

Arquitectura y Sistemas Operativos

Tecnicatura Universitaria en
Programación

ion: absolute; z-index: 999;
x 5px #ccc}.gbrtl .gbm{-m
display: block; position: a
capacity: 1; *top: -2px; *left:
/; top: -4px\0/; left: -6px\0
ne-box; display: inline-bloc
display: block; list-style: r
ne-block; line-height: 27px;
pointer; display: block; tex
ative; z-index: 1000}.gbtm{*
padding-right: 9px}#gbz .g
ad:url(//

Gestión de memoria

Unidad 7

Agenda



1. Requisitos de la gestión de memoria
2. Técnicas de Particionamiento
3. Paginación
4. Segmentación

Definiciones

Sistemas Monoprogramados

La memoria se divide entre el SO y el programa en ejecución

Sistemas Multiprogramados

La memoria se divide entre el SO y **los programas** en ejecución

La gestión de la memoria se encarga de subdividir el área de memoria asignada a los programas de usuario

Objetivo de la gestión de memoria

Una gestión de la memoria efectiva es vital en un sistema multiprogramado. Si sólo unos pocos procesos se encuentran en memoria, entonces durante una gran parte del tiempo todos los procesos esperarían por operaciones de E/S y el procesador estaría ocioso. Por tanto, es necesario asignar la memoria para asegurar una cantidad de procesos listos que consuman el tiempo de procesador disponible.



1. Requisitos de la gestión de memoria



Cinco Requisitos

5 Requisitos de la gestión de memoria

Reubicación

Protección

Compartición

Organización lógica

Organización física

Reubicación

Motivación

No es posible para el programador conocer, de antemano, qué programas residen en memoria

Se necesita poder intercambiar (swap) procesos en la memoria principal para incrementar la utilización del procesador

Solución

La reubicación se encarga de ubicar/alojar un proceso (nuevo o suspendido) en memoria principal

El proceso de reubicación facilita las traducciones de memoria, utilizando para esto los datos guardados en el PCB

Protección

Motivación

Cada proceso debe protegerse de interferencias no deseadas (sin permisos) de otros procesos, ya sea de lectura o escritura

El proceso de reubicación complejiza aún más este requisito.

Solución

Todas las referencias a memoria realizadas por un proceso deben ser verificadas en tiempo de ejecución.

El procesador es el encargado de realizar la verificación.

Compartición

Motivación

Los mecanismos de protección deben contar con la flexibilidad suficiente para que, si dos o más procesos están compartiendo determinada porción de memoria, puedan acceder a la misma.

Solución

Los mecanismos que dan soporte a la reubicación contemplan este requisito también.

Organización lógica

La memoria principal (y la secundaria) se organiza como un array de bytes

Una gestión efectiva permite:

Que la compilación de los módulos sea independiente

Que las traducciones a memoria las realice el SO

Distintos grados de protección

Compartir módulos entre procesos

Organización física



Todo sistema se organiza con 2 niveles de memoria (principal y secundaria)

Es responsabilidad del sistema que el flujo de información entre ambos niveles sea lo más eficiente posible.

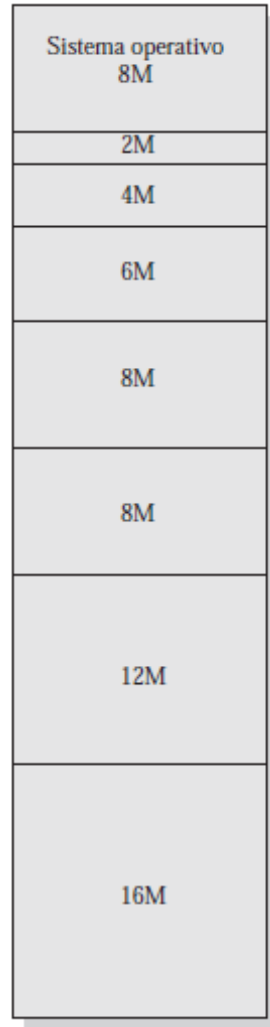


2. Técnicas de Particionamiento

Particionamiento fijo. Dos alternativas.



