***Libros: Stallings - Silberschatz***

***Anexo de Arquitectura***

**Elementos Básicos de una computadora**

-Procesador

-Memoria Principal

\*Volátil

\*Se refiere como memoria real o primaria

-Componentes de E/S

\*Dispositivos de memoria secundaria

\*Equipamiento de comunicación

\*Monitor / teclado / mouse

-Bus Sistema

Comunicación entre procesadores, memoria, dispositivos de E/S

**Registros del Procesador**

-Visibles por el usuario

-Registros que pueden ser usados por las aplicaciones

-Pueden ser referenciados por lenguaje de máquina

-Disponible para programas/aplicaciones

-Tipos de registros

\*Datos

\*Direcciones

\*Index

\*Segment pointer

\*Stack pointer

-De Control y estado

-Para control operativo del procesador

-Usados por rutinas privilegiadas del SO para controlar la ejecución de procesos

-Program Counter (PC)

\*Contiene la dirección de la proxima instrucción a ser ejecutada

-Instruction Register (IR)

\*Contiene la instrucción a ser ejecutada

-Program Status Word (PSW)

\*Contiene códigos de resultado de operaciones

\*Habilita/deshabilita Interrupciones

\*Indica el modo de ejecución (Supervisor/usuario)

**Ciclo Ejecución de Instrucción**

Dos pasos:

-Procesador lee la instrucción desde la memoria (1)

-Procesador ejecuta la instrucción (2)

Instrucción: Fetch y Execute

El procesador busca (fetch) la instrucción en la memoria – (PC) → IR

El PC se incrementa después de cada fetch para apuntar a la próxima instrucción –PC = PC + 4

La instrucción referenciada por el PC se almacena en el IR y se ejecuta

Categorías de instrucciones

Procesador – Memoria: Transfiere datos entre procesador y memoria

Procesador - E/S: Transfiere datos a/o desde periféricos

Procesamiento de Datos: Operaciones aritméticas o lógicas sobre datos

Control: Alterar secuencia de ejecución

**Interrupciones**

-Interrumpen el secuenciamiento del procesador durante la ejecución de un proceso Dispositivos de E/S más lentos que el procesador

\*Procesador debe esperar al dispositivo

Clases de interrupciones

Programa: generadas por alguna condición que ocurre como resultado de una operación ejecutada, como desbordamiento aritmético, división por cero, intentos de ejecutar una instrucción de maquina ilegal o que una referencias este fuera del espacio de memoria permitido del usuario.

Timer: generada por un temporizador dentro del procesador. Esto permite, al sistema que se encuentra operando, realizar ciertas operaciones en una base regular.

I/O: generada por un controlador I/O, para señalizar el fin normal de una operación, o también para mostrar una variedad de condiciones de error.

Error de HardWare: generada por un error, como por ejemplo un corte de electricidad o un error de paridad de la memoria.

**Necesidades del SO**

-Postergar el manejo de una interrupción en momentos críticos

-Atender en forma eficiente: la rutina de atención adecuada según el dispositivo

-Tener varios niveles de interrupción para que el SO pueda distinguir entre interrupciones de alta prioridad y de baja prioridad y responder adecuadamente

Interrupt Handler

-Programa (o rutina) que determina la naturaleza de una interrupción y realiza lo necesario para atenderla.

\*Por ejemplo, para un dispositivo particular de E/S

-Generalmente es parte del SO

**Ciclo de interrupción**

-El procesador chequea la existencia de interrupciones.

-Si no existen interrupciones, la próxima instrucción del programa es ejecutada

-Si hay pendiente alguna interrupción, se suspende la ejecución del progama actual y se ejecuta la rutina de manejo de interrupciones

**Interrupciones no enmascarables y enmascarables**

-La mayoría de las CPUs tienen dos líneas de requerimiento de interrupciones: la de no enmascarables y las enmascarables.

-La de no enmascarables se reservan para eventos tales como errores de memoria no recuperables.

-La de enmascarables puede ser “apagada” por la CPU si en ese momento se está ejecutando una secuencia crítica de instrucciones. Estas son las que usan los controladores de dispositivo cuando necesitan servicio.

