CLASE 1-Introducción I-

SISTEMA OPERATIVO

- *Es software:*
  - Se programa y diseña como cualquier software.
  - Necesita procesador y memoria para ejecutarse.
  - Tiene una característica particular:
    - Es pensado para administrar hardware para que otro software lo pueda usar=> software que administra otro software.
- *Perspectiva de arriba hacia abajo:*
  - Desde el "mundo" hacia el S.O.
  - No se involucra en los detalles de la arquitectura.
  - El s.o le oculta el hardware y muestra a los programas abstracciones mas simples de manejar.
  - Los programas son clientes que le pide servicio al S.O.
  - El S.O. da servicio a progrmas y humanos=> hay SW que lo usa sin que haya humanos.
  - No se interactua directamente con el S.O, se hace a través de un SW=> interacción amigable.
- *Perspectiva de abajo hacia arriba:*
  - S.O como administrador de recursos de 1 o + procesos.
  - Cómo administra la CPU para que los programas se puedan ejecutar.
  - Maneja la memoria secundaria, RAM, dispositivos de e/s, cpu, compartiéndolos para permitir la ejecución simultánea y la multiplexación en tiempo(cpu) y espacio(memoria).

<u>Caracteristicas</u>

- Gestiona el HW.
- Controla la ejecución de procesos
- Es una interfaz entre las aplicaciones y el HW
- Actúa como intermediario entre el usuario de una computadora y el HW de esta.

<u>Objetivos</u>

- Comodidad:
  - Facilita el uso de la computadora(HW)
- Eficiencia:
  - Permite que los recursos de un sistema se puedan usar de manera eficiente.
- Evolución:
  - Se debe construir de manera que permita desarrollar, probar y agregar nuevas funciones en el sistema sin interferir con su servicio.

<u>Componentes</u>

- Kernel o núcleo:
  - Componente de SW
  - Capa que rodea el hw
  - Se encuentra en M.P.
    - Esto se debe a que es el componente mas usado por el S.O

        - Esta porque la administra.
    - Componente principal
    - Contiene las funciones del S.O. más frecuentemente usadas y otras porciones actualmente en uso
    - Porción de código del S.O. pequeña.
    - Implementa servicios esenciales:
        - Manejo de memoria
        - Manejo de CPU
        - Administracion de procesos
        - Comunicación y concurrencia
        - Gestion de e/s
- Shell:
    - Permite la interacción con el usuario.
    - Forma de interactuar con el S.O
    - En windows es el escritorio(sistema de ventanas).
    - En linux es la terminal.
    - Hace uso del kernel(este le da acceso a los programas)
- Herramientas
    - Editores, compiladores, librerías.
    - Servicio extra que da el S.O.

<u>Servicios de un S.O</u>

- Administración y planificación del procesador (se ve despues)
- Administración de memoria (otro tema)
- Administracion de almacenamiento-Sistema de archivos.
- Administración de dispositivos
- Deteccion de errores y respuestas
    - Errores de HW internos y externos
        - Errores de memoria/CPU
        - Errores de dispositivos
    - Errores de SW
        - Errores aritmeticos(division por 0)
        - Acceso no permitido a direcciones de memoria.
    - Incapacidad del SO para conceder una solicitud de una aplicación
- Interaccion del Usuario (Shell)
- Contabilidad
    - Recoger estadísticas del uso
        - Cuanto tiempo espero por algo, cuanto uso la cpu, cuanta e/s hizo un proceso.
    - Monitorear parámetros de rendimiento
    - Anticipar necesidades de mejoras futuras
    - Dar elementos si es necesario facturar tiempo de procesamiento

<u>Complejidad</u>

- Un s.o es un software extenso y complejo
- Es desarrollado por partes
    - Cada una debe ser analizada y desarrollada entendiendo

- Su función
- Cuáles son sus entradas y sus salidas.

CLASE 2- Introducción II-

Funciones principales de un SO

Problemas que un SO debe evitar

Hay determinadas cosas que un SO tiene que garantizar.

Debe evitar

- *Que un proceso de apropie de la CPU*

  Por ejemplo: si hubiera una única CPU

  - Estoy ejecutando chrome (el SO no esta presente) y este se quiere apropiar de la CPU por ejemplo con un BUCLE INFINITO
  - ¿Cómo lo evita el SO si no esta presente?
- *Que un proceso intente ejecutar instrucciones de E/S por ejemplo*

Por ejemplo hacer un programa que quiera escribir directamente en el disco sin usar archivos, sino en un bloque específico => no debe permitirlo ya que para esto estan los archivos.

- *Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado*

Esto sucede cuando un proceso intenta pasar mas alla del espacio de memoria que se le asigno, para robarle info a otro programa ejecutandose.

Para esto no se puede andar chequeando con el S.O. si un proceso puede acceder.

Para evitar esto, el SO debe (con el apoyo del HW):

- 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
  - La protección se hace con las interrupciones

Apoyo del hardware

3 grandes apoyos del hardware para cumplir con los anteriores objetivos:

- Modos de ejecución
- Interrupción de Clock
- Protección de la Memoria

Modos de ejecución

¿Qué es?

- Bit en la cpu que representa el modo en el que la cpu se esta ejecutando.

Tipos de modos de la cpu

- Modo supervisor o kernel
    - En este modo se deben ejecutar las instrucciones privilegiadas
    - Deberian ejecutarse en este modo instrucciones del kernel.
    - Toda instrucción ejecutada en este modo se puede realizar.=> se puede hacer lo que se quiera.
    - Por ejemplo instrucciones de e/s, que cambien la palabra de estado del procesador.
    - Se encarga de
        - Gestión de procesos
            - Creación y terminación
            - Planificación
            - Intercambio
            - Sincronización y soporte para la comunicación entre procesos.
        - Gestión de memoria
            - Reserva de espacio de direcciones para procesos
            - Swapping
            - Gestión y páginas de segmentos.
        - Gestión de e/s
            - Gestion de buffers
            - Reserva de canales de e/s y de dispositivos de los procesos.
        - Funciones de soporte
- Modo usuario
    - Modo restrictivo
    - No se puede interactuar con el hardware
    - Se llevan a cabo tareas que no requieran accesos privilegiados.
    - El proceso trabaja en su propio espacio de direcciones.
    - Modo acotado de lo que puede hacer
    - Instrucciones basicas (ADD, MOD, XOR, ETC).
    - Deberian ejecutarse instrucciones como de programas como chrome.

¿Cómo cambiar de modos?

