***INTRODUCCIÓN***

**GNU/Linux (GNU No es Unix)** es un sistema operativo tipo UNIX, pero libre. Es un sistema operativo diseñado por miles de programadores. También es un sistema libre y gratuito para todo el que quiera utilizarlo. Existen diversas distribuciones, que incluyen más cosas que el sistema Linux básico. sualmente incluyen también utilidades adicionales de gestión e instalación del sistema, así como paquetes precompilados y listos para instalar muchas de las herramientas comunes de UNIX. (Red Hat, Debian, etc).

Como una característica fundamental es que GNU/Linux es de código abierto, lo que nos permite estudiarlo, personalizarlo, auditarlo, aprovechar la documentación…..

**Breve historia:** Iniciado por Richard Stallman en 1983 con el fin de crear un Unix Libre. Para asegurar que el mismo fuera libre, se necesitó crear un marco regulatorio conocido como GPL (General Public License de GNU). En 1985, Stallman crea la FSF (Free Software Foundation), con el fin de financiar el proyecto GNU. En 1990, GNU ya contaba con un editor de texto (EMACS), un compilador (GCC) y gran cantidad de bibliotecas que componen n Unix típico. Pero aún faltaba el componente principal que era el **Núcleo**

Linus Torvalds ya venía trabajando desde 1991 en un Kernel denominado Linux, el cual se distribuirá bajo licencia GPL (General Public Licence). En el año 1992, Torvalds y Stallman deciden fusionar ambos proyectos, y es allí donde nace GNU/Linux. GNU/Linux pertenece al desarrollo del software libre.

**Características generales de GNU/Linux:**

1. Es multiusuario
2. Es multitarea y multiprocesador
3. Es altamente portable
4. Posee diversos intérpretes de comandos, de los cuales algunos son programables
5. permite el manejo de usuarios y permisos
6. Todo es un archivo (hasta los dispositivos y directorios)
7. Cada directorio puede estar en una partición diferente (ejemplo: /tpm, /home, etc)
8. Es case sensitive
9. Es de código abierto

**Kernel (estructura básica):**

1. Núcleo
2. Ejecuta programas y gestiona dispositivos de hardware
3. Es el encargado de que el software y el hardware puedan trabajar juntos
4. Sus funciones más importantes son la administración de memoria y CPU
5. En sí, y en un sentido estricto, es el SO
6. Es un núcleo monolítico híbrido:
   1. Los drivers y el código del Kernel se ejecuta en modo privilegiado
   2. Lo que lo hace híbrido es la posibilidad de cargar y descargar funcionalidad a través de módulos
7. Licenciado bajo licencia GPL v2

**Intérprete de Comandos:**

1. CLI (Command Line Interface)
2. Modo de comunicación entre el usuario y el SO
3. Ejecuta programas a partir del ingreso de comandos
4. Cada usuario tiene una interfaz o shell (Bourne Shell - korn Shell - Bourne Again Shell (bash))

**Sistema de Archivos:**

1. Organiza la forma en que se almacenan los archivos en dispositivos de almacenamiento (fat, ntfs, ext2, ext3, reiser..)
2. GNU/Linux usa el Extended (v2, v3, v4)
3. Hace tiempo se está debatiendo en reemplazarlo por el Btrfs (B-tree FS) de Oracle
4. Los Directorios más importantes según FHS
   1. **/ →** tope de la estructura de directorios
   2. **/home →** se almacenan archivos de usuarios
   3. **/var →** información que varía de tamaño
   4. **/etc →** Archivos de configuración
   5. **/bin →** Archivos binarios y ejecutables
   6. **/dev →** Enlace a dispositivos
   7. **/usr →** Aplicaciones de usuarios
   8. /etc/passwd/ → información del usuario. Modificado sólo por el root
   9. /etc/shadow/ → contraseñas encriptadas de codificación irreversible del usuario

**MBR**

1. Sector reservado del disco físico (cilindro 0, cabeza 0, sector 1)
2. Existe un MBR en todos los discos
3. Si existiese más de un disco rígido en la máquina, sólo uno es designado como Primary Master Disk
4. El tamaño del MBR coincide con el tamaño estándar de sector: 512 Bytes
   1. Los primeros bytes corresponden al Master Boot Code (MBC)
   2. A partir del byte 466 está la tabla de particiones. Es de 64 bytes
   3. al final existen 2 bytes libres

**MBC**

1. El MBC es un pequeño código que permite arrancar el SO
2. La última acción del BIOS es leer el MBC. Lo lleva a memoria y lo ejecuta.
3. El proceso de inicio de un máquina y carga del sistema operativo se denomina proceso de bootstrap
4. Si se tiene un sistema instalado → MBC típico. Sino → uno diferente (boteadores)
5. El proceso de booteo puede verse como una serie de pequeños programas cuya ejecución se va encadenando
6. La finalidad del Bootloader es la de cargar una imagen de Kernel de alguna partición para su ejecución.

**Particiones**

1. Es una forma de dividir lógicamente el disco físico.
2. Es una buena práctica separar los datos del usuario de las aplicaciones y/o Sistemas Operativos instalados
3. Tener una partición de Restore de todo el sistema
4. Poder ubicar el Kernel en una partición de solo lectura, o una que ni siquiera se monta (no está disponible a los usuarios)
5. Debido al tamaño acotado en el MBR para la tabla de particiones se restringe a 4 la cantidad de particiones primarias, o 3 particiones primarias y una extendida con sus volúmenes o particiones lógicas
   1. Una de las 4 particiones puede ser extendida, la cual se subdivide en volúmenes lógicos
   2. Partición primaria: división cruda del disco, se almacena información de la misma en el MBR
   3. Partición extendida: sirve para contener unidades lógicas en su interior. Solo puede existir una partición de este tipo por disco. No se define un tipo de FS directamente sobre ella.
   4. Partición lógica: ocupa la totalidad o parte de la partición extendida y se le define un tipo de FS directamente
6. Cómo mínimo es necesario una partición para el **/**
7. Es recomendable crear al menos 2 (**/ y SWAP)**
8. Particionador destructivo : **fdisk**
9. Particionador no-destructivo: **fips - gparted - interfaz de instalación (distribuciones)**

**Gestor de Arranque:**

1. Permite la carga del sistema operativo
2. Se ejecuta lego del código de la BIOS
3. Existen 2 modos de instalación
   1. en el MBR
   2. En el sector de arranque de la partición raíz o activa
4. **GRUB - LILO - NTLDR - GAG**

**Proceso de arranque:**

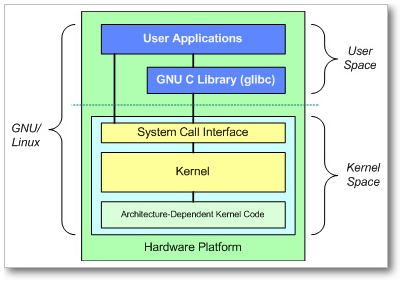
1. Se denomina bootstrap - carga
2. En las arquitecturas x86 el BIOS es el responsable de iniciar la carga del SO a través del MBC
3. Carga el programa de booteo desde el MBR
4. El gestor de arranque lanzado desde el MBC carga el Kernel
   1. Prueba e inicializa los dispositivos
   2. Luego pasa el control al proceso **init**

**Modos de Ejecución:**

1. Modo Kernel
   1. Modo privilegiado
   2. Manejo estricto de pautas de confiabilidad/seguridad
   3. Manejo de:
      1. CPU, memoria, I/O
      2. Administración multiprocesador, diagnósticos, testing
      3. Partes del FS y la interfaz de red
2. Modo User
   1. Más flexible
   2. Funciones de mantenimiento más simples, debugging
      1. Compiler, assembler, interpreter, linker/loader
      2. FS management, telecommunication, network management
      3. Editors, spreadsheets, user applications

**System Calls**

Las llamadas al sistema proporcionan una interfaz con la que poder invocar los servicios que el sistema operativo ofrece.