En caso de múltiples interrupciones:

-Deshabilitar las interrupciones mientras una interrupción está siendo procesada

-Definir prioridades a las interrupciones

-Actuar en base a esto último (algunas interrupciones tienen mayor prioridad que otras)

***Procesos I***

*Definición de proceso*

Es un programa en ejecución

Para nosotros serán sinónimos: tarea, job y proceso

Entidad que cobra vida en el SO cuando un usuario quiere ejecutar un programa

*Diferencias entre un programa y un proceso*

*Programa:*

Es estático (tiene el codigo y las definiciones de inicialización de las variables)

No tiene program counter

Existe desde que se edita hasta que se borra

Proceso:

Es dinámico

Tiene program counter (posición de la instrucción que se esta ejecutando)

Programa en ejecución

Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina

*El Modelo de Proceso*

*Multiprogramado:* pueden estar vivos un conjunto de procesos, donde cada proceso tiene su historia de ejecución, son independientes, tienen su pc.

Solo un proceso se encontrara activo en cualquier instante. (Si tenemos una sola CPU)

***Componentes de un proceso***

Proceso: Entidad de abstracción, que en un SO se define para abstraer la ejecución – Permite la idea de ejecutar programas

Un proceso (para poder ejecutarse) incluye como mínimo:

Sección de Código (texto) sale del programa o librerías - Sección de Datos (variables globales) - Stack(s) (datos temporarios: parámetros, variables temporales y direcciones de retorno)

***Stacks***

Un proceso cuenta con 1 o más stacks (una para cada modo)

– En general: modo Usuario y modo Kernel

Se crean automáticamente y su medida se ajusta en run-time.

Está formado por stack frames que son pushed (al llamar a una rutina) y popped (cuando se retorna de ella)

El stack 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***

Identificación del proceso, y del proceso padre (todo proceso es creado por otro proceso)

Identificación del usuario que lo “disparó”

Si hay estructura de grupos, grupo que lo disparó

En ambientes multiusuario, desde que terminal y quien lo ejecuto.

***Process Control Block (PCB)***

Bloque de control de procesos (soy hincha de Boca)

Estructura de datos asociada al proceso (abstracción)

Existe una por proceso.

Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina (porque es donde puedo ver toda la información del proceso)

Contiene la información asociada con cada proceso:

– PID, PPID, etc

– Valores de los registros de la CPU (PC, AC, etc)

– Planificación (estado, prioridad, tiempo consumido, etc)

– Ubicación (representación) en memoria

– Accounting

– Entrada salida (estado, pendientes, etc)

***Qué es el espacio de direcciones de un proceso?***

Es el conjunto de direcciones de memoria que ocupa el proceso

Modelo abstracto que representa la organización del proceso y como es su memoria, donde están los datos, que parte se puede escribir, etc. Define los límites del proceso.

Las posibles direcciones (apoyadas en la arquitectura) que un proceso puede usar para mantener su información.

• stack, text y datos

No incluye su PCB o tablas asociadas

Un proceso en modo usuario puede acceder sólo a su espacio de direcciones;

En modo kernel, se puede acceder a estructuras internas (PCB del proceso por ejemplo) o a espacios de direcciones de otros procesos.

***El contexto de un proceso***

Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.

Son parte del contexto, los registros de CPU, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc

Sirve cuando ocurre que un proceso es sacado de la ejecucion por otro(cambio de contexto o context switch), donde el SO guarda todo el contexto del proceso que va a salir, antes de cambiar la información del nuevo proceso.

***Cambio de Contexto (Context Switch)***

\* Se produce cuando la CPU cambia de un proceso a otro.

\* Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará después a la CPU.

\* Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto.

Que se guarda depende el S0.

- Es tiempo no productivo de CPU

-El tiempo que consume depende del soporte de HW

***Sobre el Kernel del Sistema Opetativo***

Es un conjunto de módulos de software

Se ejecuta en el procesador como cualquier otro proceso

Entonces: • ¿El kernel es un proceso? Y de ser así ¿Quién lo controla?

Diferentes enfoques de diseño

***Enfoque 1 – El Kernel como entidad independiente***

El Kernel se ejecuta fuera de todo proceso (o a la par)

Es una arquitectura utilizada por los primeros SO

Cuando un proceso es “interrumpido” o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del sistema operativo. (context switch)

El Kernel tiene su propia región de memoria

El Kernel tiene su propio Stack

Finalizada su actividad, le devuelve el control al proceso (o a otro diferente)

Importante:

– El Kernel NO es un proceso. EL concepto de proceso solo se asocia a programas de usuario

– Se ejecuta como una entidad independiente en modo privilegiado

***Enfoque 2 – El Kernel “dentro” del Proceso***

El “Código” del Kernel se encuentra dentro del espacio de direcciones de cada proceso.

