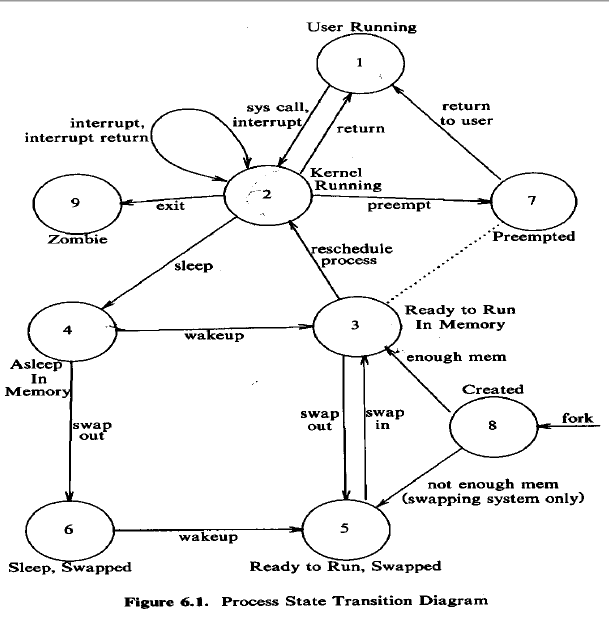
**RESUMEN ISO**

* Sistema Operativo:
  + Es software, necesita de procesador y memoria para ejecutarse.
  + Existen dos perspectivas: de arriba hacia abajo y de abajo hacia arriba.
  + De arriba hacia abajo:
    - Abstracción con respecto a la arquitectura (la arquitectura es el conjunto de instrucciones, organización de memoria, E/S, estructura de bus).
    - El SO “oculta” el hardware y presenta a los programas abstracciones más simples de manejar.
    - Los programas de aplicación son los “clientes” del SO.
    - Comparación: uso de escritorio y uso de comandos de texto.
    - Comodidad, “amigabilidad” (friendliness).
  + De abajo hacia arriba:
    - Visión del SO como un administrador de recursos.
    - Administra los recursos de hardware de uno o más procesos.
    - Provee un conjunto de servicios a los usuarios del sistema.
    - Maneja la memoria secundaria y dispositivos de I/O.
    - Ejecución simultanea de procesos.
    - Multiplexación en tiempo (CPU) y en espacio (memoria).
  + El SO gestiona el hardware, controla la ejecución de los procesos, es la interfaz entre las aplicaciones y el hardware y actúa como intermediario entre un usuario de una computadora y el hardware de la misma.
  + Objetivos:
    - Comodidad: hacer más fácil el uso de la PC.
    - Eficiencia: hacer un uso más eficiente de los recursos del sistema.
    - Evolución: permitir la introducción de nuevas funciones al sis tema sin interferir con funciones anteriores.
  + Componentes:
    - Kernel (Núcleo):
      * Es una “porción de código” que se encuentra en memoria principal y se encarga de la administración de los recursos.
      * Implementa servicios esenciales: manejo de memoria, administración de procesos, comunicación y concurrencia, y gestión de la E/S.
    - Shell: GUI / CUI o CLI.
    - Herramientas: editores, compiladores, librerías, etc.
  + Servicios de un SO:
    - Administración y planificación del procesador:
      * Imparcialidad, “justicia” en la ejecución (Fairness).
      * Que no haya bloqueos.
      * Manejo de prioridades.
    - Administración de memoria:
      * Memoria física vs memoria virtual. Jerarquías de memoria.
      * Protección de programas que compiten o se ejecutan concurrentemente.
    - Administración de almacenamiento – Sistemas de archivos:
      * Acceso a medios de almacenamiento externos.
    - Administración de dispositivos:
      * Ocultamiento de dependencias de hardware.
      * Administración de accesos simultáneos.
    - Detección de errores y respuestas:
      * Errores de hardware internos y externos: errores de memoria y errores de dispositivos.
      * Errores de software: errores aritméticos y acceso no permitido a direcciones de memoria.
      * Incapacidad del SO para conceder una solicitud de una aplicación.
    - Interacción del usuario (Shell).
    - Contabilidad:
      * Recoger estadísticas del uso.
      * Monitorear parámetros de rendimiento.
      * Anticipar necesidades de mejoras futuras.
      * Dar elementos si es necesario facturar tiempo de procesamiento.
  + Problemas que un SO debe evitar:
    - Que un proceso se apropie de la CPU.
    - Que un proceso intente ejecutar instrucciones de E/S por ejemplo.
    - Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado. Proteger los espacios de direcciones.
  + Para evitar estos problemas debe:
    - Gestionar el uso de la CPU.
    - Detectar intentos de ejecución de instrucciones de E/S ilegales.
    - Detectar accesos ilegales a memoria.
    - Proteger el vector de interrupciones. Así como las RAI (rutinas de atención de interrupciones).
  + Apoyo del Hardware: modos de ejecución, interrupción del clock, protección de la memoria (lógica vs física).
  + Modos de ejecución:
    - El bit en la CPU indica el modo actual.
    - Las instrucciones privilegiadas deben ejecutarse en modo *Supervisor o Kernel* (necesitan acceder a estructuras del Kernel, o ejecutar código que no es del proceso).
    - En modo *usuario*, el proceso puede acceder solo a su espacio de direcciones, es decir a las direcciones “propias”.
    - El Kernel del SO se ejecuta en modo supervisor.
    - El resto del SO y los programas de usuario se ejecutan en modo usuario (subconjunto de instrucciones permitidas).
    - Hay que tener en cuenta que:
      * Cuando se arranque el sistema, arranca con el bit en modo supervisor.
      * Cada vez que comienza a ejecutarse un proceso de usuario, este bit se debe poner en modo usuario (mediante una instrucción especial).
      * Cuando hay un trap o una interrupción, el bit de modo se pone en modo Kernel (Única forma de pasar a modo Kernel. No es el proceso de usuario quien hace el cambio explícitamente).
      * Como actúa: cuando el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (las llamadas instrucciones privilegiadas) esto el hardware lo detecta como una operación ilegal y produce un trap al SO.
    - En Windows:
      * En WIN2000 el modo núcleo ejecuta los servicios ejecutivos. El modo usuario ejecuta los procesos de usuario.
      * Cuando un programa se bloquea en modo usuario, a lo sumo se escribe un suceso en el registro de sucesos. Si el bloqueo se produce estando en modo supervisor se genera la BSOD (pantalla azul).
    - Procesador Intel 8088 no tenía modo dual de operación ni protección por hardware.
    - En MsDos las aplicaciones pueden acceder directamente a las funciones básicas de E/S para escribir directamente en pantalla o en disco.
  + Resumiendo:
    - Modo Kernel:
      * Modo privilegiado.
      * Manejo estricto de pautas de confiabilidad/seguridad.
      * Manejo de:
        + CPU, memoria, operaciones de E/S.
        + Administración multiprocesador, diagnosticos, testing.
        + Estructura del Filesystem.
        + Comunicaciones: interfaz de red.
    - Modo user:
      * Más flexible.
      * Funciones de mantenimiento más simples, debugging.
        + Compiler, assembler, interpreter, linker/loader.
        + File system management, telecommunication, network management.
        + Editors, spreadsheets, user applications.
  + Protección de la memoria:
    - Delimitar el espacio de direcciones del proceso.
    - Poner límites a las direcciones que puede utilizar un proceso.
      * Por ejemplo: uso de un registro base y un registro limite.
      * El SO carga estos registros por medio de instrucciones privilegiadas. Esta acción solo puede realizarse en modo Kernel.
    - La memoria principal aloja al SO y a los procesos de usuario:
      * El SO debe protegerse de ser accedido por procesos de usuario.
      * Él debe proteger el espacio de direcciones de un proceso del acceso a otros procesos.
  + Protección de la E/S:
    - Las instrucciones de E/S se definen como privilegiadas.
    - Deben ejecutarse en modo Kernel (se deberían realizar a través del SO).
  + Protección de la CPU:
    - Uso de interrupción por clock para evitar que un proceso se apropie de la CPU.
    - Se implementa normalmente a través de un clock y un contador.
    - El SO le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero el SO podría expulsar al proceso para que se ejecute otro.
    - Las instrucciones que modifican el funcionamiento del reloj son privilegiadas.
    - Se le asigna al contador el valor que se quiere que se ejecute un proceso.
    - Se la usa también para el cálculo de la hora actual, basándose en cantidad de interrupciones ocurridas cada tanto tiempo y desde una determinada fecha y hora.
  + System Calls:
    - Es la forma en que los programas de usuario acceden a los servicios del SO.
    - Los parámetros asociados a las llamadas puede pasarse de varias maneras: por registros, bloques o tablas en memoria o pilas.
    - Se ejecutan en modo supervisor.
    - Categorías:
      * Control de procesos.
      * Manejo de archivos.
      * Manejo de dispositivos.
      * Mantenimiento de información del sistema.
      * Comunicaciones.
* Elementos básicos de una computadora:
  + Procesador.
  + Memoria principal: volátil, se refiere como memoria real o primaria.
  + Componentes de E/S: dispositivos de memoria secundaria, equipamiento de comunicación, monitor, teclado, mouse, etc.
  + Bus sistema: comunicación entre procesadores, memoria, dispositivos de E/S.
* Registros del procesador:
  + Visibles por el usuario: registros que pueden ser usados por las aplicaciones.
    - Pueden ser referenciados por lenguaje máquina.
    - Disponibles para programas/aplicaciones.
    - Tipos de registros: datos, direcciones (index, segment pointer, stack pointer).
  + De control y estado: para control operativo del procesador. Usados por rutinas privilegiadas del SO para controlar la ejecución de procesos.
    - Program Counter (PC): contiene la dirección de la próxima instrucción a ser ejecutada.
    - Instruction Register (IR): contiene la instrucción a ser ejecutada.
    - Program Status Word (PSW): contiene códigos de resultados de operaciones. Habilita y deshabilita interrupciones. Indica el modo de ejecución (supervisor/usuario).
* Ciclo de ejecución de instrucción:
  + Dos pasos:
    - Procesador lee la instrucción desde la memoria.
    - Procesador ejecuta la instrucción.
* Instrucción: Fetch y Execute.
  + El procesador busca (fetch) la instrucción en la memoria: (PC) 🡪 IR
  + El PC se incrementa después de cada fetch para apuntar a la próxima instrucción: PC = PC + 4.
* Instruction Register (IR):
  + La instrucción referenciada por el PC se almacena en el IR y se ejecuta.
  + Categoría de instrucciones:
    - Procesador – Memoria: transfiere datos entre procesador y memoria.
    - Procesador – E/S: transfiere datos a/o desde periféricos.
    - Procesamiento de datos: operaciones aritméticas o lógicas sobre datos.
    - Control: alterar secuencia de ejecución.
* Interrupciones:
  + Interrumpen el secuenciamiento del procesador durante la ejecución de un proceso.
  + Dispositivos de E/S más lentos que el procesador: procesador debe esperar al dispositivo.
  + Interrupt Handler:
    - Programa o rutina que determina la naturaleza de una interrupción y realiza lo necesario para atenderla. Por ejemplo, para un dispositivo particular de E/S.
    - Generalmente es parte del SO.
  + Suspende la secuencia normal de ejecución.
  + Ciclo de interrupción:
    - El procesador chequea la existencia de interrupciones.
    - Si no existen interrupciones, la próxima instrucción del programa es ejecutada.
    - Si hay pendiente alguna interrupción, se suspende la ejecución del programa actual y se ejecuta la rutina de manejo de interrupciones.