Por ejemplo de System Calls tenemos: **open - read - fork - mmap - kill - time**

La llamada al sistema la invoca un proceso de usuario (o mejor dicho un proceso en modo usuario) y es servida por el núcleo (kernel). Una llamada al sistema implica pasar o salta del código usuario al código del núcleo. Este salto conlleva un cambio en el modo del funcionamiento del procesador. El procesador debe pasar de modo usuario a modo supervisor o privilegiado (modo kernel)

Categorías:

1. Control de procesos
2. Manejo de archivos
3. Manejo de dispositivos
4. Mantenimiento de información del sistema
5. Comunicaciones

***PROCESOS***

Un proceso simplemente es un programa en ejecución. Un proceso es un concepto manejado por el sistema operativo que consiste en el conjunto formado por:

1. Las instrucciones de un programa destinadas a ser ejecutadas por el microprocesador
2. Su estado de ejecución en un momento dado, eso es, los valores de los registros de la CPU para dicho programa.
3. Su memoria de trabajo, es decir, la memoria que ha reservado y sus contenidos
4. Otra información que permite al sistema operativo su planificación

En un sistema linux, que como ya sabemos es multitarea (sistema operativo multihilo), se pueden estar ejecutando distintas acciones a la par, y cada acción es un proceso que consta de uno o más hilos, memoria de trabajo compartida por todos los hilos e información de planificación. Cada hilo consta de instrucciones y estado de ejecución.

Cuando ejecutamos un comando en el shell, ss instrucciones se copian en algún sitio de la memoria RAM del sistema para ser ejecutadas. Cuando las instrucciones ya cumplieron su cometido, el programa es borrado de la memoria del sistema, dejándola libre para que más programas se pueden ejecutar a la vez. Por tanto cada uno de estos programas son los procesos.

Los procesos son creados y destruidos por el SO, pero lo hace a petición de otros procesos. El mecanismo por el cual un proceso crea otro proceso se denomina bifurcación o FORK. Los nuevos procesos son independientes y no comparten memoria (es decir, información) con el proceso que los ha creado.

En definitiva, es posible crear tantos hilos como procesos. La diferencia estriba en que un proceso solamente puede crear hilos para sí mismo y en que dichos hilos compartes toda la memoria reservada para el proceso.

**Process ID**

Al crearse un nuevo proceso se le asigna un identificador de proceso único (PID). Este número debe utilizarse por el administrador para referirse a un procesos dado al ejecutar un comando. Los PID son asignados por el sistema a cada nuevo proceso en orden creciente comenzando desde cero. Si antes de un reboot del sistema se llega el n´mero máximo, se vuelve a comenzar desde cero, salteando los procesos que aún siguen activos.

**Parent Process ID (PPID)**

La creación de nuevos procesos en Unix se realiza por la vía de duplicar un proceso existente invocando al comando fork(). Al proceso original se le llama padre y al nuevo proceso hijo. El PPID de un procesos es el PID de su proceso padre

**Atributos de un procesos**

1. PID
2. PPID
3. UID
   1. User ID del proceso identifica al creador del proceso. Este usuario y root son los únicos que pueden modificar el proceso
4. EUID
   1. El Effective User ID determina si el proceso tiene permiso para acceder a archivos y otros recursos del sistema.
5. GID y EGID : grupo de usuario que lanzó el proceso.

**Hilos**

Los hilos son similares a los procesos ya que ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. Los hilos son una forma de dividir un programa en dos o más tareas que corren simultáneamente, compitiendo en algunos casos, por la CPU.

La diferencia más significativa entre los procesos y los hilos, es que los primeros son típicamente independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, los hilos generalmente comparten la memoria, por lo que no necesitan costosos mecanismos de comunicación para sincronizarse.

|  |  |
| --- | --- |
| PROCESOS | HILOS |
| Manejados por el SO | Manejado por los procesos |
| Independientes de otros procesos | Relacionados con otros hilos del mismo proceso |
| Memoria privada, se necesitan mecanismo de comunicación para compartir información | Memoria compartida con el resto de hilos que forman el proceso |

**Características de proceso**

1. Es un programa en ejecución
2. Tarea - Job - Proceso
3. Es dinámico
4. Tiene PC
5. Su ciclo de vida comprende desde que se lo dispara hasta que termina

**Componentes de un proceso**

1. Sección de código (texto)
2. Sección de datos (variables globales)
3. Stacks (datos temporarios: parámetros, variables temporales y dirección de retorno)

**Stacks**

1. Un proceso cuenta con 1 o más stacks
   1. En general: Usuario y Kernel
2. Se crea automáticamente y su medida se ajusta en run-time
3. Está formado por stack frames que son pushed (al llamar a la rutina) y popped (cuando se retorna de ella)
4. El stack frame tiene los parámetros de la rutina (variables locales), y datos necesarios para recuperar el stack frame anterior (el contador de programa y el valor del stack pointer en el momento del llamado)

**Atributos de un Proceso:**

1. Identificación del proceso, y del proceso padre
2. Identificación del usuario que los disparó
3. Si hay estructura de grupos, grupo que lo disparó
4. En ambientes multiusuario, desde que terminal y quién lo ejecutó

**Estructura asociada a un Proceso (PCB - Process Control Block)**

1. Estructura asociada al proceso. Una por proceso
2. Contiene la información asociada con cada proceso:
   1. estado
   2. pc
   3. valores de registros de la CPU
3. Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina
4. Información relacionada con
   1. Planificación
   2. Ubicación en memoria
   3. Accounting
   4. Estado de I/O

**Estructura de Datos asociada a un proceso - Vista según SO**

1. **Identificar**
   1. Un identificador único asociado a este proceso, para distinguirlo del resto de los procesos
2. **Estado**
   1. Si el proceso está actualmente corriendo, está en estado de ejecución.
3. **Prioridad**
   1. Nivel de prioridad relativo a los otros procesos
4. **Contador de programa**
   1. La dirección de la siguiente instrucción del programa que se ejecutará
5. **Punteros a memoria**
   1. Incluye los punteros al código de programa y los datos asociados a dicho procesos además de cualquier bloque de memoria compartido con otros procesos.
6. **Datos de contexto**
   1. Datos que están presentes en los registros del procesador cuando el proceso está corriendo
7. **Información de estado E/S**
   1. Incluye las peticiones de E/S pendientes, dispositivo de E/S asignado a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
8. **Información de auditoría**
   1. Puede incluir la cantidad de tiempo del procesador y de tiempos de reloj utilizados, etc.

**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)**

1. Se produce cuando la CPU cambia de un proceso a otro
2. Se debe resguardar info del proceso saliente, que pasa a espera y retornará después la CPU
3. Se debe cargar la información asociada al nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada
4. Es tiempo no productivo de CPU
5. El tiempo que consume depende del soporte de HW

**Espacio de Direcciones de un Proceso:**

1. Es el conjunto de direcciones de memoria que ocupa el proceso
2. No incluye su PCB o tablas asociadas
3. Un proceso en modo usuario puede acceder sólo a su espacio de direcciones; en modo kernel, en el contexto de un proceso, se puede acceder a estructuras internas o a espacios de direcciones de otros procesos.

**Estados de un Proceso**

En su ciclo de vida, el proceso pasa por diferentes estados:

1. Nuevo (new)
   1. Es un proceso que se acaba de crear y aún no ha sido admitido en el grupo de procesos ejecutables por el SO. Típicamente se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su PCB si ha sido creado.
2. Listo para ejecutar (ready)
   1. Un proceso se prepara para ejecutar cuando así se disponga según la política que se esté implementando
3. Ejecutándose (running)
   1. El procesos está actualmente en ejecución. Se asume una computadora con un único procesador, con lo cual solo un proceso puede estar en ejecución en un instante dado.
4. En espera (waiting)
   1. Un proceso que no puede ejecutarse hasta que no se complete un evento determinado como lo podría ser una operación de E/S
5. Terminado (terminated)
   1. Es un procesos que ha sido liberado del grupo de proceso ejecutables por el SO, debido a que ha sido detenido o que ha sido abortado por alguna razón.

**Colas en la planificación de procesos**

1. Se enlazan las PCBs
2. Ejemplos
   1. De Trabajos o procesos
      1. Contiene todos los procesos en el sistema
   2. De Procesos listos
      1. Residentes en memoria principal esperando para ejecutarse
   3. De dispositivos
      1. Esperando por un dispositivo de I/O

**Módulos de la planificación**

1. Son módulos del SO que realizan distintas tareas asociadas a la planificación
2. Se ejecutan ante aquellos eventos que así lo requieren:
   1. Creación/Terminación de un proceso
   2. Eventos de Sincronización o de E/S
   3. Finalización de lapso de tiempo
   4. ETC

**Módulos de la planificación**

1. Scheduler de long term
2. Scheduler de short term
3. Scheduler de medium term
4. Otros módulos
   1. dispatcher
   2. loader
5. Pueden no existir como módulos separados de los schedulers vistos, pero la función debe cumplirse
   1. Dispatcher: hace cambio de contexto, cambio de modo de ejecución...“despacha” el proceso elegido por el short term (salta a la instrucción a ejecutar)
   2. Loader: carga en memoria el proceso elegido por el long term

**Dispatcher**

Es el planificador de corto plazo, donde su objetivo es asignar la CPU a uno de los procesos ejecutables del sistema, para lo cual sigue un determinado algoritmo o política de atención.