El Kernel se ejecuta en el MISMO contexto que algún proceso de usuario

El Kernel se puede ver como una colección de rutinas que el proceso utiliza

**Que beneficio presenta?** Una interrupción no generaría un cambio de contexto, porque el sistema operativo ya está dentro del contexto, por lo que el context switch seria saltar desde la dirección del modo usuario del proceso hacia una dirección de memoria del mismo contexto, pero que pertenece al modo kernel. En resumen, no hace falta cambiar el contexto como el caso anterior.

Dentro de un proceso se encuentra el código del programa (user) y el código de los módulos de SW del SO (kernel)

Cada proceso tiene su propio stack (uno en modo usuario y otro en modo kernel)

El proceso es el que se Ejecuta en Modo Usuario y el kernel del SO se ejecuta en Modo Kernel (cambio de modo)

El código del Kernel es compartido por todos los procesos

• En administración de memoria veremos el “como”

Cada interrupción (incluyendo las de System Call) es atendida en el contexto del proceso que se encontraba en ejecución

• Pero en modo Kernel!!! (se pasa a este modo sin necesidad de hacer un cambio de contexto completo)

• Si el SO determina que el proceso debe seguir ejecutándose luego de atender la interrupción, cambia a modo usuario y devuelve el control. Es más económico y performante

***Procesos II***

***Estados de un proceso***

En su ciclo de vida, un proceso pasa por diferentes estados.

**Nuevo (new)** Inicialización de estructuras hasta que el proceso está completo (esto lo marque el hecho de estar cargado en memoria, el proceso y la idea del espacio de direcciones) (Proceso admitido)

**Listo para ejecutar (ready):** Proceso tiene todo lo necesario para ejecutarse pero le falta la CPU, comienza el proceso de competencia por CPU, cuando es seleccionado pasa al estado, hay un planificador que lo va a seleccionar, ocurre el context switch y pasa a

**Ejecutándose (running)** pueden pasar un conjunto de cosas, que el proceso termine (exit), pasa al estado desintegración o borrado/terminated (para liberar memoria), donde se borra la PCB. Otro camino puede ser la ocurrencia de una interrupción, donde el proceso vuelve al estado de ready o ready to run, donde hay un cambio de contexto del que se va y otro por el que viene. El último caso sería que quisiera ocurrir una entrada/Salida, donde no se requiere de la CPU, ese proceso es sacado de ejecución pero no es vuelto a ready, se lo pone en un estado de espera y cuando esa operación se concreta, entonces ahí lo lleva de nuevo al estado ready.

**En espera (waiting)**

**Terminado (terminated**)

***Colas en la planificación de procesos***

Estructuras de datos que enlazan/relacionan PCBS en función del estado de cada proceso. Esto significa que hay una o varias en cada estado (5 anteriores)

La PCB se crea en un lugar concreto de la memoria, y son enlazadas por las colas de prioridad

Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo

Las PCB se enlazan en Colas siguiendo un orden determinado

Ejemplos

Cola de trabajos o procesos

\*Contiene todas las PCB de procesos en el sistema

Cola de procesos listos

\*PCB de procesos residentes en memoria principal esperando para ejecutarse Cola de dispositivos

\*PCB de procesos esperando por un dispositivo de I/O

***Módulos de la planificación***

Partes del kernel que están relacionados a la planificación de los procesos en función de sus estados.

Son módulos (SW) del Kernel que realizan distintas tareas asociadas a la planificación.

Se ejecutan ante determinados eventos que así lo requieren:

Creación/Terminación de procesos

Eventos de Sincronización o de E/S

Finalización de lapso de tiempo ⎭Etc

-Scheduler de long term

-Scheduler de short term

-Scheduler de medium term

Su nombre proviene de la frecuencia de ejecución.

Otros módulos: **Dispatcher y Loader**.

Pueden no existir como módulos separados de los schedulers vistos, pero la función debe cumplirse.

Dispatcher: hace cambio de contexto, cambio de modo de ejecución...”despacha” el proceso elegido por el Short Term (es decir, “salta” a la instrucción a ejecutar).

Loader: carga en memoria el proceso elegido por el long term.