  + Múltiples interrupciones:
    - Deshabilitar las interrupciones mientras una interrupción está siendo procesada.
    - Definir prioridades a las interrupciones.
* Evolución de un Sistema Operativo:
  + Los SO evolucionan con el objeto de:
    - Soportar nuevos tipos de hardware.
    - Brindar nuevos servicios.
    - Ofrecer mejorar y alternativas a problemas existentes: en la planificación, en el manejo de la memoria, etc.
  + Evolución histórica:
    - Procesamiento en serie:
      * No existía un SO.
      * Maquinas eran utilizadas desde una consola que contenía luces, interruptores, dispositivos de entrada e impresoras.
      * Problemas:
        + Planificación. Alto nivel de especialización. Costos.
        + Configuración: carga del compilador, fuente, salvar el programa compilado, carga y linkeo.
    - Sistemas por lotes sencillo (batch):
      * Monitor residente:
        + Software que controla la secuencia de eventos.
        + Los trabajos se colocan juntos.
        + Los programas vuelven al monitor cuando finaliza la ejecución.
        + No hay interacción con el usuario mientras se ejecutan los trabajos.
      * Sistema batch:
        + Baja utilización de la CPU.
        + Dispositivos de E/S mucho más lentos con respecto a la CPU.
        + Ante instrucción de E/S, el procesador permanece ocioso. Cuando se completa la E/S, se continúa con la ejecución del programa que se estaba ejecutando.
  + Multiprogramación:
    - La operación de los sistemas batch se vio beneficiada del spooling de las tareas, al solapar la E/S de una tarea de la ejecución de otra.
    - Al estar las tareas cargadas en disco, ya no era necesario ejecutarlas en el orden en el que fueron cargadas (job scheduling).
    - El SO mantiene varias tareas en memoria al mismo tiempo.
    - La secuencia de programas es de acuerdo a prioridad u orden de llegada.
    - Cuando el proceso necesita realizar una operación de E/S, la CPU en lugar de permanecer ociosa, es utilizada para otro proceso.
    - Después que se completa la atención de la interrupción, el control puede o no retornar al programa que se estaba ejecutando al momento de la interrupción.
  + Tiempo compartido:
    - Utiliza la multiprogramación para manejar múltiples trabajos interactivos.
    - El tiempo del procesador es compartido entre múltiples trabajos.
    - Múltiples usuarios podrían acceder simultáneamente al sistema utilizando terminales.
    - Los procesos usan la CPU por un periodo máximo de tiempo, luego del cual se le da la CPU a otro proceso.
* **PROCESOS**
  + Un proceso es un programa en ejecución.
  + Para nosotros serán sinónimos: tarea, job y proceso.
  + Diferencias entre un programa y un proceso:
    - Programa: es estático, no tiene program counter y existe desde que se edita hasta que se borra.
    - Proceso: es dinámico, tiene program counter y su ciclo de vida comprende desde que se lo “dispara” hasta que termina.
  + Modelo de proceso:
    - Multiprogramación en 4 procesos.
    - Modelo conceptual de 4 procesos secuenciales e independientes.
    - Solo un proceso se encontrara activo en cualquier instante (si tenemos una sola CPU).
  + Componentes de un proceso:
    - Proceso: entidad de abstracción.
    - Un proceso (para poder ejecutarse) incluye como mínimo:
      * Sección de código (texto).
      * Sección de datos (variables globales).
      * Stack (s) (datos temporarios: parámetros, variables temporales y direcciones de retorno).
    - Stacks:
      * Un proceso cuenta con 1 o más stacks: en general modo usuario y modo Kernel.
      * Se crean automáticamente y su medida se ajusta en run-time.
      * Está formado por *stack* *frames* que son *pushed* (al llamar a una rutina) y *popped* (cuando se retorna de ella).
      * El stack frames tiene los parámetros de la rutina (variables locales), y datos necesarios para recuperar el stack frame anterior (el contador del programa y el valor del stack pointer en el momento del llamado)
  + Atributos de un proceso:
    - Identificación del proceso, y del proceso padre.
    - Identificación del usuario que lo “disparo”.
    - Si hay estructura de grupos, grupo que lo disparo.
    - En ambientes multiusuario, desde que terminal y quien lo ejecuto.
  + Process Control Block (PCB):
    - Estructura de datos asociada al proceso (abstracción).
    - Existe una por proceso.
    - Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.
    - Contiene la información asociada con cada proceso:
      * PID, PPID, etc.
      * Valores de los registros de la CPU (PC, AC, etc.).
      * Planificación (estado, prioridad, tiempo consumido, etc.).
      * Ubicación (representación) en memoria.
      * Accounting.
      * Entrada salida (estado, pendientes, etc.).
  + Espacio de direcciones de un proceso:
    - Es el conjunto de direcciones de memoria que ocupa el proceso: stack, text y datos.
    - No incluye su PCB o tablas asociadas.
    - Un proceso en modo usuario puede acceder solo a su espacio de direcciones.
    - Un proceso en modo Kernel se puede acceder a estructuras internas o a espacios de direcciones de otros procesos.
  + El contexto de un proceso:
    - Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.
    - Son parte del contexto, los registros de CPU, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.
  + Cambio de contexto (Context Switch):
    - Se produce cuando la CPU cambia de un proceso a otro.
    - Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornara después la CPU.
    - Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto.
    - Es tiempo no productivo de CPU.
    - El tiempo que consume depende del soporte de hardware.
  + Kernel del Sistema Operativo:
    - Es un conjunto de modulos de software.
    - Se ejecuta en el procesador como cualquier otro proceso.
    - Kernel como entidad independiente:
      * El Kernel se ejecuta fuera de todo proceso.
      * Cuando un proceso es “interrumpido” o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del sistema operativo.
      * El Kernel tiene su propia región de memoria.
      * El Kernel tiene su propio Stack.
      * Finalizada su actividad, le devuelve el control al proceso (o a otro diferente).
      * Importante:
        + El Kernel NO es un proceso.
        + Se ejecuta como una entidad independiente en modo privilegiado.
    - Kernel “dentro” el proceso:
      * El “código” del Kernel se encuentra dentro del espacio de direcciones de cada proceso.
      * El Kernel se ejecuta en el MISMO contexto que algún proceso de usuario.
      * El Kernel se puede ver como una colección de rutinas que el proceso utiliza.
      * Dentro de un proceso se encuentra el código del programa (user) y el código de los módulos de software del SO (Kernel).
      * Cada proceso tiene su propio stack (uno en modo usuario y otro en modo Kernel).
      * El proceso es el que se ejecuta en *modo usuario* y el Kernel del SO se ejecuta en *modo Kernel* (cambio de modo).
      * El código del Kernel es compartido por todos los procesos.
      * Cada interrupción (incluyendo las de System Call) es atendida en el contexto del proceso que se encontraba en ejecución pero en modo Kernel.
  + Estados de un proceso:
    - En su ciclo de vida, un proceso pasa por diferentes estados:
      * New (nuevo):
        + Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre.
        + En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.
      * Ready (listo para ejecutar):
        + Luego que el Scheduler de largo plazo eligió al proceso para cargarlo en memoria, el proceso queda en estado listo.
        + El proceso solo necesita que se le asigne CPU.
        + Está en la cola de procesos listos (ready queue).
      * Running (ejecutándose):
        + El Scheduler de corto plazo lo eligió para asignarle CPU.
        + Tendrá la CPU hasta que se termine el periodo de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S.
      * Waiting (en espera):
        + El proceso necesita que se cumpla el evento esperado para continuar.
        + El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso.
        + Sigue en memoria, pero no tiene la CPU.
        + Al cumplirse el evento, pasara al estado de listo.
      * Terminated (terminado).
    - Transiciones:
      * New – Ready: por elección del Scheduler de largo plazo (carga en memoria).
      * Ready – Running: por elección del Scheduler de corto plazo (asignación de CPU).
      * Running - Waiting: el proceso “se pone a dormir”, esperando por un evento.
      * Waiting – Ready: termino la espera y compite nuevamente por la CPU.
    - Caso especial de transición:
      * Running – Ready: cuando el proceso termina su quantum (franja de tiempo) sin haber necesitado ser interrumpido por un evento, pasa al estado de Ready, para competir por CPU, pues no está esperando por ningún evento.
  + Colas en planificación de procesos:
    - Se enlazan las PCBs.
    - Ejemplos:
      * De trabajos a procesos: contiene todos los procesos en el sistema.
      * De procesos listos: residentes en memoria principal esperando para ejecutarse.
      * De dispositivos: esperando por un dispositivo de I/O.
  + Módulos de planificación:
    - Son módulos (software) del Kernel que realizan distintas tareas asociadas a la planificación.
    - Se ejecutan ante determinados eventos que así lo requieren:
      * Creación/terminación de procesos.
      * Eventos de sincronización o de E/S.
      * Finalización de lapso de tiempo, etc.
    - Scheduler de long term:
      * Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria.
      * Puede no existir este Scheduler y absorber esta tarea el de short term.
    - Scheduler de short term:
      * Decide a cuál de los procesos en la cola de listos se elige para que se use la CPU.
      * Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling.
    - Scheduler de médium term (swapping):
      * Si es necesario, reduce el grado de multiprogramación.
      * Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema.
      * Términos asociados: *swap out* (sacar de memoria), *swap in* (volver a memoria).
    - Otros módulos: **Dispatcher** y **Loader.**
    - Pueden no existir como módulos separados de los schedulers vistos, pero la función debe cumplirse.
      * Dispatcher: hace cambio de contexto, cambio de modo de ejecución, “despacha” el proceso elegido por el Short Term (es decir, “salta” a la instrucción a ejecutar).
      * Loader: carga en memoria el proceso elegido por el long term.