**Long term Scheduler**

Este planificador se encarga de las transiciones de los procesos en estado de Nuevo a Listo, es decir, cuando comienzan a competir por el CPU. También se encarga de llevar los procesos finalizados nuevamente a memoria secundaria

1. Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria
2. Puede no existir este scheduler y absorber esta tarea el de short term

**Medium Term Scheduler (swapping)**

Este planificador trata los casos en que los procesos residentes en memoria principal son demasiados, con lo cual puede darse el caso de que los procesos que ocupan memoria principal están bloqueados desperdiciando así el CPU. Es así que en algunos sistemas se intercambian procesos enteros (swap) entre memoria principal y secundaria.  
  
También es el encargado de regir las transiciones de los procesos entre memoria principal y secundaria, actúa intentado maximizar la utilización de los recursos. Por ejemplo: transfiriendo siempre a memoria secundaria procesos bloqueados o transfiriendo a memoria principal procesos bloqueados únicamente por no tener memoria.

1. Si es necesario, reduce el grado de multiprogramación
2. Saca temporariamente de memoria los procesos que sea necesario para mantener el equilibrio del sistema.
3. Términos asociados: swap out (sacar de memoria), swap in (volver a memoria)

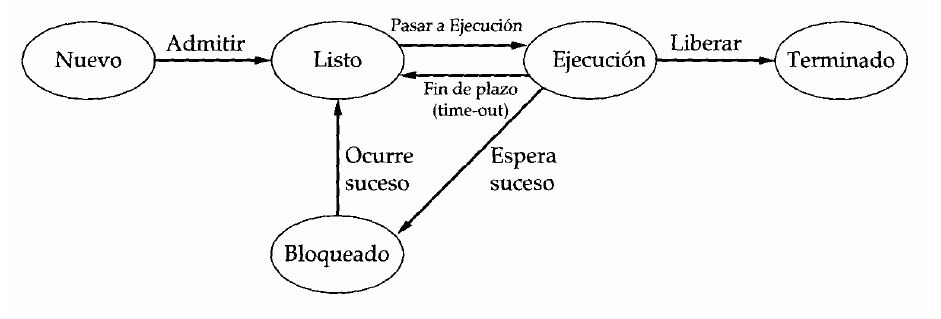
**Short Term Scheduler**

Decide qué proceso se ejecutará en un instante dado. Este planificador se puede poner en ejecución cuando se producen alguno de los siguiente eventos:

1. Interrupciones de reloj
2. Interrupciones de E/S
3. Call Systems
4. Señales

Estos procesos expulsaran al proceso en ejecución y le suministrarán el uso del procesador al siguiente proceso en la cola de listos que considere más adecuado la política implementada.

1. Decide a cuál de los procesos en la cola de listos se elige para que use la CPU
2. Términos asociado: apropiativo, no apropiativo, algoritmo de scheduling



**Sobre el estado nuevo (new) :** un usuario dispara el proceso. Un proceso es creador 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.

**Sobre el estado listo (ready) :** Luego que el scheduler de largo plazo eligió al proceso para cargarlo en memoria, el proceso queda en estado listo. El proceso sólo necesita que se le asigne CPU. Está en la cola de procesos listos (ready queue)

**Sobre el estado en ejecución (running) :** El scheduler de corto plazo lo eligió para asignar CPU. Tendrá la CPU hasta que se termine el período de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S.

**Sobre el estado de espera (waiting) :** 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, pasará al estado de listo.

**Transiciones**

1. **New-Ready :** por elección del scheduler de largo plazo (carga en memoria)
2. **Ready-Running:** por elección del scheduler de corto plazo (asignación de CPU)
3. **Running-Waiting:** el proceso “se pone a dormir”, esperando por un nuevo evento.
4. **Waiting-Ready:** terminó la espera y compite nuevamente por la CPU
5. **Caso especial : Running-Ready :** cuando el proceso termina su quantum sin haber necesitado ser interrumpirlo por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento.

**Comportamiento de los procesos**

1. CPU-bound
   1. Mayor parte del tiempo utilizando la CPU. Son aquellos procesos que en su ejecución predomina el uso del procesador.
2. I/O-bound
   1. Mayor parte del tiempo esperando por I/O. Son aquellos procesos que predominan las tareas de E/S, la interacción con periféricos.
3. La velocidad de CPU aumentó considerablemente respecto a la de los dispositivos de E/S
   1. Pensar: necesidad de atender rápidamente procesos I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para procesos CPU-bound

**Planificación**

1. Necesidad de determinar cual de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado
2. Algoritmo de planificación
   1. Algoritmo utilizado para realizar la planificación del sistema

**Algoritmos Apropiativos y No-Apropiativos**

1. En los algoritmos Apropiativos (preemptive) existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU.
2. En los algoritmos No-Apropiativo (non-preemptive) los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU
   1. Se bloquea por E/S o finaliza
   2. No hay decisiones de planificación durante las interrupciones de reloj

**Categorías de los Algoritmos de Planificación**

1. Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:
   1. Equidad: Otorgar una parte justa de la CPU a cada proceso
   2. Balance: Mantener ocupadas todas las partes del sistema
2. Ejemplos
   1. Procesos por lotes (batch)
      1. no existen usuarios que esperen una respuesta en una terminal
      2. se pueden utilizar algoritmos no apropiativos
      3. Metas propias de este tipo de algoritmos
         1. Rendimiento: maximizar el número de trabajos por hora
         2. Tiempo de retorno: minimizar los tiempos entre el comienzo y la finalización
         3. Uso de la CPU: mantener la CPU ocupada la mayor cantidad de tiempo posible
         4. Ejemplo
            1. FCFS (First Come First Served)
            2. SJF (Shortest Job First)
   2. Procesos Interactivos
      1. no sólo interacción con los usuarios
         1. un servidor, necesita de varios procesos para dar respuestas a diferentes requerimientos
      2. Son necesarios algoritmos apropiativos para evitar que un proceso acapara la CPU
      3. Metas propias de este tipo de algoritmos
         1. Tiempo de respuesta: responder a peticiones con rapidez
         2. Proporcionalidad: cumplir con expectativas de los usuarios
            1. Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto
   3. Procesos en Tiempo Real

**Algoritmos de Planificación**

**FCFS (First Come, First Served)**

Cada proceso se coloca en la cola de listos, con lo cual cuando hay que elegir un procesos para ejecutar se selecciona el más viejo en la cola de listos (FIFO) No favorece a ningún tipo de procesos, ya que van a ir ejecutándose en el orden de llegada, pero en un principio se podría decir que los procesos que demandan exclusivamente uso de procesador terminarán su ejecución en la primera ráfaga, mientras que los procesos que demandan uso de E/S deberán necesitar más ráfagas para concluir su ejecución

**SJF (Shortest Job First)**

Es una política no apropiativa que selecciona el proceso más corto primero. Al ser no apropiativa esto produce que el SO no pueda apropiarse del proceso y expulsarlo del CPU con lo cual los procesos cortos se ejecutarán constantemente lo cual puede producir inanición de lo s procesos más largos.