Modo usuario a modo kernel

- La unica manera es a través de una interrupción.
    - Cuando sucede esta, el bit se pone en modo kernel
- Se “despierta” al S.O. con una interrupción para que este corra al programa que se esta ejecutando (en modo usuario) y comience a correr una rutina del S.O. (en modo kernel).

Modo kernel a modo usuario

- Se puede hacer ya que en este modo se hace cualquier cosa.
- El s.o. simplemente vuelve a modo usuario y da poder de ejecución al programa que se desee.

Se esta en estos cambios constantemente (constantemente ocurren interrupciones).

Asi se protege la cpu de que no se ejecute una instrucción que no se debe y le da el control al SO sobre la cpu para que resuelva lo que tenga que resolver.

¿Qué sucede cuando hay interrupciones en los diferentes modos?

- Si hay una en modo kernel
    - Por ejemplo si sucede una EXCEPCION es una falla del S.O. (quien tiene el control)=>pantalla azul.

Antes no habia varios modos de ejecución, no se protegia.

Interrupción por clock

- Se usa para evitar que se apropie un proceso de la CPU y protegerla.
- Sucede constantemente (aprox milesimas de segundos). (cada que ocurre se pasa MK)
- Constantemente "despertando" al S.O. para que controle a lo que esta usando la CPU.
- Se implementa normalmente a través de un clock y un contador.
- El kernel le da valor al contador que va decrementandose con cada tick de reloj y cuando llega a 0 puede expulsar al proceso para ejecutar otro.
- Falta

Protección de la memoria

¿Qué es?

Cada programa/proceso tiene un límite de memoria para moverse el cual no puede superar.

El S.O. es quien carga (en MK) los procesos en memoria por lo que sabe donde esta cada uno. => le indica al hardware el "registro base" y "registro limite" de cada proceso.

- Al no estar presente el SO cuando un proceso se ejecuta, la CPU controla cuando un proceso quiere acceder a una zona de memoria, si dicha zona se encuentra en los limites que le indicó el SO anteriormente=> si no esta, hay un TRAP o EXCEPCION.

<u>System calls</u>

- Forma en la que los programas de usuario le piden los servicios al SO(los cuales solo se ejecutan en MK).
- Son rutinas, funciones o procedimientos que los procesos pueden invocar para pedirles servicios a los SO.
- Tiene parametros
    - Punteros, en pila, bloques de memoria.
- Se deben ejecutar en MK.
- Ejemplo
    - Count=read(file, buffer, nbytes)
    - El read se debe hacer en MK.

¿Cómo ejecuto una SC estando en Modo usuario?

- Para ejecutarla debo estar en MK.
- Se utiliza la librería(que esta en Modo Usuario) de la API de las SC, a la cual se le hace un llamado cuando se ejecuta una instrucción que genera una SC.
- La librería es la que sabe pasar al MK.
- Esta coloca un numero de llamada a la SC y fuerza una excepción(siempre es la misma), la cual hace que se cambie a MK.
- El SystemCall Handler mira el numero de la SC para saber cual fue la que genero la interrupción, ya que siempre se usa la misma interrupción.

- Cuando termina la rutina del MK, el syscall handler pasa a modo usuario desapilando
- Vuelve a la librería
- Vuelve a la ejecucion.

¿Por qué los programas que son para linux no pueden ejecutarse en windows y viceversa?

- Porque los nros de las SC que usan son diferentes.

Categorías de SC

- Control de procesos
- Manejos de archivos
- Manejo de dispositivos
- Mantenimiento de información del sistema
- Comunicaciones

Tipos de kernel

(tipos en sentido de diseño).

- Monolítico
    - Toda funcionalidad que implementa el SO se ejecuta en modo kernel, esta todo ahí.
    - Ventajas
        - Implica menos tiempo de resolución de las cosas (menos cambios de modo): se hace todo en modo kernel y luego se pasa a modo usuario.
    - La mayoria tiene este tipo (como linux y windows)
- Microkernel
    - Se trata de que el kernel sea lo mas chico que se pueda.
    - Se busca que en el Modo Usuario se encuentren diferentes componentes que apoyen al kernel
    - Ejemplo: hay un conjunto de procesos a ejecutar por el SO
        - Se puede poner en modo usuario la selección de los procesos que se van a ejecutar(cosas que puedan estar aca).
        - En el modo kernel se pone lo que si o si debe estar ahí.
    - Ventajas:
        - Estar el menor tiempo posible en modo kernel para evitar problemas (por ejemplo pantalla azul).

Clase 3- Procesos I-

Proceso

- Es un programa (lo que esta en el disco) en ejecución.
- Es una entidad abstracta generada por el S.O. para poder ejecutar los programas que se quieren ejecutar.
- Sinónimos : tarea y job.

Diferencias entre programa y proceso.

Programa	**Proceso**
Es estático: • Es lo que se compila	Es dinámico • Cambia durante su ejecución

No tiene PC • No esta en ejecución, lo que se ejecuta es el proceso creado para ejecutarlo.	Tiene PC • Se va incrementando en el ciclo de instrucción
Existe desde que se edita hasta que se borra • Existe mientras exista el medio en el que esta almacenado(hasta que se borre)	Su ciclo de vida es desde que se solicita ejecutar hasta que termina • Nace para ejecutar el programa, transita estados (ejecuta lo que tenga que ejecutar) y termina. • Existe por el SO

<u>El modelo de Proceso</u>

Multiprogramación de 4 procesos

- Pueden estar vivos todos en memoria.
- Cada proceso tiene su PC(administrado por el S.O), historia de ejecución, son independientes.
- Aprovecha el uso del HW.
- Solo un proceso se encontrara activo en cualquier instante en el caso de tener una sola CPU.

<u>Definición de proceso</u>

- Entidad que aparece con el S.O. y definida para abstraer la ejecución
- Cápsula donde se meten programas y el S.O los ejecuta.

Un proceso para poder ejecutarse incluye como mínimo

- Sección de código(texto)
    - Sale del programa o librerias
    - No se puede modificar durante la ejecución
- Sección de datos
    - Se ponen variables
    - Se puede modificar durante la ejecución
- Stacks
    - Puede necesitar mas de una.
    - Se usa para pasar parametros, variables temporales, guardar direcciones de retorno (cuando sucede una SC o una interrupción).
    - Los procesos tienen una pila para el modo usuario y otra para el modo kernel.
    - Su tamaño se ajusta según la ejecución.
    - Formado por stack frames con parámetros de la rutina y datos necesarios para recuperar el stake frame anterior.

