que completa una operación de impresión. El controlador de la línea de comunicación generará una interrupción cada vez que llega una unidad de datos. La unidad podría consistir en un único carácter o en un bloque, dependiendo de la naturaleza del protocolo de comunicaciones. En cualquier caso, es posible que se produzca una interrupción de comunicación mientras se está procesando una interrupción de la impresora.

Cuando se enfrenta con múltiples interrupciones, existen dos alternativas principales para su tratamiento:

La primera opción consiste en *inhabilitar las interrupciones* durante el procesamiento de una interrupción. Esto implica que el procesador ignora cualquier nueva señal de petición de interrupción mientras está ocupado manejando una interrupción previa. Sin embargo, esta estrategia no considera la prioridad o urgencia relativa de las interrupciones, lo que podría llevar a la pérdida de datos importantes si las operaciones de E/S no se procesan rápidamente.

La segunda estrategia implica *asignar prioridades* a las interrupciones y permitir que una interrupción de mayor prioridad interrumpa la ejecución de una de menor prioridad. Por ejemplo, en un sistema con múltiples dispositivos de E/S, cada uno tiene una prioridad asignada y se atienden en orden de prioridad. Esta estrategia garantiza que las interrupciones más urgentes se manejen primero, evitando la pérdida de datos y asegurando una respuesta rápida a eventos críticos.

**1.6. ¿Qué características distinguen a los diversos elementos de una jerarquía de memoria?**

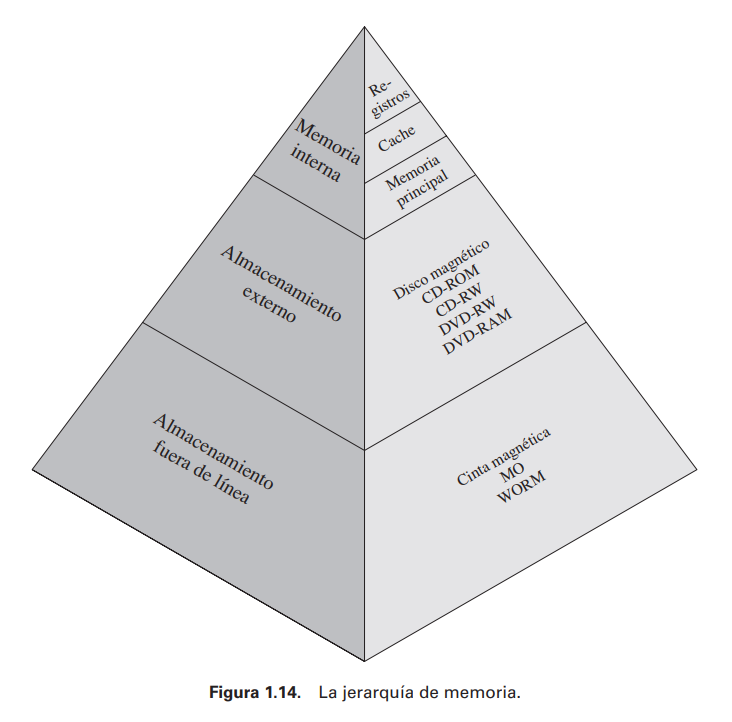
La pregunta sobre cuánta debe ser su capacidad es algo que no tiene límite. Si se dispone de una determinada capacidad, probablemente se desarrollarán aplicaciones que la usarán. La cuestión acerca de la velocidad tiene, hasta cierto tiempo, una respuesta más fácil. Para alcanzar un rendimiento máximo, la memoria debe ser capaz de mantener el ritmo del procesador. Es decir, según el procesador va ejecutando instrucciones, no debería haber pausas esperando que estén disponibles las instrucciones o los operandos. Se debe considerar también la última pregunta. Para un sistema práctico, el coste de la memoria debe ser razonable en relación con los otros componentes.

Como se podría esperar, hay un compromiso entre las tres características fundamentales de la memoria: a saber, coste, capacidad y tiempo de acceso. En cualquier momento dado, se utilizan diversas tecnologías para implementar los sistemas de memoria. En todo este espectro de tecnologías, se cumplen las siguientes relaciones:

* Cuanto menor tiempo de acceso, mayor coste por bit.
* Cuanto mayor capacidad, menor coste por bit.
* Cuanto mayor capacidad, menor velocidad de acceso.

La solución a este dilema consiste en no basarse en un único componente de memoria o en una sola tecnología, sino emplear una **jerarquía de memoria**. En la Figura 1.14 se muestra una jerarquía típica. Según se desciende en la jerarquía, ocurre lo siguiente:

1. Disminución del coste por bit.
2. Aumento de la capacidad.
3. Aumento del tiempo de acceso.
4. Disminución de la frecuencia de acceso a la memoria por parte del procesador.



**1.7. ¿Qué es una memoria cache?**

Una memoria caché es una memoria de acceso rápido que actúa como un intermediario entre el procesador y la memoria principal de un sistema computacional. Su objetivo es reducir el tiempo de acceso a los datos al proporcionar una ubicación de almacenamiento más cercana al procesador y de mayor velocidad que la memoria principal.

La necesidad de una memoria caché surge debido a la discrepancia en la velocidad entre el procesador y la memoria principal. A medida que la velocidad del procesador ha aumentado constantemente a lo largo de los años, la velocidad de acceso a la memoria principal ha permanecido relativamente más lenta. Para superar esta limitación, se emplea el principio de la proximidad, donde una memoria caché más pequeña y rápida contiene copias de partes críticas de la memoria principal.

Cuando el procesador necesita acceder a datos, primero verifica si están almacenados en la caché. Si se encuentran en la caché (lo que se conoce como un "acceso caché hit"), los datos se entregan al procesador rápidamente. En caso contrario, se realiza una lectura de la memoria principal y se carga un bloque de datos en la caché. Esta estrategia aprovecha la proximidad de referencias, ya que es probable que se acceda a datos cercanos en el tiempo nuevamente en el futuro.

**1.8. Enumere y defina brevemente las tres técnicas para las operaciones de E/S.**

Las tres técnicas para llevar a cabo operaciones de E/S son:

1. **E/S programada:** En esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. El software de E/S se escribe de manera que el procesador ejecuta instrucciones que le dan control directo sobre la operación de E/S, incluyendo la verificación del estado del dispositivo, el envío de mandatos de lectura o escritura, y la transferencia de los datos.

2. **E/S dirigida por interrupciones:** Esta técnica aborda el problema de la E/S programada, donde el procesador debe esperar inactivamente hasta que el módulo de E/S esté listo para recibir o enviar más datos. En la E/S dirigida por interrupciones, el procesador genera un mandato de E/S y continúa realizando otras tareas. El módulo de E/S interrumpe más tarde al procesador cuando está listo para intercambiar datos. Esta técnica es más eficiente que la E/S programada, ya que elimina la espera innecesaria del procesador.

3. **Acceso directo a memoria (DMA):** En el acceso directo a memoria, un módulo de DMA se encarga de la transferencia de datos entre la memoria y un dispositivo de E/S sin la intervención activa del procesador. El procesador delega la operación de E/S al módulo de DMA, que se encarga de la transferencia del bloque completo de datos, palabra por palabra, hacia o desde la memoria sin pasar por el procesador. Aunque el procesador está involucrado al principio y al final de la transferencia, durante la transferencia en sí, el procesador puede continuar con otras tareas. Aunque puede haber competencia por el uso del bus entre el módulo de DMA y el procesador, el DMA es mucho más eficiente para transferencias de E/S de múltiples palabras en comparación con las técnicas dirigidas por interrupciones o programadas.

**1.9. ¿Cuál es la diferencia entre la proximidad espacial y la temporal?**

La **proximidad espacial** se refiere a la tendencia de una ejecución a involucrar posiciones de memoria que están agrupadas. Esto refleja la tendencia de un procesador a acceder secuencialmente a las instrucciones. La proximidad espacial también refleja la tendencia de un programa a acceder de forma secuencial a las posiciones de datos, como cuando se procesa una tabla de datos.

La **proximidad temporal** hace referencia a la tendencia de un procesador a acceder a posiciones de memoria que se han utilizado recientemente. Por ejemplo, cuando se ejecuta un bucle, el procesador ejecuta el mismo juego de instrucciones repetidamente.

Tradicionalmente, la proximidad temporal se explota manteniendo en la memoria cache los valores de las instrucciones y los datos usados recientemente aprovechando una jerarquía de cache. La proximidad espacial se explota generalmente utilizando bloques de cache más grandes e incorporando mecanismos de lectura anticipada (se buscan elementos cuyo uso se prevé) en la lógica de control de la cache. **🡪 1.10**

**1.10. En general, ¿cuáles son las estrategias para aprovechar la proximidad espacial y la temporal?**

Para aprovechar la proximidad espacial y temporal en el diseño de sistemas de memoria y caché, se emplean diversas estrategias:

1. **Proximidad Espacial:**

- **Bloques de caché más grandes:** Al aumentar el tamaño de los bloques de caché, se incrementa la probabilidad de que se almacenen datos adyacentes en la memoria principal en la caché. Esto mejora la eficacia de la caché al explotar la proximidad espacial, ya que es más probable que se acceda a datos cercanos en la memoria principal simultáneamente.

**- Lectura anticipada (prefetching):** Se anticipa la necesidad de datos futuros y se los carga en la caché antes de que se soliciten explícitamente por el procesador. Esto se basa en patrones de acceso históricos o algoritmos predictivos para identificar datos que probablemente se necesitarán en el futuro cercano.

2. **Proximidad Temporal:**

- **Mantenimiento de valores recientes en caché**: Los datos e instrucciones utilizados recientemente se mantienen en la caché para aprovechar la proximidad temporal. Esto permite un acceso más rápido a estos datos si son requeridos nuevamente en un futuro cercano.

- **Políticas de reemplazo inteligente:** Se utilizan algoritmos de reemplazo en la caché que tienen en cuenta la frecuencia y recencia de acceso a los datos. Por ejemplo, el algoritmo LRU (Least Recently Used) reemplaza los datos menos recientes utilizados en la caché, lo que ayuda a mantener los datos más recientes en la caché para aprovechar la proximidad temporal.

En resumen, tanto la proximidad espacial como la temporal se aprovechan mediante el diseño de cachés con políticas de almacenamiento y reemplazo que optimizan el acceso a los datos. El objetivo es maximizar el rendimiento del sistema al minimizar los tiempos de acceso a la memoria y aumentar la eficiencia de la caché.

**2.1. ¿Cuáles son los tres objetivos de diseño de un sistema operativo?**

Un sistema operativo es un programa que controla la ejecución de aplicaciones y programas y que actúa como interfaz entre las aplicaciones y el hardware del PC. Se puede considerar que un sistema operativo tiene los siguientes tres objetivos:

* **Facilidad de uso.** Un sistema operativo facilita el uso de un computador.
* **Eficiencia.** Un sistema operativo permite que los recursos de un sistema de computación se puedan utilizar de una manera eficiente.
* **Capacidad para evolucionar.** Un sistema operativo se debe construir de tal forma que se puedan desarrollar, probar e introducir nuevas funciones en el sistema sin interferir con su servicio.

**2.2. ¿Qué es el núcleo de un sistema operativo?**

Un programa de usuario ejecuta en **modo usuario**, en el cual los usuarios no pueden acceder a ciertas áreas de memoria y no puede ejecutar ciertas instrucciones. El sistema operativo ejecuta en modo sistema, o lo que se denomina **modo núcleo**, en el cual se pueden ejecutar instrucciones privilegiadas y se puede acceder a áreas de memoria protegidas.

Una porción del sistema operativo se encuentra en la memoria principal. Esto incluye el ***kernel*,** o **núcleo,** que contiene las funciones del sistema operativo más frecuentemente utilizadas y, en cierto momento, otras porciones del sistema operativo actualmente en uso. El resto de la memoria principal contiene programas y datos de usuario. La asignación de este recurso (memoria principal) es controlada de forma conjunta por el sistema operativo y el hardware de gestión de memoria del procesador. El sistema operativo decide cuándo un programa en ejecución puede utilizar un dispositivo de E/S y controla el acceso y uso de los ficheros. El procesador es también un recurso, y el sistema operativo debe determinar cuánto tiempo de procesador debe asignarse a la ejecución de un programa de usuario particular. En el caso de un sistema multiprocesador, esta decisión debe ser tomada por todos los procesadores.

**2.3. ¿Qué es multiprogramación?**

La multiprogramación es una técnica utilizada en sistemas operativos que permite ejecutar múltiples programas de manera concurrente en una computadora, de manera que parece que están ejecutándose al mismo tiempo. Aunque en realidad el procesador ejecuta instrucciones de un programa a la vez, la multiprogramación logra la ilusión de concurrencia al alternar rápidamente entre los diferentes programas en ejecución. En otras palabras, La multiprogramación es una técnica sistemas operativos donde varios programas se cargan en la memoria simultáneamente. El CPU cambia rápidamente entre los programas, permitiendo la imitación de una ejecución simultánea.

La multiprogramación aprovecha los períodos de inactividad del procesador, como las esperas de E/S (Entrada/Salida) o los tiempos de espera de memoria, para ejecutar otros programas en segundo plano. De esta manera, se optimiza el uso de los recursos del sistema y se aumenta la eficiencia del procesador, permitiendo que múltiples usuarios o tareas compartan la misma computadora de manera efectiva.

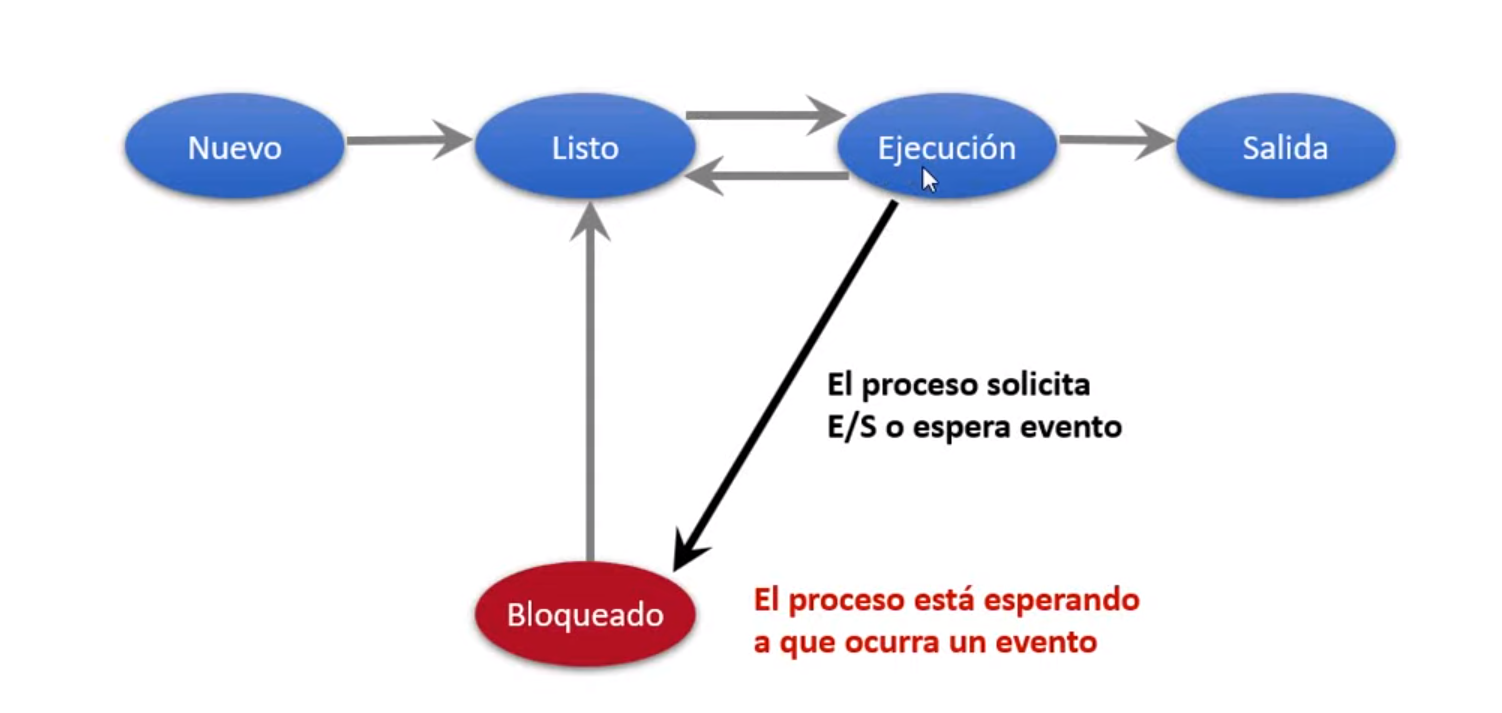
**2.4. ¿Qué es un proceso?** 🡪 Mas información en Resumen Unidad 1

* Un programa en ejecución.
* Una instancia de un programa ejecutándose en un computador.
* La entidad que se puede asignar o ejecutar en un procesador.
* Una unidad de actividad caracterizada por un solo hilo secuencial de ejecución, un estado actual, y un conjunto de recursos del sistema asociados.

