

## TAREA 9

### Sistemas Operativos

#### Integrantes:

Mora Ayala José Antonio  
Ramírez Cotonieto Luis Fernando  
Torres Carrillo Josehf Miguel Ángel  
Tovar Jacuinde Rodrigo

#### Profesor:

Cortés Galicia Jorge

# Paginación

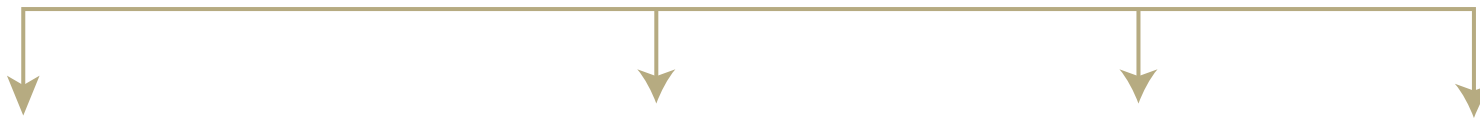
Permite que el espacio de direcciones físicas de un proceso no sea contiguo.



Evita el considerable problema de encajar fragmentos de memoria de tamaño variable en el almacén de respaldo.



Tradicionalmente el soporte de paginación se gestionaba mediante hardware. Actualmente en algunos diseños se comparte hardware y S.O.



## Método básico

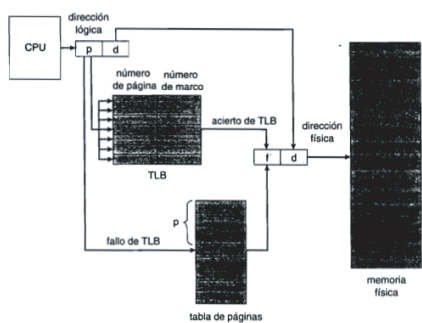
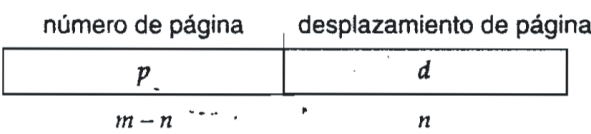
Implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados marcos y descomponer la memoria lógica en bloques del mismo tamaño denominados paginas.



Cuando hay que ejecutar un proceso, el almacén de respaldo se encarga de ello, dividiendo bloques de tamaño fijo que tienen el mismo tamaño de los marcos de memoria.



El tamaño de la pagina (al igual que el tamaño del marco) está definido por el hardware. El tamaño de la pagina es, normalmente, a una potencia de 2, variando entre los 512 bytes y 16 MB.



## Soporte de Hardware

La mayoría de los sistemas operativos asignan una tabla de paginas para cada proceso, almacenando con un puntero a la tabla de paginas, junto con los otros valores de los registros.



La implementación de hardware en su caso más simple, es la asignación de la tabla de paginas como un conjunto de registros, estos se construyen con lógica de muy alta velocidad para que la traducción sea eficiente.



Cada acceso a la memoria debe pasar a través del mapa de paginación, por lo que la eficiencia es de las consideraciones más importantes.



Para los problemas de velocidad se suele dar solución utilizando una cache de hardware especial de pequeño tamaño y con capacidad de acceso rápido, denominado buffer de consulta de traducción.



Algunos buffers almacenan identificadores del espacio de direcciones.



## Protección



La protección de memoria en un entorno paginado se consigue mediante una serie de bits de protección asociados a cada marco.

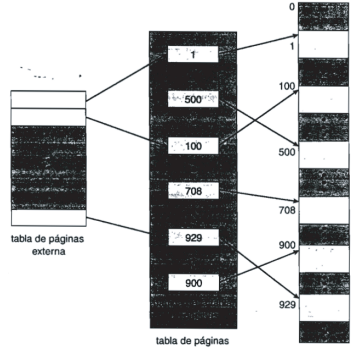


Todo intento de escribir en una pagina de solo lectura provocará una interrupción de hardware al sistema operativo (o una violación de protección de memoria).

## Estructura de la tabla de paginas

### Paginación jerárquica

Utiliza un algoritmo de paginación de dos niveles, en el que la propia tabla de paginas está también paginada. También se conoce con el nombre de tabla de paginas con correspondencia ((mapeo) directa.



## Tablas de paginas hash

Una técnica común para gestionar los espacios de direcciones superiores a 32 bits consiste en utilizar una tabla de hash de paginas, donde el valor de una tabla hash es el número de pagina hash.