<u>Conjunto de atributos de un proceso</u>

- Identificación del proceso y del proceso padre
    - Todo proceso tiene un proceso padre
- Identificación del usuario que lo disparó
- Si hay estructura de grupos, el grupo que lo disparó
- En ambientes multiusuario, la terminal que lo disparó.

<u>PCB</u>

- Estructura de datos asociada al proceso
- Acá se almacenan los atributos del proceso.
- Es un gran registro donde se guardan los atributos y los punteros o información de la memoria que indica donde estan las secciones anteriormente mencionadas.
- Hay una para cada proceso
- Es lo primero que debe crear el SO para cuando crea un proceso y es lo último que se borra cuando el proceso termina.
- Contiene info asociada con cada proceso:
    - Ubicación en memoria:
        - Como esta el proceso almacenado en memoria.
        - PID, PPID
        - Valores de registros de la CPU
        - Planificación
        - Accounting
            - Cuanta memoria viene ocupando, lo máximo que ocupo, etc.
        - Entrada salida.

<u>Espacio de direcciones de un proceso</u>

- Es un conjunto de direcciones de memoria que ocupa el proceso.
- Limitan al proceso para protegerlo de otro proceso=> cada uno es una “burbuja”.
- Modelo abstracto que representa cómo un proceso utiliza la memoria.
- Son las posibles direcciones que un proceso tiene para mantener su información=> estas direcciones dependen de la arquitectura. En la información no se incluye su PCB o tablas asociadas.
- En modo usuario: el proceso accede sólo a su espacio de direcciones
- En modo kernel: se puede acceder a espacios de direcciones de otros procesos o a estructuras internas (PCB)

<u>Contexto de un proceso</u>

- Toda la información que el SO necesita para administrar el proceso y la CPU para ejecutarlo correctamente.
- Incluye:
    - Archivos que se abrieron
    - Parte de la memoria que se esta usando
    - Registros de la cpu (PC)
    - Prioridad del proceso
    - Si tiene e/s pendientes.

Cambio de Contexto (Context Switch)

- Es cuando un proceso es sacado de la ejecución para meter otro proceso
- Cuando un proceso se va a sacar, el SO debe guardar el contexto del proceso saliente antes de meter el contexto del proceso que va a tomar la CPU.
    - Lo que se almacena depende del diseño del SO.
- Pasos
    - Guardar contexto e información del proceso saliente
    - Guardar contexto e información del proceso entrante

    - Ejecutar el proceso entrante desde la instrucción siguiente a la última ejecutada en dicho contexto.
- Utilización del SO de la cpu para algo que no es una ejecución(tiempo no productivo)=> consume ciclos de cpu por lo que debe estar optimizada (depende de la arquitectura).

<u>Kernel</u>

No es un proceso

Es un conjunto de módulos de software

Presenta diferentes **enfoques de su diseño**

Enfoque 1

- Es una entidad que esta a la par de los procesos
    - Se ejecuta fuera de todo proceso
- Se ejecuta como una entidad independiente en modo privilegiado.
- Arquitectura usada por los primeros SO
- Cada vez que hay una interrupción hay un "Context Switch" porque se resguarda el contexto del proceso y luego se pasa el control al kernel => es un desperdicio de tiempo de CPU porque tiene una gran sobrecarga. (desventaja)
- El kernel tiene su propia
    - Región de memoria (desde y hasta)
    - Stack
- Cada vez que termina su actividad le devuelve al proceso el control (sobrecarga).

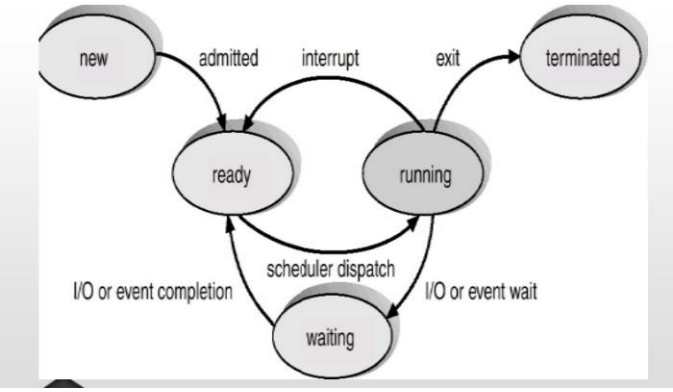
Enfoque 2

- El mismo kernel forma parte de todos los procesos y contextos de todos los procesos
    - Está dentro del espacio de direcciones de cada proceso
    - Dentro de un proceso se encuentra el codigo del programa (user) y el código de los módulos de SW del SO (kernel)
    - El código del kernel se comparte por todos los procesos
- Las interrupciones no generan Context Switch con el kernel porque este ya esta dentro del proceso => ventaja
    - Cuando sucede una se atiende en el contexto del proceso que se estaba ejecutando
    - Debe estar si o si en el modo kernel por lo que
        - Solo se salta a una dirección de memoria del mismo contexto que pertenece al kernel (asi se pasa a este modo)
            - Si se necesita, luego de atenderla vuelve al modo usuario
- El espacio de direcciones de un proceso esta acotado a la arquitectura por lo que si tengo un espacio pequeño y meto el kernel, reduzco al proceso su espacio de direcciones=> desventaja (es menos peor que la desventaja del enfoque 1)
- Se necesitan 2 stacks porque el mismo proceso se ejecuta en 2 modos diferentes
    - Una pila para modo usuario
    - Una pila para modo kernel

Clase 4 –Procesos –II -

- En su ciclo de vida pasa por diferentes estados (en cada SO hay diferentes estados pero como mínimo estan estos)

- Nuevo (new)
- Listo para ejecutar(ready)
- Ejecutándose(running)
- En espera (waiting)
- Terminado (terminated)



New

- Estado de inicialización (estado 0)
- Un usuario “dispara” el proceso
- El proceso escreado por otro proceso (Proceso padre)
- Se inicializan las estructuras de datos de cada proceso
- El proceso queda en la cola de procesos para ser cargado en memoria

Ready

- El proceso tiene todo lo necesario para ejecutarse pero le falta la CPU
    - Comienza a competir por la cpu (cola de procesos listos Ready Queue) y una vez que es seleccionado pasa a estado Running.

Running

- Durante este estado puede suceder que
    - Termine (exit)
        - Pasa al estado terminated.
    - Sea interrumpido
        - Por ejemplo a cada proceso se le da 3 segundos (Quantum o time slice) y cuando se termina el tiempo vuelve a ready (Context switch)
    - Quiera hacer una e/S
        - El proceso se saca de ejecución pero no vuelve a ready (porque para pasar a este estado debe haber terminado su tarea)
        - El proceso es llevado a un estado de espera y luego cuando termina vuelve a ready.