Un proceso es un programa en ejecución en un sistema operativo. Puede ser una aplicación de usuario, un servicio del sistema, o incluso parte del propio sistema operativo. Cada proceso tiene su propio espacio de memoria asignado, que incluye el código ejecutable, los datos, la pila de ejecución y otras secciones necesarias para su funcionamiento.

Los procesos pueden interactuar entre sí y con el sistema operativo mediante el intercambio de datos, señales y solicitudes de servicios. Cada proceso se ejecuta de forma independiente de otros procesos, y el sistema operativo se encarga de gestionar los recursos compartidos, como la CPU, la memoria y los dispositivos de E/S, para garantizar una ejecución segura y eficiente.

Además, los procesos pueden tener diferentes estados, como en ejecución, listo, bloqueado o terminado, dependiendo de su actividad y necesidades de recursos en un momento dado. La gestión de procesos es una parte fundamental de los sistemas operativos modernos, ya que permite la multitarea y la ejecución concurrente de múltiples programas en un mismo sistema.



Lo que se necesita para enfrentarse a estos problemas es una forma sistemática de monitorizar y controlar la ejecución de varios programas en el procesador. El concepto de proceso proporciona los fundamentos. Se puede considerar que un proceso está formado por los siguientes tres componentes:

* Un programa ejecutable.
* Los datos asociados que necesita el programa (variables, espacio de trabajo, *buffers*, etc.).
* El contexto de ejecución del programa.

**2.5. ¿Cómo utiliza el sistema operativo el contexto de ejecución de un proceso?**

El **contexto de ejecución**, o **estado del proceso**, es el conjunto de datos interno por el cual el sistema operativo es capaz de supervisar y controlar el proceso. Esta información interna está separada del proceso, porque el sistema operativo tiene información a la que el proceso no puede acceder. El contexto incluye toda la información que el sistema operativo necesita para gestionar el proceso y que el procesador necesita para ejecutar el proceso apropiadamente. El contexto incluye el contenido de diversos registros del procesador, tales como el contador de programa y los registros de datos. También incluye información de uso del sistema operativo, como la prioridad del proceso y si un proceso está esperando por la finalización de un evento de E/S particular.

El proceso puede verse como una estructura de datos. Un proceso puede estar en ejecución o esperando ejecutarse. El **estado** completo del proceso en un instante dado se contiene en su contexto. Esta estructura permite el desarrollo de técnicas potentes que aseguran la coordinación y la cooperación entre los procesos. Se pueden diseñar e incorporar nuevas características en el sistema operativo (por ejemplo, la prioridad), expandiendo el contexto para incluir cualquier información nueva que se utilice para dar soporte a dicha característica.

**2.6. Liste y explique brevemente cinco responsabilidades relacionadas con la gestión de almacenamiento de un sistema operativo típico.**

**GESTIÓN DE MEMORIA**

Un entorno de computación que permita programación modular y el uso flexible de los datos puede ayudar a resolver mejor las necesidades de los usuarios. Los gestores de sistema necesitan un control eficiente y ordenado de la asignación de los recursos. Para satisfacer estos requisitos, el sistema operativo tiene cinco responsabilidades principales de gestión de almacenamiento:

1. **Aislamiento de procesos.** El sistema operativo debe evitar que los procesos independientes interfieran en la memoria de otro proceso, tanto datos como instrucciones.
2. **Asignación y gestión automática.** Los programas deben tener una asignación dinámica de memoria por demanda, en cualquier nivel de la jerarquía de memoria. La asignación debe ser transparente al programador. Por tanto, el programador no debe preocuparse de aspectos relacionados con limitaciones de memoria, y el sistema operativo puede lograr incrementar la eficiencia, asignando memoria a los trabajos sólo cuando se necesiten.
3. **Soporte a la programación modular.** Los programadores deben ser capaces de definir módulos de programación y crear, destruir, y alterar el tamaño de los módulos dinámicamente.
4. **Protección y control de acceso.** La compartición de memoria, en cualquier nivel de la jerarquía de memoria, permite que un programa direccione un espacio de memoria de otro proceso. Esto es deseable cuando se necesita la compartición por parte de determinadas aplicaciones.Otras veces, esta característica amenaza la integridad de los programas e incluso del propio sistema operativo. El sistema operativo debe permitir que varios usuarios puedan acceder de distintas formas a porciones de memoria.
5. **Almacenamiento a largo plazo.** Muchas aplicaciones requieren formas de almacenar la información durante largos periodos de tiempo, después de que el computador se haya apagado.

Normalmente, los sistemas operativos alcanzan estos requisitos a través del uso de la memoria virtual y las utilidades de los sistemas operativos. El sistema operativo implementa un almacenamiento a largo plazo, con la información almacenada en objetos denominados ficheros. El fichero es un concepto lógico, conveniente para el programador y es una unidad útil de control de acceso y protección para los sistemas operativos.

**2.7. Explique la distinción entre una dirección real y una dirección virtual.**

La memoria virtual es una utilidad que permite a los programas direccionar la memoria desde un punto de vista lógico, sin importar la cantidad de memoria principal física disponible. La memoria virtual fue concebida como un método para tener múltiples trabajos de usuario residiendo en memoria principal de forma concurrente, de forma que no exista un intervalo de tiempo de espera entre la ejecución de procesos sucesivos, es decir, mientras un proceso se escribe en almacenamiento secundario y se lee el proceso sucesor. Debido a que los procesos varían de tamaño, si el procesador planifica un determinado número de procesos, es difícil almacenarlos compactamente en memoria principal. Se introdujeron los sistemas de paginación, que permiten que los procesos se compriman en un número determinado de bloques de tamaño fijo, denominados páginas. Un programa referencia una palabra por medio de una **dirección virtual**, que consiste en un número de página y un desplazamiento dentro de la página. Cada página de un proceso se puede localizar en cualquier sitio de memoria principal. El sistema de paginación proporciona una proyección dinámica entre las direcciones virtuales utilizadas en el programa y una **dirección real**, o dirección física, de memoria principal.

*Conclusión:*

* **Dirección Real**: Es la ubicación física exacta en la memoria física de la computadora. Es como una dirección de casa en un mapa: te lleva directamente al lugar específico donde se encuentra la información en la memoria física de la computadora.
* **Dirección Virtual**: Es una dirección lógica o imaginaria utilizada por los programas. En lugar de referirse directamente a una ubicación física en la memoria, se refiere a una ubicación abstracta que luego se traduce en una dirección real por el sistema operativo. Es como una dirección postal que te guía a una ubicación general, pero el correo se entrega a una dirección real específica. La memoria virtual permite que los programas se ejecuten como si tuvieran acceso a toda la memoria física, aunque en realidad se utiliza solo una parte de ella. Esto es útil cuando se tienen múltiples programas en ejecución y se necesita administrar eficientemente la memoria disponible.

La dirección virtual sería como la dirección de código postal, que es la dirección lógica o abstracta utilizada por el programa. Mientras que la dirección real sería como la dirección física de una casa, que es la ubicación exacta en la memoria física de la computadora. Cuando el programa necesita acceder a una parte específica de su código o datos en la memoria, el sistema operativo se encarga de traducir esa dirección virtual en una dirección real, permitiendo así el acceso a la información almacenada en la memoria principal. Esto es fundamental para garantizar que los programas puedan ejecutarse de manera eficiente y sin conflictos en un entorno de memoria compartida.

**2.8. Describa la técnica de planificación round-robin o turno rotatorio.**

Una responsabilidad clave de los sistemas operativos es la gestión de varios recursos disponibles para ellos (espacio de memoria principal, dispositivos de E/S, procesadores) y para planificar su uso por parte de los distintos procesos activos.

La planificación y la gestión de recursos son esencialmente problemas de investigación, y se pueden aplicar los resultados matemáticos de esta disciplina. Adicionalmente, medir la actividad del sistema es importante para ser capaz de monitorizar el rendimiento y realizar los ajustes correspondientes.

La Figura 2.11 sugiere los principales elementos del sistema operativo relacionados con la planificación de procesos y la asignación de recursos en un entorno de multiprogramación. El sistema operativo mantiene un número de colas, cada una de las cuales es simplemente una lista de procesos esperando por algunos recursos. La cola a corto plazo está compuesta por procesos que se encuentran en memoria principal (o al menos una porción mínima esencial de cada uno de ellos está en memoria principal) y están listos para ejecutar, siempre que el procesador esté disponible. Cualquiera de estos procesos podría usar el procesador a continuación. Es responsabilidad del planificador a corto plazo, o *dispatcher*, elegir uno de ellos. Una estrategia común es asignar en orden a cada proceso de la cola un intervalo de tiempo; esta técnica se conoce como ***round-robin*** o **turno rotatorio**. En efecto, la técnica de turno rotatorio emplea una cola circular. Otra estrategia consiste en asignar niveles de prioridad a los distintos procesos, siendo el planificador el encargado de elegir los procesos en orden de prioridad.

La cola a largo plazo es una lista de nuevos trabajos esperando a utilizar el procesador. El sistema operativo añade trabajos al sistema transfiriendo un proceso desde la cola a largo plazo hasta la cola a corto plazo. En este punto, se debe asignar una porción de memoria principal al proceso entrante. Por tanto, el sistema operativo debe estar seguro de que no sobrecarga la memoria o el tiempo de procesador admitiendo demasiados procesos en el sistema. Hay una cola de E/S por cada dispositivo de E/S. Más de un proceso puede solicitar el uso del mismo dispositivo de E/S. Todos los procesos que esperan utilizar dicho dispositivo, se encuentran alineados en la cola del dispositivo. De nuevo, el sistema operativo debe determinar a qué proceso le asigna un dispositivo de E/S disponible.

*Conclusión:*

La técnica de planificación round-robin, o turno rotatorio, es una estrategia utilizada por los sistemas operativos para asignar tiempo de procesador a los procesos en ejecución de manera equitativa y justa.

Imagina que tienes una lista de tareas por hacer y quieres asegurarte de que todas tengan la misma oportunidad de ser completadas. En lugar de enfocarte en una tarea a la vez, decides dividir tu tiempo entre todas las tareas de manera igualitaria. Esto es lo que hace el planificador round-robin.

En un sistema operativo, los procesos listos para ejecutar se colocan en una cola. El planificador round-robin toma esta cola y asigna un pequeño intervalo de tiempo de procesador a cada proceso en un orden circular. Cada proceso obtiene una rebanada de tiempo de CPU para realizar su trabajo antes de pasar al siguiente proceso en la cola. Una vez que todos los procesos han recibido su turno, el planificador vuelve al principio de la cola y comienza de nuevo, rotando continuamente a través de los procesos disponibles.

Esta técnica asegura que ningún proceso monopolice el tiempo de CPU y que todos los procesos tengan la oportunidad de avanzar en su ejecución de manera justa. Es como compartir un pastel equitativamente entre amigos: todos reciben una porción antes de que se repita el ciclo.

*Además:*

La cola a corto y largo plazo son estructuras utilizadas por los sistemas operativos para gestionar los procesos en ejecución y su acceso a los recursos del sistema.

- **Cola a corto plazo**: Esta cola contiene los procesos que están listos para ser ejecutados en la CPU. Estos procesos ya están en la memoria principal y solo están esperando su turno para ser ejecutados por el procesador. El planificador a corto plazo, también conocido como dispatcher, elige de manera eficiente qué proceso se ejecutará a continuación. En el contexto de la técnica de planificación round-robin, esta cola sería utilizada para almacenar los procesos que están listos para recibir una rebanada de tiempo de CPU.

- **Cola a largo plazo**: Por otro lado, la cola a largo plazo contiene los procesos que aún no están en memoria principal y están esperando ser admitidos en el sistema para su ejecución. Estos pueden ser procesos recién creados o procesos que previamente se habían suspendido o terminado y que ahora deben ser cargados en memoria. El sistema operativo debe decidir cuántos de estos procesos admitir en la memoria principal en función de la disponibilidad de recursos y evitar la sobrecarga del sistema.

En resumen, la cola a corto plazo maneja los procesos que ya están en memoria y listos para ser ejecutados, mientras que la cola a largo plazo maneja los procesos que están esperando ser cargados en memoria para su ejecución futura. Ambas colas son importantes para garantizar una gestión eficiente de los recursos del sistema y un equilibrio adecuado entre la concurrencia y la carga del sistema.

**2.9. Explique la diferencia entre un núcleo monolítico y un micronúcleo.**

Hasta hace relativamente poco tiempo, la mayoría de los sistemas operativos estaban formados por un gran **núcleo monolítico**. Estos grandes núcleos proporcionan la mayoría de las funcionalidades consideradas propias del sistema operativo, incluyendo la planificación, los sistemas de ficheros, las redes, los controladores de dispositivos, la gestión de memoria y otras funciones. Normalmente, un núcleo monolítico se implementa como un único proceso, con todos los elementos compartiendo el mismo espacio de direcciones. Una **arquitectura micronúcleo** asigna sólo unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC), y la planificación básica. Ciertos procesos proporcionan otros servicios del sistema operativo, algunas veces denominados servidores, que ejecutan en modo usuario y son tratados como cualquier otra aplicación por el micronúcleo. Esta técnica desacopla el núcleo y el desarrollo del servidor. Los servidores pueden configurarse para aplicaciones específicas o para determinados requisitos del entorno. La técnica micronúcleo simplifica la implementación, proporciona flexibilidad y se adapta perfectamente a un entorno distribuido. En esencia, un micronúcleo interactúa con procesos locales y remotos del servidor de la misma forma, facilitando la construcción de los sistemas distribuidos.

*Conclusión:*

Imagina que el sistema operativo es como una gran ciudad y el núcleo es el centro de operaciones, donde se toman todas las decisiones importantes. Ahora, hay dos formas diferentes de organizar este centro:

- **Núcleo Monolítico:** Es como tener un mega-edificio que alberga todas las funciones importantes del sistema operativo: desde la planificación y la gestión de memoria hasta las redes y los controladores de dispositivos. Todo está integrado en un solo lugar, compartiendo el mismo espacio y recursos. Es como tener un edificio enorme donde viven todas las funciones del sistema operativo juntas.