* + Diagrama incluyendo Swapping:
    - 1- Ejecución en modo usuario.
    - 2- Ejecución en modo Kernel.
    - 3- El proceso está listo para ser ejecutado cuando sea elegido.
    - 4- Proceso en espera en memoria principal.
    - 5- Proceso listo, pero el swapper debe llevar al proceso a memoria principal antes que el Kernel lo pueda elegir para ejecutar.
    - 6- Proceso en espera en memoria secundaria.
    - 7- Proceso retornando desde el modo Kernel al user. Pero el Kernel se apropia, hace un context switch para darle la CPU a otro proceso.
    - 8- Proceso recientemente creado y en transición: existe, pero aún no está listo para ejecutar, ni está dormido.
    - 9- El proceso ejecuto la system call *exit* y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, código resultando del exit. Es el estado final.
  + Comportamiento de los procesos:
    - Procesos alternan ráfagas de CPU y de I/O.
    - CPU-Bound:
      * Mayor parte del tiempo es utilizando la CPU.
    - I/O-Bound (I/O = E/S):
      * Mayor parte del tiempo esperando por I/O.
    - La velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S:
      * Pensar: necesidad de atender rápidamente proceso I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para procesos CPU-bound.
  + Planificación:
    - Necesidad de determinar cuál de todos los procesos que están listos para ejecutarse, se ejecutara a continuación en un ambiente multiprogramado.
    - Algoritmo de planificación: algoritmo utilizado para realizar la planificación del sistema.
    - Algoritmos Apropiativo (preemtive):
      * En estos existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU.
    - Algoritmos No Apropiativo (nonpreemptive):
      * En estos los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU.
      * Se bloque por E/S o finaliza.
      * No hay decisiones de planificación durante las interrupciones de reloj.
    - Categorías de los algoritmos de planificación:
      * Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:
        + Equidad: otorgar una parte justa de la CPU a cada proceso.
        + Balance: mantener ocupadas todas las partes del sistema.
      * Ejemplos:
        + Procesos por lotes (batch).
        + Procesos interactivos.
        + Procesos en Tiempo Real.
    - Procesos Batch:
      * No existen usuarios que esperen una respuesta en una terminal.
      * Se pueden utilizar algoritmos no apropiativo.
      * Metas propias de este tipo de algoritmo:
        + Rendimiento: maximizar el número de trabajos por hora.
        + Tiempo de retorno: minimizar los tiempos entre el comienzo y la finalización.
        + Uso de la CPU: mantener la CPU ocupada la mayor cantidad de tiempo posible.
      * Ejemplos: FCFS (First Come First Served) y SJF (Shortest Job First).
    - Procesos Interactivos:
      * No solo interacción con los usuarios: un servidor, necesita de varios procesos para dar respuesta a diferentes requerimientos.
      * Son necesarios algoritmos apropiativo para evitar que un proceso acapare la CPU.
      * Metas propias de este tipo de algoritmos:
        + Tiempo de respuesta: responder a peticiones con rapidez.
        + Proporcionalidad: cumplir con expectativas de los usuarios.

Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto.

* + - * Ejemplos: Round Robin, Prioridades, Colas Multinivel, SRTF (Shortest remaining time first.
    - Política vs Mecanismo:
      * Existen situaciones en las que es necesario que la planificación de uno o varios procesos se comporte de manera diferente.
      * El algoritmo de planificación debe estar parametrizado, e manera que los procesos/usuarios pueden indicar los parámetros para modificar la planificación.
      * El Kernel implementa el mecanismo.
      * El usuario/proceso/administrador utiliza los parámetros para determinar la política.
      * Ejemplo:
        + Un algoritmo de planificación por prioridades y una System Call que permite modificar la prioridad de un proceso (man nice).
        + Un proceso puede determinar las prioridades de los procesos que el crea, según la importancia de los mismos.
  + Creación de procesos:
    - Un proceso es creado por otro proceso.
    - Un proceso padre tiene uno o más procesos hijos.
    - Se forma un árbol de procesos.
    - Actividades en la creación:
      * Crear la PCB.
      * Asignar un PID (Process IDentification) único.
      * Asignarle memoria para regiones: stack, text y datos.
      * Crear estructuras de datos asociadas: fork (copiar el contexto, regiones de datos, text y stack).
    - Relación entre procesos padre e hijo:
      * Con respecto a la ejecución:
        + El padre puede continuar ejecutándose concurrentemente con su hijo.
        + El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución.
      * Con respecto al espacio de direcciones:
        + El hijo es un duplicado del proceso padre (caso Unix).
        + Se crea el proceso y se le carga adentro el programa (caso Windows).
    - Creación de procesos en UNIX:
      * System Call fork () crea nuevo proceso.
      * System Call execve(), usada después del fork, carga un nuevo programa en el espacio de direcciones.
    - Creación de procesos en Windows:
      * System Call CreateProcess() crea un nuevo proceso y carga el programa para ejecución.
    - Terminación de procesos:
      * Ante un (exit), se retorna el control al SO: el proceso padre puede esperar recibir un código de retorno (via wait).
      * Proceso padre puede terminar la ejecución de sus hijos (kill): la tarea asignada al hijo se terminó. Cuando el padre termina su ejecución habitualmente no se permite a los hijos continuar, pero existe la opción. Terminación en cascada.
  + Procesos Cooperativos:
    - Afecta o es afectado por la ejecución de otros procesos en el sistema.
    - Sirven para:
      * Para compartir información (por ejemplo, un archivo).
      * Para acelerar el computo (separar una tarea en sub-tares que cooperan ejecutándose paralelamente).
      * Para planificar tareas de manera tal que se puedan ejecutar en paralelo.
  + Procesos Independientes:
    - El proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.
* **MEMORIA**
  + Administración de memoria:
    - División lógica de la memoria física para alojar múltiples procesos: garantizado protección. Depende del mecanismo provisto por el hardware.
    - Asignación eficientemente: contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos.
  + Requisitos:
    - Reubicación:
      * El programador no debe ocuparse de conocer donde será colocado en la memoria RAM.
      * Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.
      * Las referencias a la memoria se deben “traducir” según ubicación actual del proceso.
    - Protección:
      * Los procesos NO deben referenciar (acceder) a direcciones de memoria de otros procesos (salvo que tengan permiso).
      * El chequeo se debe realizar durante la ejecución: NO es posible anticipar todas las referencias a memoria que un proceso puede realizar.
    - Compartición:
      * Permitir que varios procesos accedan a la misma porción de memoria: ejemplo, rutinas comunes, librerías, espacios explícitamente comparticos, etc.
      * Permite un mejor uso (aprovechamiento) de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones.
  + Abstracción (espacio de direcciones):
    - Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
    - El tamaño depende de la arquitectura del procesador:
      * 32 bits: 0 .. 2³² - 1
      * 64 bits: 0 .. 264 - 1
    - Es independiente de la ubicación “real” del proceso en la memoria RAM.
  + Direcciones:
    - Lógicas:
      * Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
      * Representa una dirección en el “espacio de direcciones del proceso”.
    - Físicas:
      * Referencia una localidad en la memoria física (RAM).
        + Dirección absoluta.
    - En caso de usar direcciones lógicas, es necesaria algún tipo de conversión a direcciones físicas.
    - Conversión de direcciones: una forma simple de hacer esto es utilizando registros auxiliares.
      * Registro base: dirección de comienzo del espacio de direcciones del proceso en la RAM.
      * Registro límite: dirección final del proceso o medida del proceso (tamaño de su espacio de direcciones).
      * Ambos valores se fijan cuando el ED del proceso es cargado a memoria.
      * Varían entre procesos (Context Switch).
    - Direcciones lógicas vs físicas:
      * Si la CPU traba con direcciones lógicas para acceder a memoria principal, se deben transformar en direcciones físicas.
        + Resolución de direcciones (address-binding): transformar la dirección lógica en la dirección física correspondiente.
      * Resolución en momento de compilación (archivos .com de DOS) y en tiempo de carga.
        + Direcciones lógicas y físicas son idénticas.
        + Para reubicar un proceso es necesario recompilarlo o recargarlo.
      * Resolución en tiempo de ejecución.
        + Direcciones lógicas y físicas son diferentes.
        + Direcciones lógicas son llamadas “direcciones virtuales”.
        + La reubicación se puede realizar fácilmente.
        + El mapeo entre “virtuales” y “físicas” es realizado por hardware: Memory Management Unit (MMU).
    - Memory Management Unit (MMU):
      * Dispositivo de Hardware que mapea direcciones virtuales a físicas.
        + Es parte del procesador.
        + Re-programar el MMU es una operación privilegiada: solo puede ser realizada en Kernel Mode.
      * El valor en el “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.
        + Los procesos nunca usan direcciones físicas.
    - Problemas del esquema
      * El esquema de Registro Base + Limite presenta problemas:
        + Necesidad de almacenar el espacio de direcciones de forma continua en la memoria física.
        + Fragmentación.
        + Mantener “partes” del proceso que no son necesarias.
      * Solución: paginación y/o segmentación.
    - Paginación:
      * Memoria física es dividida lógicamente en pequeños trozos de igual tamaño 🡪 **Marcos.**
      * Memoria lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos 🡪 **Páginas.**
      * El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la se coloca cada página.
      * La dirección lógica se interpreta como un número de página y un desplazamiento dentro de la misma.
    - Segmentación:
      * Esquema que soporta el “punto de vista de un usuario”.
      * Un programa es una colección de segmentos. Un segmento es una unidad lógica como: programa principal, procedimientos y funciones, variables locales y globales, stack, etc.
      * Puede causar fragmentación.
      * Todos los segmentos de un programa puede no tener el mismo tamaño (código, datos, rutinas).
      * Las direcciones lógicas consisten en dos partes: selector de segmento y desplazamiento dentro del segmento.
      * Arquitectura:
        + Tabla de segmentos: permite mapear la dirección lógica en física. Cada entrada contiene:

Base: dirección física de comienzo del segmento.

Limit: longitud del segmento.

* + - * + Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.
        + Segment-table length register (STLR): cantidad de segmentos de un programa.
      * Ventajas sobre paginación: compartir y proteger.
    - Segmentación paginada:
      * La paginación: es transparente al programador y elimina fragmentación externa.
      * La segmentación: es visible al programador y facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
      * Cada segmento es dividido en páginas de tamaño fijo.
  + Memoria Virtual (MV): Motivación.
    - Podemos pensar también que, no todo el espacio de direcciones del procesos se necesita en todo momento:
      * Rutinas o Librerias que se ejecutan una única vez (o nunca).
      * Partes del programa que no vuelven a ejecutarse.
      * Regiones de memoria alocadas dinámicamente y luego liberadas.
      * Etc.
    - Como se puede trabajar:
      * El SO puede traer a memoria las “piezas” de un proceso a medida que este las necesita.
      * Definiremos como “conjunto residente” a la porción del espacio de direcciones del proceso que se encuentra en memoria (alguna bibliografía lo llama “Working Set”).
      * Con el apoyo del hardware se detecta cuando se necesita una porción del proceso que no está en su conjunto residente.
    - Ventajas:
      * Mas procesos puede ser mantenidos en memoria:
        + Solo son cargadas algunas secciones de cada proceso.
        + Con más procesos en memoria principal es más probable que existan más procesos Ready.
      * Un proceso puede ser más grande que la memoria principal:
        + El usuario no se debe preocupar por el tamaño de sus programas.
        + La limitación la impone el hardware y el bus de direcciones.
    - ¿Qué se necesita para memoria virtual?
      * El hardware debe soportar paginación por demanda (y/o segmentación).
      * Un dispositivo de memoria secundaria (disco) que de él apoyo para almacenar las secciones del proceso que no están en Memoria Principal (área de intercambio).
      * El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.
    - Memoria Virtual con Paginación:
      * Cada proceso tiene su tabla de páginas.
      * Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en la memoria principal.
      * Cada entrada en la tabla de páginas tiene bits de control (entre otros):
        + Bit V: indica si la página está en memoria.
        + Bit M: indica si la página fue modificada. Si se modificó, en algún momento, se deben reflejar los cambios en Memoria Secundaria.
      * Entrada en la tabla de páginas de x86:
        + Una entrada valida tiene:

Bit V = 1.

Page Frame Number (PFN) – Marco de memoria asociado

Flags que describen su estado y protección.

* + Fallo de páginas (Page Fault):
    - Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en la memoria principal. Bit V = 0.
      * La página no se encuentra en su conjunto residente.
      * El bit V es controlado por el hardware.
    - El hardware detecta la situación y genera un trap al SO.
    - El SO podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la página que se necesite se cargue.
    - El SO busca un “Frame o Marco Libre” en la memoria y genera una operación de E/S al disco para copiar en dicho Frame la página del proceso que se necesita utilizar.
    - El SO puede asignarle la CPU a otro proceso mientras se completa la E/S.
      * La E/S se realizara y avisara mediante interrupción su finalización.
    - Cuando la operación de E/S finaliza, se notifica al SO y este:
      * Actualiza la tabla de páginas del proceso.
        + Coloca el Bit V en 1 en la página en cuestión.
        + Coloca la dirección base del Frame donde se colocó la página.
      * El proceso que genero el Fallo de Pagina vuelve a estado de Ready (listo).
      * Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes género el fallo de página.
  + Performance
    - Tasa de Page Faults 0 <= p <= 1
      * Si p = 0 no hay page faults.
      * Si p = 1, cada a memoria genera un page fault
    - Effective Access Time (EAT)
      * EAT = (1 – p) x memory Access + p x (page\_fault\_overhead + [swap\_page\_out] + swap\_page\_in + restar\_overhead)
  + Tabla de páginas:
    - Cada proceso tiene su tabla de páginas.
    - El tamaño de la tabla de páginas depende del espacio de direcciones del proceso.
    - Puede alcanzar un tamaño considerable.
    - Formas de organizar:
      * Tabla de 1 nivel: tabla única lineal.
        + Direcciones de 32 bits (20bits y 12 bits).
        + Direcciones de 64 bits (52 bits y 12 bits).
      * Tabla de 2 niveles o más (multinivel).
      * Tabla invertida: Hashing.
        + Utilizada en Arquitecturas donde el espacio de direcciones es muy grande (las tablas de páginas ocupan muchos niveles y la traducción es costosa).
        + Hay una entrada por cada frame. Hay una sola tabla para todo el sistema.
        + Usada en PowerPC, UltraSPARC, y IA-64.
        + El número de página es transformado en un valor de HASH.
        + El HASH se usa como índice de la tabla invertida para encontrar el marco asociado.
        + Solo se mantienen los PTEs de páginas presentes en memoria física.

Tabla invertida organizada como tabla hash en memoria principal: (Se busca indexadamente por un número de página virtual. Si está presente en tabla, se extrae el marco de página y sus protecciones. Si no está presente en tabla, corresponde a un fallo de página).