Terminated

- Es lo opuesto al nuevo
- Se “Desinializan” las estructuras que se inicializaron en New, se borran para liberar memoria.

Waiting

- Durante este proceso, se le da la CPU a otro proceso.
- El proceso necesita que se cumpla el evento esperado para continuar
    - Terminación de una e/s solicitada
    - Llegada de una señal por parte de otro proceso
- Una vez que el proceso sale de aca vuelve a ready para competir por la cpu (sigue en memoria pero no tiene la CPU)

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).
- Runnig-Waiting: se pone a dormir el proceso esperando por un evento.
- Waiting-Ready: terminó la espera y compite nuevamente por la CPU.
- Running- Ready:
    - Caso especial
    - Cuando el proceso termina su quantum sin haber necesitado de ser interrumpido por un evento pasa al estado de ready para competir por CPU.
    - Se da en algoritmos apropiativos

<u>Colas de planificación de procesos</u>

- Estructuras de datos que enlazan las PCB de los procesos en función del estado de ese proceso
- Van cambiando los enlaces de las PCB, no las PCB.
- Puede haber varias colas por estado
    - Por ejemplo por dispositivo (si se esperan discos una para disco1, disco2, etc)
- No siempre se comportan como una cola(que entre y salga el primero)
    - Puede estar ordenado por prioridad por ejemplo.
- Ejemplos
    - Cola de trabajos o procesos
        - Contiene todas la PCB
    - Cola de procesos listos
        - PCB de procesos residentes en memoria principal esperando para ejecutarse
    - Cola de dispositivos
        - PCB de procesos esperando por un dispositvio de I/O

<u>Modulos de planificación</u>

- Son partes del kernel que estan relacionadas a la planificación de los procesos en relación a sus estados
- Los planificadores se ejecuntan ante ciertos eventos:
    - Creación/Terminación de procesos
    - Ecentos de sincronización o de E/S.
    - Finalización de lapso de tiempo.

Existen diferentes(pueden estar implementados juntos o separados)

- Scheduler de long term
- Scheduler de short term
- Scheduler de medium term

Su nombre proviene de la frecuencia de ejecución.

Hay más

- Dispatcher
- Loader.

Scheduler de long term

- Realiza la actividad de admitir procesos de New A Ready (admitted)
    - Indica que proceso se selecciona para cargarlo y que esté en estado de Ready
- Esta ligado al loader
    - Es quien carga el espacio de direcciones del proceso seleccionado.

Scheduler de short term

- Selecciona entre los procesos en Ready el que va a tomar la CPU (de Ready a Running)
- Agarra al primero que está en la cola
- Ligado al dispatcher
    - Es e que hace el context switch

Medium term scheduler

- Si es necesario reduce el grado de multiprogramación(cantidad de procesos en memoria listos para ejecutarse)
    - Se trata de tener un alto grado pero esto siempre no es bueno
- Saca un proceso de memoria para bajar el grado porque se necesita hacer eso
- Se encarga de seleccionar el proceso que va a subir (de los que quedaron en el estado cando bajaron)
- Distingue el estado Suspend el cual
    - Tiene a los procesos que tiene su información cargada pero no fueron cargados en memoria

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 M.P. antes de 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 CS para darle la CPU a otro proceso
8. Proceso recientemente creado y en transición
    - Existe pero aun no esta listo para ejecutar ni esta dormido
9. el proceso ejecutó la SC exit y esta en ESTADO ZOMBIE

- Ya no existe mas pero se registran datos sobre su uso. Es el estado final

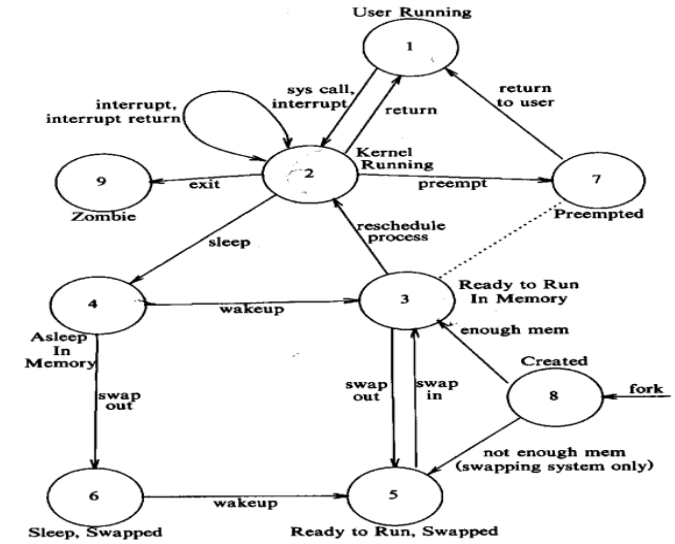
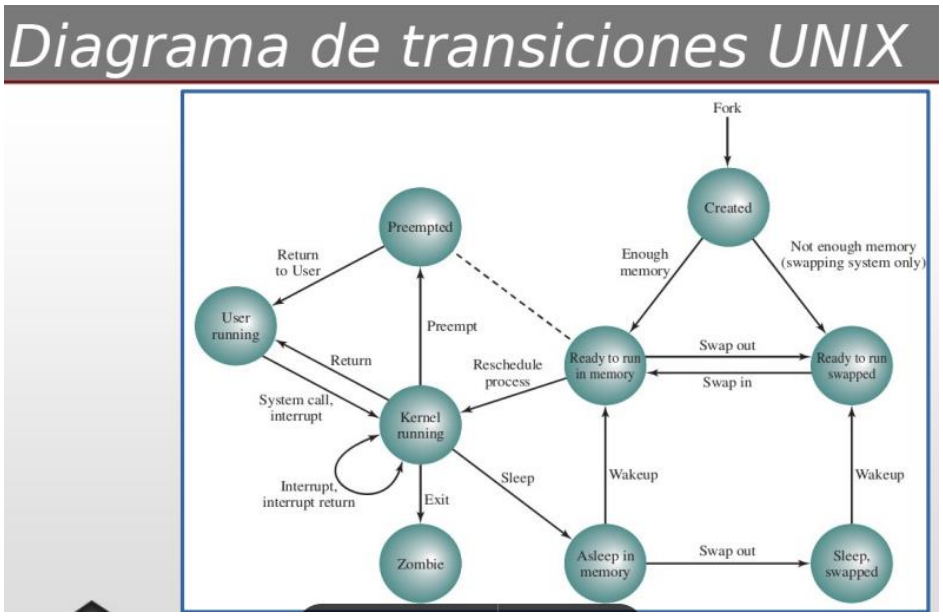


Figure 6.1. Process State Transition Diagram



Clase 5-procesos III-

<u>Comportamiento de los procesos</u>

La selección del algoritmo de planificación depende del tipo del S.O. que se ejecuta

- Hay diferentes S.O. con diferentes objetivos
    - Esta diferencia esta dada por el comportamiento de los procesos
- Cuando se piensa un algoritmo se piensa la naturaleza de los procesos que quieren ejecutarse en ese algoritmo
    - CPU-bound
        - Durante su tiempo de vida requieren mayor uso de la CPU porque hacen cómputos o calculos

    - I/O-bound
        - Mayor parte del tiempo esperando por I/O porque hacen mucha sobrecarga de archivos, procesos de archivos, etc
- La velocidad de la CPU es mucho mas rapida que la de los dispositivos de E/S.