- **Micronúcleo:** Ahora, imagina que en lugar de un mega-edificio, tienes un pequeño núcleo que solo se encarga de las funciones esenciales: almacenamiento, comunicación entre procesos y planificación básica. Todo lo demás, como las redes, los sistemas de archivos y los controladores de dispositivos, se manejan por separado, como edificios independientes llamados "servidores". Estos servidores funcionan como aplicaciones normales, pero están especializados en ciertas tareas del sistema operativo. Es como tener un pequeño núcleo central que coordina diferentes servicios, cada uno en su propio edificio especializado.

Entonces, la diferencia clave es que el núcleo monolítico tiene todo integrado en un solo lugar, mientras que el micronúcleo divide las funciones en partes más pequeñas y flexibles, lo que permite una mayor personalización y adaptabilidad, especialmente en entornos distribuidos.

**2.10. ¿En qué consiste el uso de multihilos o multithreading?**

El ***Multitheading*** es una técnica en la cual un proceso, ejecutando una aplicación, se divide en una serie de hilos o *threads* que pueden ejecutar concurrentemente. Se pueden hacer las siguientes distinciones:

* ***Thread* o hilo.** Se trata de una unidad de trabajo. Incluye el contexto del procesador (que contiene el contador del programa y el puntero de pila) y su propia área de datos para una pila (para posibilitar el salto a subrutinas). Un hilo se ejecuta secuencialmente y se puede interrumpir de forma que el procesador pueda dar paso a otro hilo.
* **Proceso.** Es una colección de uno o más hilos y sus recursos de sistema asociados (como la memoria, conteniendo tanto código, como datos, ficheros abiertos y dispositivos). Esto corresponde al concepto de programa en ejecución. Dividiendo una sola aplicación en múltiples hilos, el programador tiene gran control sobre la modularidad de las aplicaciones y la temporización de los eventos relacionados con la aplicación.

La técnica *multithreading* es útil para las aplicaciones que llevan a cabo un número de tareas esencialmente independientes que no necesitan ser serializadas. Un ejemplo es un servidor de bases de datos que escucha y procesa numerosas peticiones de cliente. Con múltiples hilos ejecutándose dentro del mismo proceso, intercambiar la ejecución entre los hilos supone menos sobrecarga del procesador que intercambiar la ejecución entre diferentes procesos pesados. Los hilos son también útiles para estructurar procesos que son parte del núcleo del sistema operativo.

*Conclusión:*

El uso de multihilos o multithreading consiste en la capacidad de un programa para realizar múltiples tareas simultáneamente dentro de un solo proceso. Los hilos son como subprocesos dentro de un programa que pueden ejecutar diferentes partes del código de forma independiente. Esto significa que un programa puede realizar múltiples acciones al mismo tiempo, como procesar datos, manejar la interfaz de usuario y realizar operaciones de red, todo dentro del mismo proceso. Esto es especialmente útil en aplicaciones que requieren una respuesta rápida y que tienen múltiples tareas que se pueden realizar en paralelo.

**3.1. ¿Qué es una traza de instrucciones?**

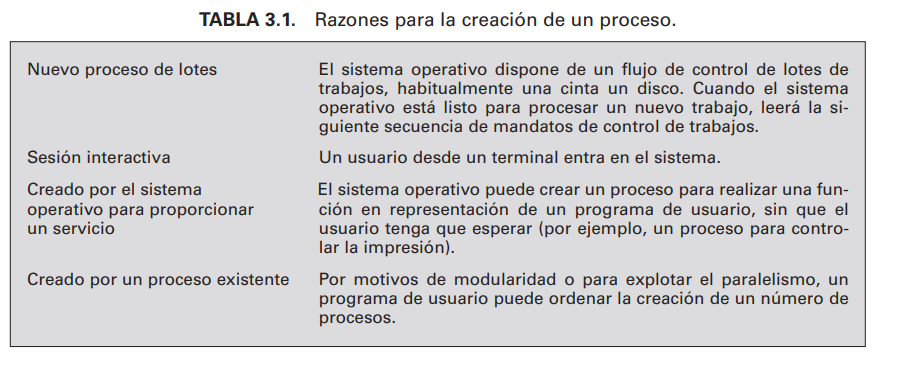
Una traza de instrucciones es una secuencia ordenada de las instrucciones ejecutadas por un proceso durante su ejecución. En el contexto del procesamiento, cada proceso tiene su propia traza de instrucciones, que representa la secuencia de acciones realizadas por el procesador mientras ejecuta ese proceso en particular. Por lo tanto, la traza de instrucciones de un proceso proporciona una visión detallada de las operaciones específicas llevadas a cabo por ese proceso a lo largo del tiempo.

Además, las trazas de instrucciones de varios procesos pueden ser analizadas para comprender cómo se entrelazan las ejecuciones de los diferentes procesos en el procesador. Esto permite caracterizar el comportamiento del procesador en términos de cómo gestiona y ejecuta múltiples procesos concurrentemente.

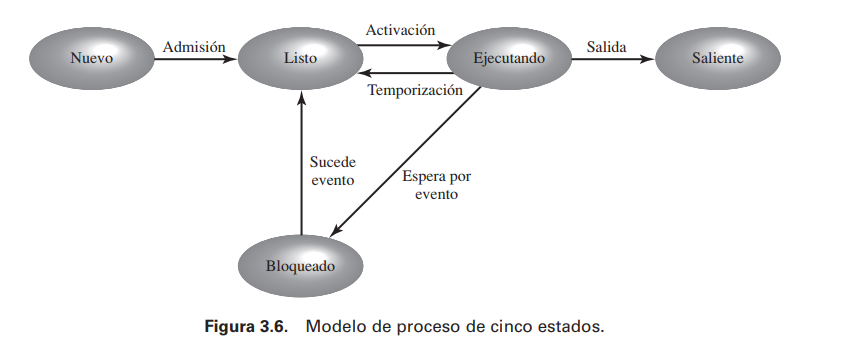
**3.2. ¿Cuáles son los eventos comunes que llevan a la creación de un proceso?**

Existen cuatro eventos comunes que llevan a la creación de un proceso, como se indica en la Tabla 3.1. En un entorno por lotes, un proceso se crea como respuesta a una solicitud de trabajo. En un entorno interactivo, un proceso se crea cuando un nuevo usuario entra en el sistema. En ambos casos el sistema operativo es responsable de la creación de nuevos procesos. Un sistema operativo puede, a

petición de una aplicación, crear procesos. Por ejemplo, si un usuario solicita que se imprima un fichero, el sistema operativo puede crear un proceso que gestione la impresión. El proceso solicitado puede, de esta manera, operar independientemente del tiempo requerido para completar la tarea de impresión.



3.3. Para el modelo de procesamiento de la Figura 3.6, defina brevemente cada estado.



* **Ejecutando.** El proceso está actualmente en ejecución. Para este capítulo asumimos que el computador tiene un único procesador, de forma que sólo un proceso puede estar en este estado en un instante determinado.
* **Listo.** Un proceso que se prepara para ejecutar cuando tenga oportunidad.
* **Bloqueado.** Un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.
* **Nuevo.** Un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su bloque de control de proceso (BCP) si ha sido creado.
* **Saliente.** Un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razón.

**3.4. ¿Qué significa la expulsión de un proceso?**

Una de las transiciones que podemos observar en la Figura 3.6 es la de:

**Ejecutando** 🡪 **Listo.** La razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos. Es de particular importancia el caso en el cual el sistema operativo asigna diferentes niveles de prioridad a diferentes procesos. Supóngase, por ejemplo, que el proceso A está ejecutando a un determinado nivel de prioridad, y el proceso B, a un nivel de prioridad mayor, y que se encuentra bloqueado. Si el sistema operativo se da cuenta de que se produce un evento al cual el proceso B está esperando, moverá el proceso B al estado de Listo. Esto puede interrumpir al proceso A y poner en ejecución al proceso B. Decimos, en este caso, que el sistema operativo **ha expulsado al proceso** A**(\*)**. Adicionalmente, un proceso puede voluntariamente dejar de utilizar el procesador. Un ejemplo son los procesos que realiza alguna función de auditoría o de mantenimiento de forma periódica.

**(\*)** En general, el término expulsión (*preemption*) se define como la reclamación de un recurso por parte de un proceso antes de que el proceso que lo poseía finalice su uso. En este caso, el recurso es el procesador. El proceso está ejecutando y puede continuar su ejecución pero es expulsado por otro proceso que va a entrar a ejecutar.

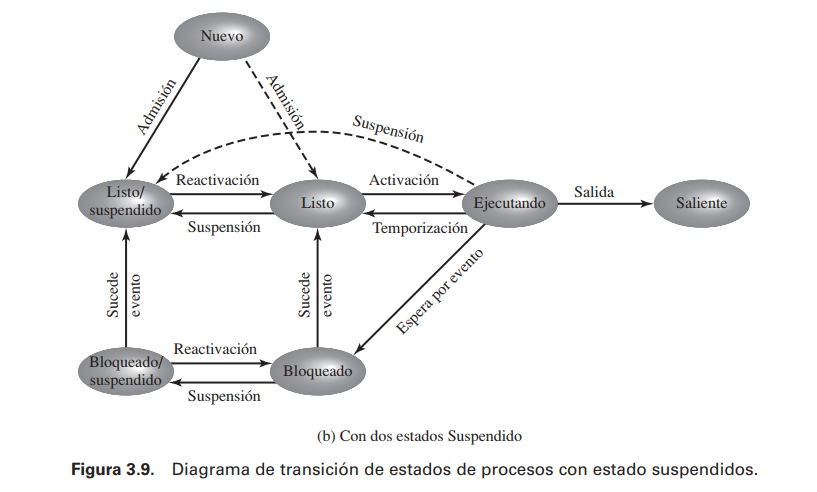
**3.5. ¿Que es el *swapping* y cuál es su objetivo?**

El swapping es una técnica utilizada en los sistemas operativos para gestionar la memoria cuando la capacidad de la memoria principal es insuficiente para alojar todos los procesos activos. Consiste en mover parte o la totalidad de un proceso de la memoria principal al almacenamiento secundario, como el disco duro, liberando así espacio en la memoria principal para otros procesos.

Cuando no hay procesos en la memoria principal que estén listos para ser ejecutados de inmediato, es decir, no hay procesos en estado "Listo", el sistema operativo puede seleccionar uno de los procesos bloqueados y transferirlo al disco, colocándolo en un estado llamado "Suspendido". Luego, puede cargar otro proceso de la cola de procesos suspendidos en la memoria principal para su ejecución.

El objetivo del swapping es optimizar el uso de la memoria principal y mejorar el rendimiento del sistema al garantizar que los procesos activos tengan acceso rápido a la memoria cuando sea necesario. Aunque el swapping implica operaciones de E/S, como la transferencia de datos entre la memoria principal y el disco, generalmente mejora el rendimiento del sistema al permitir una gestión más eficiente de la memoria.´

**3.6. ¿Por qué la Figura 3.9 tiene dos estados bloqueados?**



**Bloqueado.** El proceso está en memoria principal y esperando un evento.

**Bloqueado/Suspendido.** El proceso está en almacenamiento secundario y esperando un evento.

Entre estos 2 estados, sus transiciones quedarían:

**Bloqueado 🡪 Bloqueado/Suspendido. Suspensión.** Si no hay procesos listos, entonces al menos uno de los procesos bloqueados se transfiere al disco para hacer espacio para otro proceso que no se encuentra bloqueado. Esta transición puede realizarse incluso si hay procesos listos disponibles, si el sistema operativo determina que el proceso actualmente en ejecución o los procesos listos que desea ejecutar requieren más memoria principal para mantener un rendimiento adecuado.

**Bloqueado/Suspendido** 🡪 **Bloqueado. Reactivación.** La incursión de esta transición puede parecer propia de un diseño más bien pobre. Después de todo, si hay un proceso que no está listo para ejecutar y que no está en memoria principal, ¿qué sentido tiene traerlo? Pero considérese el siguiente escenario: un proceso termina, liberando alguna memoria principal. Hay un proceso en la cola de Bloqueados/Suspendidos con mayor prioridad que todos los procesos en la cola de Listos/Suspendidos y el sistema operativo tiene motivos para creer que el evento que lo bloquea va a ocurrir en breve. Bajo estas circunstancias, sería razonable traer el proceso Bloqueado a memoria por delante de los procesos Listos.

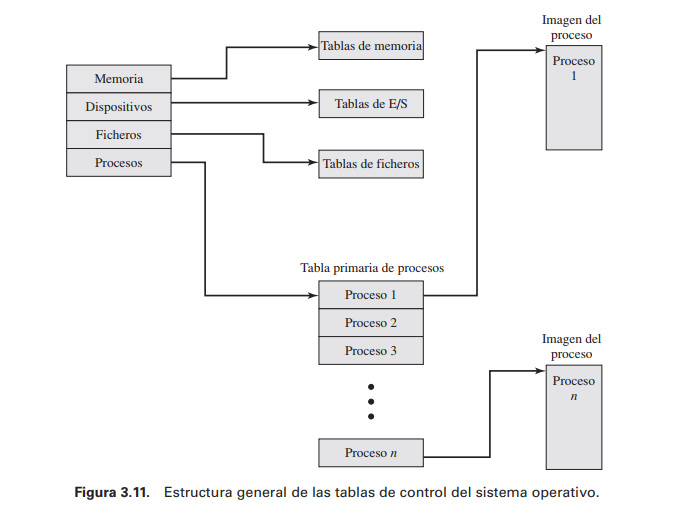
**3.7. Indique cuatro características de un proceso suspendido.**

Podemos analizar el concepto de procesos suspendidos, definiendo un proceso suspendido como el que cumple las siguientes características:

1. El proceso no está inmediatamente disponible para su ejecución.
2. El proceso puede estar o no a la espera de un evento, si es así, la condición de bloqueo es independiente de la condición estar suspendido, y si sucede el evento que lo bloquea, eso no habilita al proceso para su ejecución inmediata.
3. El proceso fue puesto en estado suspendido por un agente: bien el proceso mismo, el proceso padre o el sistema operativo, con el propósito de prevenir su ejecución.
4. El proceso no puede ser recuperado de este estado hasta que el agente explícitamente así lo indique.

**3.8. ¿Para qué tipo de entidades el sistema operativo mantiene tablas de información por motivos de gestión?**

Si el sistema operativo se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para proporcionar esta información es el siguiente: el sistema operativo construye y mantiene tablas de información sobre cada entidad que gestiona. Se indica una idea general del alcance de esta tarea en la Figura 3.11, que muestra cuatro diferentes tipos de tablas mantenidas por el sistema operativo: memoria, E/S, ficheros y procesos. A pesar de que los detalles difieren de un sistema operativo a otro, fundamentalmente, todos los sistemas operativos mantienen información de estas cuatro categorías.