**Round Robin T fijo y T variable**

Es una política que se base en la temporización por reloj. se hace uso de una medida unitaria común a todos los procesos, el QUANTUM el cual determina cuánto tiempo podrá hacer uso un proceso del procesador

Quantum pequeño : demasiado costo temporal de cambio de contexto, pero útiles para procesos demandantes de E/S

Quantum grande: los procesos de E/S Pueden tardar demasiado tiempo hasta volver a poder hacer uso del CPU, pero favorece a los procesos CPU-Bound. Cuando un proceso es expulsado del CPU es colocado al final de la cola de listos Queue y se selecciona otro FIFO circular

**SRTF (Shortest Remaining Time First)**

El SRTF se fenie como una versión apropiativa del SJF,es decir, que selecciona el proceso el cual le reste menos tiempo de ejecución y le otorga el cPU. Como es de suponer favorece a los proceso de E/S, y no así a los demandantes de cPU, y aunque ellos mismo suelen necesitar de muchos más ciclos de ejecución que las breves ráfagas de CPU de las que hacen uso los procesos de E/S

**Cola Multinivel**

Para los procesos interactivos de podría hacer uso de los siguiente tipos de algoritmos

1. SJF
2. SRTF
3. VRR con tiempo variable

Para los procesos batch se podría hacer uso de

1. FCFS
2. RR Quantum alto

**Inanición**

Significa que un proceso se le niega sistemáticamente el uso de un recurso compartido, por lo cual pasa largos períodos de tiempo sin poder hacer un de dicho recurso, también se conoce como Starvation. Esto provocó que el proceso no pueda finalizar y la tarea muere.

Puede provocarla aquellos que manejan políticas de prioridades, también en el SJF.

Se puede solucionar mediante el cambio de prioridad de un proceso durante su vida, es decir, aumentando a medida que aumente al tiempo de espera o disminuyendo si ha recibido una determinada cantidad de tiempo de CPU. Para el caso de SJF siempre provoca canción y no se pueden tomar medidas al respecto.

**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

1. Crear la PCB
2. Asignar PID (Process IDentification) único
3. Asignar memoria para regiones
   1. Stack
   2. Text
   3. Datos
4. Crear estructuras de datos asociadas
   1. Fork (copiar el contexto, regiones de datos, text, stack)

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

El Hijo es un duplicado del proceso padre (casi Unix). Se crea el proceso y se le carga dentro el programa (caso windows)

Creación de procesos:

1. UNIX
   1. System Call fork(), crea un nuevo proceso
   2. System Call execve(), usada después del fork, carga un nuevo programa en el espacio de direcciones
2. Windows:
   1. System Call CreateProcess() crea un nuevo procesos y carga el programa para ejecución

Terminación de procesos:

1. Ante un (exit) se retorna el control al SO
   1. El proceso padre puede recibir un código de retorno (via wait)
2. Proceso padre puede terminar la ejecución de sus hijos (kill)
   1. La tarea asignada al hijo se terminó
   2. Cuando el padre termina su ejecución
      1. Habitualmente no se permite a los hijos continuar, pero existe la opción
      2. Terminación es cascada

**Pasos que se suceden al llamar a una System Call:**

1. El proceso de Usuario llama a una Syscall pode medio de la GlibC
2. La GlibC pone los parámetros para la Syscall en el Stack y eleva una interrupción
3. Cambia a Kernel Mode
4. Coloca el PC y PSW (Flags) en el stack (puede que se coloquen más registros, dependiente de la arquitectura)
5. Se coloca en la PC la dirección de la rutina de atención de interrupción que se extrae de la IDT y se continúa la ejecución
6. Se sacan los parámetros a la Syscall Del stack
7. De ser necesario se pueden guardar otros más registros del proceso actual en el Stack o en la PCB, depende de la implementación del SO
8. Se cambia a Kernel Stack, guardando la dirección del stack en User Mode en la PCB
9. Se colocan los parámetros para la Syscall en el Stack
10. Se ejecuta al Syscall
11. Si la Syscall bloquea el Proceso
    1. De ser necesario se guarda más información sobre el proceso bloqueado, registros, estados, etc
    2. Se ejecuta el Short Term Scheduler para seleccionar un nuevo proceso
    3. Se realiza el context switch
       1. Se cargan los registros del nuevo proceso
       2. Se acomoda la dirección del Stack, dejando apuntado a la dirección que el HW dejó previo a que el proceso seleccionado sea suspendido
       3. Se acomodan los datos necesario en la PCB o estructuras utilizadas
12. Se cambia a User Mode
13. Se ejecuta RET
14. Se sacan de la pila el PWD y PC
15. Continúa la ejecución del proceso actual

**Resumen de la Práctica - Procesos**

Pasos del proceso de inicio de sistema de **GNU/Linux,** desde que se prende la PC hasta obtener el login del sistema:

1. Se empieza a ejecutar el código BIOS
2. La BIOS ejecuta el POST, autoprueba de encendido, verifica dispositivos de HW
3. La BIOS lee el sector de arranque del disco MBR
   1. Es un registro de arranque principal conocido como arranque maestro, es el primer sector (SECTOR 0) de un dispositivo de almacenamiento de datos. A veces se emplea para el arranque del SO, con BOOTSTRAP, otras veces es usado para almacenar una tabla de particiones y, en ocasiones, se usa sólo para identificar un dispositivo de disco individual.
4. Se carga el gestor de arranque MBC
   1. Si el disco es BOOTEABLE los primeros 446 bytes del MBR están ocupados por un pequeño trozo de código denominado Master Boot Code, o cargador inicial, BOOTSTRAP LOADER, que es cargado por la BIOS para comenzar el procesos de carga. El BL recorre la tabla de particiones buscando una partición activa, y si lo encuentra carga su sector inicial cargando su código en memoria y transfiriendo el control, Dicho código es capaz de cargar cualquier programa que se encuentre en cualquier parte del disco, a su vez inicializará directamente el SO o tal vez una utilidad conocida como gestor de arranque GRUB.
5. El BOOTLOADER carga el Kernel y el INIT RD , o el comando INIT RD
6. El init carga los procesos necesarios para la puesta en marcha del sistema, vaciando el subdirectorio /tmp, lanza el GET TY que se encarga de ejecutar el login en el sistema para permitir la autentificación a los usuarios.

Los procesos INIT los ejecuta el Kernel y es el primer proceso que se ejecuta en el sistema. Su objetivo es cargar los subprocesos necesarios para la puesta en marcha del sistema operativo. Paso 5 y 6.

El runlevel es un estado de operación presente que determinan los servicios que tendremos disponibles en cada uno de ellos. Es una forma de tener diferentes modos de trabajos (procesos) que cada uno de ellos, con características bien definidas. su objetivo es definir el modo de operación de los Sistemas Operativos

1. INIT 0 → Alto o cierre del sistema - apagado
2. INIT 1 → Modo de usuario único - monousuario
3. INIT 2 → Multiusuario
4. INIT 3 → Multiusuario con soporte de red - inicia normalmente
5. INIT 4 → Multiusuario con soporte de red
6. INIT 5 → Multiusuario gráfico X11
7. INIT 6 → Reinicio

Están definidos en la carpeta /etc/inittab/. no Otros sistemas utilizan otro sistema de arranque.

La carpeta /etc/inittab/ → almacena todas las tareas que se van a realizar después de que el proceso init se ejecuta, pero que se mantienen en segundo plano. Por ejemplo se almacenan funciones comunes como:

1. Inicio → activación del SWAP
2. Configuración de la Red
3. Variables del Sistema
4. Etc

**Upstart - Arranque del sistema**

Permite la ejecución de trabajos (jobs) en forma asincrónica a través de eventos (event-based). Estos jobs tienen como principal objetivo definir servicios o tareas a ser ejecutadas por INIT. Las tareas son scripts de texto plano que definen las acciones /tareas a ejecutar ante determinadas eventos. El upstart es el reemplazo de System V, su principal diferencia es que el upstart es estrictamente asincrónico (dependency-based).

El upstart es un reemplazo basado en eventos, y no en niveles. los servicios se pueden levantar o desactivar en respuesta a ciertos eventos, y este procedimiento permite por ejemplo manejar el reinicio de servicios que mueren en forma inesperada. Upstart opera asíncronamente. Las tareas y los servicios son ejecutados ante eventos (arranque del equipo o inserción de un dispositivo USB) definidos como jobs. Los jobs se almacenan en el directorios /etc/init. Son scripts en texto plano que definen las acciones a ejecutar. Es compatible con System V.

**System D**

Es un sistema que centraliza la administración de demonios y librerías del sistema

1. Mejora el paralelismo de booteo
2. Puede ser controlado por Systemctl
3. Compatible con System V
4. El demonio systemd reemplaza al proceso init → este pasa a tener PID1
5. Los runlevels son reemplazados por targets
6. Al igual que con upstart el archivo /etc/initab no existe más
7. Las unidades de trabajo son denominadas units de tipo:
   1. Service: controla un servicio en particular
   2. Socket: encapsula IPC, un socket del sistema o file system FIFO (.socket) → socket based activation
   3. Target: agrupa units o establece puntos de sincronización durante el booteo (.target) → dependencia de unidades
   4. Snapshot: almacena el estado de un conjunto de unidades que puede ser reestablecido más tarde (.snapshot)
   5. Las units pueden tener dos estados
      1. Active
      2. Inactive
8. cgroup : permite organizar un grupo de procesos en forma jerárquica. Agrupa conjuntos de procesos relacionados.