Particiones de igual tamaño

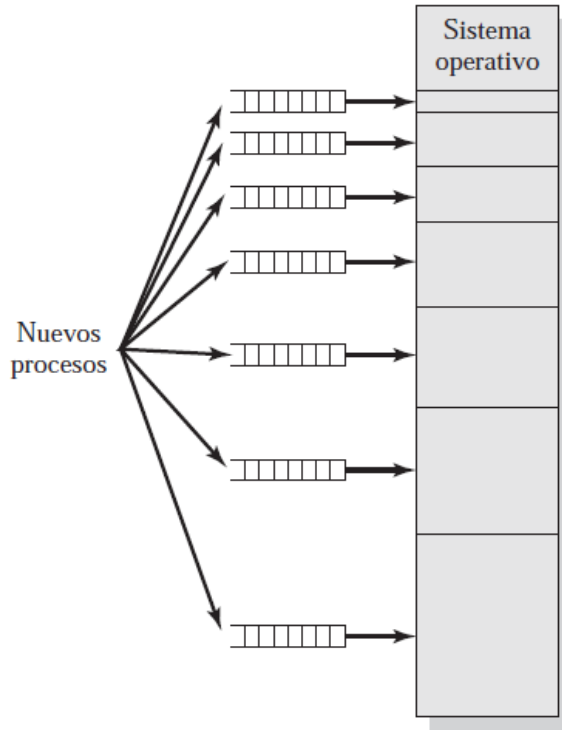


Particiones de distinto tamaño

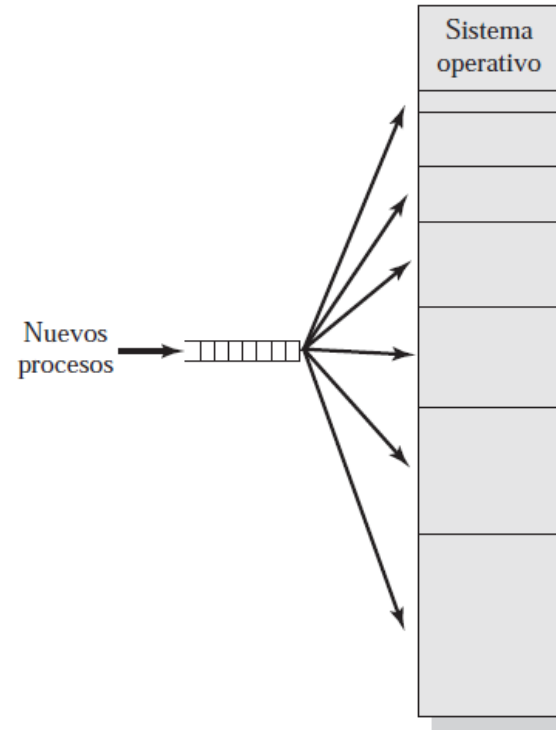
- Particiones fijas: Un programa podría ser demasiado grande para caber en una partición. En este caso, el programa debe usar overlays, de forma que sólo se necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndolo (overlaying) a cualquier programa o datos que haya allí.
- La utilización de la memoria principal es ineficiente. Cualquier programa ocupa una partición entera malgastando espacio interno de memoria. Este fenómeno se conoce como fragmentación interna.
- Ambos problemas se pueden mejorar, aunque no resolver, utilizando particiones de tamaño diferente.

Particionamiento fijo de distinto tamaño. Asignación de memoria.

- La forma más sencilla consiste en asignar cada proceso a la partición más pequeña dentro de la cual cabe. En este caso, se necesita una cola de planificación para cada partición, que mantenga procesos en disco destinados a dicha partición.
- La ventaja de esta técnica es que los procesos siempre se asignan de tal forma que se minimiza la memoria malgastada dentro de una partición.
- Desventaja: Si todos los procesos ocupan menos memoria que las particiones mas grandes, estas no se utilizan.