<u>Planificación</u>

- Determinar cuál de esos procesos listos para ejecutarse va a tomar la CPU cuando llegue el momento de hacerlo => en un ambiente multiprogramado.
- El sistema se planifica a través del ALGORITMOS DE PLANIFICACIÓN, pueden ser
    - Apropiativos
    - No apropiativos

Algoritmos apropiativos

- Cuando un proceso se esta ejecutando en la CPU, se lo puede expulsar de esta más alla de lo que esté haciendo el proceso
    - Por ejemplo porque se le terminó el tiempo o porque llega un proceso de mayor prioridad

Algoritmos no apropiativos

- Un proceso ejecutandose se van de la CPU por su voluntad (porque termina o porque quiere hacer e/s)

Categorias de algoritmos de planificación

Todos los algoritmos tienen un objetivo principal: productividad del recurso (la CPU),pero hay otras metas correspondientes a diferentes algoritmos:

- 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 por lotes (batch)

- Lo opuesto a interactivos
- Por ejemplo a la noche un banco hace un balance de cuentas
- No hay un usuario esperando una respuesta en una terminal
    - El proceso inicia, se ejecuta y termina => NO APROPIATIVO
- Metas :
    - Rendimiento
        - Maximizar el nro de trabajos por hora
    - Tiempo de retorno
        - Minimizar los tiempos entre el comienzo y la finalización
    - Tiempo en espera puede verse afectado

    - Uso de la cpu
        - Mantenerla ocupada la mayor cantidad de tiempo posible
- Ejemplos :
    - FCFS
    - SJF
        - Mejor prestacion que el FCFS porque tiene menor promedio

Procesos interactivos

- Es de interacción tanto con usuarios como servidores
    - El servidor necesita vaarios procesos para dar rta a diferentes requerimientos
- Son necesarios algoritmos apropiativos para evitar que un proceso se quede con la CPU
- Metas:
    - Tiempo de rta
        - Responder a peticiones con rapidez
    - Proporcionalidad
        - Cumplir con expectativas de los usuarios
            - Si el usuario pone STOP al reproductor, que la musica deje de ser reproducida en un tiempo considerablemente corto
- Ejemplos
    - Round Robin
    - Prioridades
    - Colas multinivel
    - SRTF

Politica VS Mecanismo

- Situaciones en las que es necesario que la planificación de 1 o + procesos se comporte de manera diferente
- El algoritmo de planificación debe estar parametrizado para que los procesos/usuarios puedan indicar como se modifica la planificación.
- El kernel implementa el mecanismo (es el mismo para todos)
- El usuario/proceso/administración utiliza los parametros para determinar la politica
- La politica depende de la version (va a variar el Q)
    - Si es de uso general va a tener un Q corto (Q=3) para garantizar la interactividad
    - Si es server va a tener un Q largo

Clases 6-Procesos 4-

<u>Creación de procesos</u>

- Un proceso siempre es creado por otro proceso
- El proceso 0 nace ya creado (no lo crea otro proceso)
    - El loader carga una imagen de un proceso ya existente.
- Se forma un arbol de procesos
    - Cada proceso tiene un padre
    - Un proceso puede tener 0,, 1 o + hijos

Actividades en la creación

- Crear la PCB

    - Se crea cuando se realiza la SC que crea un proceso
- Asignar PID(Proceess IDentificatio) unico
- Asignar memoria para regiones
    - Stack, text y datos
- Crear estructuras de datos asociadas.
    - Fork

Relación entre procesos Padre e Hijo

- Respecto a la ejecución hay 2 modelos
    - Cuando un proceso padre crea un hijo, continua ejecutandose concurrentemente con su hijo=> AMBOS COMPITEN POR LA CPU
    - Cuando un proceso padre crea un hijo, espera a que este termine para poder seguir ejecutandose.
- Respecto al espacio de direcciones
    - En unix: cuando el proceso padre crea un hijo, este es un duplicado del padre
        - Lógicamente el espacio de direcciones es una copia del espacio del padre.
    - En windows: cuando el proceso padre crea un hijo, el espacio de direcciones de este es uno nuevo, vacío

Creación de Procesos

En UNIX

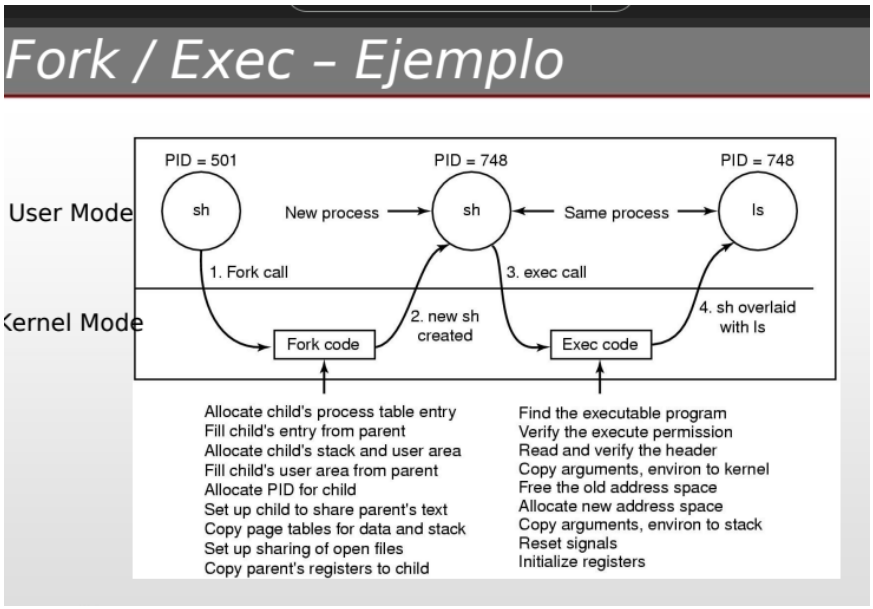
- SC fork()
    - Crea un nuevo proceso
    - No necesariamente pasa antes del execve
    - El proceso es igual al padre
        - Copia del espacio de direcciones
        - Copia los registros
            - El PC es igual por lo que estan en el mismo punto de ejecucion
    - Unica SC con un comportamiento particular
    - Devuelve un numero (1 para el padre y 1 para el hijo, se pregunta cual es para saber donde estoy)
        - >0 = es el ID del proceso hijo que se creo (estoy en el padre)
        - <0 = falló, el hijo no existe
        - = 0 Estoy en el hijo
    - Terminación de procesos
        - EXIT
            - Se retorna el control al S.O.
            - El padre puede esperar recibir un codigo de retorno (x wait)
            - Se lo usa cuando se requiere que el padre espere a los hijos
        - KILL
            - El padre puede terminar la ejecución de sus hijos
            - La tarea asignada al hijo terminó
            - Cuando el padre termina su ejecución:
                - Normalmente no se permite a los hijos continuar pero puede permitirse
                - Terminación en cascada