**Cronología**

1. System v → sincronico
2. System D → asincrónico
3. Upstart → asincrónico

***MEMORIA***

**Administración de Memoria:**

La gestión de memoria se encarga de administrar la división lógica de la memoria para alojar múltiples procesos, debe asignar eficientemente la memoria para contener el mayor número de procesos como sea posible, lo que hace que cuantos más procesos estén en memoria, más procesos competirán por la CPU. Más probabilidad que la CPU no esté ociosa.   
Se proponen 5 requisitos:

1. **Reubicación**
   1. Los programadores no saben dónde estará el programa en memoria cuando se ejecute, mientras el programa se ejecuta, puede ser movido al disco y devuelto a memoria principal en una posición diferente (reubicado). Se deben traducir las referencias a la memoria según la dirección actual del proceso
2. **Protección**
   1. Los procesos no deberían ser capaces de referenciar el espacio de memoria de otro proceso sin permiso, es imposible comprobar las direcciones absolutas de los programas puesto que éstos pueden ser reubicados. El chequeo se debe realizar durante la ejecución ya que el SO no puede anticipar todas las referencias a memoria que un proceso puede realizar.
3. **Compartición**
   1. Permitir que varios procesos accedan a la misma zona (porción) de memoria. Lleva a un mejor uso de la memoria, evitando copias innecesarias de instrucciones.
4. **Organización Lógica**
   1. Los programas son escritos en módulos que se pueden escribir y compilar por separado. También se les puede dar diferente grado de protección (WRX). Y puede ser módulos compartidos.
5. **Organización Física**
   1. La memoria disponible para un programa y sus datos puede ser insuficiente, el solapamiento permite asignar la misma zona de memoria a diferentes módulos. El programador no sabe cuánto espacio habrá disponible.

**Espacio de Direcciones (abstracción):**

Es el rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos. El tamaño depende de la arquitectura y debe ser independiente de la ubicación real del proceso en memoria.

**Direcciones:**

1. **Lógicas**
   1. Referencian a una localidad de memoria independiente de la asignación actual de los datos en la memoria. Se debe realizar una traducción a una dirección física.
2. **Físicas**
   1. La dirección absoluta en la memoria principal

**Swapped Out:** Un proceso puede ser temporalmente sacado de la memoria a un disco de manera de permitir la ejecución de procesos

**Swapped In (si se descarga considerando las direcciones físicas):** se debe cargar en el mismo espacio de memoria que ocupaba antes

**Swapped In (si se descarga considerando las direcciones lógicas):** se puede cargar en cualquier espacio de direcciones de memoria

**Esquema con Múltiples Particiones**

La memoria es dividida en varias regiones (particiones), los procesos (su espacio de direcciones) se colocan en las particiones según su tamaño. Existen 2 técnicas:

1. **Particiones Fijas**
   1. Regiones definidas con límites fijados
   2. Particiones de IGUAL tamaño
      1. Cualquier proceso cuyo tamaño es menor igual que la partición puede ser colocado en una partición libre
      2. Si todas están ocupadas → Swap
      3. Ineficiencia
         1. Programa más grandes que el tamaño de las particiones no podrán ejecutarse, por más que la memoria total sea más grande que el programa
         2. Programa pequeños, ocuparán una partición desperdiciando mucho espacio.
   3. Particiones de DISTINTO tamaño
      1. Evita el problema de las particiones de igual tamaño
         1. Procesos pequeños pueden usar particiones pequeña
         2. Se pueden prever algunas particiones grandes para procesos grandes
      2. Complejidad en el algoritmo de selección de partición para un proceso
   4. **Problemas:**
      1. Problemas de fragmentación interna: espacio libre dentro de las particiones de las particiones asignadas que no puede ser utilizado
      2. Un programa más grande que una partición no puede ejecutarse
   5. **Algoritmo de Ubicación**
      1. Particiones del mismo tamaño:
         1. Como todas las particiones tienen el mismo tamaño, no importa qué partición asignar
      2. Particiones de diferente tamaño:
         1. Se puede asignar a cada proceso la partición más pequeña en la que cabe
         2. Cola para cada partición
         3. Los procesos se asignan de manera que se minimiza la memoria desperdiciada de una partición
2. **Particiones Dinámicas**
   1. El tamaño y el número de particiones es variable
   2. Al proceso se le asigna exactamente la cantidad de memoria que necesita
   3. Aparecen huecos en la memoria. Esto se conoce como fragmentación externa
   4. Se debe realizar una compactación para desplazar a los procesos de forma que estén juntos y todo el espacio libre esté en un solo bloque
   5. **Algoritmo de Ubicación**
      1. El sistema operativo debe decidir qué bloque libre asignar a un proceso
      2. **Algoritmo del mejor ajuste (best fit)**
         1. Elige el bloque que tiene el tamaño más cerca al solicitado
         2. Peor rendimiento de todos
         3. Como se busca el bloque más pequeño por proceso, se produce el menor volumen de fragmentación, pero hay que compactar más a menudo.
      3. **Algoritmo del primer ajuste (first fit)**
         1. Es el más rápido
         2. Puede haber muchos procesos cargado en la zona inicial de la memoria, que debe ser examinada cuando se busca un bloque libre.
      4. **Algoritmo del siguiente ajuste (Next-fit)**
         1. A menudo se asigna un bloque de memoria en la última parte de la memoria donde está el mayor bloque
         2. El mayor bloque de memoria se parte en pequeños bloques
         3. Se necesita compactar para obtener un bloque grande en la última zona final de memoria
   6. **Problemas**
      1. Cada vez que entra y sale un proceso se genera huecos en la memoria, en los que eventualmente un proceso no podría entrar, pero si entraría si unimos todos los huecos (compactación)
      2. Problemas de fragmentación externa
         1. Espacio libre entre las particiones asignadas. Puede ser utilizado siempre y cuando el espacio requerido por un procesos, sea menor.

**Fragmentación**

**Interna:** Es el espacio dentro de las particiones fijas que queda inutilizado dado que raramente el proceso que ocupa una partición fija lo ocupa en su totalidad

**Externa:** Cada vez que entran y salen procesos se generan huecos en la memoria, que luego eventualmente otro proceso no podría entrar. Pero si se podría hacer uso de estos huevos si se unen mediante la compactación.

**Paginación**

Partición de la memoria en pequeños pedazos del mismo tamaño (Chunks - Marcos) y dividir cada proceso en trozos del mismo tamaño. Los trozos de un proceso se llaman páginas y los de la memoria se llaman marcos de página (frames).

El sistema operativo mantiene una tabla de página para cada proceso

1. Contiene la ubicación del marco de página (frame) de cada página del proceso
2. La dirección de memoria consiste en un número de página y un desplazamiento (offset) dentro de la página

**Segmentación**

Todos los segmentos de todos los programas no tienen porqué ser del mismo tamaño. Hay un tamaño máximo de segmento. El direccionamiento consta de dos partes -un número de segmento y un desplazamiento dentro de éste (offset).

Como los segmentos no son iguales, la segmentación es similar al particionamiento dinámico.

**Arquitectura de la Segmentación**

1. Tabla de segmentos: permite mapear la dirección lógica en física. Cada entrada contiene
   1. Base: dirección física de comienzo del segmento
   2. Limit: longitud del segmento
2. Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos
3. Segment-table length register (STLR): cantidad de segmentos de un programa

**Segmentación Paginada**

1. La Paginación
   1. Transparente al programador
   2. Elimina Fragmentación externa
2. Segmentación
   1. Es visible al programador
   2. Facilita modularidad,estructura de datos grandes y da mejor soporte a la compartición y protección
3. Cada segmento es dividido en páginas de tamaño fijo

**Memoria Virtual**

En una computadora con memoria virtual las direcciones de los programas, generadas por la CPU, hacen referencia a un espacio mayor que el espacio físico realmente disponible en la memoria principal o memoria física.

Este esquema permite hablar de paginación, disociar completamente la memoria virtual de la memoria física. Donde cada proceso tiene una vista lógica de su memoria, mapeándose en memoria física.

