Memorias

La clasificaremos en dos tipos **REAL** y **VIRTUAL**.

La **memoria real** o principal es en donde son ejecutados los programas y procesos de una computadora y es el espacio **real** que existe en **memoria** para que se ejecuten los procesos.

La **memoria virtual**, es el espacio reservado en disco o área de intercambio, donde se obtiene una extensión de la RAM.

# Clasificación

# MEMORIA REAL

## **Memoria Real Contigua Simple:**

Acá observaremos las características que posee este tipo de memoria.

* Sistema operativo sencillo o básico.
  + - Control de lotes.
    - Monitor residente.
    - Ejecutor.
* Mono programación.
* Mono usuario.
* Registros frontera.

## **Memoria Real Contigua Particionada:**

Acá observaremos las características generales de las memorias particionadas.

* Sistema operativo más complejo.
* Context switch.
* Control de invasión.
* Registro límite.
* El proceso debe ser más chico que la memoria.
* No se puede cambiar de clase o partición.
* Produce fragmentación interna.

## **Memoria Real Contigua Particionada Fija:**

La memoria principal de divide en particiones estáticas en tiempo de generación del sistema. Un proceso puede cargarse en una partición de igual o mayor tamaño que el proceso. Tiene la ventaja de ser sencillo de implementar y su poca sobrecarga para el sistema operativo. Las desventajas son el uso ineficiente de la memoria (debido a la fragmentación interna) y que debe fijarse el número máximo de procesos activos. Pueden ser de dos tipos:

* **Equitativa**: Se divide la memoria en particiones de igual tamaño.
* **Proporcional:** La memoria se divide en porciones de distintos tamaños, cada una con su Job Queue. Los procesos son asignados a la partición que produzca menor fragmentación interna.

Memoria equitativa: Memoria proporcional:



Ejemplo de Ejerció de “**Memoria Real Contigua Particionada Fija Proporcional**”. Dado una memoria con una capacidad de 1000kb, posee tres particiones de 600kb, 200kb y 100kb, el sobrante es para el S.O. Dado los siguientes procesos con sus tamaños y permanencia.

|  |  |  |  |
| --- | --- | --- | --- |
| **Arribo** | **Proceso** | **Tamaño** | **Tiempo de Permanencia** |
| 0 | A | 450kb | 5 |
| 0 | B | 150kb | 3 |
| 0 | C | 90kb | 5 |
| 0 | D | 250kb | 3 |
| 0 | E | 180kb | 2 |
| 0 | F | 80KB | 4 |
| 0 | G | 50kb | 4 |

Se pide:

1. Construye las particiones.
2. Ubica a los procesos en cada partición, según corresponda.
3. Analiza que pasa en el tiempo T0, T3 y T5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | RQ |  | T0 | T3 | T5 |
| S.O=100KB |  |  | S.O | S.O | S.O |
| PART1=600KB | D |  | A | A | D |
| PART2=200KB | E |  | B | E |  |
| PART3=100KB | G,F |  | C | C | F |

Acá podemos observar que es lo que pasa en cada una de las particiones en los Tiempos T0, T3 y T5. Lo primero es ubicar los procesos por su tamaño, sabiendo que los procesos en las particiones se van a ubicar con el Algoritmo Best Fit (mejor hueco o la partición que produzca menos fragmentación interna) y en cada una de ellas entra un solo proceso.

***Nota****: Un error clásico es tratar de asignar dos procesos a la misma partición.*

## **Memoria Real Contigua Particionada Variable:**

* Multiusuario.
* Multiproceso.
* Sistema operativo complejo:
  + - Context switch.
    - Diferentes tamaños.
    - Mayor control de huecos.
    - Interrupciones.
* Una sola Job Queue.
* Tabla de espacios ocupados y libres.
* Fragmentación externa:
  + - Compactación.

Las particiones se crean de forma dinámica, de tal forma que cada proceso se carga en una partición del mismo tamaño que el proceso. Tiene la ventaja de no poseer fragmentación interna y el uso más eficiente de la memoria. La desventaja es el uso ineficiente del procesado, debido a la necesidad de compactación para evitar la fragmentación externa, que se da cuando un proceso finaliza su ejecución y deja un hueco libre en la memoria que es demasiado chico para que otro proceso lo ocupe.

***Compactación***, es reubicar los procesos con el objetivo de generar un hueco suficiente para poder alojar uno nuevo, esto surge con la fragmentación externa en donde se posee espacio suficiente pero no contiguo (y obviamente está asociado a la asignación contigua de datos), existen varios métodos, en donde se pueden combinar:

* Manual o Automática.
* Total o Parcial.
* Bruta o Inteligente.