Una cola de procesos por partición



Una única cola

- Una técnica óptima es emplear una única cola para todos los procesos. En el momento de cargar un proceso en la memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso.
- Si todas las particiones están ocupadas, se debe llevar a cabo una decisión para enviar a swap a algún proceso.

Particionamiento fijo. Conclusiones

Descripción

- La memoria principal se divide en particiones estáticas en tiempo de generación del sistema. Un proceso se puede cargar en una partición con igual o superior tamaño.

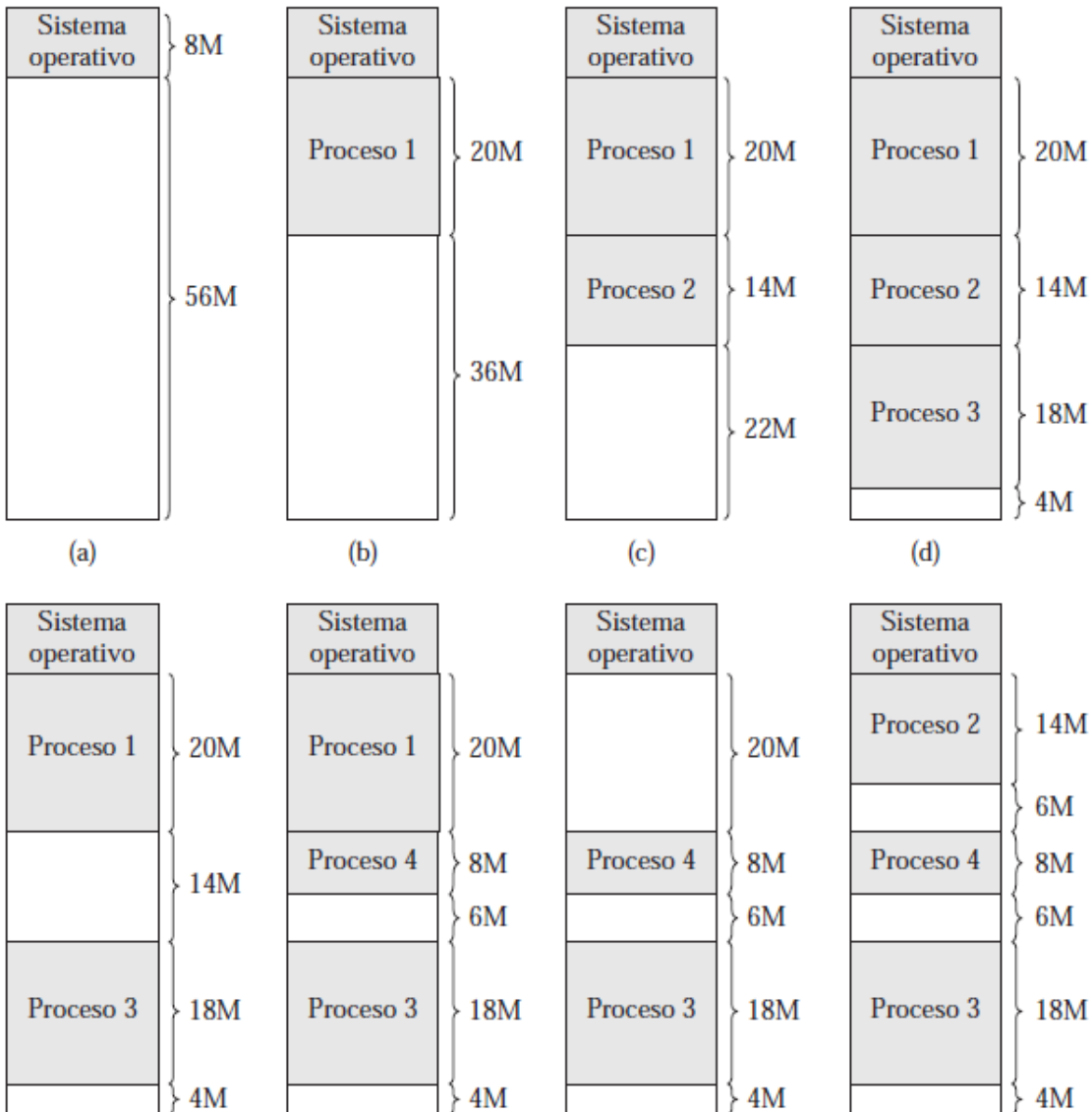
Virtudes

- Sencilla de implementar; poca sobrecarga para el sistema operativo.

Defectos

- Uso ineficiente de la memoria, debido a la fragmentación interna; debe fijarse el número máximo de procesos activos.

Particionamiento dinámico.



- Las particiones son de longitud y número variable. Cuando se lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera.
- Como muestra este ejemplo, el método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos en la memoria. Este fenómeno se conoce como fragmentación externa, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental
- Una técnica para eliminar la fragmentación externa es la compactación: de vez en cuando, el sistema operativo desplaza los procesos en memoria, de forma que se encuentren contiguos y de este modo toda la memoria libre se encontrará unida en un bloque.
- La compactación consume tiempo y malgasta tiempo de procesador.

Particionamiento dinámico. Conclusiones

Descripción

- Las particiones se crean de forma dinámica, de tal forma que cada proceso se carga en una partición del mismo tamaño que el proceso.

Virtudes

- No existe fragmentación interna; uso mas eficiente de memoria principal.