La memoria virtual generalmente hace creer al proceso que dispone de más memoria física de la que realmente dispone, ya que la memoria física no es más que una proyección parcial de la memoria virtual.

Es espacio de direcciones no necesariamente tiene que esta contiguo a la memoria.

El SO debe ser capaz de realizar de manera correcta el movimiento de la s páginas entre la memoria principal y la secundaria. El SO debe disponer de las tablas de páginas y hacer uso intensivo de esta para una correcta gestión de la memoria.  
  
**Memoria Virtual con Paginación por Demanda**

1. Cada proceso tiene su tabla de páginas
2. Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en la memoria principal
3. Cada entrada en la tabla de páginas tiene bts de control (entre otros)
   1. Bit V: indica si la página está en memoria
   2. Bit M: indica si la página fue modificada. si se modificó, en algún momento se deben reflejar los cambios en memoria secundaria

**Fallo de páginas (Page Fault)**

1. Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en memoria principal. Bit v = 0
   1. La página no se encuentra en su conjunto residente
   2. el bit v es controlado por el HW
2. El HW detecta la situación y genera un trap al S.O.
3. El SO podrá colocar al proceso en estado de Blocked (espera) mientras gestiona que la página que se necesite se cargue
4. 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
5. El SO puede asignarle la CPU a otro proceso mientras se completa la E/S
   1. La E/S se realizará y avisará mediante interrupción su finalización
6. Cuando la operación de E/S finaliza, se notifica al SO y este
   1. Actualizar la tabla de páginas del proceso
      1. Coloca el Bit V en 1 en la página en cuestión
      2. Coloca la dirección base del Frame donde se colocó la página
   2. El proceso que generó el Fallo de Página vuelve a estado de Ready (listo)
   3. Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes generó el fallo de página

**Tabla de Páginas**

1. Cada proceso tiene su tabla de páginas
2. El tamaño de la tabla de páginas depende del espacio de direcciones del proceso
3. puede alcanzar un tamaño considerable

**Tamaño de la Página**

1. Pequeño
   1. Menor fragmentación interna
   2. Más páginas requeridas por proceso → tabla de páginas más grandes
   3. Más páginas pueden residir en memoria
2. Grande
   1. Mayor fragmentación interna
   2. La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente → más rápido mover páginas hacía memoria principal

**Translation Lookaside Buffer 1**

Cada referencia en el espacio virtual puede causar 2 o más accesos a la memoria física

1. Uno o más para obtener la entrada en la tabla de páginas
2. Uno para obtener los datos

Para solucionar este problema, una memoria caché de alta velocidad es usada para almacenar entradas de páginas TBL

**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(hit), 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, de no estar se genera un page fault. La TLB se actualiza para incluir la nueva entrada

**Asignación de Marcos**

1. Cuántas páginas de un proceso se pueden encontrar en memoria ?
   1. Tamaño del conjunto residente
2. Asignación Dinámica
   1. El número de marcos de cada proceso varía
3. Asignación fija
   1. Número de arco fijo para cada proceso
   2. Asignación equitativa
   3. Asignación proporcional: se asigna acorde al tamaño del proceso

**Reemplazo de Páginas - Selección de una Víctima**

1. **LRU**
   1. Reemplaza a la página que no ha sido referenciada por más tiempo. Con lo cual cada página deberá tener información del instante de su última referencia
2. **FIFO**
   1. Trata a los frames en uso como una cola circular. Es muy simple de implementar. la página más vieja de la memoria es la reemplazada, aunque esta mismo podría ser prontamente necesitada.
3. **OPT**
   1. Selecciona la página cuya próxima referencia se encuentra lo más lejana posible a la actual. Es imposible de implementar puesto que no se conoce lo que va a hacer un proceso en el futuro.
4. **Segunda Chance**
   1. Se hace uso de un bit adicional, bit de referencia R. Cuando la página es recién cargada en memoria este bit se pone en 0, y cuando la página es referenciada se pone en 1. La víctima a reemplazar por esta política se busca en orden FIFO. Se selecciona la primera página cuyo bit R está en 0. Mientras se realiza esta búsqueda todo los bits R que se encuentres en 1 se llevan a 0

**De Mejor a Peor**

1- Óptimo

2- Segunda Chance

3- LRU

4- FIFO

**Thrashing (hiperpaginación)**

Concepto: decimos que un sistema está en thrashing cuando pasa más tiempo paginado que ejecutando procesos. Como consecuencia, hay un baja importante de performance en el sistema.

**Ciclo de Thrashing**

1. El SO 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. comienza los page-faults y robo de frames a otros procesos.
5. Por swapping de páginas, y encolamiento de 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 del thrashing**

1. Se puede limitar el thrashing usando algoritmos de reemplazo local
2. Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos
3. Si bien perjudica la performance del sistema, es controlable

Si un proceso cuenta con todos los frames que necesita, no habría thrashing. Veremos algunas técnicas como la estrategia de Working Set, con el modelo de localidad y la estrategia PFF (Frecuencia de Fallos de Páginas)

**El modelo de localidad**

1. Cercanía de referencias o principio de cercanía
2. Las referencias a datos y programa dentro de un proceso tienden a agruparse
3. En cortos períodos de tiempo, el proceso necesitará pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos...)
4. Un programa se compone de varias localidades
5. Ejemplo: Cada rutina será una nueva localidad: se referencia sus direcciones (cercanas) cuando se está ejecutando
6. Para prevenir la hiperactividad, un proceso debe tener en memoria sus páginas más activas (menos page-faults)

**Working Set**

1. Se basa en el modelo de localidad
2. Ventana del working: las referencias de memoria más recientes
3. Working set: es el conjunto de páginas que tienen las más recientes referencias a página

**Prevención del thrashing**

1. SO monitorea c/ proceso, dándole tantos frames hasta su WSS
2. Si quedan frames, puede iniciar otro proceso
3. Si D crece, excediendo m, se elige un proceso para suspender, reasginándose sus frames…

Así, se mantiene alto el grado de multiprogramación optimizando el uso de la CPU

**Problema del modelo del WS**

1. Mantener un registro de los wss
2. La ventana es móvil
3. m = cantidad de frames disponibles
4. WSS = medida del working set del proceso p
5. D = demanda total de frames
6. si D > m, habrá thrashing

**Prevención del thrashing por PFF**

1. PFF: frecuencia de page faults
2. PFF HIGHT: se necesitan más frames
3. PFF DOWN: los procesos tienen muchos frames asignados

**PFF**

1. Establecer límites superior e inferior de las PFFś deseadas
2. Excede PFF max → le doy un frame más
3. Por debajo del PFF mínimo → le saco frame
4. 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 tienen un baja utilización o algún parámetro de la memoria lo indica ( poca memoria libre - Mucha memoria modificada)

Tareas:

1. Limpia las páginas modificadas sincronizandose con el swap
2. Reduce el tiempo de swap posteriormente ya que las páginas están limpias
3. Puede sincronizar varias páginas contiguas reduciendo el tiempo total de transferencia
4. Mantener el número de páginas libre en el sistema a un cierto número
5. No liberarlas del todo hasta que haga falta realmente

**Memoria Compartida**

Gracias al uso de la tabla de páginas varios procesos pueden compartir un marco de memoria; para ellos 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 (sólo lectura) por ejemplo: editores de texto, compiladores, etc. Los datos son privados a cada proceso y se encuentran en páginas no compartidas.

**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 ellos modifica una página compartida la página es copiada. COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas.

**Mapeo de Archivo en Memoria**

1. Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales
2. El contenido del archivo no se sube a memoria hasta que se generan page faults
3. El contenido de la página que genera el PF es obtenido desde el archivo asociado
   1. no del área de intercambio
4. Cuando un proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente
5. Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el sistema de archivos
6. Utilizando comúnmente para asociar librerías compartidas

**Área de Intercambio**

1. Sobre el Área utilizada
   1. Área dedicada, separada del sistema de archivos(por ejemplo: en linux)
   2. Un archivo dentro del sistema de archivos (por ejemplo: windows)
2. Técnicas para la administración
   1. 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.
   2. 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 (página a página) de la localización de las páginas en disco
3. Cuando una página no está en memoria, sino en disco, cómo podemos saber en qué parte del área de intercambio está? Rta: el PTE de dicha página tiene el bit v=0 y todos los demás bits sin usar !