## ***Algoritmo de ubicación:***

Debido a que la compactación de memoria consume una gran cantidad de tiempo, el diseñador del sistema operativo debe ser inteligente a la hora de decidir cómo asignar la memoria a los procesos. A la hora de cargar un proceso a la memoria principal, y siempre que haya más de un bloque de memoria libre de suficiente tamaño, el sistema operativo debe decidir qué bloque libre asignar.

Pueden usarse tres algoritmos de colocación:

**Best-fit (mejor ajuste)**: Escoge el bloque libre del tamaño más cercano al de la petición.

**First-fit (primer ajuste)**: Comienza a analizar la memoria desde el principio y asigna al proceso el primer hueco en el que entre.

**Next-fit (siguiente ajuste)**: Comienza a analizar la memoria desde la última colocación y elige el siguiente bloque disponible que sea suficientemente grande.

Ejemplo de Ejerció de “**Memoria Real Contigua Particionada Variable**”, se manejan con una tabla en donde registra el desplazamiento de los procesos. Aparecen los parámetros, ***quien, desde, cuanto, estado.*** Suponiendo que tenemos una memoria total de 1000kb, en donde se utiliza 150kb para el S.O y arriban los siguientes procesos.

|  |  |  |  |
| --- | --- | --- | --- |
| **Arribo** | **Proceso** | **Tamaño** | **Tiempo de Permanencia** |
| 0 | A | 450kb | 5 |
| 0 | B | 150kb | 8 |
| 0 | C | 200kb | 6 |
| 0 | D | 460kb | 6 |
| 0 | E | 180kb | 2 |

Se pide:

1. Construye las particiones.
2. Ubica a los procesos en cada partición, según corresponda.
3. Analiza que pasa en el tiempo T0, T3 y T5.

T0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RQ |  | **QUIEN** | **DESDE** | **CUANTO** | **ESTADO** |
| E,D |  | S.O | 0 | 149 | OCUPADO |
|  |  | A | 150 | 599 | OCUPADO |
|  |  | B | 600 | 749 | OCUPADO |
|  |  | C | 750 | 949 | OCUPADO |
|  |  |  | 950 | 999 | LIBRE |

Si observamos la memoria se encuentra ocupada por los procesos que pudieron entrar, el resto de ellos quedaron en la Ready Queue, esperando que alguno de ellos finalice y genere el espacio suficiente para poder alojarse.

T5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RQ |  | **QUIEN** | **DESDE** | **CUANTO** | **ESTADO** |
| E,D |  | S.O | 0 | 149 | OCUPADO |
|  |  |  | 150 | 599 | LIBRE |
|  |  | B | 600 | 749 | OCUPADO |
|  |  | C | 750 | 949 | OCUPADO |
|  |  |  | 950 | 999 | LIBRE |

Si observamos el proceso **A** finalizo, por lo tanto, libero un espacio de 450kb, y el siguiente en la Ready Queue es el proceso **D** el cual tiene el tamaño de 460kb. Este no entraría en ningún hueco disponible ya que tengo uno de 450kb y otro de 50kb, pero si observamos la memoria posee espacio suficiente pero no contiguo, estas son las consecuencias de la Fragmentación Externa, en donde la solución es la ***compactación***. Si compactamos de forma ***Manual-> Total*** quedaría.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RQ |  | **QUIEN** | **DESDE** | **CUANTO** | **ESTADO** |
| E,D |  | S.O | 0 | 149 | OCUPADO |
|  |  | B | 150 | 299 | OCUPADO |
|  |  | C | 300 | 499 | OCUPADO |
|  |  |  | 500 | 999 | LIBRE |
|  |  |  |  |  |  |

Ahora si estamos en condiciones de ubicar el proceso D

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RQ |  | **QUIEN** | **DESDE** | **CUANTO** | **ESTADO** |
| E |  | S.O | 0 | 149 | OCUPADO |
|  |  | B | 150 | 299 | OCUPADO |
|  |  | C | 300 | 499 | OCUPADO |
|  |  | D | 500 | 959 | OCUPADO |
|  |  |  | 960 | 999 | LIBRE |

## **Memoria Real Contigua Particionada Colegas (Buddy System)**

El sistema *buddy* es un compromiso razonable para eliminar las desventajas de los esquemas de particionamiento tanto fijo, como variable, pero en los sistemas operativos contemporáneos, la memoria virtual basada en paginación y segmentación es superior.