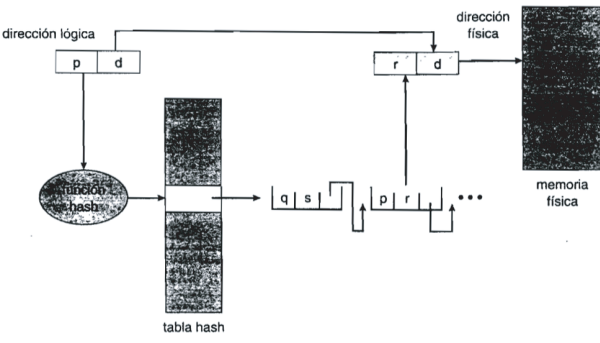


También se ha propuesto una técnica llamada tablas de paginas en cluster, que son similares a las tablas de paginas. Las tablas de paginas en cluster son particularmente útiles para espacios de direcciones dispersos.



## Tablas de paginas invertidas

La tabla de paginas incluye una entrada por cada pagina de proceso que está utilizando. Estas tablas pueden ocupar gran cantidad de memoria física, para esto son las tablas de pagina invertida, donde se tiene una entrada por cada pagina real.



# Memoria Virtual

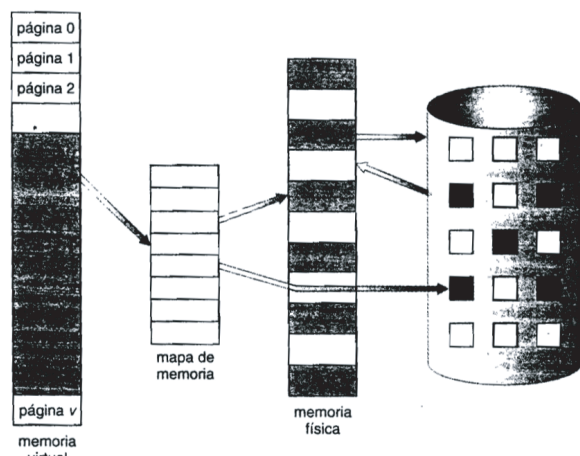
La memoria virtual incluye la separación de la memoria lógica, tal como la percibe el usuario y con respecto a la memoria física.



La memoria virtual facilita enormemente la tarea de programación porque el programador ya no tiene que preocuparse por la cantidad de memoria física disponible, ahora puede concentrarse en el problema que deba resolver su programa.



El espacio de direcciones virtuales de un proceso hace referencia a la forma lógica de almacenar en un proceso de memoria. Los espacios que contienen este tipo de huecos se denominan espacios de direcciones dispersos. Permite que dos o más procesos compartan los archivos y la memoria mediante mecanismos de compartición de paginas.



## Paginación bajo demanda

Con la memoria virtual basada en paginación bajo demanda, solo se cargan las paginas cuando así se solicita durante la ejecución del programa, de este modo, las paginas a las que nunca se acceda no llegarán a cargarse en la memoria física.



Cuando hay que cargar un proceso, en lugar de cargar el proceso completo, el paginador solo carga en la memoria de las paginas necesarias, de este modo, evita cargar en la memoria de las paginas que no vayan a ser utilizadas, reduciendo así el tiempo de carga y la cantidad de memoria física necesaria.



Si una pagina no ha sido cargada a la memoria, provoca una interrupción de fallo de pagina.



## Copia durante la escritura

Esta técnica permite la creación rápida de procesos y minimizar el número de nuevas paginas que deben asignarse al proceso recién creado.

## Hardware

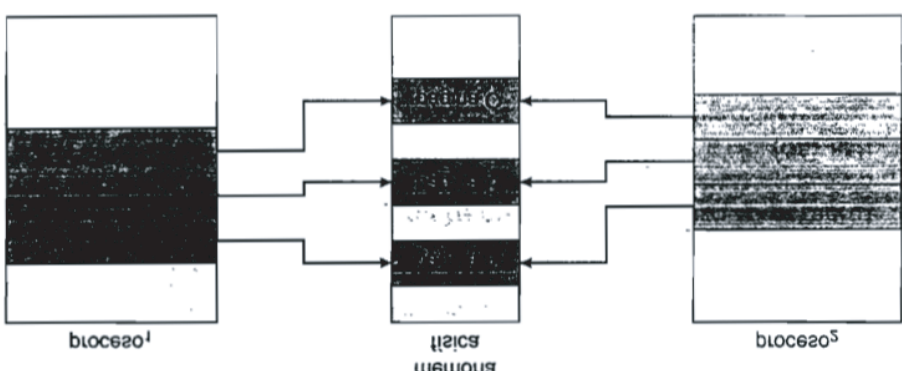
Una tabla de paginas, esta tabla permite marcar una entrada como inválida mediante un bit válido-inválido, o mediante un valor especial de una serie de bits de protección.



Memoria secundaria: Esta memoria almacena aquellas paginas que no estén en la memoria principal. Es usualmente un disco de alta velocidad.



Un requisito fundamental para la paginación bajo demanda, es la necesidad de poder reiniciar cualquier instrucción después de un fallo de pagina. Puesto que guardamos el estado del proceso interrumpido.



## Rendimiento de la paginación bajo demanda

El tiempo puede afectar de gran manera, de los 10 a los 200 ns.