Defectos

- Uso ineficiente del procesador, debido a la necesidad de compactación para evitar la fragmentación externa.

Sistema Buddy

- Este sistema surge como mejora al particionamiento estático y dinámico
- Lo que se hace es dividir los bloques de memoria a la mitad hasta obtener un bloque que se ajuste lo mejor posible a la petición de memoria
- El sistema Buddy propone que los bloques de memoria disponible son de tamaño 2^k , con $L \leq k \leq U$, donde:

2^L = bloque de tamaño más pequeño

2^U = bloque de tamaño máximo.

Algoritmo:

1. Antes de comenzar se asume que se tiene un único bloque de tamaño 2^U .
2. Si el proceso a cargar tiene un tamaño s , tal que $2^{U-1} < s < 2^U$ se asigna el bloque de tamaño 2^U entero.
1. Si la relación del paso anterior no se cumple, entonces el bloque se divide en 2 bloques de tamaño 2^{U-1} .
2. Si el proceso a cargar tiene un tamaño s , tal que $2^{U-2} < s < 2^{U-1}$ se asigna el bloque de tamaño 2^{U-1} entero.

En caso de que no se cumpla el paso 4, volver al paso 3.

El proceso se repite hasta encontrar el bloque de menor tamaño que pueda almacenar el proceso.

Sistema Buddy. Ejemplo

Bloque de 1 Mbyte	1 M						$2^U = 1 \text{ M}$
A Solicitar 100 K	A = 128 K	128 K	256 K	512 K			
B Solicitar 240 K	A = 128 K	128 K	B = 256 K	512 K			
C Solicitar 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K		
D solicitar 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K	
Liberar B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K	
Liberar A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K	
E Solicitar 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K	
Liberar C	E = 128 K	128 K	256 K	D = 256 K	256 K		
Liberar E	512 K		D = 256 K	256 K			
Liberar D	1 M						