- SC execve()
    - Generalmente se usa despues del fork
    - No necesariamente tiene que pasar despues de un fork
    - Carga un nuevo programa en el espacio de direcciones

En windows

- Solo una SC CreateProcess()
    - Crea un nuevo proceso y carga el programa para ejecución

Ejemplo de shell

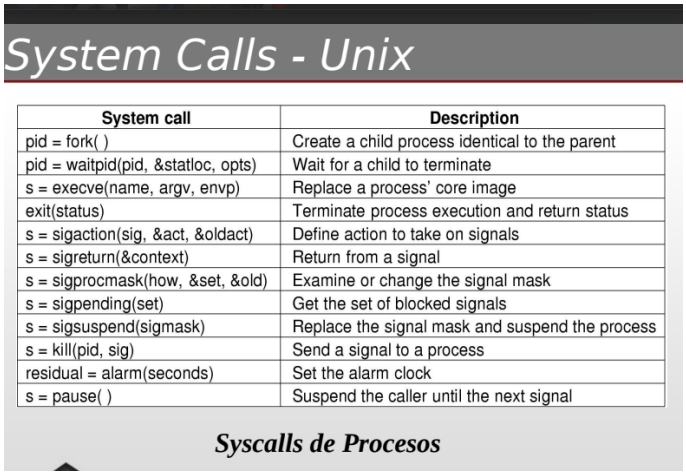


1. Sucede un fork
    a. Tengo un nuevo proceso que tmb esta ejecutando bash (PID=728)
2. Ejecuto exec
    a. No cambia el proceso (sigue siendo 748) sino el espacio de direcciones.

Procesos cooperativos e independientes

- Independiente
    - Programar una solución a un problema a partir de un conjunto de un procesos
    - Todos los procesos trabajan en conjunto para obtener una solución
    - Afecta o e afectado por la ejecución de otros procesos en el sistema
    - Necesita de herramientas del S.O por ejemplo que se avise cuando un proceso terminó
    - Cambia el modelo de programación
    - Beneficios
        - Acelerar el computo
            - Separar tarea en subtareas que cooperan ejecutandose paralelamente
        - Compartir informacion
            - Un archivo
        - Planificar tareas de manera tal que se puedan ejecutar en paralelo

- Independiente
    - Todo lo opuesto a cooperativo
    - El proceso no afecta ni puede ser afectado por la ejecución de otros procesos
    - *No comparte ningún tipo de dato*
    - *Por ejemplo:*
        - *Abro un archivo y cuento los caracteres que tiene y lo cierro*



Clase 7-Memoria I-

<u>Memoria</u>

- La organización y adminstraciónde la M.P. es uno de los factores + importantes en el diseño del S.O.
    - El espacio de direcciones debe estar en M.P. para que el proceso se pueda ejecutar.
- Los programas y datos deben estar en almacenamiento principal para
    - Poderlos ejecutar
    - Referenciarlos directamente

El s.o. debe

- Llevar un registro de las partes de memoria que se estan usando y las que no
    - Responsable de que los espacios de direcciones lógicos esten en la ram

    - Debe saber donde estan, el estado de la RAM(lo que esta libre o no), quien lo esta ejecutando (si es código o datos),
- Asignar espacio en M.P a los procesos cuando estos la necesitan.
- Liberar espacio de memoria asignada a los procesos que terminaron.
- Lograr que el programador se abstraiga de la alocación de los programas
    - No tenemos que preocuparnos sobre en qué lugar de la memoria va a ir un programa
    - Nos abstraemos de la ubicación física en la RAM
- Brindar seguridad en los procesos apra que unos no accedan a secciones privadas de otros
- Brindar posibilidad de acceso compartido a determinadas secciones de la memoria
    - Por ejemplo si 2 procesos quieren usar word, que el codigo de word no este 2 veces en RAM sino que sea un código común
- Garantizar la performance del sistema

Se espera de un S.O. un uso eficiente de la memoria con el objetivo de alojar el mayor nro de procesos.

Administración de Memoria

- División lógica de la Memoria Física para alojar múltiples procesos
    - Garantizando protección entre los multiples procesos
    - Esta forma de usar la memoria fisica depende del mecanismo de administración del HW
        - El s.o no inventa como administrarla sino que el HW tiene una forma de hacerlo
        - => el s.o se adapta y los programadores de S.O tienen que hacer que se use de la forma + eficiente.
- Asignación eficiente
    - Contener la mayor cantidad de procesos para garantizar el mayor uso de la CPU para estos.

Requisitos

- Reubicación
    - Mientras un proceso se ejecuta puede ser sacado y traído a la memoria (SWAP) y posiblemente colocarse en diferentes direcciones cuando se vuelve a traer
    - El programador no debe ocuparse de conocer donde será colocado el proceso en RAM
    - Las referencias a memoria se deben traducir según la ubicación actual del proceso
    - Ligado a que lo haga el HW
- Protección
    - Ligado al HW(debe controlar que esa dirección que se quiere usar no corresponda a otro proceso)=> no seria eficiente que el S.O. se ponga a analizar que la dirección corresponda a otro proceso o no
    - Los procesos no deben referenciar/acceder a direcciones de memoria de otros procesos (salvo que tengan permiso)
    - El chequeo debe realizarse durante la ejecución
        - No es posible anticipar todas las referencias a memoria que un proceso puede hacer.
- Compartición

- Permitir que varios procesos compartan/accedan a la misma porción de memoria
  - El ejemplo del word
- Permite un mejor uso/aprovechamiento de la RAM para evitar copias repetidas de instrucciones.