**Gestión de E/S**

1. Legible para el usuario
   1. Usados para comunicarse con el usuario
      1. impresoras, terminales]: pantalla, teclado, mouse
2. Legible para la máquina
   1. Utilizados para comunicarse con los componentes electrónicos
      1. Discos, cintas, sensores, etc
3. Comunicación
   1. Usados para comunicarse con dispositivos remotos
      1. Líneas digitales, modems, interfaces de red, etc

**Problemas que Surgen**

1. Amplia Variedad
   1. Manejan diferentes cantidad de datos
   2. En velocidades diferentes
   3. En formatos diferentes
2. La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

**Interfaz de E/S**

1. Es deseable manejar todos los dispositivos de E/S de una manera uniforme, estandarizada
2. 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
3. Los dispositivos de E/S pueden resultar extremadamente lentos respecto a la memoria
4. El uso de la multiprogramación permite que los procesos esperen por la finalización de la E/S mientras que otro se ejecuta
5. E/S no puede alcanzar la velocidad de la CPU

**Aspectos de los dispositivos de E/S**

1. Unidad de Transferencia
   1. Dispositivos por bloques (Discos)
      1. Operaciones: Read - Write - Seek
   2. Dispositivos por Carácter (keyboards - mouse - serial ports )
      1. Operaciones: Get - Put
2. Formas de acceso
   1. Secuencial o aleatorio
3. Tipo de Acceso
   1. Compartido: Disco Rígido
   2. Exclusivo: Impresora
   3. Read only : CDROM
   4. Write only : Pantalla
   5. Read/Write : Disco

**Subsistema de E/S - Servicios**

1. Planificación
   1. Organización de los requerimientos a los dispositivos
   2. EJ: Planificación de requerimientos a disco para minimizar movimientos
2. Buffering - Almacenamiento de los datos en memoria mientras se transfieren
   1. Solucionar problemas de velocidad entre los dispositivos
   2. Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos
3. Caching
   1. Mantener en memoria copia de los datos de reciente acceso para mejorar la performance
4. Spooling - Administrar la cola de requerimientos de un dispositivo
   1. Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo Ej: impresora
   2. Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo
5. Reserva de Dispositivos
   1. Acceso exclusivo
6. Manejo de Errores
   1. El SO debe administrar errores ocurridos (lectura de un disco -. dispositivo no disponible - errores de escritura)
   2. La mayoría retorna un número de error o código cuando la E/S falla
   3. Logs de errores
7. Formas de realizar E/S
   1. Bloqueante : el proceso se suspende hasta que el requerimiento de E/S se completa
      1. Fácil de usar y entender
      2. No es suficiente bajo algunas necesidades
   2. No Bloqueante : el requerimiento de E/S retorna en cuanto es posible
      1. Ejemplo : interfaz de usuario que recibe input desde el teclado/mouse y se muestra en el screen
      2. Ejemplo : aplicación de video que lee frames desde un archivo mientras se va mostrando en pantalla

**Estructura de Datos**

1. El kernel mantiene la información de estado de cada dispositivo o componente
   1. Archivos abiertos
   2. Conexiones de red
2. Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc

**Lectura sobre un archivo en un disco**

1. Determinar el dispositivo que almacena los datos
   1. Traducir el nombre del archivo en la representación del disco
2. Lectura física de los datos en la memoria
3. Marcar los datos como disponibles al proceso que realizó el requerimiento
   1. Desbloqueo
4. Retornar el control al proceso

**Drivers**

1. Contienen el código dependiente del dispositivo
2. Manejan un tipo dispositivo
3. Traducen los requerimientos abstractos en los comandos para el dispositivo
   1. Escribe sobre los registros del controlador
   2. Acceso a la memoria mapeada
   3. Encola requerimientos
4. Comúnmente las interrupciones de los dispositivos están asociadas a una función del driver
5. Interfaz entre el SO y el HARD
6. Forman parte del espacio de memoria del Kernel
   1. En general se cargan como módulos
7. Los fabricantes de HW implementan el driver en función de una API especificada por el SO : open() - close() - read() - write() - entre otros
8. Para agregar nuevo HW sólo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel
9. Linux distingue 3 tipos de dispositivos
   1. Carácter: E/S programada o por interrupciones
   2. Bloque: DMA
   3. Red: Ports de comunicaciones
10. Los Drivers se implementan como módulos
    1. Se cargan dinámicamente
11. Debe tener al menos estas operaciones
    1. init\_module : para instalarlo
    2. cleanup\_module : para desinstalarlo
    3. Operaciones que debe contener para E/S
       1. open → abre el dispositivo
       2. release → cerrar el dispositivo
       3. read → leer bytes en el dispositivo
       4. write → escribir bytes en el dispositivo
       5. ioctl → orden de control sobre el dispositivo

**Performance**

1. E/S es uno de los factores que más afectan a la performance del sistema
   1. Utiliza mucho CPU para ejecutar los drivers y el código del subsistema de E/S
   2. Provoca context switches ante las interrupciones y bloqueos de los procesos
   3. Utiliza el bus de memoria en copa de datos
      1. Aplicaciones (espacio usuario) - kernel
      2. Kernel (memoria física) - controladora

**Para Mejorar**

1. Reducir el número de context switches
2. Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación
3. Reducir la frecuencia de las interrupciones, utilizando
   1. Transferencias de gran cantidad de datos
   2. Controladoras más inteligentes
   3. Polling, si se minimiza la espera activa
4. Utilizar DMA

***Administración de Archivos***

1. Los necesitamos para
   1. Almacenar grandes cantidades de datos
   2. Tener almacenamiento a largo plazo
   3. Permitir a distintos procesos acceder al mismo conjunto de información

**Archivo**

1. Entidad con nombre
2. Espacio lógico continuo y direccionable
3. Provee a los programas de datos - entrada
4. Permite a los programas guardar datos - salida
5. El programa mismo es información que debe guardarse

**Sistema de Manejo de Archivos**

1. Servicios:
   1. Crear
   2. Borrar
   3. Buscar
   4. Copiar
   5. Leer
   6. Escribir
   7. etc
   8. Facilita el acceso a los archivos por parte de las aplicaciones
   9. 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 cuando a archivo**

1. Cumplir con la gestión de datos
2. Cumplir con las solicitudes del usuario
3. Minimizar / Eliminar la posibilidad de perder o destruir datos
   1. Garantizar la integridad del contenido de los archivos
4. Dar soporte a E/S a distintos dispositivos
5. Brindar un conjunto de interfaces de E/S para tratamiento de archivos
6. Proveer soporte de E/S para múltiples usuarios

**Tipos de Archivos**

1. Regulares
   1. Texto plano : Source File
   2. Binarios
      1. Object File
      2. Executable File
2. Directorios
   1. Archivo que mantienen la estructura en el File Systems

**Atributos de un Archivo**

1. Nombre
2. Identificador
3. Tipo
4. Localización
5. Tamaño
6. Protección, Seguridad y Monitoreo

**Directorios**

1. Contiene información acerca de archivos y directorios que están dentro de él
2. El directorio es, en sí mismo, un archivo
3. Interviene en la resolución entre el nombre y el archivo mismo
4. Operaciones en directorios
   1. Search for a file
   2. Create a file (directory entry)
   3. Delete a file (directory entry)
   4. List a directory
   5. Rename a file

**Estructuras de Directorios**

1. Los archivos pueden ubicarse siguiente un path desde el directorio raíz y sus sucesivas referencias (full pathname del archivo o path absoluto)
2. Distintos archivos pueden tener el mismo nombre el full pathname es único
3. El directorio actual se lo llama “directorio de trabajo”
4. Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por su path absoluto como por su path relativo indicando solamente la ruta al archivo desde el directorio de trabajo

**Compartir archivos**

1. En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos
2. Debe ser realizado bajo un esquema de protección
   1. Derechos de acceso
   2. Manejo de accesos simultáneos

**Protección**

1. El propietario/administrador debe ser capaz de controlar
   1. Que se puede hacer : derechos de acceso
   2. Quién lo puede hacer
2. 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
3. Execution
   1. El usuario puede ejecutar
4. Reading
   1. El usuario puede leer el archivo
5. Appending
   1. El usuario puede agregar datos pero no modificar o borrar el contenido del archivo
6. Updating
   1. El usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos
7. Changing protection
   1. El usuario puede modificar los derechos de acceso
8. Deletion
   1. El usuario puede borrar el archivo