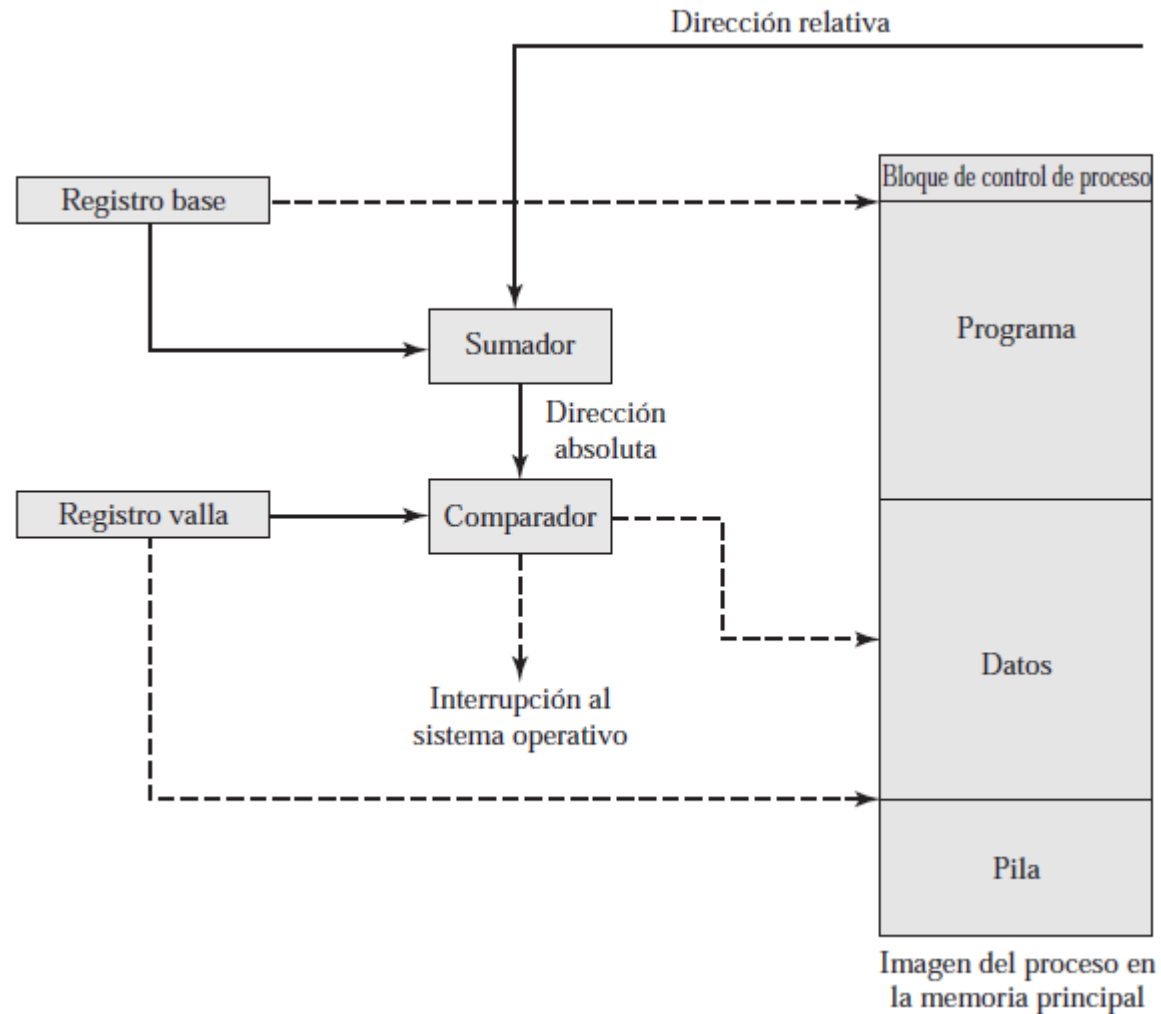
Tipos de direcciones de memoria

- **Dirección lógica o relativa:** es una referencia a una dirección de memoria y es independiente de la ubicación actual. Debe ser traducida a una dirección física antes de poder ser utilizada. Se dice que es relativa, ya que se define en relación a un punto conocido. Este punto conocido es otra dirección de memoria, normalmente conocida por el procesador, que es el registro base
- **Dirección física o absoluta:** es una dirección real en la memoria principal.