***Long term Scheduler***

Es el que realiza la actividad de admitir procesos de nuevo al estado de listo. Cuando un proceso se crea, cual se selecciona para cargarlo a memoria a su espacio de direcciones y que pase a estado ready.

Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria.

Puede no existir este scheduler y absorber esta tarea el de short term.

***Medium Term Scheduler (swapping)***

Si es necesario, reduce el grado de multiprogramación

Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema.

Términos asociados: swap out (sacar de memoria), swap in (volver a memoria).

***Short Term Scheduler***

Decide a cuál de los procesos en la cola de listos se elige para que use la CPU.

Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling

***Sobre el estado suspendido***

Estado en el que el medium term tiene que distinguir entre un proceso que esta todo inicializado (estructura) pero que falto descargarlo en la memoria.

***Sobre el estado nuevo (new)***

Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre.

En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria

***Sobre el estado listo (ready)***

Luego que el scheduller 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 asignarle 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***

**New-Ready:** Por elección del scheduler de largo plazo (carga en memoria)

**Ready-Running**: Por elección del scheduler de corto plazo (asignación de CPU)

**Running-Waiting:** el proceso “se pone a dormir”, esperando por un evento.

**Waiting-Ready:** Terminó la espera y compite nuevamente por la CPU.

***Caso especial: running-ready***

Cuando el proceso termina su quantum (franja de tiempo) sin haber necesitado ser interrumpirlo por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento...

Se trata de un caso distinto a los anteriores, porque el procesos es expulsado de la CPU contra su voluntad

Esta situación se da en algoritmos apropiativos

***Explicación por estado***

1. Ejecución en modo usuario

2. Ejecución en modo kernel

3. El proceso está listo para ser ejecutado cuando sea elegido.

4. Proceso en espera en memoria principal.

5. Proceso listo, pero el swapper debe llevar al proceso a memoria ppal antes que el kernel lo pueda elegir para ejecutar.

6. Proceso en espera en memoria secundaria.

7. Proceso retornando desde el modo kernel al user. Pero el kernel se apropia, hace un context switch para darle la CPU a otro proceso.

8. Proceso recientemente creado y en transición: existe, pero aún no está listo para ejecutar, ni está dormido.

9. El proceso ejecutó la system call exit y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, codigo resultante del exit. Es el estado final.

***Procesos III***

***Comportamiento de los procesos (cont.)***

\*CPU-bound

Mayor parte del tiempo utilizando la CPU

\*I/O-bound (I/O = E/S)

Mayor parte del tiempo esperando por I/O

\*La velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S

Pensar: Necesidad de atender rápidamente procesos I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para procesos CPU bound

***Planificación***

\*Planificación:

–Necesidad de determinar cuál de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado

\*Algoritmo de Planificación

–Algoritmo utilizado para realizar la planificación del sistema

***Algoritmos Apropiativos y No Apropiativos***

En los algoritmos Apropiativos (preemtive) existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU (Se lo puede expulsar más allá de lo que está haciendo el proceso).

En los algoritmos No Apropiativo (nonpreemptive) los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU

– Se bloquea por E/S o finaliza

– No hay decisiones de planificación durante las interrupciones de reloj

***Categorías de los Algoritmos de Planificación***

Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:

Equidad: Otorgar una parte justa de la CPU a cada proceso

Balance: Mantener ocupadas todas las partes del sistema

Ejemplos:

Procesos por lotes (batch)

Procesos Interactivos

Procesos en Tiempo Real

***Procesos Batch***

No existen usuarios que esperen una respuesta en una terminal.

Se pueden utilizar algoritmos no apropiativos

Metas propias de este tipo de algoritmos:

\*Rendimiento: Maximizar el número de trabajos por hora

\*Tiempo de Retorno: Minimizar los tiempos entre el comienzo y la finalización

\*El Tiempo es espera se puede ver afectado

\*Uso de la CPU: Mantener la CPU ocupada la mayor cantidad de tiempo posible

Ejemplos de Algoritmos:

\*FCFS – First Come First Served

\*SJF – Shortest Job First

***Procesos Interactivos***

No solo interacción con los usuarios

Un servidor, necesita de varios procesos para dar respuesta a diferentes requerimientos

Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU

Metas propias de este tipo de algoritmos:

\*Tiempo de Respuesta: Responder a peticiones con rapidez

\*Proporcionalidad: Cumplir con expectativas de los usuarios

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

Ejemplos de Algoritmos:

\*Round Robin

\*Prioridades

\*Colas Multinivel

\*SRTF – Shortest remaining time first

***Política Versus Mecanismo***

Existen situaciones en las que es necesario que la planificación de uno o varios procesos se comporte de manera diferente

El algoritmo de planificación debe estar parametrizado, de manera que los procesos/usuarios pueden indicar los parámetros para modificar la planificación

El Kernel implementa el mecanismo

El usuario/proceso/administrador utiliza los parámetros para determinar la Política

Ejemplo:

\*Un algoritmo de planificación por prioridades y una System Call que permite modificar la prioridad de un proceso (man nice)

\*Un proceso puede determinar las prioridades de los procesos que el crea, según la importancia de los mismos

***Procesos IV***

***Creación de procesos***

Un proceso es creado por otro proceso

Un proceso padre tiene uno o más procesos hijos.

Se forma un árbol de procesos

***Actividades en la creación***

Crear la PCB

Asignar PID (Process Identification) único

Asignarle memoria para regiones

–Stack, Text y Datos

Crear estructuras de datos asociadas

–Fork (copiar el contexto, regiones de datos, text y stack)

***Relación entre procesos Padre e Hijo***

Con respecto a la Ejecución:

El padre puede continuar ejecutándose concurrentemente con su hijo

El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución.

Con respecto al Espacio de Direcciones:

El hijo es un duplicado del proceso padre (caso Unix)

Se crea un nuevo espacio de direcciones copiando el del padre

Se crea el proceso y se le carga adentro el programa (caso Windows)

Se crea un nuevo espacio de direcciones vacío

¿Cómo funciona fork?

***Creación de Procesos***

En UNIX: (2 System Calls)

system call fork() crea nuevo proceso igual al llamador

system call execve(), generalmente usada después del fork, carga un nuevo programa en el espacio de direcciones.

En Windows: (1 System Call)

System call CreateProcess () crea un nuevo proceso y carga el programa para ejecución.

***¿Cómo funciona fork?***

***Ejemplo SysCall fork***

***Terminación de procesos***

\*Ante un (exit), se retorna el control al sistema operativo

\*El proceso padre puede esperar recibir un código de retorno (via wait). Generalmente se lo usa cuando se requier que el padre espere a los hijos

Proceso padre puede terminar la ejecución de sus hijos (kill)

\*La tarea asignada al hijo se terminó

\*Cuando el padre termina su ejecución

\* Habitualmente no se permite a los hijos continuar, pero existe la opción. \*Terminación en cascada

***Ejemplo SysCall fork+wait+exi***

***Fork / Exec – Ejemplo***

***Memoria I***

1. La organización y administración de la “memoria principal” es uno de los factores más importantes en el diseño de los S. O.
2. Los programas y datos deben estar en el almacenamiento principal para:
   1. Poderlos ejecutar.
   2. Referenciarlos directamente.
3. El SO debe:
   1. Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no.
   2. Asignar espacio en memoria principal a los procesos cuando estos la necesitan.
   3. Libera espacio de memoria asignada a procesos que han terminado.
4. Se espera de un S.O. un uso eficiente de la memoria con el fin de alojar el mayor número de procesos
5. El S.O. debe:
   1. Lograr que el programador se abstraiga de la alocación de los programas
   2. Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros
   3. Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código en común, etc.)
   4. Garantizar la performance del sistema

***Administración de Memoria***

1. División Lógica de la Memoria Física para alojar múltiples procesos

– Garantizando protección

– Depende del mecanismo provisto por el HW

1. Asignación eficiente

– Contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos

***Requisitos***

Reubicación

\*) El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM

\*) Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.

\*) Las referencias a la memoria se deben “traducir” según ubicación actual del proceso.

Protección

1. Los procesos NO deben referenciar – acceder - a direcciones de memoria de otros procesos

\*) Salvo que tengan permiso

1. El chequeo se debe realizar durante la ejecución:

\*) NO es posible anticipar todas las referencias a memoria que un proceso puede realizar

Compartición

1. Permitir que varios procesos accedan a la misma porción de memoria.

\*Ej.: Rutinas comunes, librerías, espacios explícitamente compartidos, etc

2) Permite un mejor uso – aprovechamiento - de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones

Abstracción - Espacio de Direcciones

1. Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
2. El tamaño depende de la Arquitectura del Procesador
   1. 32 bits: 0 .. 232 - 1
   2. 64 bits: 0 .. 264 – 1
3. Es independiente de la ubicación “real” del proceso en la Memoria RAM

Direcciones

1. Lógicas
   1. Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
   2. Representa una dirección en el “Espacio de Direcciones del Proceso”
2. Físicas
   1. Referencia una localidad en la Memoria Física (RAM)

• Dirección absoluta

En caso de usar direcciones Lógicas, es necesaria algún tipo de conversión a direcciones Físicas.