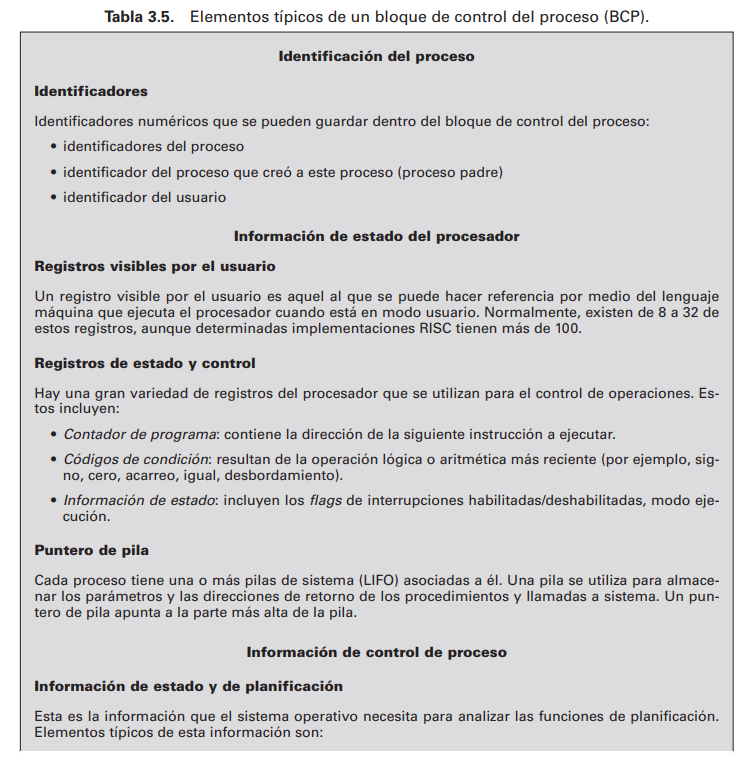
****

**3.9. Indique tres categorías generales de información que hay en el bloque de control de proceso.**

Cada proceso está asociado a un número de atributos que son utilizados por el sistema operativo para controlar el proceso. Normalmente, el conjunto de estos atributos se denomina **bloque de control del proceso (BCP).** Nos podemos referir al conjunto de programa, datos, pila, y atributos, como la **imagen del proceso.**

Podemos agrupar la información del bloque de control del proceso en tres categorías generales:

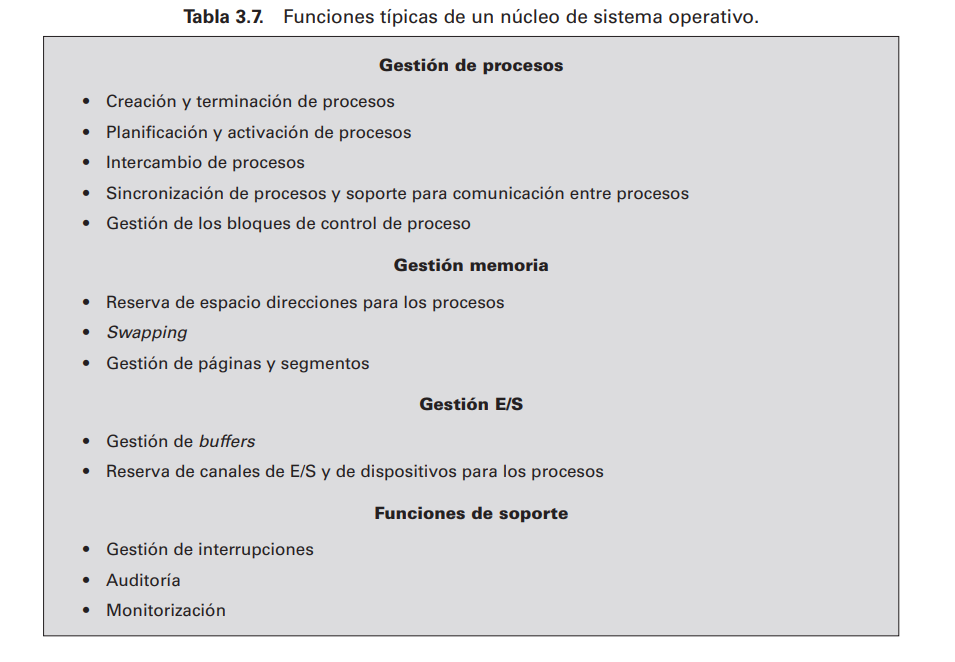
* Identificación del proceso
* Información de estado del procesador
* Información de control del proceso

****

**3.10. ¿Por qué se necesitan dos modos (usuario y núcleo)?**

Antes de continuar con nuestra explicación sobre cómo el sistema operativo gestiona los procesos, necesitamos distinguir entre los modos de ejecución del procesador, normalmente asociados con el sistema operativo y los asociados con los programas de usuario. Muchos procesadores proporcionan al menos dos modos de ejecución. Ciertas instrucciones se pueden ejecutar en modos privilegiados únicamente. Éstas incluirían lectura y modificación de los registros de control, por ejemplo la palabra de estado de programa; instrucciones de E/S primitivas; e instrucciones relacionadas con la gestión de memoria. Adicionalmente, ciertas regiones de memoria sólo se pueden acceder en los modos más privilegiados.

El modo menos privilegiado a menudo se denomina **modo usuario**, porque los programas de usuario típicamente se ejecutan en este modo. El modo más privilegiado se denomina **modo sistema**, **modo control** o **modo núcleo**. Este último término se refiere al núcleo del sistema operativo, que es la parte del sistema operativo que engloba las funciones más importantes del sistema. La Tabla 3.7 lista las funciones que normalmente se encuentran dentro del núcleo del sistema operativo.



El motivo por el cual se usan los otros modos es claro. Se necesita proteger al sistema operativo y a las tablas clave del sistema, por ejemplo los bloques de control de proceso, de la interferencia con programas de usuario. En modo núcleo, el software tiene control completo del procesador y de sus instrucciones, registros, y memoria. Este nivel de control no es necesario y por seguridad no es recomendable para los programas de usuario.

Aparecen dos cuestiones: ¿cómo conoce el procesador en que modo está ejecutando y cómo este modo puede modificarse? En lo referente la primera cuestión, existe típicamente un bit en la palabra de estado de programa (PSW) que indica el modo de ejecución. Este bit se cambia como respuesta a determinados eventos. Habitualmente, cuando un usuario realiza una llamada a un servicio del sistema operativo o cuando una interrupción dispara la ejecución de una rutina del sistema operativo, este modo de ejecución se cambia a modo núcleo y; tras la finalización del servicio, el modo se fija de nuevo a modo usuario.

**3.11. ¿Cuáles son los pasos que realiza el sistema operativo para la creación de un proceso?**

Una vez que el sistema operativo decide, por cualquier motivo, crear un proceso procederá de la siguiente manera:

1. **Asignar un identificador de proceso único al proceso**. En este instante, se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso.
2. **Reservar espacio para proceso**. Esto incluye todos los elementos de la imagen del proceso. Para ello, el sistema operativo debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario. Estos valores se pueden asignar por defecto basándonos en el tipo de proceso, o pueden fijarse en base a la solicitud de creación del trabajo remitido por el usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el sistema operativo como parte de la solicitud de la creación de proceso. Si existe una parte del espacio direcciones compartido por este nuevo proceso, se fijan los enlaces apropiados. Por último, se debe reservar el espacio para el bloque de control de proceso (BCP).
3. **Inicialización del bloque de control de proceso**. La parte de identificación de proceso del BCP contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre. En la información de estado de proceso del BCP, habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto entrada del programa) y los punteros de pila de sistema (fijados para definir los límites de la pila del proceso). La parte de información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. Por ejemplo, el estado del proceso se puede inicializar normalmente a Listo o Listo/Suspendido. La prioridad se puede fijar, por defecto, a la prioridad más baja, a menos que una solicitud explicita la eleve a una prioridad mayor. Inicialmente, el proceso no debe poseer ningún recurso (dispositivos de E/S, ficheros) a menos que exista una indicación explícita de ello o que haya sido heredados del padre.
4. **Establecer los enlaces apropiados**. Por ejemplo, si el sistema operativo mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o en la cola de Listos/Suspendidos.
5. **Creación o expansión de otras estructuras de datos**. Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o de análisis de rendimiento del sistema.

**3.12. ¿Cuál es la diferencia entre interrupción y *trap*?**

La principal diferencia entre una interrupción y una trampa (trap) radica en su origen y propósito:

1. **Interrupción:** Ocurren debido a eventos externos al proceso en ejecución, como la finalización de una operación de E/S. Las interrupciones pueden provenir de dispositivos de hardware u otros elementos externos al procesador. Cuando se produce una interrupción, el procesador suspende temporalmente la ejecución del proceso actual, guarda su estado y ejecuta un manejador de interrupciones específico para manejar el evento. Después de procesar la interrupción, el control se devuelve al proceso interrumpido o se asigna a otro proceso, dependiendo del diseño del sistema operativo.

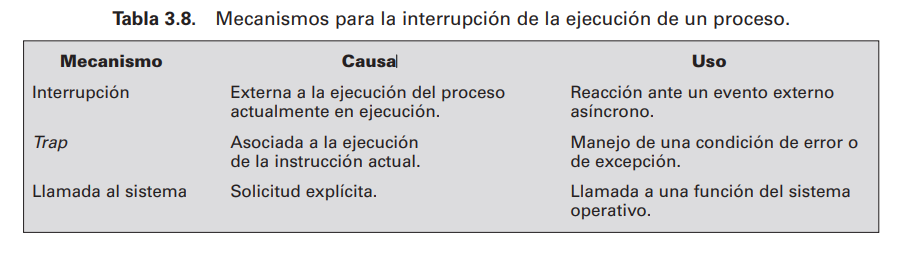
2. **Trampa (Trap):** Se generan debido a condiciones específicas dentro del proceso en ejecución, como errores o excepciones. Por ejemplo, un intento de acceso no permitido a un archivo puede provocar una trampa. Las trampas se utilizan para capturar eventos internos del programa que requieren atención del sistema operativo. Cuando se activa una trampa, el procesador detiene la ejecución normal del programa, guarda su estado y ejecuta un manejador de trampas para manejar la situación. Posteriormente, el control puede volver al proceso original o tomar acciones específicas según el tipo de trampa.

En resumen, las interrupciones son eventos externos que requieren atención del sistema operativo, mientras que las trampas son eventos internos del proceso que necesitan ser manejados por el sistema operativo. Ambos mecanismos permiten que el sistema operativo tome el control del procesador para realizar acciones específicas en respuesta a eventos importantes.

**3.13. Dé tres ejemplos de interrupción.**

Dentro de una **interrupción** ordinaria, el control se transfiere inicialmente al manejador de interrupción, que realiza determinadas tareas internas y que posteriormente salta a una rutina del sistema operativo, encargada de cada uno de los tipos de interrupciones en particular. Algunos ejemplos son:

* **Interrupción de reloj.** El sistema operativo determinar si el proceso en ejecución ha excedido o no la unidad máxima de tiempo de ejecución, denominada **rodaja de tiempo** (*time slice*). Esto es, una rodaja de tiempo es la máxima cantidad de tiempo que un proceso puede ejecutar antes de ser interrumpido. En dicho caso, este proceso se puede pasar al estado de Listo y se puede activar otro proceso.
* **Interrupción de E/S.** El sistema operativo determina qué acción de E/S ha ocurrido. Si la acción de E/S constituye un evento por el cual están esperando uno o más procesos, el sistema operativo mueve todos los procesos correspondientes al estado de Listos (y los procesos en estado Bloqueado/Suspendido al estado Listo/Suspendido). El sistema operativo puede decidir si reanuda la ejecución del proceso actualmente en estado Ejecutando o si lo expulsa para proceder con la ejecución de un proceso Listo de mayor prioridad.
* **Fallo de memoria.** El procesador se encuentra con una referencia a una dirección de memoria virtual, a una palabra que no se encuentra en memoria principal. El sistema operativo debe traer el bloque (página o segmento) que contiene la referencia desde memoria secundaria a memoria principal. Después de que se solicita la operación de E/S para traer el bloque a memoria, el proceso que causó el fallo de memoria se pasa al estado de Bloqueado; el sistema operativo realiza un cambio de proceso y pone a ejecutar a otro proceso. Después de que el bloque de memoria solicitado se haya traído, el proceso pasará al estado Listo.



**3.14. ¿Cuál es la diferencia entre cambio de modo y cambio de proceso?**

**Cambio de modo**. En el Capítulo 1, discutimos la inclusión de una fase de interrupción como parte del ciclo de una instrucción. Recordando esto, en la fase de interrupción, el procesador comprueba que no exista ninguna interrupción pendiente, indicada por la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador pasa a la fase de búsqueda de instrucción, siguiendo con el programa del proceso actual. Si hay una interrupción pendiente, el proceso actúa de la siguiente manera:

1. Coloca el contador de programa en la dirección de comienzo de la rutina del programa manejador de la interrupción.
2. Cambia de modo usuario a modo núcleo de forma que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.

El procesador, acto seguido, pasa a la fase de búsqueda de instrucción y busca la primera instrucción del programa de manejo de interrupción, que dará servicio a la misma. En este punto, habitualmente, el contexto del proceso que se ha interrumpido se salvaguarda en el bloque de control de proceso del programa interrumpido.

Una pregunta que se puede plantear es, ¿qué constituye el contexto que se debe salvaguardar? La respuesta es que se trata de toda la información que se puede ver alterada por la ejecución de la rutina de interrupción y que se necesitará para la continuación del proceso que ha sido interrumpido. De esta forma, se debe guardar la parte del bloque de control del proceso que hace referencia a la información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador, y la información de la pila.

¿Se necesita hacer algo más? Eso depende de qué ocurra luego. El manejador de interrupción es habitualmente un pequeño programa que realiza unas pocas tareas básicas relativas a la interrupción. Por ejemplo, borra el *flag* o indicador que señala la presencia de interrupciones. Puede enviar una confirmación a la entidad que lanzó dicha interrupción, como por ejemplo el módulo de E/S. Y puede realizar algunas tareas internas variadas relativas a los efectos del evento que causó la interrupción. Por ejemplo, si la interrupción se refiere a un evento de E/S, el manejador de interrupción comprobará la existencia o no de una condición de error. Si ha ocurrido un error, el manejador mandará una se- ñal al proceso que solicitó dicha operación de E/S. Si la interrupción proviene del reloj, el manejador la va a pasar el control al activador, el cual decidirá pasar a otro proceso debido a que la rodaja de tiempo asignada a ese proceso ha expirado.

¿Qué pasa con el resto de información del bloque de control de proceso? Si a esta interrupción le sigue un cambio de proceso a otro proceso, se necesita hacer algunas cosas más. Sin embargo, en muchos sistemas operativos, la existencia de una interrupción no implica necesariamente un cambio de proceso. Es posible, por tanto, que después de la ejecución de la rutina de interrupción, la ejecución se reanude con el mismo proceso. En esos casos sólo se necesita salvaguardar la información del estado del procesador cuando se produce la interrupción y restablecerlo cuando se reanude la ejecución del proceso. Habitualmente, las operaciones de salvaguarda y recuperación se realizan por hardware.

**Cambio del estado del proceso**. Está claro, por tanto, que el cambio de modo es un concepto diferente del cambio de proceso8. Un cambio de modo puede ocurrir sin que se cambie el estado del proceso actualmente en estado Ejecutando. En dicho caso, la salvaguarda del estado y su posterior restauración comportan sólo una ligera sobrecarga. Sin embargo, si el proceso actualmente en estado Ejecutando, se va a mover a cualquier otro estado (Listo, Bloqueado, etc.), entonces el sistema operativo debe realizar cambios sustanciales en su entorno. Los pasos que se realizan para un cambio de proceso completo son:

1. Salvar el estado del procesador, incluyendo el contador de programa y otros registros.
2. Actualizar el bloque de control del proceso que está actualmente en el estado Ejecutando. Esto incluye cambiar el estado del proceso a uno de los otros estados (Listo, Bloqueado, Listo/Suspendido, o Saliente). También se tienen que actualizar otros campos importantes, incluyendo la razón por la cual el proceso ha dejado el estado de Ejecutando y demás información de auditoría.
3. Mover el bloque de control de proceso a la cola apropiada (Listo, Bloqueado en el evento *i*, Listo/Suspendido).
4. Selección de un nuevo proceso a ejecutar; esta cuestión se analiza con más detalle en la Parte Cuatro.
5. Actualizar el bloque de control del proceso elegido. Esto incluye pasarlo al estado Ejecutando.
6. Actualizar las estructuras de datos de gestión de memoria. Esto se puede necesitar, dependiendo de cómo se haga la traducción de direcciones; estos aspectos se cubrirán en la Parte Tres.
7. Restaurar el estado del procesador al que tenía en el momento en el que el proceso seleccionado salió del estado Ejecutando por última vez, leyendo los valores anteriores de contador de programa y registros.