El proceso de traducción de direcciones de memoria contempla 2 pasos:

1. El procesador traduce la dirección relativa sumando la dirección base del proceso.
2. Verifica que la dirección traducida se encuentre dentro de los límites del proceso. De no ser así se genera una interrupción conocida como *segmentation fault* (falla de segmento)

Traducción a dirección física



Dirección relativa = 1502

0000010111011110

Proceso de usuario
(2700 bytes)

Particionado



3. Paginación

Paginación

Se divide la memoria en particiones de tamaño fijo denominados marcos y cada proceso también es dividido en porciones fijas del mismo tamaño denominados páginas

Características

Para funcionar, el SO mantiene tablas de páginas por cada proceso.

Cada página de una tabla contiene la dirección del marco donde fue cargada

Cada vez que un proceso es suspendido, su tabla de páginas es reiniciada (eliminada)

Cuando un proceso es cargado en memoria, el SO busca los marcos libres y se los asigna a las páginas del proceso.

Ventajas

Se elimina la fragmentación externa

La fragmentación interna se reduce afectando sólo a la última página de cada proceso.

Se elimina la necesidad de cargar los procesos en posiciones contiguas

Paginación. Ejemplo

Marco número	Memoria principal
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Quince marcos disponibles

Memoria principal
0 A.0
1 A.1
2 A.2
3 A.3
4
5
6
7
8
9
10
11
12
13
14

(b) Cargar proceso A

Memoria principal
0 A.0
1 A.1
2 A.2
3 A.3
4 B.0
5 B.1
6 B.2
7
8
9
10
11
12
13
14

(c) Cargar proceso B

Memoria principal
0 A.0
1 A.1
2 A.2
3 A.3
4 B.0
5 B.1
6 B.2
7 C.0
8 C.1
9 C.2
10 C.3
11
12
13
14

(d) Cargar proceso C

Memoria principal
0 A.0
1 A.1
2 A.2
3 A.3
4
5
6
7 C.0
8 C.1
9 C.2
10 C.3
11
12
13
14

(e) Intercambiar B

Memoria principal
0 A.0
1 A.1
2 A.2
3 A.3
4 D.0
5 D.1
6 D.2
7 C.0
8 C.1
9 C.2
10 C.3
11 D.3
12 D.4
13
14