Conversión de Direcciones

Una forma simple de hacer esto es utilizando registros auxiliares

1. Registro Base
   1. Dirección de comienzo del Espacio de Direcciones del proceso en la RAM
2. Registro Limite
   1. Dirección final del proceso o medida del proceso – Tamaño de su Espacio de Direcciones

Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria.

Varían entre procesos (Context Switch)

Dir. Lógicas vs. Físicas

1. Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas.
   1. Resolución de direcciones (address-binding): transformar la dirección lógica en la dirección física correspondiente
2. Resolución en momento de compilación (Archivos .com de DOS) y en tiempo de carga
   1. Direcciones Lógicas y Físicas son idénticas
   2. Para reubicar un proceso es necesario recompilarlo o recargarlo.
3. Resolución en tiempo de ejecución
   1. Direcciones Lógicas y Físicas son diferentes
   2. Direcciones Lógicas son llamadas “Direcciones Virtuales”
   3. La reubicación se puede realizar fácilmente
   4. El mapeo entre “Virtuales” y “Físicas” es realizado por hardware

• Memory Management Unit (MMU)

Memory Management Unit (MMU)

1. Dispositivo de Hardware que mapea direcciones virtuales a físicas
   1. Es parte del Procesador
   2. Re-programar el MMU es una operación privilegiada

• Solo puede ser realizada en Kernel Mode

1. El valor en el “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.
   1. Los procesos nunca usan direcciones físicas

Mecanismos de asignación de memoria

1. Particiones Fijas: El primer esquema implementado
   1. La memoria se divide en particiones o regiones de tamaño Fijo (pueden ser todas del mismo tamaño o no)
   2. Alojan un proceso cada una
   3. Cada proceso se coloca de acuerdo a algún criterio (First Fit, Best Fit, Worst Fit, Next Fit) en alguna partición
2. Particiones dinámicas: La evolución del esquema anterior
   1. Las particiones varían en tamaño y en número
   2. Alojan un proceso cada una
   3. Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

Fragmentación

La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua

1. Fragmentación Interna:
   1. Se produce en el esquema de particiones Fijas
   2. Es la porción de la partición que queda sin utilizar
2. Fragmentación Externa:
   1. Se produce en el esquema de particiones dinámicas
   2. Son huecos que van quedando en la memoria a medida que los procesos finalizan
   3. Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar
   4. Para solucionar el problema se puede acudir a la compactación, pero es muy costosa

Problemas del esquema

\*) El esquema de Registro Base + Limite presenta problemas:

– Necesidad de almacenar el Espacio de Direcciones de forma continúa en la Memoria Física

– Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas

– Fragmentación – Mantener “partes” del proceso que no son necesarias

– Los esquemas de particiones fijas y dinámicas no se usan hoy en día

Solución:

– Segmentación

– Paginación

Segmentación

Esquema que se asemeja a la “visión del usuario”. El programa se divide en partes/secciones

Un programa es una colección de segmentos. Un segmento es una unidad lógica como:

1. Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.

Puede causar Fragmentación

Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas).

Las direcciones Lógicas consisten en 2 partes:

Selector de Segmento

Desplazamiento dentro del segmento

Segmentación (cont.) – Arquitectura

\*) Tabla de Segmentos

Permite mapear la dirección lógica en física. Cada entrada contiene:

Base: Dirección física de comienzo del segmento

Limit: Longitud del Segmento

\*) Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.

\*) Segment-table length register (STLR): cantidad de segmentos de un programa

Paginación

1. Memoria Física es dividida lógicamente en pequeños trozos de igual tamaño **⎝ Marcos**
2. Memoria Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos **⎝ Paginas**
3. El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el **Marco** en la que se coloca cada página.
4. La dirección lógica se interpreta como:
   1. – un número de página y un desplazamiento dentro de la misma.

Segmentación Paginada

1. La paginación
   1. Transparente al programador
   2. Elimina Fragmentación externa.
2. Segmentación
   1. Es visible al programador
   2. 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