La desventaja de un esquema de particionamiento fijo es que limita el número de procesos activos y puede utilizar el espacio ineficiente si existe un mal ajuste entre los tamaños de partición disponibles y los tamaños de los procesos; un esquema de particionamiento dinámico es más complejo de mantener e incluye la sobrecarga de la compactación.

## **Memoria Real No Contigua Paginación:**

La memoria se divide en pequeños frames (marcos) y los procesos se dividen en pequeñas porciones (llamadas páginas) del mismo tamaño que los frames. Al momento de cargar un proceso a memoria, se le asigna un número de marcos igual al de las páginas que tiene dicho proceso, sin necesidad de que sean contiguas, por lo que no existe fragmentación externa y solo un mínimo de fragmentación interna en la última página del proceso. El sistema operativo mantiene una lista de marcos libres. Un proceso solo es cargado en memoria cuando hay suficientes marcos libres para su número de páginas.

El tamaño de las páginas y los frames es conveniente que sea potencia de 2, de esta forma la dirección de memoria relativa al origen del programa es igual a la dirección lógica, compuesta por el número de página y el desplazamiento adentro de la misma.

En la paginación existen 2 algoritmos de asignación ***First Fit o Next Fit***, dado que todas las paginas y marcos poseen el mismo tamaño y pueden alojarse de manera no contigua por lo tanto cada proceso posee una tabla donde indica ***Pagina/Marco***, esta combinación indicara en la memoria donde esta ubicada la pagina del proceso.

Ejemplo de Ejercicio de “**Memoria Real No Contigua Paginación”**, se posee una memoria de 1500kb con un S.O que trabaja con paginación simple, donde cada página posee un tamaño de 150kb. Se deberá analizar los tiempos T0 y T5, sabiendo:

|  |  |  |  |
| --- | --- | --- | --- |
| **Arribo** | **Proceso** | **Tamaño** | **Tiempo de Permanencia** |
| 0 | A | 450kb | 5 |
| 0 | B | 150kb | 8 |
| 0 | C | 200kb | 6 |
| 0 | D | 460kb | 6 |
| 0 | E | 180kb | 2 |

* Indicar cuantas paginas necesitan cada programa.
* Construir las tablas y asignar un marco en cada cado para los procesos.
* Como queda la memoria en cada caso.

|  |  |  |
| --- | --- | --- |
| **Proceso** | **Tamaño** | **Paginas** |
| A | 450kb | 3 |
| B | 150kb | 1 |
| C | 200kb | 2 |
| D | 450kb | 3 |
| E | 180kb | 2 |

**Memoria**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | A | A | B | C | C | D | D | D |  |

***Notas:*** Recuerden que la memoria es como un vector arranca siempre en cero.

Tablas T0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | | **B** | | **C** | | **D** | |
| **PAGINA** | **MARCO** | **PAGINA** | **MARCO** | **PAGINA** | **MARCO** | **PAGINA** | **MARCO** |
| 0 | 0 | 0 | 3 | 0 | 4 | 0 | 6 |
| 1 | 1 |  |  | 1 | 5 | 1 | 7 |
| 2 | 2 |  |  |  |  | 2 | 8 |
|  |  |  |  |  |  |  |  |

En la RQ se encuentra el proceso E, el cual debe aguardar para cargarse ya que no posee espacio porque pese a que la asignación es no contigua, el proceso debe estar cargado todo en memoria.

T5 en este tiempo finaliza el proceso A, se generan dos paginas libre, aca se puede resolver aplicando cualquiera de los dos algoritmos de asignación, probemos con First Fit como quedaria:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | B | C | C | D | D | D |  |

***First Fit***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | E |  | B | C | C | D | D | D |  |

|  |  |
| --- | --- |
| **E** | |
| **PAGINA** | **MARCO** |
| 0 | 0 |
| 1 | 1 |

***Next Fit*** *(Recuerden que este algoritmo posee un puntero a la última dirección que asigno un marco)*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E |  |  | B | C | C | D | D | D | E |

|  |  |
| --- | --- |
| **E** | |
| **PAGINA** | **MARCO** |
| 0 | 9 |
| 1 | 0 |

## **Memoria Real No Contigua Segmentación:**

Un programa de usuario se puede subdividir utilizando la segmentación, en la cual se divide al programa y sus datos asociados en segmentos de distintos tamaños. Debido a esto, la segmentación es similar al particionamiento variable. La diferencia es que con la segmentación un programa podría ocupar más de una partición y estas particiones no necesitan ser contiguas. La segmentación elimina la fragmentación interna pero sufre de fragmentación externa como en el caso del particionamiento variable, solo que en menor medida debido a que los programas son divididos en varias piezas pequeñas.