* + - La forma de organizarla depende del hardware subyacente.
  + Tamaño de la página:
    - Pequeño:
      * Menor fragmentación interna.
      * Más paginas requeridas por proceso 🡪 tablas de páginas más grandes.
      * Mas paginas pueden residir en memoria.
    - Grande:
      * Mayor fragmentación interna.
      * La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente 🡪 más rápido mover páginas hacia la memoria principal.
  + Translation Lookaside Buffer 1:
    - Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoria física.
      * Uno (o más) para obtener la entrada en la tabla de páginas.
      * Uno para obtener los datos.
    - Para solucionar este problema, una memoria cache de alta velocidad es usada para almacenar entradas de páginas.
      * TLB.
  + Translation Lookaside Buffer 2:
    - Contiene las entradas de la tabla de páginas que fueron usadas más recientemente.
    - Dada una dirección virtual, el procesador examina la TLB.
    - Si la entrada de la tabla de páginas se encuentra en la TLB (init), es obtenido el frame y armada la dirección física.
  + Translation Lookaside Buffer 3:
    - Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en la tabla de páginas del proceso.
    - Se controla si la página está en la memoria (si no está, se genera un Page Fault).
    - La TLB es actualizada para incluir la nueva entrada.
    - El cambio de contexto genera la invalidación de las entradas de la TLB.
  + Asignación de Marcos:
    - ¿Cuantas páginas de un proceso se pueden encontrar en memoria?: tamaño del Conjunto Residente.
    - Asignación dinámica: el número de marcos para cada proceso varía.
    - Asignación fija: número fijo de marcos para cada proceso.
      * Asignación Equitativa. Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso.
      * Asignación proporcional: se asigna acorde al tamaño del proceso.
  + Reemplazo de páginas:
    - ¿Qué sucede si ocurre un fallo de página y todos los marcos están ocupados? 🡪 “se debe seleccionar una página victima”.
    - ¿Cuál sería el reemplazo óptimo? 🡪 que la pagina a ser removida no sea referenciada en un futuro próximo.
    - La mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado.
    - Alcance del reemplazo:
      * Reemplazo global:
        + El fallo de página de un proceso puede reemplazar la página de cualquier proceso.
        + El SO no controla la tasa de page-faults de cada proceso.
        + Puede tomar frames de otro proceso aumentado la cantidad de frames asignados a él.
        + Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad.
      * Reemplazo local:
        + El fallo de página de un proceso solo puede reemplazar sus propias páginas – De su Conjunto Residente.
        + No cambia la cantidad de frames asignados.
        + El SO puede determinar cuál es la tasa de page-faults de cada proceso.
        + Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.
    - Algoritmos de reemplazo:
      * OPTIMO.
      * FIFO.
      * LRU (Least Recently User).
      * 2da chance.
      * NRU (Non Recently Used): utiliza bits R y M.
  + Thrashing (hiperpaginación):
    - Concepto: decimos que un sistema está en thrashing cuando pasa más tiempo paginando que ejecutando procesos.
    - Como consecuencia, hay una baja importante de performance en el sistema.
    - Ciclo del Thrashing:
      * 1) El Kernel monitorea el uso de la CPU.
      * 2) Si hay baja utilización 🡪 aumenta el grado de multiprogramación.
      * 3) Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos.
      * 4) Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos.
      * 5) Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU.
      * 6) Vuelve a 1).
    - El Scheduler de CPU y el Thrashing:
      * 1) Cuando se decrementa el uso de la CPU, el Scheduler long term aumenta el grado de multiprogramación.
      * 2) El nuevo proceso inicia nuevos page-faults, y por lo tanto, más actividad de paginado.
      * 3) Se decrementa el uso de la CPU.
      * 4) Vuelve a 1).
    - Control de Thrashing:
      * Se puede limitar el Thrashing usando algoritmos de reemplazo local.
      * Con este algoritmo, si un proceso entra en Thrashing no roba frames a otros procesos.
      * Si bien perjudica la performance del sistema, es controlable.
    - Conclusión:
      * Si un proceso cuenta con todos los frames que necesita, no habría Thrashing.
      * Una manera de abordar esta problemática es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad.
      * Otra estrategia con el mismo espíritu es la del algoritmo PFF (Frecuencia de Fallos de Página).
  + El modelo de localidad:
    - Cercanía de referencias o principio de cercanía.
    - Las referencias a datos y programa dentro de un proceso tienden a agruparse.
    - La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento.
    - En cortos periodos de tiempo, el proceso necesita pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos).
    - Un programa se compone de varias localidades.
    - Ejemplo: cada rutina será una nueva localidad: se referencian sus direcciones (cercanas) cuando se está ejecutando.
    - Para prevenir la hiperactividad, un proceso debe tener en memoria sus páginas más activas (menos page faults).
  + El modelo de working set:
    - Se basa en el modelo de localidad.
    - Ventana del working set (Δ): las referencias de memoria más recientes.
    - Working set: es el conjunto de páginas que tiene las más recientes Δ referencias a páginas.
    - La selección del Δ:
      * Δ chico: no cubrirá la localidad.
      * Δ grande: puede tomar varias localidades.
    - Medida del working set:
      * M = cantidad frames disponibles.
      * WSS = medida del working set del proceso p.
      * WSS = D
      * D = demanda total de frames.
      * Si D > m, habrá Thrashing.
  + Prevención de Thrashing:
    - SO monitorea cada proceso, dándole tantos frames hasta su WSS.
    - Si quedan frames, puede iniciar otro proceso.
    - Si D crece, excediendo m, se elige un proceso para suspender, reasignándose sus frames.
    - Así, se mantiene alto el grado de multiprogramación optimizando el uso de la CPU.
  + Problema del modelo del WS:
    - Mantener un registro de los WSS.
    - La ventana es móvil.
  + Prevención del Thrashing por PFF
    - PFF en lugar de calcular el working set de los procesos, utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al Working Set.
      * PFF: frecuencia de page faults.
      * PFF alta 🡪 se necesitan más frames.
      * PFF baja 🡪 los procesos tienen muchas frames asignados.
    - Esquema de PFF
      * Establecer tasa de PF aceptable:
        + Si la tasa actual es baja, el proceso pierde frames.
        + Si la tasa actual es alta, el proceso gana frames.
    - Establecer límites superior e inferior de las PFF´s deseadas.
    - Excede PFF max. 🡪 le doy un frame más.
    - Por debajo del PFF mínimo 🡪 le saco frame.
    - Puede llegar a suspender un proceso si no hay más frames. Sus frames se reasignan a procesos de alta PFF.
  + Demonio de paginación:
    - Proceso creado por el SO durante el arranque que apoya a la administración de la memoria.
    - Se ejecuta cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica.
      * Poca memoria libre.
      * Mucha memoria modificada.
    - Tareas:
      * Limpiar páginas modificadas sincronizándolas con el swap.
      * Reducir el tiempo de swap posterior ya que las páginas están “limpias”.
      * Reducir el tiempo de transferencia al sincronizar varias páginas contiguas.
      * Mantener un cierto número de páginas libre en el sistema.
      * Demorar la liberación de una página hasta que haga falta realmente.
    - Ejemplos: en Linux (proceso “kswapd”). En Windows (proceso “system”).
  + Memoria compartida:
    - Gracias al uso de la tabla de páginas varios procesos puede compartir un marco de memoria; para ello ese marco debe estar asociado a una página en la tabla de páginas de cada proceso.
    - El número de página asociado al marco puede ser diferente en cada proceso.
    - Código compartido:
      * Los procesos comparten una copia de código (solo lectura) por ejemplo editores de texto, compiladores, etc.
      * Los datos son privados a cada proceso y se encuentran en páginas no compartidas.
  + Mapeo de archivo en memoria:
    - Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales.
    - El contenido del archivo no se sube a memoria hasta que se general Page Faults.
    - El contenido de la página que genera el PF es obtenido desde el archivo asociado (no del área de intercambio).
    - Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente.
    - Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el sistema de archivos.
    - Es utilizado comúnmente para asociar librerías compartidas o DLLs.
  + Copia en Escritura:
    - La copia en escritura (Copy-on-Write, COW) permite a los procesos padre e hijo compartir inicialmente las mismas páginas de memoria.
      * Si uno de ellos modifica una página compartida la página es copiada.
    - COW permite crear procesos de forma más eficiente debido a que solo las páginas modificadas son duplicadas.
  + Área de intercambio:
    - Sobre el área utilizada:
      * Área dedicada, separada del sistema de archivos (por ejemplo, en Linux).
      * Un archivo dentro del sistema de archivos (por ejemplo, en Windows).
    - Técnicas para la administración:
      * Cada vez que se crea un proceso se reserva una zona del área de intercambio igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco de su área de intercambio. La lectura se realiza sumando el número de página virtual a la dirección de comienzo del área asignada al proceso.
      * No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria. Problema: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco.
    - Cuando una página no está en memoria, sino en disco, ¿cómo podemos saber en qué parte del área de intercambio esta?
      * El PTE de dicha página tiene el bit V=0 y todos los demás bits sin usar.
    - Área de intercambio – Linux:
      * Permite definir un número predefinido de áreas de Swap.
      * Swap\_info es un arreglo que contiene estas estructuras.
      * Cada área es dividida en un número fijo de slots según el tamaño de la página.
      * Cuando una página es llevada a disco, Linux utiliza el PTE para almacenar 2 valores:
        + En número de área.
        + El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64 gb).
* **INTRODUCCION A IPC**
  + Concurrencia y Paralelismo
    - Es común dividir una solución en diferentes “tareas o programas” que, independientemente o colaborativamente, solucionan el problema.
    - Es común contar con un conjunto de procesadores para ejecutar nuestras soluciones de forma paralela.
    - Necesidades:
      * Comunicar procesos: compartir información entre procesos.
      * Sincronizar procesos: acceso a información compartida.
  + Modelo de procesos concurrentes:
    - Productor: repeat … produce un elemento … enviar elemento a consumidor … until false;
    - Consumidor: repeat … esperar que elemento a consumir … consume elemento … until false;
  + Definición – Condición de carrera:
    - El resultado final depende del orden en que se ejecuten los procesos
  + Definición – Sección Critica:
    - Sección de código en un proceso que accede a recursos compartidos con otros procesos y que NO puede ser ejecutada mientras otro procesos este en su sección critica.
      * Se protegen datos, no código.
      * El SO también presente secciones críticas.
    - Condiciones:
      * **Exclusión Mutua:** dos procesos no pueden estar simultáneamente dentro de sus SC.
      * No se pueden hacer suposiciones en cuanto a velocidades o cantidad de CPUs.
      * Ningún proceso que se ejecute fuera de su SC puede bloquear otros procesos.
      * **Espera Limitada:** ningún proceso tiene que esperar “por siempre” para entrar en su SC.
  + **IPC (Inter-Process Communication):**
    - Mecanismo para comunicar y sincronizar procesos.
    - Consiste de:
      * Semáforos:
        + Es una herramienta de sincronización.
        + Sirve para solucionar el problema de la sección crítica.
        + Sirve para solucionar problemas de sincronización.
        + Es una variable entera: inicializada en un valor no negativo.
        + Dos operaciones:

Wait (también llamadas down o p): decrementa el valor. El proceso no puede comunicar ante un valor negativo, se bloquea.

Signal (también llamadas up o v): incrementa el valor. Desbloqueo de un proceso que espere el semáforo.

* + - * + Operaciones atómicas: cuando un proceso modifica su valor, otros procesos no pueden modificarlo simultáneamente.
      * Sistema de mensajes (pasaje de mensajes):
        + Dos primitivas básicas: send y receive.
        + Se establece un link de comunicación entre dos o más procesos.
        + Ese link puede ser:

Unidireccional o bidireccional.

Simétrico o asimétrico.

* + - * + La operación send puede ser por copia o por referencia.
        + Los mensajes de medida fija o variable.
        + Comunicación directa:

Cada proceso que quiere comunicarse con otro deberá explícitamente indicar quien recibe o manda la comunicación.

Send (P, mensaje) envía un mensaje al proceso P.

Receibe (Q, mensaje) recibe un mensaje desde el proceso Q.

* + - * + Comunicación indirecta:

Usa un mailbox o port.

Un mailbox puede verse como un objeto donde se ponen y sacan mensajes.

Cada mailbox tiene una identificación única.

El sistema operativo debe proveer los mecanismo para que un proceso pueda:

Crear un nuevo mailbox.

Compartir un mailbox.

Enviar y recibir mensajes a través de mailbox.

Destruir mailbox.

Capacidad del link (¿Cuántos mensajes puede mantener el link):

Cero: no puede haber mensajes esperando. Es lo que se llama Rendezvous: el emisor debe esperar que el receptor reciba el mensaje para poder mandar otro. Hay sincronismo.

Capacidad limitada: la cola tiene una longitud finita.

Capacidad ilimitada: tiene una longitud “infinita”. El emisor nunca espera.

* + - * + Emisor y receptor puede ser bloqueantes o no bloqueantes.
        + Caso receptor:

Si el mensaje ya se mandó, lo recibe.

Si no hay mensajes: o se bloquea o continúa sin recepción.

* + - * + Caso emisor:

Si hay un proceso esperando o hay capacidad en el link, envía.

Si no hay un proceso esperando o el link se llenó: o se bloquea o continúa su ejecución sin enviar.