Por tanto, el cambio de proceso, que implica un cambio en el estado, requiere un mayor esfuerzo que un cambio de modo.

**4.1. La Tabla 3.5 enumera los elementos típicos que se encuentran en un bloque de control de proceso para un sistema operativo monohilo. De éstos, ¿cuáles deben pertenecer a un bloque de control de hilo y cuáles deben pertenecer a un bloque de control de proceso para un sistema multihilo?**

En un sistema operativo monohilo, un bloque de control de proceso típicamente incluye los siguientes elementos:

**1.** **\*\*Identificador de proceso\*\*:** Un identificador único para el proceso.

**2. \*\*Estado del proceso\*\*:** Indica si el proceso está listo, en ejecución, esperando, etc.

**3. \*\*Contador de programa\*\*:** Guarda la dirección de la próxima instrucción a ejecutar.

**4. \*\*Registros del procesador\*\*:** Contiene los valores de los registros del CPU del proceso.

**5. \*\*Información de programación\*\*:** Prioridad de planificación, tiempo de CPU utilizado, tiempo de espera, etc.

**6. \*\*Información de gestión de memoria\*\*:** Tamaño de espacio de direcciones, lista de páginas asignadas, etc.

**7. \*\*Información de gestión de recursos\*\*:** Lista de archivos abiertos, identificadores de recursos asignados, etc.

**8. \*\*Información de control de acceso\*\*:** Permisos de acceso al proceso y a los recursos.

Para un sistema multihilo, algunos de estos elementos deben pertenecer al bloque de control de proceso, mientras que otros deben estar en el bloque de control de hilo:

Elementos que deben estar en el bloque de control de proceso:

1. Identificador de proceso.

2. Estado del proceso.

3. Información de gestión de memoria.

4. Información de gestión de recursos.

5. Información de control de acceso.

Elementos que deben estar en el bloque de control de hilo:

1. Identificador de hilo: Un identificador único para el hilo.

2. Estado del hilo: Indica si el hilo está listo, en ejecución, esperando, etc.

3. Contador de programa: Guarda la dirección de la próxima instrucción a ejecutar.

4. Registros del procesador: Contiene los valores de los registros del CPU del hilo.

5. Información de programación: Prioridad de planificación, tiempo de CPU utilizado, tiempo de espera, etc.

En un sistema multihilo, cada hilo dentro de un proceso comparte la misma memoria y los mismos recursos, por lo que la información relacionada con la gestión de memoria y los recursos se mantiene en el bloque de control de proceso. Mientras que la información específica del hilo, como el identificador de hilo, estado del hilo y registros del procesador, se mantiene en el bloque de control de hilo.

**4.2. Enumere las razones por las que un cambio de contexto entre hilos puede ser más barato que un cambio de contexto entre procesos.**

Ciertamente, hay varias razones por las cuales un cambio de contexto entre hilos puede ser más barato que un cambio de contexto entre procesos:

1. \*\*Memoria compartida\*\*: Los hilos dentro de un proceso comparten el mismo espacio de memoria, lo que significa que no es necesario copiar datos entre ellos durante un cambio de contexto. En cambio, los procesos tienen su propio espacio de memoria y, por lo tanto, necesitan copiar datos si desean compartir información.

2. \*\*Menor sobrecarga del sistema operativo\*\*: Cambiar entre hilos generalmente implica cambiar de contexto dentro del mismo espacio de direcciones de memoria, lo que puede ser gestionado más eficientemente por el sistema operativo. Por otro lado, cambiar entre procesos implica cambiar entre espacios de direcciones diferentes, lo que generalmente implica una sobrecarga adicional, como el cambio de tablas de páginas y la limpieza de la caché de memoria.

3. \*\*Comunicación más eficiente\*\*: Los hilos dentro de un proceso pueden comunicarse directamente mediante variables compartidas o mecanismos de sincronización de bajo nivel, como semáforos o mutex. En cambio, los procesos deben usar mecanismos de comunicación interprocesos (IPC) más costosos, como sockets, pipes o colas de mensajes.

4. \*\*Mayor velocidad de creación y destrucción\*\*: Crear y destruir hilos suele ser más rápido que crear y destruir procesos. Los hilos comparten recursos con el proceso principal, lo que significa que no se necesita un nuevo espacio de direcciones ni una copia de la imagen del proceso.

5. \*\*Facilidad de coordinación\*\*: Dado que los hilos comparten el mismo espacio de memoria, es más fácil coordinar y sincronizar la ejecución entre ellos. Por el contrario, los procesos deben utilizar mecanismos de sincronización más complejos y costosos, lo que puede afectar negativamente al rendimiento.

En resumen, los cambios de contexto entre hilos dentro de un mismo proceso suelen ser más baratos que los cambios de contexto entre procesos debido a la menor sobrecarga del sistema operativo, la memoria compartida y la comunicación más eficiente entre los hilos.

**4.3. ¿Cuáles son las dos características diferentes y potencialmente independientes en el concepto de proceso?**

Se muestra cómo el concepto de proceso es más complejo y sutil de lo que se ha visto hasta este momento y, de hecho, contiene dos conceptos diferentes y potencialmente independientes: uno relativo a la propiedad de recursos y otro relativo a la ejecución. En muchos sistemas operativos esta distinción ha llevado al desarrollo de estructuras conocidas como hilos (*threads*).

* **Propiedad de recursos.** Un proceso incluye un espacio de direcciones virtuales para el manejo de la imagen del proceso; como ya se explicó en el Capítulo 3 la imagen de un proceso es la colección de programa, datos, pila y atributos definidos en el bloque de control del proceso. De vez en cuando a un proceso se le puede asignar control o propiedad de recursos tales como la memoria principal, canales E/S, dispositivos E/S y archivos. El sistema operativo realiza la función de protección para evitar interferencias no deseadas entre procesos en relación con los recursos.
* **Planificación/ejecución.** La ejecución de un proceso sigue una ruta de ejecución (traza) a través de uno o más programas. Esta ejecución puede estar intercalada con ese u otros procesos. De esta manera, un proceso tiene un estado de ejecución (Ejecutando, Listo, etc.) y una prioridad de activación y ésta es la entidad que se planifica y activa por el sistema operativo.

**4.4. Dé cuatro ejemplos generales del uso de hilos en un sistema multiprocesador monousuario.**

* **Trabajo en primer plano y en segundo plano.** Por ejemplo, en un programa de hoja de cálculo, un hilo podría mostrar menús y leer la entrada de usuario, mientras otro hilo ejecuta los mandatos de usuario y actualiza la hoja de cálculo. Esta forma de trabajo a menudo incrementa la velocidad que se percibe de la aplicación, permitiendo al programa solicitar el siguiente mandato antes de que el mandato anterior esté completado.
* **Procesamiento asíncrono.** Los elementos asíncronos de un programa se pueden implementar como hilos. Por ejemplo, se puede diseñar un procesador de textos con protección contra un fallo de corriente que escriba el *buffer* de su memoria RAM a disco una vez por minuto. Se puede crear un hilo cuyo único trabajo sea crear una copia de seguridad periódicamente y que se planifique directamente a través del sistema operativo; no se necesita código adicional en el programa principal que proporcione control de tiempo o que coordine la entrada/salida.
* **Velocidad de ejecución.** Un proceso multihilo puede computar una serie de datos mientras que lee los siguientes de un dispositivo. En un sistema multiprocesador pueden estar ejecutando simultáneamente múltiples hilos de un mismo proceso. De esta forma, aunque un hilo pueda estar bloqueado por una operación de E/S mientras lee datos, otro hilo puede estar ejecutando.
* **Estructura modular de programas.** Los programas que realizan diversas tareas o que tienen varias fuentes y destinos de entrada y salida, se pueden diseñar e implementar más fácilmente usando hilos.

**4.5. ¿Qué recursos son compartidos normalmente por todos los hilos de un proceso?**

Todos los hilos de un proceso comparten **el mismo espacio de direcciones de la memoria**, todos los hilos se suspenden al mismo tiempo. De forma similar, la finalización de un proceso finaliza todos los hilos de ese proceso.

El "espacio de direcciones" se refiere al conjunto de direcciones de memoria disponibles para un proceso o programa en un sistema computacional. Este espacio de direcciones representa la gama completa de ubicaciones de memoria que el proceso puede acceder durante su ejecución.

El espacio de direcciones se organiza en unidades de tamaño fijo, como bytes o palabras, y puede dividirse en varias secciones, cada una con un propósito específico.

**4.6. Enumere tres ventajas de los ULT sobre los KLT.**

Existen dos amplias categorías de implementación de hilos: hilos de nivel de usuario (*user-level threads*, ULT) e hilos de nivel de núcleo (*kernel-level threads,* KLT)4. Los últimos son también conocidos en la literatura como hilos soportados por el núcleo (*kernel-supported threads*) o procesos ligeros (*lightweight processes*).

El uso de ULT en lugar de KLT, presenta las siguientes ventajas:

1. El cambio de hilo no requiere privilegios de modo núcleo porque todas las estructuras de datos de gestión de hilos están en el espacio de direcciones de usuario de un solo proceso. Por consiguiente, el proceso no cambia a modo núcleo para realizar la gestión de hilos. Esto ahorra la sobrecarga de dos cambios de modo (usuario a núcleo; núcleo a usuario).
2. La planificación puede especificarse por parte de la aplicación. Una aplicación se puede beneficiar de un simple algoritmo de planificación cíclico, mientras que otra se podría beneficiar de un algoritmo de planificación basado en prioridades. El algoritmo de planificación se puede hacer a medida sin tocar el planificador del sistema operativo.
3. Los ULT pueden ejecutar en cualquier sistema operativo. No se necesita ningún cambio en el nuevo núcleo para dar soporte a los ULT. La biblioteca de los hilos es un conjunto de utilidades a nivel de aplicación que comparten todas las aplicaciones.

**1.** **\*\*Hilos de nivel de usuario (ULT)\*\*:**

- Los hilos de nivel de usuario son gestionados completamente por la biblioteca de hilos en el espacio de usuario del programa, sin intervención directa del sistema operativo.

- La creación, planificación y sincronización de los hilos de nivel de usuario se realizan mediante llamadas a funciones proporcionadas por la biblioteca de hilos, que generalmente se ejecutan en el espacio de usuario del programa.

- El sistema operativo no es consciente de la existencia de estos hilos y los trata como procesos normales. El núcleo del sistema operativo asigna recursos solo al proceso en el que se están ejecutando los hilos, y no tiene conocimiento ni control sobre los hilos individuales dentro del proceso.

**2. \*\*Hilos de nivel de núcleo (KLT)\*\*:**

- Los hilos de nivel de núcleo, también conocidos como hilos soportados por el núcleo o procesos ligeros, son gestionados directamente por el núcleo del sistema operativo.

- El sistema operativo asigna recursos como el tiempo de CPU, la memoria y los recursos de E/S a cada hilo de nivel de núcleo, y gestiona su planificación, sincronización y cambio de contexto.

- Los hilos de nivel de núcleo generalmente requieren llamadas al sistema operativo para operaciones como la creación, planificación y sincronización de hilos, lo que puede resultar en un mayor overhead en comparación con los hilos de nivel de usuario.

- El sistema operativo trata cada hilo de nivel de núcleo como una entidad independiente y puede planificar y ejecutar los hilos de manera más flexible y eficiente, aprovechando las características del hardware subyacente.

**4.7. Enumere dos desventajas de los ULT en comparación con los KLT.**

Hay dos desventajas de los ULT en comparación con los KLT:

1. En un sistema operativo típico muchas llamadas al sistema son bloqueantes. Como resultado, cuando un ULT realiza una llamada al sistema, no sólo se bloquea ese hilo, sino que se bloquean todos los hilos del proceso.
2. En una estrategia pura ULT, una aplicación multihilo no puede sacar ventaja del multiproceso. El núcleo asigna el proceso a un solo procesador al mismo tiempo. Por consiguiente, en un determinado momento sólo puede ejecutar un hilo del proceso. En efecto, tenemos multiprogramación a nivel de aplicación con un solo proceso. Aunque esta multiprogramación puede dar lugar a una mejora significativa de la velocidad de la aplicación, hay aplicaciones que se podrían beneficiar de la habilidad de ejecutar porciones de código de forma concurrente.

**4.8. Defina *jacketing (revestimiento)*.**

Otra forma de solucionar el problema de hilos que se bloquean es una técnica denominada *jacketing* (revestimiento). El objetivo de esta técnica es convertir una llamada al sistema bloqueante en una llamada al sistema no bloqueante. Por ejemplo, en lugar de llamar directamente a una rutina del sistema de E/S, un hilo puede llamar a una rutina *jacket* de E/S a nivel de aplicación. Con esta rutina *jacket*, el código verifica si el dispositivo de E/S está ocupado. Si lo está, el hilo entra en estado Bloqueado y pasa el control (a través de la biblioteca de hilos) a otro hilo. Cuando este hilo recupera de nuevo el control, chequea de nuevo el dispositivo de E/S.

**4.9. Defina brevemente las siguientes arquitecturas:**

Procesadores paralelos:

- SIMD (única instrucción múltiples flujos de datos)

- MIMD (múltiples instrucciones múltiples flujos de datos):  
 - Memoria compartida (fuertemente acoplados):

- Maestro/esclavo

- Multiprocesadores simétricos (SMP)

- Memoria distribuida (débilmente acoplados:

- *Clusters*

Claro, aquí tienes definiciones breves de cada arquitectura:

**1. \*\*SIMD (Single Instruction, Multiple Data)\*\*:**

- En SIMD, una sola instrucción es ejecutada simultáneamente en múltiples elementos de datos.

- Los procesadores SIMD son eficientes para aplicaciones que realizan operaciones idénticas en grandes conjuntos de datos, como procesamiento de imágenes, gráficos por computadora y procesamiento de señales.

**2. \*\*MIMD (Multiple Instruction, Multiple Data)\*\*:**

- En MIMD, múltiples unidades de procesamiento ejecutan instrucciones independientes en conjuntos de datos independientes.

**a. \*\*Memoria compartida (fuertemente acoplados)\*\*:**

- En sistemas con memoria compartida, varios procesadores comparten el mismo espacio de memoria física y pueden acceder a los mismos datos directamente.

- Ejemplos incluyen el modelo **maestro/esclavo**, donde un procesador coordina y distribuye tareas a los demás, y los sistemas **SMP**, donde todos los procesadores tienen acceso igualitario a la memoria.

**b. \*\*Memoria distribuida (débilmente acoplados)\*\*:**

- En sistemas con memoria distribuida, cada procesador tiene su propio espacio de memoria local y se comunica con otros procesadores a través de mensajes.

- Estos sistemas pueden formar clústeres de computadoras interconectadas, como en el caso de los **clusters**, donde cada nodo tiene su propio procesador y memoria.

En resumen, SIMD y MIMD son dos categorías de procesadores paralelos, y dentro de MIMD, se pueden distinguir entre sistemas con memoria compartida (como maestro/esclavo y SMP) y sistemas con memoria distribuida (como los clusters). Cada uno de estos tipos de arquitectura tiene aplicaciones y características específicas.

**4.10. Enumere los aspectos principales de diseño de un sistema operativo SMP.**

Las principales claves de diseño incluyen las siguientes características:

* **Procesos o hilos simultáneos concurrentes.** Las rutinas del núcleo necesitan ser reentrantes para permitir que varios procesadores ejecuten el mismo código del núcleo simultáneamente. Debido a que múltiples procesadores pueden ejecutar la misma o diferentes partes del código del núcleo, las tablas y la gestión de las estructuras del núcleo deben ser gestionas apropiadamente para impedir interbloqueos u operaciones inválidas.
* **Planificación.** La planificación se puede realizar por cualquier procesador, por lo que se deben evitar los conflictos. Si se utiliza multihilo a nivel de núcleo, existe la posibilidad de planificar múltiples hilos del mismo proceso simultáneamente en múltiples procesadores. En el Capítulo 10 se examina la planificación multiprocesador.