**Derechos de Acceso**

1. Owners (propietarios)
   1. Tiene todos los derechos
   2. Puede dar derechos a otros usuarios. Se determinan clases
      1. Usuario específico
      2. Grupos de usuarios
      3. Todos (archivos públicos)

**Disco**

**Conceptos**

1. Sector
   1. Unidad de almacenamiento utilizada en los discos rígidos
2. Bloque/Cluster
   1. Conjunto de sectores consecutivos
3. File System
   1. Define la forma en que los datos son almacenados
4. FAT : File Allocation Table
   1. Contiene información sobre en qué lugar están alocados los distintos archivos

**Preasignación**

1. Se necesita saber cuánto espacio va a ocupar el archivo en el momento de su creación
2. Se tiende a definir espacios mucho más grandes que lo necesario
3. Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo

**Asignación Dinámica**

1. El espacio se solicita a medida que se necesita
2. Los bloques de datos pueden quedar de manera no contigua
3. **Continua**
   1. Conjunto continuo de bloques son utilizados
   2. Se requiere una pre-asignación
      1. Se debe conocer el tamaño del archivo durante su creación
   3. File Allocation Table : es simple
      1. Sólo una entrada que incluye Bloque de inicio y longitud
   4. El archivo puede ser leído con una única operación
   5. Puede existir fragmentación externa
      1. Compactación
   6. Problemas
      1. Encontrar bloques libres continuos en el disco
      2. Incremento del tamaño de un archivo
4. **Encadenada**
   1. Asignación en base a bloques individuales
   2. Cada bloque tiene un puntero al próximo bloque del archivo
   3. FAT : única entrada por archivo : bloque de inicio y tamaño del archivo
   4. No hay fragmentación externa
   5. Útil para acceso secuencial (no random)
   6. Los archivos pueden crecer bajo demanda
   7. No se requieren bloques contiguos
5. **Indexada**
   1. Asignación en base a bloques individuales
   2. No se produce fragmentación externa
   3. El acceso random a un archivo es eficiente
   4. FAT : única entrada con la dirección del bloque de índices (index node / i-node)

**Gestión de Espacio Libre**

1. Control sobre cuáles de los bloques de disco están disponibles
2. Alternativas
   1. Tabla de bits
   2. Bloques libres encadenados
   3. Indexación
3. Bloques encadenados
   1. Se tiene un puntero al primer bloque libre
   2. Cada bloque libre tiene un puntero al siguiente bloque libre
   3. Ineficiente para la búsqueda de bloques libres → hay que realizar varias operaciones de E/S para obtener un grupo libre
   4. Problemas con la pérdida de un enlace
   5. Difícil encontrar bloques libres consecutivos
4. Indexación (agrupamiento)
   1. Variante de bloques libre encadenados
   2. el primer bloque libre contiene las direcciones de N bloques libres
   3. Las n-1 primeras direcciones son bloques libres
   4. la N-ésima dirección referencia otro bloque con N direcciones de bloques libres
5. Recuento
   1. Variante de indexación
   2. esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua)
   3. En lugar de tener N direcciones libres (índice) se tiene
      1. La dirección del primer bloque libre
      2. Los N bloques libres contiguos que le siguen

**Unix - Manejo de Archivos**

**Tipos de Archivos**

1. Archivo común
2. Directorio
3. Archivos especiales (dispositivos /dev/sda)
4. Named pipes (comunicación entre procesos)
5. Links (comparten el i-nodo, solo dentro del mismo filesystem)
6. Links simbólicos (para filesystems diferentes)

**Estructura del Volumen**

1. Boot Block: código para bootear el SO
2. Superblock: atributos sobre el file system
3. I-NODE Table: tabla que contiene todos los I-NODOS
   1. I-NODO: estructura de control que contiene información clave de un archivo
4. Data Blocks : bloques de datos de los archivos

**Información del I-Nodo**

1. Identificación de usuario y grupos de archivos
2. Instantes del último acceso y de la última modificación
3. Contador con él numero de HORD-LINK al archivo
4. El tipo de archivo
5. 15 apuntadores a bloques de disco
   1. 12 apuntan a bloques directorios
   2. 1ero : apuntan a bloques indirectos
   3. 2do : a la dirección de un bloque indirecto doble
   4. 3ero : a un bloque indirecto triple
6. Bloque de carga : es el primero de cada sistema, está reservado para un programa de carga
7. Super bloque (512 bytes): es el bloque uno del dispositivo. Contiene información sobre el sistema de archivos tales como:
   1. tamaño de bloques
   2. nombre del sistema de archivos
   3. número de bloques reservados para inodos
   4. inodos libres
   5. comienzo de cadena de bloques libre
   6. nombre del volumen
   7. última actualización
   8. último backup

**VFS - Virtual File System**

1. Interfaz uniforme para el acceso de los procesos a los archivos
2. Es una capa de abstracción sobre los distintos FileSystem utilizados
3. Asume que los archivos son objetos que comparten propiedades más allá del File System que los almacena
4. Objetos
   1. Superblock object
      1. Representa el File System montado
   2. Inode object
      1. Representa un archivo
   3. D-Entry object
      1. Representa una entrada en un directorio
   4. File object
      1. Representa un archivo abierto asociado a un proceso

**Windows File Systems Soportados**

1. CD-ROM File System (CDFS)
2. Universal Disk Format (UDF)
3. File Allocation Table
   1. FAT12
   2. FAT16
   3. FAT32
   4. **FAT**
      1. File Allocation Table es un sistema de archivos utilizado originalmente por DOS y Windows 9x. Windows soporta FAT Filesystems
         1. por compatibilidad con otro SO en sistemas multi boot
         2. para permitir upgrades desde versiones anteriores
         3. para formato de dispositivos como diskettes
      2. 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.
      3. Se utiliza un esquema de Asignación Encadenada
      4. La única diferencia es que el puntero al próximo bloque están en la FAT y no en los bloques
      5. Bloques libres y dañados tienen códigos especiales
4. New Technology File System (NTFS)
   1. NTFS es el filesystem nativo de windows
   2. NTFS usa 64bit para referenciar clusters
   3. Es simple más rápido para ciertas operaciones pero NTFS soporta
      1. Tamaños de archivo y discos mayores
      2. mejor performance en discos grandes
      3. nombres de archivos de hasta 255 caracteres
      4. atributos de seguridad

***Caché de Disco***

1. Buffers en memoria principal para almacenamiento temporario de sectores de disco
2. Contienen una copia de algunos sectores de disco
3. MINIMIZAR LA FRECUENCIA DE ACCESO AL DISCO
4. Cuando un proceso quiere acceder a un bloque de la caché hay dos alternativas
   1. Se copia el espacio de direcciones de usuario
   2. Se trabaja como memoria compartida (no se copia permitiendo acceso a varios procesos)

**Estrategia de Reemplazo**

1. Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado
2. Es una lista de bloques, donde el último es el más recientemente usado (LRU)
3. Cuando un bloque se referencia o entra en la caché queda al final de la lista
4. No se mueven los bloques en la memoria: se asocian punteros
5. Otra alternativa: Least Frequently Used. Se reemplaza el que tenga menor número de referencias

**Unix System V**

**Objetivos y Estructura**

1. Minimizar la frecuencia de acceso a disco
2. Es una estructura forma para buffers
3. El kernel asigna un espacio en la memoria durante la inicialización para esta estructura
4. Un buffer tiene dos partes: el header y el lugar donde se almacena el bloque de disco traído a memoria
5. Header
   1. Identifica por número de dispositivo y número de bloque
   2. Tiene punteros
      1. 2 punteros para la hash queue
      2. 2 punteros para el free list
      3. un puntero al bloque en memoria
   3. Estado

**Estado de los Buffers**

1. Free o disponible
2. Busy o no disponible (en uso por algún proceso)
3. El kernel está escribiendo a disco o leyendo del disco
4. Delayed write : buffers que hayan sido modificado en memoria, pero el bloque original en disco todavía no fue actualizado

**Free List**

1. Organiza los buffers disponibles, es decir, los buffers donde se puede cargar un nuevo bloque de disco
2. No necesariamente los bloques están vacíos
3. Se ordena según LRU (Least Recent Used)

**Hash Queues**

1. Son colas para optimizar la búsqueda de un buffer en particular
2. Se organizan según una función de hash usando (dispositivo, #bloque)