* + - * Memoria compartida:
        + Tradicionalmente cada proceso cuenta con su espacio de direcciones: direcciones virtuales.
        + Un proceso no puede acceder al espacio de otro: protección de la memoria.
        + Los procesos siguen “viendo” un espacio virtual: cada región compartida puede estar en diferente lugar del espacio de direcciones de cada proceso.
        + La técnica permite a dos o más procesos compartir un segmento de memoria.
        + Permite la transferencia de datos entre procesos: comunicación.
        + Se requieren mecanismos de sincronización: semáforos.
* **SUBSISTEMA DE ENTRADA/SALIDA**
  + Interfaz de I/O – Metas:
    - Generalidad:
      * Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada.
      * Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock.
    - Eficiencia:
      * Los dispositivos de I/O puede resultar extremadamente lentos respecto a la memoria y la CPU.
      * El uso de la multiprogramación permite que un proceso espere por la finalización de su I/O mientras que otro proceso se ejecuta.
  + Aspectos de los dispositivos de I/O:
    - Unidad de transferencia:
      * Dispositivos por bloques (discos): operaciones 🡪 read, write, seek.
      * Dispositivos por carácter (keyboards, mouse, serial ports): operaciones 🡪 get, put.
    - Formas de acceso: secuencial o aleatorio.
    - Tipo de acceso: compartido (disco rígido) o exclusivo (impresora).
    - Tipo de acceso: read only (CDROM), write only (pantalla) o read/write (disco).
  + Servicios:
    - Planificación:
      * Organización de los requerimientos a los dispositivos.
      * Ej: planificación de requerimientos a disco para minimizar tiempos.
    - Buffering – Almacenamiento de los datos en memoria mientras se transfieren:
      * Solucionar problemas de velocidad entre los dispositivos.
      * Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos.
    - Caching – Mantener en memoria copia de los datos de reciente acceso para mejorar performance.
    - Spooling – Administrar la cola de requerimientos de un dispositivo:
      * Algunos dispositivos de acceso exclusivo, no puede atender distintos requerimientos al mismo tiempo: por ejemplo, impresora.
      * Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo.
    - Reserva de dispositivos: acceso exclusivo.
    - Manejo de errores:
      * El SO debe administrar errores ocurridos (lectura de un disco, dispositivo no disponible, errores de escritura).
      * La mayoría retorna un número de error o código cuando la I/O falla.
      * Logs de errores.
    - Formas de realizas I/O:
      * Bloqueante: el proceso se suspende hasta que el requerimiento de I/O se completa.
        + Fácil de usar y entender.
        + No es suficiente bajo algunas necesidades.
      * No bloqueante: el requerimiento de I/O retorna en cuanto es posible.
        + Ejemplo: interfaz de usuario que recibe input desde el teclado/mouse y se muestra en el screen.
        + Ejemplo: aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla.
  + Estructuras de datos:
    - El Kernel mantiene la información de estado de cada dispositivo o componente: archivos abiertos, conexiones de red, etc.
    - Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc.
  + Desde el requerimiento de I/O hasta el Hardware:
    - Consideremos la lectura sobre un archivo en un disco:
      * Determinar el dispositivo que almacena los datos (traducir el nombre del archivo en la representación del dispositivo).
      * Traducir requerimiento abstracto en bloques de disco (Filesystem).
      * Realizar la lectura física de los datos (bloques) en la memoria.
      * Marcar los datos como disponibles al proceso que realizo el requerimiento (desbloquearlo).
      * Retornar el control al proceso.
  + Drivers:
    - Contienen el código dependiente del dispositivo.
    - Manejan un tipo dispositivo.
    - Traducen los requerimientos abstractos en los comandos para el dispositivo:
      * Escribe sobre los registros del controlador.
      * Acceso a la memoria mapeada.
      * Encola requerimientos.
    - Comúnmente las interrupciones de los dispositivos están asociadas a una función del driver.
    - Interfaz entre el SO y el HARD.
    - Forman parte del espacio de memoria del Kernel: en general se cargan como módulos.
    - Los fabricantes del hardware implementan el driver en función de una API especificada por el SO: open(), close(), read(), write(), etc.
    - Para agregar nuevo hardware solo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel.
    - Ejemplo en Linux:
      * Linux distingue 3 tipos de dispositivos:
        + Carácter: I/O programa o por interrupciones.
        + Bloque: DMA.
        + Red: ports de comunicaciones.
      * Los drivers se implementan como modulos: se cargan dinámicamente.
      * Debe tener al menos estas operaciones:
        + Init\_module: para instalarlo.
        + Cleanup\_module: para desintalarlo.
      * Operaciones que debe contener para I/O:
        + Open: abre el dispositivo.
        + Reléase: cerrar el dispositivo.
        + Read: leer bytes del dispositivo.
        + Write: escribir bytes en el dispositivo.
        + Ioctl: orden de control sobre el dispositivo.
  + Performance:
    - I/O es uno de los factores que más afectan a la performance del sistema:
      * Utiliza mucho la CPU para ejecutar los drivers y el código del subsistema de I/O.
      * Provoca Context Switches ante las interrupciones y bloqueos de los procesos.
      * Utiliza el bus de memoria en copia de datos:
        + Aplicaciones (espacio usuario) – Kernel.
        + Kernel (memoria física) – controladora.
    - Mejorar la performance:
      * Reducir el número de contexs switches.
      * Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación.
      * Reducir la frecuencia de las interrupciones utilizando:
        + Transferencias de gran cantidad de datos.
        + Controladores más inteligentes.
        + Polling, si se minimiza la espera activa.
      * Utilizar DMA.
* **ARQUITECTURA DE ENTRADA/SALIDA**
  + Variedad en los dispositivos de I/O:
    - Legible para el usuario: usados para comunicarse con el usuario (impresoras, terminales: pantalla, teclado, mouse).
    - Legible para la maquina: utilizados para comunicarse con los componentes electrónicos (discos, cintas, sensores, etc.).
    - Comunicación: usados para comunicarse con dispositivos remotos (líneas digitales, módems, interfaces de red, etc).
  + Problemas que surgen:
    - Amplia variedad:
      * Manejan diferente cantidad de datos.
      * En velocidades diferentes.
      * En formatos diferentes.
    - La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM.
  + Hardware y Software involucrado:
    - Buses.
    - Controladores.
    - Dispositivos.
    - Puertos de E/S – Registros.
    - Drivers.
    - Comunicación con controlador del dispositivo: I/O Programada, interrupciones, DMA.
  + Comunicación: CPU – Controladora
    - ¿Cómo puede la CPU ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo?
      * La controladora tiene uno o más registros: registros para señales de control y registros para datos.
    - La CPU se comunica con la controladora escribiendo y leyendo dichos registros.
  + Comandos de I/O:
    - CPU emite direcciones: para identificar el dispositivo.
    - CPU emite comandos:
      * Control - ¿Qué hacer?: ejemplo, girar el disco.
      * Test – Controlar el estado: ejemplo, power? Error?
      * Read/Write: transferir información desde/hacia el dispositivo.
  + Mapeo de E/S y E/S aislada:
    - Correspondencia en memoria (Memory mapped I/O):
      * Dispositivos y memoria comparten el espacio de direcciones.
      * I/O es como escribir/leer en la memoria.
      * No hay instrucciones especiales para I/O: ya se dispone de muchas instrucciones para la memoria.
    - Isolated I/O (Aislada, uso de puertos de E/S):
      * Espacio separado de direcciones.
      * Se necesitan líneas de I/O. Puertos de E/S.
      * Instrucciones especiales: conjunto limitado.
  + Técnicas de I/O – Programada:
    - CPU tiene control directo sobre la I/O:
      * Controla el estado.
      * Comandos para leer y escribir.
      * Transfiere los datos.
    - CPU espera que el componente de I/O complete la operación.
    - Se desperdician ciclos de CPU.
  + Polling:
    - En la I/O Programada, es necesario hacer polling del dispositivo para determinar el estado del mismo.
      * Listo para recibir comandos.
      * Ocupado.
      * Error.
    - Ciclo de “Busy-wait” para realizar la I/O.
    - Puede ser muy costoso si la espera es muy larga.
  + Técnicas de I/O – manejada por interrupciones:
    - Soluciona el problema de la espera de la CPU.
    - La CPU no repite el chequeo sobre el dispositivo.
    - El procesador continúa la ejecución de instrucciones.
    - El componente de I/O envía una interrupción cuando termina.
  + Técnicas de I/O – DMA (Direct Memory Access):
    - Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo.
    - El procesador es interrumpido luego de que el bloque entero fue transferido.
* **ADMINISTRACION DE ARCHIVOS (FileSystem)**
  + Necesitamos archivos porque:
    - Almacenar grandes cantidades de datos.
    - Tener almacenamiento a largo plazo.
    - Permitir a distintos procesos acceder al mismo conjunto de información.
  + Archivo:
    - Entidad abstracta con nombre.
    - Espacio lógico continuo y direccionable.
    - Provee a los programas de datos (entrada).
    - Permite a los programar guardar datos (salida).
    - El programa mismo es información que debe guardarse.
  + Archivos – Punto de vista del usuario:
    - Que operaciones se pueden llevar a cabo.
    - Como nombrar un archivo.
    - Como asegurar la protección.
    - Como compartir archivos.
    - No tratar con aspectos físicos.
  + Archivos – Punto de vista del diseño:
    - Implementar archivos.
    - Implementar directorios.
    - Manejo del espacio en disco.
    - Manejo del espacio libre.
    - Eficiencia y mantenimiento.
  + Sistema de manejo de archivos:
    - Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos: crear, borrar, buscar, copiar, leer, escribir, etc.
    - Facilita el acceso a los archivos por parte de las aplicaciones.
    - Permite la abstracción al programador, en cuanto al acceso de bajo nivel (el programador no desarrolla el software de administración de archivos).
  + Objetivos del SO en cuanto a archivos:
    - Cumplir con la gestión de datos.
    - Cumplir con las solicitudes del usuario.
    - Minimizar/eliminar la posibilidad de perder o destruir datos: garantizar la integridad del contenido de los archivos.
    - Dar soporte de E/S a distintos dispositivos.
    - Brindar un conjunto de interfaces de E/S para tratamiento de archivos.
  + Tipos de archivos:
    - Archivos regulares:
      * Texto plano: Source File.
      * Binarios: Objetc File y Executable File.
    - Directorios:
      * Archivos que mantienen la estructura en el FileSystem.
  + Atributos de un archivo:
    - Nombre, Identificador, Tipo, Localización, Tamaño, Protección, Seguridad y Monitoreo (Owner, Permisos, Password), (Momento en que el usuario lo modifico, creo, accedió por última vez), (ACLs).
  + Directorios:
    - Contiene información acerca de archivos y directorios que están dentro de él.
    - El directorio es, en sí mismo, un archivo.
    - Interviene en la resolución entre el nombre y el archivo mismo.
    - Operaciones en directorios:
      * Buscar un archivo.
      * Crear un archivo (entrada de directorio).
      * Borrar un archivo.
      * Listar el contenido.
      * Renombrar archivos.
      * Etc.
    - El uso de los directorios ayuda con:
      * La eficiencia: localización rápida de archivos.
      * Uso del mismo nombre de archivo: diferentes usuarios puede tener el mismo nombre de archivo.
      * Agrupación: agrupación lógica de archivos por propiedades/funciones: ejemplo, programas Java, juegos, librerías, etc.
    - Estructura de directorios:
      * Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias (**full pathname** del archivo o **PATH absoluto**).
      * Distintos archivos puede tener el mismo nombre pero el fullpathname es único.
      * El directorio actual se lo llama “directorio de trabajo (working directory).
      * Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por **PATH absoluto** como por su **PATH relativo** indicando solamente la ruta al archivo desde el directorio de trabajo.
  + Identificación absoluta y relativa:
    - Tanto en archivos como directorios se pueden identificar de manera:
      * Absoluta: el nombre incluye todo el camino del archivo.
        + /var/www/index.html
        + C:\windows\winhelp.exe
      * Relativa: el nombre se calcula relativamente al directorio en el que se esté.
        + Si estoy en el directorio /var/spool/mail
        + Entonces es: ../../www/index.html
  + Compartir archivos:
    - En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos.
    - Debe ser realizado bajo un esquema de protección:
      * Derechos de acceso.
      * Manejo de accesos simultáneos.
  + Protección:
    - El propietario/administrador debe ser capaz de controlar:
      * Que se puede hacer: derechos de acceso.
      * Quien lo puede hacer.
    - Derechos de acceso:
      * Los directorios también tienen permisos, los cuales pueden permitir el acceso al mismo para que el usuario pueda usar el archivo siempre y cuando tenga permisos.
      * Execution: el usuario puede ejecutar.
      * Reading: el usuario puede leer el archivo.
      * Appending: el usuario puede agregar datos pero no modificar o borrar el contenido del archivo.
      * Updating: el usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos.
      * Changing protection: el usuario puede modificar los derechos de acceso.
      * Deletion: el usuario puede borrar el archivo.
      * Owners (propietarios): tiene todos los derechos. Puede dar derechos a otros usuarios. Se determinan clases (usuario específico, grupos de usuarios y todos (archivos públicos).
  + Metas del Sistema de Archivos:
    - Brindar espacio en disco a los archivos de usuario y del sistema.
    - Mantener un registro del espacio libre. Cantidad y ubicación del mismo dentro del disco.
  + Conceptos:
    - Sector: unidad de almacenamiento utilizada en los discos rígidos.