Abstracción- Espacio de direcciones

- Todas las posibles direcciones(rango) a memoria que un proceso puede usar para direccionar sus instrucciones y datos.
- Logicamente el proceso puede direccionar tantas direcciones como el bus o la arquitectura le permita
  - Si un proceso se ejecuta en una arquitectura de 32 bits=> 0.. 2^32 – 1
  - Si un proceso se ejecuta en una arquitectura de 64 bits=> 0.. 2^64 –1
- Es independiente de la ubicación real del proceso en la RAM
  - La direccion 100 en el espacio de direcciones del proceso, no necesariamente es la direccion 100en la ram.

El codigo del S.O tambien esta en todos los espacios de direcciones de todos los procesos.

Direcciones

Lógicas

- Son aquellas que hacen referencia a los espacios de direcciones => referencia una dirección en el Espacio de direcciones
- Son las que usan los procesos

Físicas

- Referencia una localidad en la memoria física ram
- Se accede a través de direcciones físicas=> cada dirección lógica se tiene que traducir a una dirección física para acceder a la RAM: CONVERSIÓN DE DIRECCIONES

Conversión de direcciones

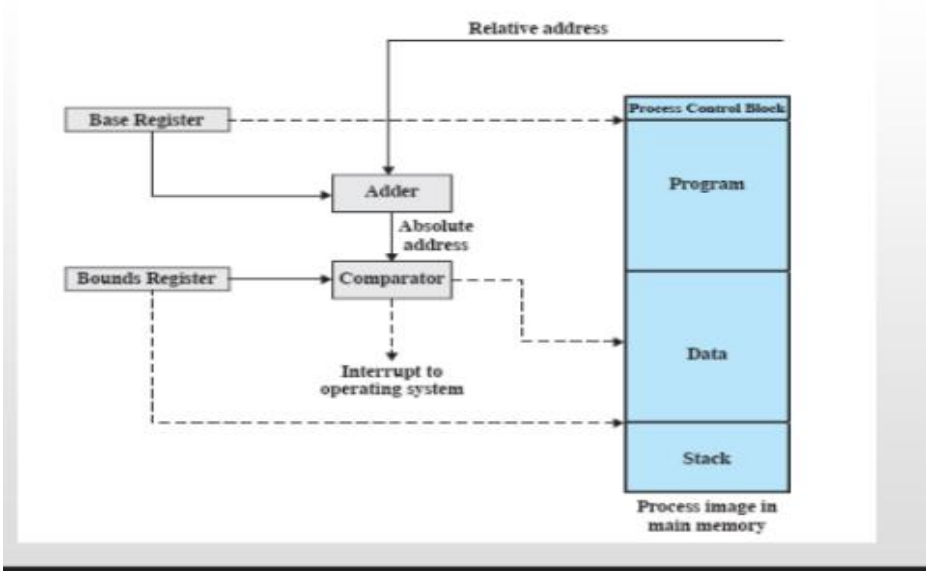
La forma simple es usando registros auxiliares:

- Registro base
  - Direccion de comienzo del Espacio de direcciones del proceso en la RAM
- Registro limite
  - Direccion final del proceso o tamaño de su Espacio de Direcciones

Ambos valores se fijan cuando se carga el espacio de direcciones del proceso, en memoria

La dirección lógica se interpreta como un desplazamiento que se va a ir sumando al registro base

- Sumo dirección física al registro base (ADDER)
- (comparator)Si esa suma es menor al registro límite, esa dirección lógica se transforma en una dirección física.(data)
- (comparator)Si esa suma es mayor al registro límite hay una interrupcion por software al S.O. para que este decida que hacer con el proceso que hizo un ACCESO INVALIDO A MEMORIA(interrupt to operating system)



Direcciones lógicas vs físicas

- Las direcciones con las que trabaja la CPU son lógicas
    - Por ejemplo ADD #10, 40
    - La cpu toma esa dirección 10 como una lógica, la tiene que traducir en una dirección física para llegar a ram y poder ejecutar la instrucción=>RESOLUCIÓN DE DIRECCIONES(adress-binding).
- La resolución se puede hacer en 2 momentos
    - Momento de compilación
        - Direccion logica= dirección fisica
        - Ya no hay s.o. con este tipo.
        - Se podian pisar 2 procesos con la misma dirección
    - Tiempo de carga
        - Cuando el espacio de direcciones se carga en la ram, convierto todas las direcciones.
        - Permite que la compilación sea independiente pero tiene la desventaja que
            - Si un proceso se carga no se puede descargar o hay que recargarlo en el mismo lugar
    - Tiempo de ejecución
        - Direccion física y lógica son diferentes
        - Las direcciones lógicas son llamadas “direcciones virtuales”
        - La reubicación se puede hacer fácilmente
        - La cpu traduce la dirección lógica a una física para poder acceder a la dirección lógica
        - Tiene que acompañar a la velocidad del procesador
        - Las arquitecturas tienen MMU(Unidad de manejo de traduccion)
            - Antes de que la dirección salga por el bus para acceder a la ram, MMU hace la suma y comparación.

MMU

- Dispositivo de hardware (muy ligado a la CPU, suele estar en el mismo chip) al que se le dan datos cuando hay un Context Switch

    - Registro base y limite
- Se reprograma solo en modo kernel
- Suma a la direccion lógica y obtiene la direccion física=> mapea direcciones virtuales a fisicas
- Los procesos nunca usan direcciones fisicas, lo que hace es sumar a cada direccioón generada por el proceso, el valor del "registro de locación"

<u>Mecanismos de asignación de memoria</u>

Particion: el proceso esta a partir de una dirección hasta un limite. Se va particionando la memoria RAM

Formas de particionar:

Particiones fijas

- De antemano el S.O. divide lógicamente a la memoria RAM con un tamaño fijo
- Todas pueden ser del mismo tamaño o no
- Cada una aloja un proceso, el cual se coloca en alguna según algun criterio
    - *First Fit*
    - *Best Fit*
    - *Worst Fit*
    - *Next Fit*
- Genera fragmentación interna

Particiones dinámicas

- A medida que los procesos aparecen, la memoria se va particionando según el tamaño del espacio de direcciones del proceso
- Varian en tamaño y nro
- Cada una aloja un proceso
- Genera fragmentación externa