**4.11. Dé ejemplos de servicios y funciones que se encuentran en un sistema operativo monolítico típico que podrían ser subsistemas externos en un sistema operativo micronúcleo.**

Comencemos con las definiciones:

- **\*\*Sistema Operativo Monolítico\*\*:** En un sistema operativo monolítico, todas las funciones del sistema operativo se ejecutan en modo kernel en el mismo espacio de direcciones. Todas las funcionalidades, como la gestión de procesos, la gestión de memoria, el sistema de archivos y los controladores de dispositivos, forman parte de un solo y grande núcleo. Ejemplos de sistemas operativos monolíticos incluyen versiones antiguas de Linux y Windows (antes de Windows NT).

- **\*\*Sistema Operativo Micronúcleo\*\*:** En un sistema operativo micronúcleo, el núcleo es pequeño y solo proporciona servicios básicos como la comunicación entre procesos y la gestión de memoria. Funcionalidades como la gestión de procesos, el sistema de archivos y los controladores de dispositivos se implementan como subsistemas externos que se ejecutan en modo de usuario, fuera del núcleo. Ejemplos de sistemas operativos micronúcleo incluyen MINIX y QNX.

Ahora, respecto a los servicios y funciones que podrían ser subsistemas externos en un sistema operativo micronúcleo:

**1. \*\*Gestión de Procesos\*\*:**

- En un sistema operativo monolítico, la gestión de procesos se realiza dentro del núcleo.

- En un sistema operativo micronúcleo, la gestión de procesos podría ser un subsistema externo que se ejecuta en modo de usuario. Este subsistema sería responsable de crear, destruir y gestionar procesos, así como de la planificación de la CPU.

**2. \*\*Sistema de Archivos\*\*:**

- En un sistema operativo monolítico, el sistema de archivos está integrado en el núcleo del sistema operativo.

- En un sistema operativo micronúcleo, el sistema de archivos podría ser un subsistema externo que se ejecuta en modo de usuario. Este subsistema sería responsable de proporcionar acceso a los archivos y directorios almacenados en los dispositivos de almacenamiento.

**3. \*\*Controladores de Dispositivos\*\*:**

- En un sistema operativo monolítico, los controladores de dispositivos están integrados en el núcleo del sistema operativo.

- En un sistema operativo micronúcleo, los controladores de dispositivos podrían ser subsistemas externos que se ejecutan en modo de usuario. Cada controlador de dispositivo sería responsable de interactuar con un dispositivo de hardware específico y proporcionar una interfaz para que otros componentes del sistema operativo accedan al dispositivo.

**4. \*\*Redes y Comunicaciones\*\*:**

- En un sistema operativo monolítico, las funciones de red y comunicaciones están integradas en el núcleo del sistema operativo.

- En un sistema operativo micronúcleo, las funciones de red y comunicaciones podrían ser subsistemas externos que se ejecutan en modo de usuario. Estos subsistemas serían responsables de proporcionar conectividad de red, protocolos de comunicación y servicios de red, como TCP/IP.

En resumen, un sistema operativo micronúcleo separa las funciones del sistema operativo en subsistemas externos que se ejecutan en modo de usuario, mientras que un sistema operativo monolítico integra todas las funciones en un solo núcleo que se ejecuta en modo kernel.

**4.12. Enumere y explique brevemente siete ventajas potenciales de un diseño micronúcleo en comparación con un diseño monolítico.**

Aquí tienes una lista de las ventajas potenciales de un diseño de micronúcleo en comparación con un diseño monolítico, junto con una breve explicación de cada una:

**1. \*\*Interfaces uniformes\*\*:**

- El micronúcleo impone una interfaz uniforme en las peticiones realizadas por un proceso, ya que todos los servicios se proporcionan a través de paso de mensajes. Esto simplifica la interacción de los procesos con el sistema operativo.

**2. \*\*Extensibilidad\*\*:**

- La arquitectura de micronúcleo facilita la extensibilidad del sistema operativo, permitiendo agregar nuevos servicios o realizar múltiples servicios en la misma área funcional. Esto se logra mediante la adición o modificación de servidores relacionados sin necesidad de reconstruir todo el núcleo.

**3. \*\*Flexibilidad\*\*:**

- Las características existentes en un sistema de micronúcleo pueden ser eliminadas o modificadas para realizar una implementación más pequeña y eficiente. Esto permite que el sistema operativo sea flexible y se adapte a diferentes requisitos y entornos de uso.

**4. \*\*Portabilidad\*\*:**

- Un sistema operativo basado en un micronúcleo puede ser más portable, ya que gran parte del código específico del procesador está en el micronúcleo. Los cambios necesarios para migrar el sistema a un nuevo procesador son menores y tienden a estar unidos en grupos lógicos.

**5. \*\*Fiabilidad\*\*:**

- La estructura modular de un micronúcleo permite una mayor verificación y prueba rigurosa del código. Un micronúcleo pequeño puede ser verificado de manera más exhaustiva, lo que contribuye a mejorar la fiabilidad del sistema operativo.

**6. \*\*Soporte de sistemas distribuidos\*\*:**

- Un micronúcleo facilita el soporte de sistemas distribuidos, incluyendo clusters controlados por sistemas operativos distribuidos. La arquitectura de micronúcleo permite la comunicación entre procesos y servicios a través de paso de mensajes, lo que facilita la creación de sistemas distribuidos.

**7. \*\*Soporte de sistemas operativos orientados a objetos (OOOS)\*\*:**

- Un micronúcleo es compatible con enfoques orientados a objetos en el diseño y desarrollo de extensiones modulares para el sistema operativo. Esto se logra mediante el uso de componentes con interfaces claramente definidas, que pueden ser interconectadas para construir el software a través de bloques de construcción.

En resumen, un diseño de micronúcleo ofrece varias ventajas, como interfaces uniformes, extensibilidad, flexibilidad, portabilidad, fiabilidad, soporte de sistemas distribuidos y soporte de sistemas operativos orientados a objetos, lo que lo convierte en una opción atractiva para el desarrollo de sistemas operativos.

**4.13. Explique la desventaja potencial de rendimiento de un sistema operativo micronúcleo.**

Una potencial desventaja que se cita a menudo de los micronúcleos es la del rendimiento. Lleva más tiempo construir y enviar un mensaje a través del micronúcleo, y aceptar y decodificar la respuesta, que hacer una simple llamada a un servicio.

Esta pérdida de rendimiento es más notable en los micronúcleos de primera generación, donde se observa una significativa reducción del rendimiento, incluso después de optimizar el código del micronúcleo. Para abordar este problema, se han explorado diferentes enfoques:

**1. \*\*Aumento del tamaño del micronúcleo\*\*:** Al reintroducir servicios críticos y manejadores en el micronúcleo, se reducen el número de cambios de modo usuario-núcleo y de espacio de direcciones de proceso. Sin embargo, esto puede sacrificar la fortaleza del diseño del micronúcleo, como sus mínimas interfaces y su flexibilidad.

**2. \*\*Reducción del tamaño del micronúcleo\*\*:** Diseñando un micronúcleo más pequeño, se eliminan las pérdidas de rendimiento y se mejora la flexibilidad y la fiabilidad. Ejemplos de micronúcleos pequeños de segunda generación incluyen L4, que consta de 12 Kbytes de código y 7 llamadas al sistema. Experimentaciones han demostrado que estos sistemas pueden funcionar tan bien o mejor que sistemas operativos monolíticos como UNIX.

En resumen, aunque los sistemas operativos micronúcleo pueden enfrentar desafíos de rendimiento, las estrategias para optimizar el diseño del micronúcleo, ya sea aumentando o reduciendo su tamaño, pueden mitigar estas desventajas y llevar a sistemas operativos más eficientes y flexibles.

**4.14. Enumere cuatro funciones que le gustaría encontrar incluso en un sistema operativo micronúcleo mínimo.**

El micronúcleo debe incluir aquellas funciones que dependen directamente del hardware y aquellas funciones necesarias para mantener a los servidores y aplicaciones operando en modo usuario.

Estas funciones entran dentro de las categorías generales de gestión de memoria a bajo nivel, **intercomunicación de procesos** (IPC), y E/S y manejo de interrupciones.

**4.15. ¿Cuál es la forma básica de comunicación entre procesos o hilos en un sistema operativo micronúcleo?**

La forma básica de comunicación entre procesos o hilos en un sistema operativo micronúcleo es a través de paso de mensajes (message passing). En este modelo de comunicación, los procesos o hilos intercambian datos y señales enviando y recibiendo mensajes entre sí. Esto se realiza típicamente a través de las siguientes operaciones básicas:

**1. \*\*Enviar mensaje\*\*:** Un proceso o hilo envía un mensaje a otro proceso o hilo especificando el destino y los datos que se desean transmitir. Esto puede implicar colocar los datos en un área de memoria compartida o estructuras de datos dedicadas para la comunicación.

**2. \*\*Recibir mensaje\*\*:** Un proceso o hilo espera y recibe mensajes que le son enviados por otros procesos o hilos. Para recibir un mensaje, el proceso o hilo especifica el remitente del mensaje y, en algunos casos, el tipo de mensaje que espera recibir.

**3. \*\*Responder a mensajes\*\*:** Después de recibir un mensaje, un proceso o hilo puede optar por responder al remitente enviando un mensaje de vuelta. Esto completa el ciclo de comunicación.

El paso de mensajes proporciona una forma flexible y segura de comunicación entre procesos o hilos en un sistema operativo micronúcleo. Permite la comunicación entre procesos o hilos que se ejecutan en diferentes espacios de direcciones y puede ser implementada de manera eficiente incluso en sistemas distribuidos. Además, el paso de mensajes puede ofrecer mecanismos de sincronización para garantizar la coordinación adecuada entre los procesos o hilos que participan en la comunicación.

**5.1. Enumere cuatro aspectos de diseño para los cuales el concepto de concurrencia es relevante.**

**Concurrencia:** los procesos se entrelazan en el tiempo para ofrecer la apariencia de ejecución simultánea. Aunque no se consigue procesamiento paralelo real, e ir cambiando de un proceso a otro supone cierta sobrecarga, la ejecución entrelazada proporciona importantes beneficios en la eficiencia del procesamiento y en la estructuración de los programas. En un sistema de múltiples procesadores no sólo es posible entrelazar la ejecución de múltiples procesos sino también solaparlas.

**¿Qué aspectos de diseño y gestión surgen por la existencia de la concurrencia?**

Pueden enumerarse las siguientes necesidades:

1. El sistema operativo debe ser capaz de seguir la pista de varios procesos. Esto se consigue con el uso de bloques de control de proceso y fue descrito en el Capítulo 4.
2. El sistema operativo debe ubicar y desubicar varios recursos para cada proceso activo. Estos recursos incluyen:
   * **Tiempo de procesador.** Esta es la misión de la planificación, tratada en la Parte Cuatro.
   * **Memoria.** La mayoría de los sistemas operativos usan un esquema de memoria virtual. El tema es abordado en la Parte Tres.
   * **Ficheros.** Tratados en el Capítulo 12.
   * **Dispositivos de E/S.** Tratados en el Capítulo 11.
3. El sistema operativo debe proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos. Esto involucra técnicas que relacionan memoria, ficheros y dispositivos de E/S. En el Capítulo 15 se encuentra tratado en general el tema de la protección.
4. El funcionamiento de un proceso y el resultado que produzca, debe ser independiente de la velocidad a la que suceda su ejecución en relación con la velocidad de otros procesos concurrentes. Este es el tema de este capítulo.

**5.2. ¿En qué tres contextos aparece la concurrencia?**

La concurrencia aparece en tres contextos diferentes:

**• Múltiples aplicaciones.** La multiprogramación fue ideada para permitir compartir dinámicamente el tiempo de procesamiento entre varias aplicaciones activas.

**• Aplicaciones estructuradas.** Como extensión de los principios del diseño modular y de la programación estructurada, algunas aplicaciones pueden ser programadas eficazmente como un conjunto de procesos concurrentes.

**• Estructura del sistema operativo.** Las mismas ventajas constructivas son aplicables a la programación de sistemas y, de hecho, los sistemas operativos son a menudo implementados en sí mismos como un conjunto de procesos o hilos.

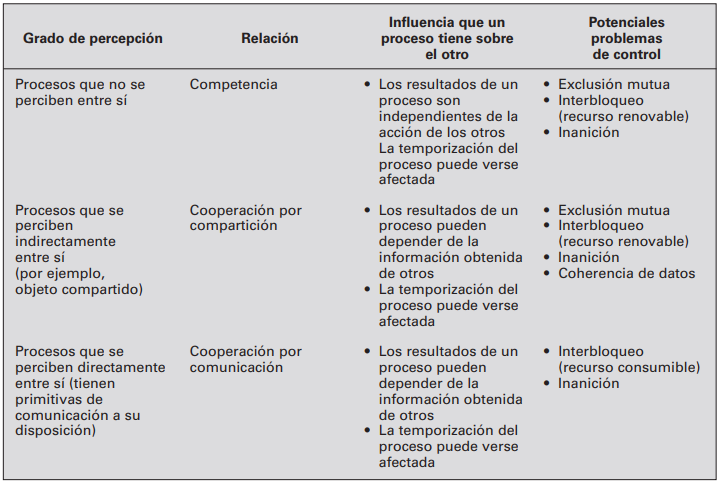
**5.3. ¿Cuál es el requisito básico para la ejecución de procesos concurrentes?**

Se descubre que el requisito básico para conseguir ofrecer procesos concurrentes es la capacidad de hacer imperar la exclusión mutua; esto es, la capacidad de impedir a cualquier proceso realizar una acción mientras se le haya permitido a otro.

5.4. Enumere tres grados de percepción entre procesos y defina brevemente cada uno.

Podemos clasificar las formas en que los procesos interaccionan en base al grado en que perciben la existencia de cada uno de los otros. Tres posibles grados de percepción más las consecuencias de cada uno:

* **Procesos que no se perciben entre sí.** Son procesos independientes que no se pretende que trabajen juntos. El mejor ejemplo de esta situación es la multiprogramación de múltiples procesos independientes. Estos bien pueden ser trabajos por lotes o bien sesiones interactivas o una mezcla. Aunque los procesos no estén trabajando juntos, el sistema operativo necesita preocuparse de la **competencia** por recursos. Por ejemplo, dos aplicaciones independientes pueden querer ambas acceder al mismo disco, fichero o impresora. El sistema operativo debe regular estos accesos.
* **Procesos que se perciben indirectamente entre sí.** Son procesos que no están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero que comparten accesos a algún objeto, como un *buffer* de E/S. Tales procesos exhiben **cooperación** en la compartición del objeto común.
* **Procesos que se perciben directamente entre sí.** Son procesos capaces de comunicarse entre sí vía el ID del proceso y que son diseñados para trabajar conjuntamente en cierta actividad. De nuevo, tales procesos exhiben **cooperación**.



**5.5 ¿Cuál es la diferencia entre procesos en competencia y procesos cooperantes?**

La diferencia principal entre procesos en competencia y procesos cooperantes radica en la naturaleza de su interacción y la forma en que comparten recursos:

**1. \*\*Competencia entre procesos por recursos\*\*:**

- En este caso, los procesos concurrentes compiten por el uso del mismo recurso.

- Cada proceso opera de manera independiente y no tiene conocimiento explícito de la existencia de otros procesos.

- Los procesos compiten por recursos como dispositivos de E/S, memoria, tiempo de procesador, etc.