    - Bloque/Cluster: conjunto de sectores consecutivos.
    - File System: define la forma en que los datos son almacenados.
    - FAT (File Allocation Table): contiene información sobre en qué lugar están alocados los distintos archivos.
  + Pre-asignación:
    - Se necesita saber cuánto espacio va a ocupar el archivo en el momento de su creación.
    - Se tiene a definir espacios mucho más grandes que lo necesario.
    - Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo.
  + Asignación dinámica:
    - El espacio se solicita a medida que se necesita.
    - Los bloques de datos puede quedar de manera no contigua.
  + Formas de asignación:
    - Continua:
      * Conjunto continuo de bloques son utilizados.
      * Se requiere una pre-asignación: se debe conocer el tamaño del archivo durante su creación.
      * File Allocation Table (FAT) es simple: solo una entrada que incluye Bloque de inicio y longitud.
      * El archivo puede ser leído con una única operación.
      * Puede existir fragmentación externa: compactación.
      * Problemas de la técnica: encontrar bloques libres continuos en el disco, e incremento del tamaño de un archivo.
    - Encadenada:
      * Asignación en base a bloques individuales.
      * Cada bloque tiene un puntero al próximo bloque del archivo.
      * File Allocation Table (FAT): única entrada por archivo: Bloque de inicio y tamaño del archivo.
      * No hay fragmentación externa.
      * Útil para acceso secuencial (no ramdon).
      * Los archivos puede crecer bajo demanda.
      * No se requieren bloques contiguos.
      * Se pueden consolidar los bloques de un mismo archivo para garantizar cercanía de los bloques de un mismo archivo.
    - Indexada:
      * Asignación en base a bloques individuales.
      * No se produce fragmentación externa.
      * El acceso “random” a un archivo es eficiente.
      * File Allocation Table (FAT): única entrada con la dirección del bloque de índices (index node / i-node).
      * Variante: asignación por secciones.
      * Variante: niveles de indirección.
  + Gestión de espacio libre:
    - Control sobre cuáles de los bloques de disco están disponibles.
    - Alternativas: tablas de bits, bloques libres encadenados, indexación.
    - Tabla de bits:
      * Tabla (vector) con 1 bit por cada bloque de disco.
      * Cada entrada: 0 = bloque libre, 1 = bloque en uso.
      * Ventaja: fácil encontrar un bloque o grupo de bloques libres.
      * Desventaja: tamaño del vector en memoria. Tamaño disco bytes / tamaño bloque en sistema archivo. Eje: disco 16Gb con bloques de 512 bytes 🡪 32 Mb.
    - Bloques encadenados:
      * Se tiene un puntero al primer bloque libre.
      * Cada bloque libre tiene un puntero al siguiente bloque libre.
      * Ineficiencia para la búsqueda de bloques libres 🡪 hay que realizar varias operaciones de E/S para obtener un grupo libre.
      * Problemas con la pérdida de un enlace.
      * Difícil encontrar bloques libres consecutivos.
    - Indexación (o agrupamiento):
      * Variante de “bloques libres encadenados”.
      * El primer bloque libre contiene las direcciones de N bloques libres.
      * Las N-1 primeras direcciones son bloques libres.
      * La N-ésima dirección referencia otro bloque con N direcciones de bloques libres.
    - Recuento:
      * Variante de indexación.
      * Esta estrategia considera las situaciones que varios bloques contiguos puede ser solicitados o liberados a la vez (en especial con asignación contigua).
      * En lugar de tener N direcciones libres (índice) se tiene:
        + La dirección del primer bloque libre.
        + Los N bloques libres contiguos que le siguen. (#bloque, N siguientes bloques libres).
  + UNIX – Manejo de archivos:
    - Tipos de archivos:
      * Archivo común.
      * Directorio.
      * Archivos especiales (dispositivos /dev/sda).
      * Named pipes (comunicación entre procesos).
      * Links (comparten el i-nodo, solo dentro del mismo filesystem).
      * Links simbólicos (para filesystems diferentes).
    - Estructura del volumen:
      * Boot block: código para bootear el SO.
      * Superblock: atributos sobre el File System – Bloques/Clusters libres.
      * I-NODO table: tabla que contiene todos los i-nodos: el I-NODO es una estructura de control que contiene información clave de un archivo.
      * Data Blocks: bloques de datos de los archivos.
  + Windows – File Systems Soportados:
    - CD-ROM Fyle System (CDFS).
    - Universal Disk Format (UDF).
    - File Allocation Table: FAT12, FAT16, FAT32.
    - New Technology File System (NTFS).
    - Windows FAT:
      * FAT es un sistema de archivos utilizado originalmente por DOS y Windows 9x.
      * Windows soporta FAT file systems por:
        + Por compatibilidad con otro SO en sistemas multiboot.
        + Para permitir upgrades desde versiones anteriores.
        + Para formato de dispositivos como diskettes.
      * Las distintas versiones de FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters: FAT12, FAT16, FAT32.
      * Se utiliza un mapa de bloques del sistema de archivos, llamado FAT.
      * La FAT tiene tantas entradas como bloques.
      * La FAT, su duplicado y el directorio razin se almacenan en los primeros sectores de la partición.
      * Se utiliza un esquema de asignación encadenada.
      * La única diferencia es que el puntero al próximo bloqu esta en la FAT y no en los bloques.
      * Bloques libres y dañados tienen códigos especiales.
    - FAT12:
      * El identificador de cluster de 12 bits FAT12 limita una partición para almacenar un máximo de 212 (4096) grupos.
        + Windows utiliza tamaños de cluster de 512 bytes a 8 KB de tamaño, lo que limita un FAT12 tamaño de volumen a 32MB.
        + Windows usa FAT12 como el formato para todos disquetes de 5 pulgadas y disquetes de 3.5 pulgadas y discos que almacenan hasta 1,44 MB de datos.
    - FAT16:
      * Con un identificador de cluster de 16 bits, puede abordar 216 (65536) grupos.
        + En Windows el rango de tamñaos de cluster FAT16 son 512 bytes (el tamaño del sector) a 64 KB, que limieta los tamaños de volumen FAT16 a 4GB.
        + El tamaño del cluster que usa Windows depende del tamaño de un volumen.
    - FAT32:
      * Es el más reciente basado en FAT formato del sistema de archivos.
      * FAT32 utiliza identificadores de cluster de 32 bits pero reserva los 4 bits altos, por lo que en efector tiene identificadores de 28 bits.
        + Porque los tamaños de cluster FAT32 pueden ser tan grandes como 32 KB, FAT32 tiene una capacidad teórica para abordar 8TB volúmenes.
        + Los números de cluster de mayor potencia de FAT32 lo permiten administrar discos de manera más eficiente que FAT16; puede manejar volúmenes de hasta 128GB con 512 bytes.
  + Windows – NTFS:
    - NTFS es el filesystems nativo de Windows.
    - NTFS usa 64-bit para referenciar clusters: teóricamente permite tener volúmenes de hasta 16 Exabytes (16 billones de GB).
    - ¿Por qué usar NTFS en lugar de FAT?: FAT es simple, más rápido para ciertas operaciones, pero NTFS soporta:
      * Tamaños de archivo y de discos mayores.
      * Mejora performance en discos grandes.
      * Nombres de archivos de hasta 255 caracteres.
      * Atributos de seguridad.
      * Transaccional.
* **Cache de disco**
  + Disk Cache:
    - Buffers en memoria principal para almacenamiento temporario de bloques de disco.
    - Objetivo: minimizar la frecuencia de acceso al disco.
  + Algunas observaciones:
    - Cuando un proceso quiere acceder a un bloque de la cache hay dos alternativas:
      * Se copia el bloque al espacio de direcciones del usuario.
      * Se trabaja como memoria compartida (no se copia permitiendo acceso a varios procesos).
  + Estrategia de reemplazo:
    - Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado.
    - Es una lista de bloques, donde el último es el más recientemente usado (LRU, Least Recently Used).
    - Cuando un bloque se referencia o entra en la cache queda al final de la lista.
    - No se mueven los bloques en la memoria: se asocian punteros.
    - Otra alternativa: Least Frecuently User. Se reemplaza el que tenga menor número de referencias.
* **Buffer Cache – Unix System V**
  + Objetivo y estructura:
    - Minimizar la frecuencia de acceso a disco.
    - Es una estructura formada por buffers.
    - El Kernel asigna un espacio en la memoria durante la inicialización para esta estructura.
    - Un buffer tiene dos partes: el header y el lugar donde se almacena el bloque de disco traído a memoria.
  + El header:
    - Identifica por número de dispositivo y numero de bloque.
    - Estado.
    - Punteros a:
      * 2 punteros para la hash queue.
      * 2 punteros para la free list.
      * 1 puntero al bloque en memoria.
  + Estados de los buffers:
    - Free o disponible.
    - Busy o no disponible (en uso por algún proceso).
    - Se está escribiendo o leyendo del disco.
    - Delayed Write (DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original en disco.
  + Free List:
    - Organiza los buffers disponibles para ser utilizados para cargar nuevos bloques de disco.
    - No necesariamente los buffers están vacios.
    - Se ordena según LRU (Least Recent Used).
  + Hash Queues:
    - Son colas para optimizar la búsqueda de un buffer en particular.
    - Se organizan según una función de hash usando (dispositivo, #bloque).
  + Funcionamiento del buffer cache:
    - Cuando un proceso quiere acceder a un archivo, utiliza su i-nodo para localizar los bloques de datos donde se encuentra éste.
    - El requerimiento llega al buffer cache quien evalua si puede satisfacer el requerimiento o si debe realizar la E/S.
    - Se puede dar 5 escenarios (los mismos para algoritmo de asignación):
      * 1) El Kernel encuentra el bloque en la hash queue y el buffer está libre.
      * 2) El Kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
      * 3) Idem 2, pero el bloque libre está marcado como DW.
      * 4) El Kernel no encuentra el bloque en la hash queue y la free list esta vacia.
      * 5) El Kernel encuentra el bloque en la hash queue pero está BUSY.
* **Threads**
  + Analicemos estas situaciones:
    - Procesador de texto: ingreso de caracteres, auto-guardado, análisis ortográfico/gramatical.
    - Aplicaciones que muestran una animación, o un gráfico a medida que se ingresan datos.
    - Acceso simultaneo a diferentes fuentes de E/S.
    - Tendencia de los procesadores actuales a contar con varios núcleos (multiprocesadores).
  + Los lenguajes de programación:
    - Brindan herramientas que nos permiten separar las diferentes “tareas” de los programas en unidades de ejecución diferentes:
      * Java – heredar de “Thread”, implementar la interface “Runnable”.
      * Delphi – heredar de “Thread”.
      * C#, C, etc.
      * Ruby – Thread.new {código}.
      * PHP – Heredar de Thread.
      * Javascript – HTML5 Web Workers.
  + Primero SO – Procesos:
    - Programa en ejecución.
    - Unidad de asignación de los recursos.
    - Conceptos relacionados con proceso:
      * Espacio de direcciones.
      * Punteros a los recursos asignados (Stacks, archivos, etc.).
      * Estructuras asociadas: PCB, tablas.
    - Único hilo de ejecución por proceso.
  + SO actuales – Threads:
    - Unidad básica de utilización de CPU.
    - Proceso:
      * Espacio de direcciones.
      * Unidad de propiedad de recursos.
      * Conjunto de threads (eventualmente uno).
    - Thread:
      * Unidad de contexto (hilo de ejecución).
      * Contexto del procesador.
      * Stacks de usuario y Kernel.
      * Variables propias.
      * Acceso a la memoria y recursos del proceso.
  + Procesos e Hilos:
    - ¿Por qué dividir una aplicación en threads?:
      * Respuestas percibidas por los usuarios, paralelismo/ejecución en background.
        + Ejemplo: el servicio de impresión de Word ejecuta en background y nos permite seguir editando.
      * Aprovechar las ventajas de múltiples procesadores.
        + Con n CPUs puede ejecutarse n threads al mismo tiempo.
      * Características complejas:
        + Sincronización.
        + Escalabilidad: una cantidad de threads por proceso excesiva implica más cambios de contexto entre hilos del mismo proceso.
  + Ventajas:
    - Sincronización de procesos.
    - Mejorar tiempos de respuesta.
    - Compartir recursos.
    - Economía.
    - Analicemos uso de RPC, o servidor de archivos.
  + Algunos conceptos:
    - Hyper Threading:
      * Permite al software programa para ejecutar multiples hilos (multi-threaded) procesar los hilos en paralelo dentro de un único procesador.
      * Simular dos procesadores lógicos dentro de un único procesador físico:
        + Duplica solo algunas “secciones” de un procesador: registros de control (MMU, Interrupciones, Estado, etc) y registros de propósito general (AX, BX, PC, Stack, etc.).
      * Resultado: mejoraría en el uso del procesador (entre 20% y 30%).
    - Sistemas Dual-Core: una CPU con dos cores por procesador físico. Un circuito integrado tiene 2 procesadores completos. Los 2 procesadores combinan cache y controlador.
    - Sistemas Dual-processor (DP): tiene 2 procesadores físicos en el mismo chasis. Pueden estar en la misma motherboard o no. Cache y controlador independientes.
    - En ambos casos, las APIC (Advanced Programmable Interrupt Controllers) están separadas por procesador. De esta manera proveen administración de interrupciones x procesador.
  + Estructuras de un hilo:
    - Cada hilo dentro de un proceso contara con:
      * Un estado de ejecución.
      * Un contexto de procesador.
      * Una pila en modo usuario y otra en modo supervisor.
      * Almacenamiento para variables locales.
      * Acceso a memoria y recursos del proceso (archivos abiertos, señales, además de la parte de código y datos) que compartirá con el resto de los hilos.
    - La estructura de un hilo está constituida por:
      * Program Counter.
      * Un conjunto de registros.
      * Un espacio de stack.
  + Análisis en hilos de:
    - Context Switch.
    - Creación.
    - Destrucción.
    - Planificación.
    - Protección.
  + Estados de un Thread:
    - Ejecución, Listo, Bloqueado.
    - Planificación: sobre los threads.
    - Eventos sobre procesos afectan todos sus Threads.