<u>Fragmentación</u>

Se produce cuando una localidad de memoria no se puede usar porque no esta de forma contigua

Fragmentación interna

- Es la porción de partición que queda sin usar

Fragmentación externa

- Son huecos que van quedando en la memoria a medida que los procesos finalizan
- Al no encontrarse de forma contigua puede estar el caso de que haya memoria libre para alocar un proceso pero que no se pueda usar
- Solución:
    - Compactacion:
        - Unir huecos libres en un unico hueco grande para que entre el espacio de direcciones
        - Para ir reacomodando el espacio de direcciones se necesita un almacenamiento secundario para ir cargando y descargandolos ahí
        - Es muy costoso

El hardware limitaba al S.O. a tener los espacios de direcciones continuos por el tema del registro base y limite

Problemas del esquema de registro base + limite

- Necesidad de almacenar el Espacio de Direcciones de forma continua en Memoria Fisica
- Los primeros SO definian particiones fijas de memoria luego evolucionaron a particiones dinamicas
- Fragmentación
- Mantener partes del proceso que no son necesarias
- Ya no se usan las particiones fijas y dinámicas

Solucion:

- Paginación
- Segmentación

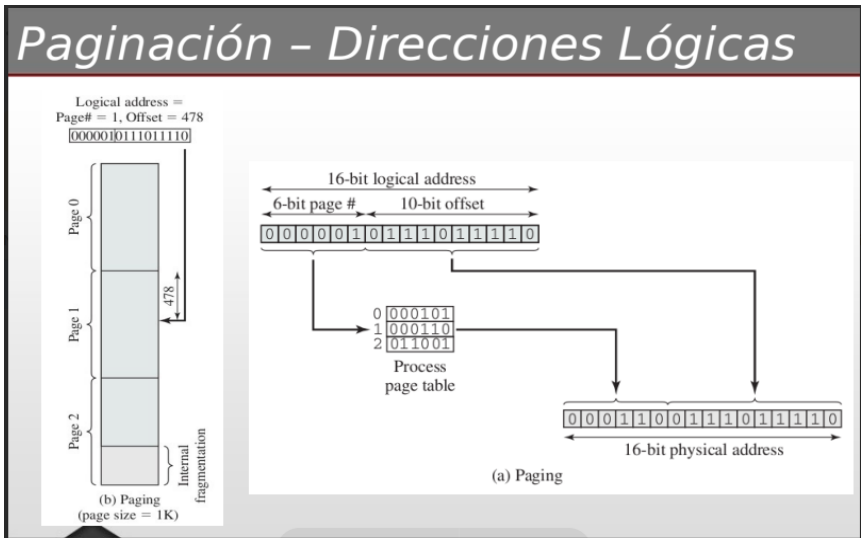
Paginación

- La memoria fisica se divide lógicamente en pequeñas partes de igual tamaño => Marcos
- La memoria lógica (espacio de direcciones) se divide en partes de igual tamaño que cada marco => Páginas
- Tabla de páginas
    - Se usa para la base
    - El S.O. carga las paginas del proceso en los marcos, a partir de acá se crea la tabla donde se guarda el numero de marco donde fue a parar la página
    - El S.O. debe mantener la tabla de páginas y conocer el estado de c/ marco (libre o no) para poder cargar una pagina en uno disponible
    - Le sirve a la MMU para traducir una dirección lógica a una física
- Cualquier pagina del espacio de direcciones puede ir a parar a cualquiera de los marcos:
    - Se rompe la continuidad
- La dirección lógica se interpreta como un nro de página y un desplazamiento dentro de la misma.
- Ya no hay registro base, el limite es el tamaño
- Para la base se usa la TABLA DE PÁGINAS
    - Hay una para cada proceso
    - Carga las paginas del proceso en los marcos, ahí se crea una tabla de paginas donde se guarda el nro de marco donde fue a parar la pagina
    - El s.o carga las paginas y mantiene la tabla de paginas junto con el estado de cada marco (si esta ocupado o no) pq cuando tiene que cargar una pagina busca un marco libre
    - Esta tabla le sirve a la MMU para traducir una dirección lógica a una física.
- Puede generar fragmentación interna pero solo en la última página

Ejemplo

- A la direccion se la interpreta como dos valores
    - Los primeros 6 bits son el nro de pagina y el resto son un desplazamiento dentro del página (apartir del comienzo de esta)
    - La mmu agarra la direccion y la interpreta de esa manera

- SACA LOS 6 BITS Q REPRESENTA LA PAGINA Y LOS CAMBIA POR LOS 6 BITS (a partir de la tabla de paginas) Q REPRESENTAN LA BASE DEL MARCO Y SE GENERA LA DIRECCION FISICA (solo se cambian los primeros 6 bits por los del marco, dp quedan los mismos bits de desplazamiento



Segmentación

- Se piensa cada parte de un espacio de direcciones como un segmento
- Cada segmento se puede cargar al espacio de direcciones de manera discontinua
- El segmento se mueve
- Todo lo que esta en un segmento hay cosas similares
    - Un segmento de datos no tiene codigo por ejemplo
- Cada uno tiene un tamaño
- Se carga en memoria RAM buscando espacios libres para cargarlo
- No hay BASE ni LIMITE=> tabla de segmentos
    - Guarda la direccion base de donde se cargo el segmento
    - Guarda el limite para que cuando se traduzca una direccion, se controle que no se pase del segmento que le corresponde
- Puede generar fragmentacion externa
    - La ram carga y descarga segmentos por lo que quedan huecos de forma dinamica.
- Todos los segmentos de un programa pueden NO tener el mismo tamaño (códigos, datos, rutinas). La base y límite del segmento son dinámicos.
- Las direcciones lógicas consisten en 2 partes:
    - Selector de segmento.
    - Desplazamiento dentro del segmento (sobre registro base y límite).
- La segmentación posee ventaja sobre la paginación respecto de: la protección de espacios de memoria y la compartición de bloques de memoría.
- Cuando uno compila → el compilador deja huellas del segmento.
- **Segmentación paginada (mix de las dos anteriores):**
- Paginación
    - Transparente al programador.
    - Elimina fragmentación externa.
- Segmentación

    - Visible al programador.
    - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
- Segmentación paginada: cada segmento es dividido en páginas de tamaño fijo.
    - Cada segmento tiene una tabla de páginas que le corresponde.
    - Toma las ventajas de ambas: compartición, protección(de parte de la segmentación), evitar fragmentaciones (de parte de la paginación).
    - Se sigue guardando en paginas
- La unidad de trabajo para subir o bajar de la RAM es la pagina