- Cada proceso debe dejar inalterado el estado de los recursos que utiliza, y no debe verse afectado por la ejecución de otros procesos.

**2. \*\*Cooperación entre procesos vía compartición\*\*:**

- En este caso, los procesos interactúan entre sí al compartir recursos como variables, ficheros o bases de datos compartidas.

- Aunque los procesos pueden tener acceso a los mismos datos compartidos, pueden operar de manera independiente sin tener conocimiento explícito de los otros procesos.

- Los procesos deben cooperar para garantizar la integridad de los datos compartidos.

- Los mecanismos de control se utilizan para asegurar que los datos compartidos se manipulen adecuadamente y se mantenga su integridad.

**3. \*\*Cooperación entre procesos vía comunicación\*\*:**

- En este caso, los procesos participan en un esfuerzo común que los vincula a todos ellos a través de la comunicación directa.

- La comunicación proporciona una manera de sincronizar o coordinar actividades entre los procesos.

- A diferencia de la cooperación vía compartición, donde los procesos no tienen conocimiento explícito de los demás procesos pero son conscientes de la necesidad de mantener la integridad de los datos, en la cooperación vía comunicación, los procesos están directamente involucrados en la coordinación de actividades.

En resumen, mientras que los procesos en competencia compiten por recursos sin tener conocimiento de los demás procesos, los procesos cooperantes pueden compartir recursos de manera compartida o a través de la comunicación directa para coordinar actividades.

**5.6. Enumere los tres problemas de control asociados con los procesos en competencia y defina brevemente cada uno.**

Los tres problemas de control asociados con los procesos en competencia son:

**1. \*\*Exclusión Mutua\*\*:** Este problema se refiere a la necesidad de garantizar que solo un proceso tenga acceso a un recurso crítico en un momento dado. Es esencial que, durante la ejecución de una sección crítica de un proceso, ningún otro proceso pueda acceder al mismo recurso para evitar condiciones de carrera y garantizar la consistencia de los datos. La exclusión mutua asegura que los procesos no interfieran entre sí mientras acceden al recurso compartido.

**2. \*\*Interbloqueo (Deadlock)\*\*:** El interbloqueo ocurre cuando dos o más procesos quedan atrapados en un estado de espera perpetua porque cada uno de ellos está esperando que el otro libere un recurso que necesita para avanzar. Por ejemplo, si dos procesos necesitan acceder a dos recursos en el orden opuesto, pueden bloquearse mutuamente mientras esperan que el otro libere el recurso que necesitan. El interbloqueo impide el progreso de los procesos y puede causar una parálisis en el sistema.

**3. \*\*Inanición (Starvation)\*\*:** La inanición ocurre cuando un proceso queda permanentemente excluido de acceder a un recurso compartido, a pesar de que intenta acceder a él repetidamente. Esto puede suceder si otros procesos tienen prioridad sobre el proceso inaniciado y continúan obteniendo acceso al recurso antes de que el proceso inaniciado pueda hacerlo. La inanición puede llevar a una degradación del rendimiento del sistema y a la injusticia en la asignación de recursos.

Estos problemas de control son críticos en la gestión de procesos concurrentes y requieren estrategias y mecanismos adecuados para abordarlos y evitar que ocurran en el sistema. La exclusión mutua es fundamental para garantizar la coherencia de los datos, mientras que el interbloqueo y la inanición deben evitarse para garantizar un funcionamiento adecuado y equitativo del sistema.

**5.7. Enumere las condiciones necesarias para la exclusión mutua.**

Cualquier mecanismo o técnica que vaya a proporcionar exclusión mutua debería cumplimentar los siguientes requisitos:

1. La exclusión mutua debe hacerse cumplir: sólo se permite un proceso al tiempo dentro de su sección crítica, de entre todos los procesos que tienen secciones críticas para el mismo recurso u objeto compartido.
2. Un proceso que se pare en su sección no crítica debe hacerlo sin interferir con otros procesos.
3. No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo ni inanición.
4. Cuando ningún proceso esté en una sección crítica, a cualquier proceso que solicite entrar en su sección crítica debe permitírsele entrar sin demora.
5. No se hacen suposiciones sobre las velocidades relativas de los procesos ni sobre el número de procesadores.
6. Un proceso permanece dentro de su sección crítica sólo por un tiempo finito.

**5.8. ¿Qué operaciones pueden ser realizadas sobre un semáforo?**

**Definición:** Los semáforos son una herramienta de programación utilizada en sistemas operativos y lenguajes de programación para proporcionar concurrencia y sincronización entre procesos o hilos. Fueron introducidos por Edsger Dijkstra en 1965 como una solución para problemas de exclusión mutua y sincronización en sistemas concurrentes.

El concepto fundamental detrás de los semáforos es permitir que los procesos o hilos cooperen mediante señales simples. Un proceso puede ser detenido en un punto específico hasta que reciba una señal específica. Los semáforos son implementados como variables especiales que pueden tener un valor entero no negativo.

Para conseguir el efecto deseado, el semáforo puede ser visto como una variable que contiene un valor entero sobre el cual sólo están definidas tres operaciones:

1. Un semáforo puede ser inicializado a un valor no negativo.
2. La operación semWait decrementa el valor del semáforo. Si el valor pasa a ser negativo, entonces el proceso que está ejecutando semWait se bloquea. En otro caso, el proceso continúa su ejecución.
3. La operación semSignal incrementa el valor del semáforo. Si el valor es menor o igual que cero, entonces se desbloquea uno de los procesos bloqueados en la operación semWait. Aparte de estas tres operaciones no hay manera de inspeccionar o manipular un semáforo.

**5.9. ¿Cuál es la diferencia entre semáforos binarios y semáforos generales?**

La diferencia principal entre semáforos binarios y semáforos generales radica en su comportamiento y en la cantidad de valores que pueden tomar.

**1. \*\*Semáforos No-binarios: semáforo con contador o** **semáforo general** **\*\*:**

- Pueden tomar valores enteros no negativos.

- La operación `semWait` decrementa el valor del semáforo. Si el valor resultante es negativo, el proceso que ejecuta `semWait` se bloquea.

- La operación `semSignal` incrementa el valor del semáforo. Si el valor es menor o igual a cero, se desbloquea uno de los procesos bloqueados en `semWait`.

**2. \*\*Semáforos Binarios o Mutex\*\*:**

- Solo pueden tomar los valores 0 y 1.

- La operación `semWaitB` (o `semWait` en algunos contextos) comprueba el valor del semáforo. Si el valor es cero, el proceso que ejecuta `semWaitB` se bloquea. Si el valor es uno, se cambia el valor a cero y el proceso continúa su ejecución.

- La operación `semSignalB` (o `semSignal` en algunos contextos) comprueba si hay algún proceso bloqueado en el semáforo. Si hay procesos bloqueados, se desbloquea uno de ellos. Si no hay procesos bloqueados, el valor del semáforo se establece en uno.

En resumen, los semáforos binarios solo pueden tener dos estados (0 o 1) y son útiles para sincronizar el acceso a recursos compartidos cuando solo se necesita un control binario, como la exclusión mutua. Por otro lado, los semáforos generales pueden tener valores enteros no negativos y son más versátiles para controlar la concurrencia en situaciones donde se necesitan múltiples permisos o recursos.

**5.10. ¿Cuál es la diferencia entre semáforos fuertes y semáforos débiles?**

La diferencia entre semáforos fuertes y semáforos débiles radica en la política de ordenación de los procesos que están bloqueados y esperan acceder al recurso controlado por el semáforo.

**1. \*\*Semáforos Fuertes\*\*:**

- Utilizan una política FIFO (primero en entrar, primero en salir) para la ordenación de los procesos en la cola de espera.

- En un semáforo fuerte, el proceso que ha estado esperando más tiempo es el primero en ser desbloqueado cuando se libera el recurso.

**2. \*\*Semáforos Débiles\*\*:**

- No especifican un orden específico para la extracción de procesos de la cola de espera.

- En un semáforo débil, el sistema operativo o el planificador de procesos puede decidir el orden de desbloqueo de los procesos basándose en alguna política de planificación predeterminada.

En resumen, en los semáforos fuertes, se garantiza que los procesos se desbloquearán en el orden en que se bloquearon, lo que puede ser útil en situaciones donde se necesita un comportamiento predecible o justo. Por otro lado, los semáforos débiles no garantizan un orden específico de desbloqueo y dependen del planificador del sistema operativo para determinar qué proceso se desbloquea en qué momento.

**5.11. ¿Qué es un monitor?**

Un monitor es un módulo software consistente en uno o más procedimientos, una secuencia de inicialización y datos locales. Las principales características de un monitor son las siguientes:

1. Las variables locales de datos son sólo accesibles por los procedimientos del monitor y no por ningún procedimiento externo.
2. Un proceso entra en el monitor invocando uno de sus procedimientos.
3. Sólo un proceso puede estar ejecutando dentro del monitor al tiempo; cualquier otro proceso que haya invocado al monitor se bloquea, en espera de que el monitor quede disponible.

**5.12. ¿Cuál es la diferencia entre bloqueante y no bloqueante con respecto a los mensajes?**

La diferencia entre bloqueante y no bloqueante con respecto a los mensajes radica en cómo se comportan los procesos que envían y reciben mensajes:

**- \*\*Envío bloqueante, recepción bloqueante\*\*:** Tanto el proceso emisor como el receptor se bloquean hasta que el mensaje se entrega. Esta combinación, conocida como rendezvous, asegura que tanto el proceso emisor como el receptor estén sincronizados y esperen activamente hasta que la comunicación se complete.

**- \*\*Envío no bloqueante, recepción bloqueante\*\*:** El proceso emisor puede continuar su ejecución después de enviar el mensaje, pero el receptor se bloquea hasta que recibe el mensaje solicitado. Esto permite que el proceso emisor avance sin esperar la confirmación de que el mensaje ha sido recibido, mientras que el receptor se bloquea hasta que la comunicación se complete.

**- \*\*Envío no bloqueante, recepción no bloqueante\*\*:** En esta combinación, ni el proceso emisor ni el receptor se bloquean. Ambos pueden continuar ejecutándose sin esperar la confirmación de la entrega o recepción del mensaje. Esto proporciona la máxima flexibilidad, pero también requiere que los procesos implementen mecanismos para manejar la confirmación de la recepción del mensaje.

Cada combinación tiene sus propias ventajas y desventajas. Por ejemplo, el envío no bloqueante y la recepción bloqueante pueden ser útiles para evitar bloqueos completos de procesos mientras se asegura que la comunicación se complete de manera sincronizada. Sin embargo, el envío no bloqueante puede llevar a la generación excesiva de mensajes si no se maneja adecuadamente, lo que puede consumir recursos del sistema.

**5.13. ¿Qué condiciones están asociadas generalmente con el problema lectores/escritores?**

El problema lectores/escritores se define como sigue: Hay un área de datos compartida entre un número de procesos. El área de datos puede ser un fichero, un bloque de memoria principal o incluso un banco de registros del procesador. Hay un número de procesos que sólo leen del área de datos (lectores) y otro número que sólo escriben en el área de datos (escritores). Las siguientes condiciones deben satisfacerse.

1. Cualquier número de lectores pueden leer del fichero simultáneamente.

2. Sólo un escritor al tiempo puede escribir en el fichero.

3. Si un escritor está escribiendo en el fichero ningún lector puede leerlo.

**6.1. Cite ejemplos de recursos reutilizables y consumibles.**

Primero que nada se puede definir el **interbloqueo** como el bloqueo *permanente* de un conjunto de procesos que o bien compiten por recursos del sistema o se comunican entre sí. Un conjunto de procesos está interbloqueado cuando cada proceso del conjunto está bloqueado esperando un evento (normalmente la liberación de algún recurso requerido) que sólo puede generar otro proceso bloqueado del conjunto. El interbloqueo es permanente porque no puede producirse ninguno de los eventos. A diferencia de otros problemas que aparecen en la gestión de procesos concurrentes, no hay una solución eficiente para el caso general.

Un **recurso reutilizable** es aquél que sólo lo puede utilizar de forma segura un proceso en cada momento y que no se destruye después de su uso. Los procesos obtienen unidades del recurso que más tarde liberarán para que puedan volver a usarlas otros procesos. Algunos ejemplos de recursos reutilizables incluyen procesadores, canales de E/S, memoria principal y secundaria, dispositivos, y estructuras de datos como ficheros, bases de datos y semáforos.

Un **recurso consumible** es aquél que puede crearse (producirse) y destruirse (consumirse). Normalmente, no hay límite en el número de recursos consumibles de un determinado tipo. Un proceso productor desbloqueado puede crear un número ilimitado de estos recursos. Cuando un proceso consumidor adquiere un recurso, el recurso deja de existir. Algunos ejemplos de recursos consumibles son las interrupciones, las señales, los mensajes y la información en *buffers* de E/S.

**6.2. ¿Cuáles son las tres condiciones que deben cumplirse para que sea posible un interbloqueo?**

Deben presentarse tres condiciones de gestión para que sea posible un interbloqueo:

1. **Exclusión mutua.** Sólo un proceso puede utilizar un recurso en cada momento. Ningún proceso puede acceder a una unidad de un recurso que se ha asignado a otro proceso.
2. **Retención y espera.** Un proceso puede mantener los recursos asignados mientras espera la asignación de otros recursos.
3. **Sin expropiación.** No se puede forzar la expropiación de un recurso a un proceso que lo posee.

**6.3. ¿Cuáles son las cuatro condiciones que producen un interbloqueo?**

Si se cumplen estas tres condiciones se puede producir un interbloqueo, pero aunque se cumplan puede que no lo haya. Para que realmente se produzca el interbloqueo, se requiere una cuarta condición:

1. **Espera circular.** Existe una lista cerrada de procesos, de tal manera que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.

**6.4. ¿Cómo se puede prever la condición de retención y espera?**

La condición de retención y espera puede eliminarse estableciendo que un proceso debe solicitar al mismo tiempo todos sus recursos requeridos, bloqueándolo hasta que se le puedan conceder simultáneamente todas las peticiones. Esta estrategia es insuficiente en dos maneras. En primer lugar, un proceso puede quedarse esperando mucho tiempo hasta que todas sus solicitudes de recursos puedan satisfacerse, cuando, de hecho, podría haber continuado con solamente algunos de los recursos. En segundo lugar, los recursos asignados a un proceso pueden permanecer inutilizados durante un periodo de tiempo considerable, durante el cual se impide su uso a otros procesos. Otro problema es que un proceso puede no conocer por anticipado todos los recursos que requerirá.

Hay también un problema práctico creado por el uso de una programación modular o una estructura multihilo en una aplicación. La aplicación necesitaría ser consciente de todos los recursos que se solicitarán en todos los niveles o en todos los módulos para hacer una solicitud simultánea.

**6.5. Enumere dos maneras cómo se puede prever la condición de sin expropiación.**

Esta condición se puede impedir de varias maneras. En primer lugar, si a un proceso que mantiene varios recursos se le deniega una petición posterior, ese proceso deberá liberar sus recursos originales y, si es necesario, los solicitará de nuevo junto con el recurso adicional. Alternativamente, si un proceso solicita un recurso que otro proceso mantiene actualmente, el sistema operativo puede expropiar al segundo proceso y obligarle a liberar sus recursos. Este último esquema impediría el interbloqueo sólo si no hay dos procesos que posean la misma prioridad.

Esta estrategia es sólo práctica cuando se aplica a recursos cuyo estado se puede salvar y restaurar más tarde, como es el caso de un procesador

**6.6. ¿Cómo se puede prever la condición de espera circular?**

La condición de espera circular se puede impedir definiendo un orden lineal entre los distintos tipos de recursos. Si a un proceso le han asignado recursos de tipo *R*, posteriormente puede pedir sólo aquellos recursos cuyo tipo tenga un orden posterior al de *R*.