* + Tipos – Usuario – ULT:
    - La aplicación se encarga de la gestión.
    - El Kernel no se entera que hay hilos.
    - Bibliotecas:
      * Crear, destruir, planificar, etc.
    - Ejemplos: Java VM, POSIX Threads, Solaris Threads.
    - Ventajas:
      * Intercambio entre hilos.
      * Planificación independiente.
      * Portabilidad (no depende del SO multithreading).
    - Desventajas:
      * Bloqueo del proceso durante una System Call.
      * No se puede multiplexar en distintos procesadores.
      * No hay protección entre hilos.
  + Tipos – Kernel – KLT:
    - El Kernel se encarga de la gestión y planificación.
    - Ventajas:
      * Se puede multiplexar el proceso en distintos procesadores.
      * Independencia de bloqueos entre Threads de un mismo proceso.
    - Desventajas:
      * Cambios de modo de ejecución en el switch entre hilos del mismo proceso.
      * La creación y administración de los KLTs es más lenta que los ULTs.
    - Ejemplos: Windows NT/2000 y Linux.
  + Tipos de Threads – Combinaciones:
    - Es posible combinar ULT y KLT.
    - ULT: planificación y sincronización.
    - LWP – Lightweight Process: asocia ULT y KLT.
    - Este enfoque aprovecha las ventajas de ambos tipos.
  + Modelos de Multithreading:
    - Relación entre ULT y KLT.
    - Tipos:
      * Uno a uno:
        + Cada ULT mapea con un KLT.
        + Cuando se necesita un ULT se debe crear un KLT.
        + Ejemplos: Windows – OS/2.
      * Muchos a uno:
        + Muchos ULT mapean a un único KLT.
        + Usado en sistemas que no soportan KLT.
        + Si se bloquea un ULT, se bloquea el proceso.
        + Java sobre un sistema no soporta KLT.
      * Muchos a muchos:
        + Muchos ULT mapean a muchos KLT.
        + Ejemplo: Solaris.