(f) Cargar proceso D

0	0
1	1
2	2
3	3

Tabla de páginas del proceso A

0	—
1	—
2	—

Tabla de páginas del proceso B

0	7
1	8
2	9
3	10

Tabla de páginas del proceso C

0	4
1	5
2	6
3	11
4	12

Tabla de páginas del proceso D

13
14

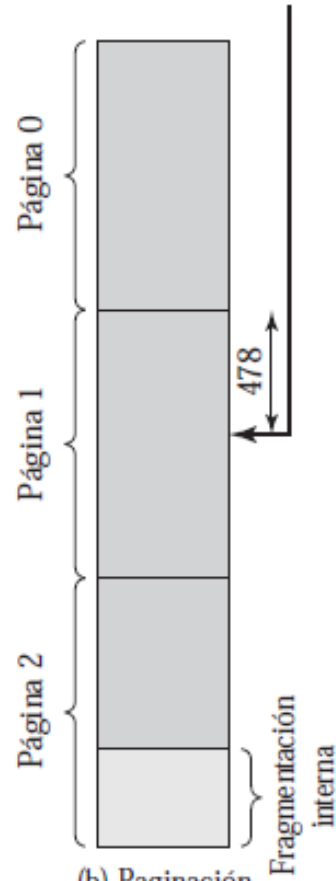
Lista de marcos libres



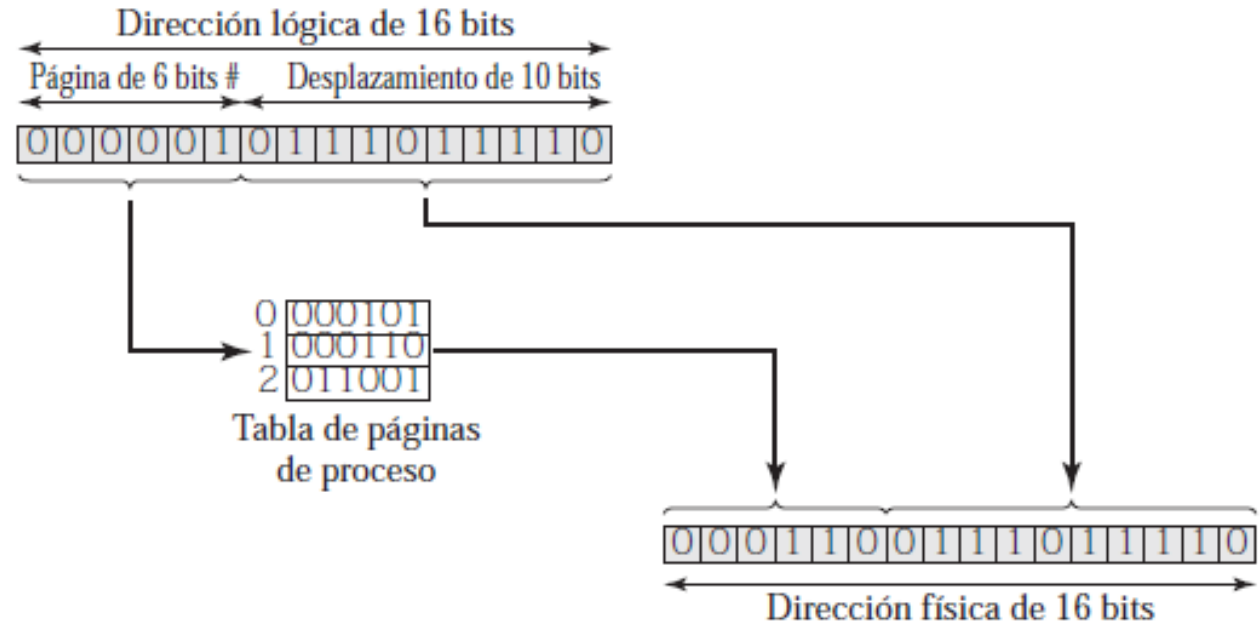
Traducción a direcciones físicas

Dirección lógica =
Página# = 1, Desplazamiento = 478

0000010111011110



(b) Paginación
(tamaño página = 1K)





4. Segmentación

Segmentación

Es una técnica avanzada que divide la memoria principal y los procesos en porciones fijas

Características

Ve al espacio de direcciones de memoria de un proceso como un conjunto de segmentos (código + datos)

Cada uno de estos segmentos es una unidad lógica que permite al programador organizar el código

Permite al SO ubicarlos en porciones de memoria distintas que no necesariamente deben ser contiguas.

Al igual que con la paginación, el SO crea tablas de segmentos por cada proceso y una lista de bloques de memoria libres .

Ventajas

Se elimina la fragmentación interna

Se elimina la necesidad de cargar los procesos en posiciones contiguas

Posibilita que varios procesos compartan el segmento de código y el de datos.

Traducción a direcciones físicas

Dirección lógica =
Segmento# = 1, desplazamiento = 752

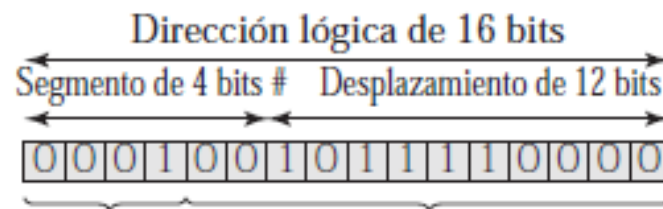
0001001011110000

Segmento 0
750 bytes

Segmento 1
1950 bytes

752

(c) Segmentación



	Longitud	Base
0	001011101110	0000010000000000
1	011110011110	0010000000100000

Tabla de segmentos de proceso

+

0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0

Dirección física de 16 bits

Muchas gracias