**6.7. ¿Cuál es la diferencia entre predicción, detección y prevención del interbloqueo?**

Existen tres estrategias para el tratamiento del interbloqueo. En primer lugar, se puede prevenir el interbloqueo adoptando una política que elimine una de las condiciones (las 4 condiciones enumeradas previamente). En segundo lugar, se puede predecir el interbloqueo tomando las apropiadas decisiones dinámicas basadas en el estado actual de asignación de recursos. En tercer lugar, se puede intentar **detectar** la presencia del interbloqueo (se cumplen las 4 condiciones) y realizar las acciones pertinentes para recuperarse del mismo.

**Virtualización 🡪 Proceso de crear entornos virtuales que simulan recursos de hardware.**

Se refiere a la abstracción de los recursos de una computadora, llamada Hypervisor o VMM (Virtual Machine Monitor) que crea una capa de abstracción entre el hardware de la máquina física (host) y el sistema operativo de la máquina virtual (virtual machine, guest), dividiéndose el recurso en uno o más entornos de ejecución, es decir es la ejecución múltiples sistemas operativo lógicos-virtuales (guests) por encima de un sistema operativo “real” ejecutándose sobre la máquina física (el hardware de base).

**TIPOS DE VIRTUALIZACIÓN**

• Virtualización asistida por software (hosted): es necesario un sistema operativo para poder virtualizar (vmware server / virtualbox). 🡪 **Tipo 1**

• Paravirtualización o bare-metal (nativa): no necesitan un sistema operativo, son un sistema operativo y utilizan acceso a hardware específico (intel vt o amd-v). 🡪 **Tipo 2**

**HYPERVISOR 🡪 Es el software que permite la virtualización.**

Un hipervisor (en inglés hypervisor) o monitor de máquina virtual (virtual machine monitor) es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos en una misma computadora. *Es el que permite que el hardware físico se convierta en hardware lógico, y así dividir por ejemplo un disco duro fisico en 4 discos duros virtuales.*

**Tipos de Hypervisores**

•Virtualización de Hardware. **Tipo 1**. Utiliza el sistema operativo real de tu máquina (como VMware o VirtualBox).

•Virtualización a nivel de Sistema Operativo. **Tipo 2**. Funciona directamente en el hardware de tu computadora (como Xen o Microsoft Hyper-V).

**INFRAESTRUCTURA DE VIRTUALIZACIÓN**

Son los recursos (Servidores, Almacenamiento, Red) necesarios para conseguir un entorno virtual. Separando los distintos elementos logramos conseguir:

* + Disponibilidad
  + Flexibilidad
  + Automatización

**MAQUINAS VIRTUALES Y CONTENEDORES**

**Contenedor**, es una máquina virtual ligera, que comparte funciones con el kernel del sistema operativo, que está realizando las funciones de hipervisor. Openvz, lxc – linux containers, parallels, docker y virtuozzo son los sistemas más conocidos para crear contenedores.

**Máquina virtual**, es una maquina a todos los efectos, y para ello el sistema realiza una reserva de los recursos físicos según las características especificadas para la máquina.

**¿QUE ES EL CLOUD COMPUTING?**

Es la utilización de recursos (servidores, aplicaciones, …) que se encuentran en internet y la gestión de la infraestructura (servidores, almacenamiento, red) se encarga un tercero para tratar de obtener una optimización total de la infraestructura mediante la automatización para conseguir una flexibilidad y adaptabilidad de los recursos informáticos.

¿Qué se quiere resolver con la utilización de cloud computing?

* Costes de la infraestructura: se impone la filosofía del pago por uso, además de la reducción de costes provenientes del consumo energético, mantenimiento, etc... Todos los costes asociados a un **CPD(\*)** se trasladan a un tercero. Outsourcing.
* Escalabilidad: no importa si se necesita uno o cien servidores, 1 gb o 100 tb de almacenamiento. El aumento de recursos es casi instantáneo.
* Flexibilidad: disponibilidad ante picos de trabajo se puede ampliar la infraestructura, desechándola al finalizar. Se puede tener la infraestructura funcionando únicamente en el horario laboral, la infraestructura esta siempre disponible, desde cualquier lugar. Al encontrarse en internet, lo único que necesitamos es conexión para poder acceder a nuestros recursos.

**Existen tres conceptos de nubes:**

**1.\ Publica:** Son recursos informáticos accesibles desde internet, normalmente compartidos con otros clientes.

* Ventajas:
  + Alto retorno de la inversión inicial
  + Servicios ofrecidos por un tercero
  + Pago por uso
* Desventajas:
  + Dependencia de las comunicaciones
  + Seguridad

**2.\ Privada:** Son recursos informáticos para nosotros, pueden ser en nuestro **CPD** o en **CPD** externo, el acceso es exclusivo para nosotros (VPN), suele ser un servicio a medida de las necesidades.

* Ventajas:
  + Si se realiza en nuestro **CPD**, las ventajas de la nube sin el problema de las comunicaciones
* Desventajas:
  + Costes de mantenimiento altos
  + Bajo retorno de la inversión

**3.\ Hibrida:** Es una mezcla de las nubes públicas y privadas. Se intentan obtener lo mejor de ambas, sin las desventajas que tienen y se suele utilizar como paso intermedio para una migración total a la nube.

**(\*)** “**CPD**" es la abreviatura de "Centro de Procesamiento de Datos", también conocido como centro de datos. En el contexto de la nube, cuando se menciona "CPD", se refiere al centro de datos donde se alojan los servidores y otros recursos de infraestructura de la nube. Este centro de datos puede estar ubicado en las instalaciones de la empresa o puede ser proporcionado por un proveedor de servicios en la nube.

**Compiladores**

En primer lugar, para que el ordenador o el procesador puedan  
comprender las instrucciones que contiene un programa desarrollado previamente, el código fuente escrito en los lenguajes de programación actuales debe convertirse a un formato legible por máquina.  
 De este procedimiento, dependiendo del lenguaje de programación, se  
encarga un compilador o un intérprete. ¿Qué son exactamente estos dos  
programas? Y ¿en qué se diferencian?

Un intérprete y un compilador son dos tipos de programas que se utilizan para ejecutar código, pero tienen diferencias fundamentales en cómo procesan y ejecutan ese código.

**1. \*\*Compilador\*\*:**

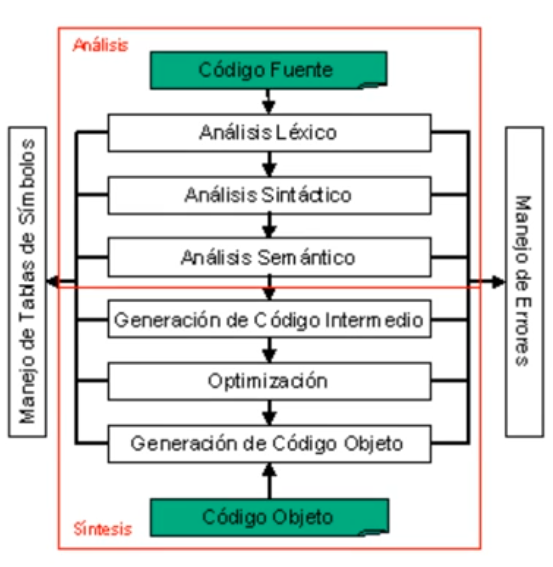
* Un compilador es un programa informático que **traduce todo el código fuente**  
  de un proyecto de software **a código máquina** antes de ejecutarlo.
* Solo entonces **el procesador ejecuta** el software, **obteniendo todas las instrucciones** en código máquina antes de comenzar.
* De esta manera, el procesador cuenta con todos los componentes necesarios para ejecutar el software, procesar las entradas y generar los resultados.
* No obstante, en muchos casos, durante el proceso de compilación tiene lugar un paso intermedio fundamental: antes de generar la traducción final en código máquina, la mayoría de los compiladores suelen **convertir el código fuente en un código intermedio (también llamado código objeto)** que, a menudo, es compatible con diversas plataformas y que, además, también puede ser utilizado por un intérprete.
* Al producir el código**, el compilador determina qué instrucciones van a enviarse al procesador y en qué orden**. Si las instrucciones no son interdependientes, incluso es posible que puedan procesarse en paralelo.
* Ejemplos de lenguajes compilados son C, C++, y Rust.

**2. \*\*Intérprete\*\*:**

* Un intérprete es un programa que lee y **ejecuta el código fuente línea por línea** o instrucción por instrucción en tiempo de ejecución.
* No produce un archivo ejecutable independiente; en cambio**, el código fuente se lee y se interpreta cada vez que se ejecuta el programa**.
* Esto permite una mayor flexibilidad, ya que los cambios en el código fuente se reflejan inmediatamente en la ejecución del programa sin necesidad de volver a compilar.
* Cuando un intérprete está ejecutando un programa, **utiliza sus propias bibliotecas internas** para traducir el código fuente del programa a instrucciones que la computadora pueda entender y ejecutar. Una vez que una línea de código fuente se ha traducido a estas instrucciones legibles por la máquina, se envía directamente al procesador para su ejecución.
* Solo **se interrumpe prematuramente si se produce un fallo durante el procesamiento**, lo que simplifica mucho la resolución de los errores, ya que la línea de código problemática se detecta inmediatamente después de ocurrir el fallo.
* Ejemplos de lenguajes interpretados son Python, JavaScript y Ruby.

En resumen, mientras que un compilador traduce todo el código de una vez a un formato ejecutable, un intérprete ejecuta el código directamente sin una fase de compilación previa. Cada enfoque tiene sus propias ventajas y desventajas en términos de rendimiento, portabilidad y facilidad de desarrollo.

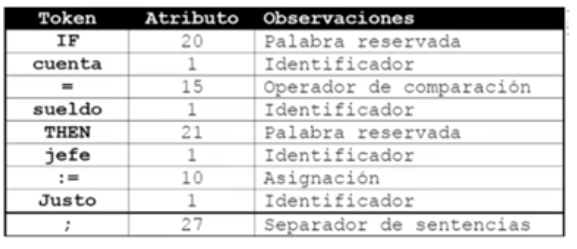
**Fase de un Compilador**



***Video explicativo:*** [***https://www.youtube.com/watch?v=5I\_0N7S41HQ&ab\_channel=AnaGabriela***](https://www.youtube.com/watch?v=5I_0N7S41HQ&ab_channel=AnaGabriela)

**1. \*\*Análisis léxico\*\*:**

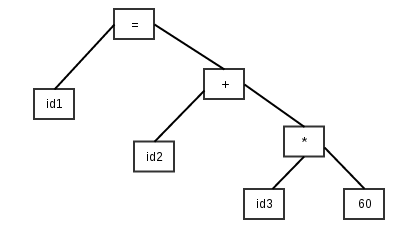
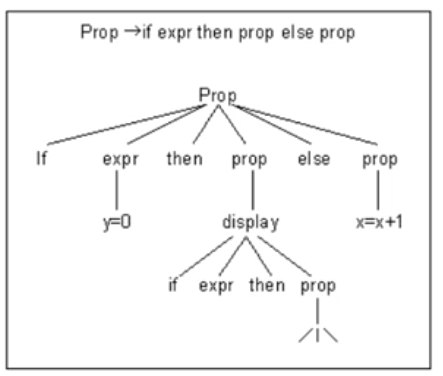
- En esta fase, el código fuente se divide en unidades léxicas más pequeñas llamadas "tokens", como identificadores, palabras clave, operadores, etc.



**Expresión** 🡪 precio = costo + impuesto \* 60  
**Análisis léxico** 🡪 id1 = id2 + id3 \* 60

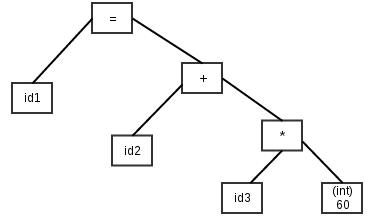
**2. \*\*Análisis sintáctico\*\*:**

- Se verifica la estructura gramatical (Jerárquica) del código fuente utilizando reglas definidas por la gramática del lenguaje. Se construye un árbol sintáctico que representa la estructura del programa.



**3. \*\*Análisis semántico\*\*:**

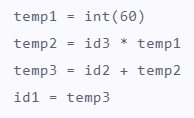
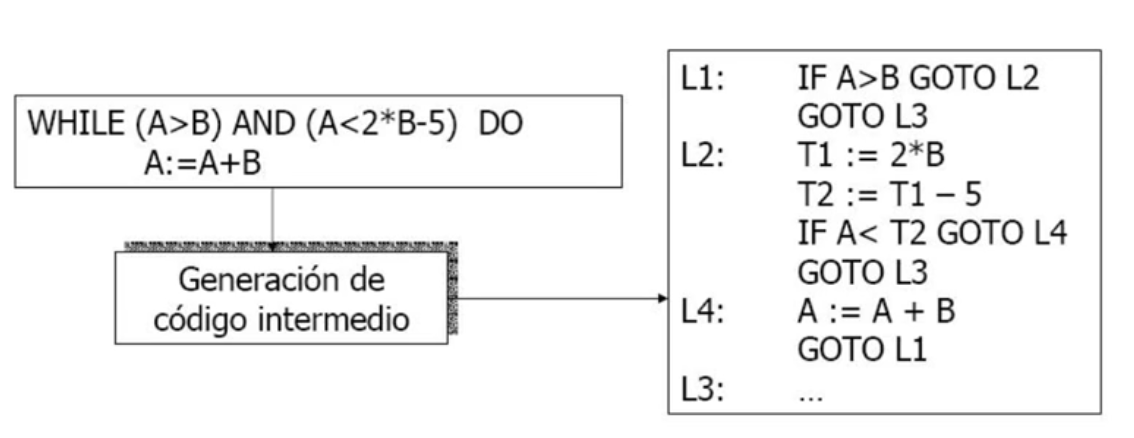
- Se comprueba si el programa cumple con las reglas de significado del lenguaje. Se detectan errores semánticos como el uso incorrecto de variables o tipos de datos.



*El análisis semántico lograría reconocer la constante como un numero entero*

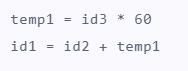
**4. \*\*Generación de código intermedio\*\*:**

- El compilador puede generar un código intermedio independiente de la plataforma que representa el programa en un nivel más abstracto que el código máquina, facilitando la optimización y la portabilidad.



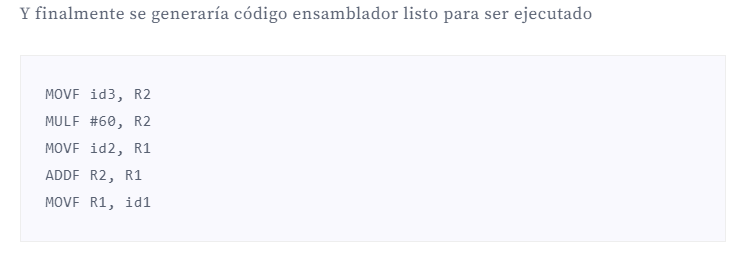
**5. \*\*Optimización de código\*\*:**

- Se realizan transformaciones en el código intermedio para mejorar su rendimiento, como la eliminación de código muerto o la reorganización de instrucciones para reducir la cantidad de operaciones.



**6. \*\*Generación de código objetivo\*\*:**

- Finalmente, se genera el código objetivo específico para la plataforma de destino, como código máquina o código de bytes para una máquina virtual. Este código es el que realmente ejecutará el programa.



**Linker**

Es una herramienta del proceso de compilación que se encarga de combinar múltiples archivos objeto y bibliotecas para crear un único programa ejecutable o biblioteca compartida. Esencialmente, su función principal es resolver las referencias entre diferentes partes del código y vincularlas de manera adecuada.