- Sea  $p$  la probabilidad de que se produzca un fallo de página ( $0 \leq p \leq 1$ ). Cabe esperar que  $p$  esté próxima a cero, es decir, que sólo vayan a producirse unos cuantos fallos de página. El tiempo de acceso efectivo será entonces:
- $$\text{tiempo de acceso efectivo} = (1 - p) \times m + p \times \text{tiempo de fallo de página.}$$
- Para calcular el tiempo de acceso efectivo, debemos conocer cuánto tiempo se requiere para dar servicio a un fallo de página. Cada fallo de página hace que se produzca la siguiente secuencia:
1. Interrupción al sistema operativo.
  2. Guardar los registros de usuario y el estado del proceso.
  3. Determinar que la interrupción se debe a un fallo de página.
  4. Comprobar que la referencia de página era legal y determinar la ubicación de la página en el disco.
  5. Ordenar una lectura desde el disco para cargar la página en un marco libre:
    - a. Esperar en la cola correspondiente a este dispositivo hasta que se dé servicio a la solicitud de lectura.
    - b. Esperar el tiempo de búsqueda y/o latencia del dispositivo.
    - c. Comenzar la transferencia de la página hacia un marco libre.
  6. Mientras estamos esperando, asignar la CPU a algún otro usuario (planificación de la CPU, que será opcional).
  7. Recibir una interrupción del subsistema de E/S de disco (E/S completada).
  8. Guardar los registros y el estado del proceso para el otro usuario (si se ejecuta el paso 6).
  9. Determinar que la interrupción corresponde al disco.
  10. Corregir la tabla de páginas y otras tablas para mostrar que la página deseada se encuentra ahora en memoria.
  11. Esperar a que se vuelva a asignar la CPU a este proceso.
  12. Restaurar los registros de usuario, el estado del proceso y la nueva tabla de páginas y reanudar la ejecución de la instrucción interrumpida.



# ALGORITMOS DE SUSTITUCION DE PAGINAS

## Sustitución de paginas no usada recientemente

variables

Estructura pagina paginaEntrante

instrucciones : tipo\_de\_elemento;

rotulo : numero\_de\_instrucciones;

bitR : entero;

bitM : entero;

bitV-I : caracter;

array de tipo pagina tablapaginacion;

entero i,j;

Funcion ComprobarTabla(tablapaginacion)

booleano bandera <- falso;

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i][1] = V entonces

bandera <- verdadero;

finSi;

siguiente;

finLoop;

retornar bandera;

Inicio

Si (ComprobarTabla(tablapaginacion) == verdadero)

entonces

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i]->bitV-I = v entonces ( v

significa valido, se puede asignar)

tablapaginacion[i] <- paginaEntrante;

ColaP.push(pagina);

finSi;

siguiente;

finLoop;

sino

Desde i=0 hasta numero de filas del arreglo

Si (tablapaginacion[i]->bitM == 0 Y

tablapaginacion[i]->bitR==0) entonces

tablapaginacion[i] <- paginaEntrante;

finSi;

ActualizarBitRM(tablapaginacion[i]);

finSi;

Fin

```
finSi;  
finSi;  
ColaMarcosC<- ColaMarcosC->nextPagina;  
finSi;  
finLoop;  
finSi;  
Fin
```

## Sustitucion segunda oportunidad de pagina mejorado

variables

```
Estructura página paginaEntrante;  
instrucciones: tipo_de_elemento;  
bitReferencia : entero;  
bitMoficacion : entero;  
array de tipo página tablapaginacion;  
entero i,j;  
colaCircular de tipo paginas ColaMarcosC;  
Funcion ComprobarTabla(tablapaginacion)  
booleana bandera <- falso;  
Desde i=0 hasta numero de filas del arreglo  
Si tablapaginacion[i][1] = 0 entonces  
bandera <- verdadero;  
finSi;  
siguiente;  
finLoop;  
retornar bandera;
```

Inicio

```
Si (ComprobarTabla(tablapaginacion) == verdadero)  
entonces
```

```
Desde i=0 hasta número de filas del arreglo  
Si tablapaginacion[i]->bitV-l = 0 entonces  
tablapaginacion[i]<- paginaEntrante;  
tablapaginacion[i]->bitReferencia =1;  
ColaMarcosC.push(paginaEntrante);
```

```
finSi;  
siguiente;
```

```
finLoop;
```

sino

```
Mientras (ColaMarcosC->bitReferencia!= 0 Y  
bitMoficacion != 0 )
```

```
pagina paginaAUX <- ColaMarcosC.pop();
```

```
Desde i=0 hasta número de filas del arreglo
```

```
Si tablapaginacion[i] == paginaAUX
```

## Sustitución de paginas FIFO

FIFO

variables

Estructura pagina paginaEntrante

instrucciones : tipo\_de\_elemento;

bitV-I : caracter;

array de tipo pagina tablapaginacion;

entero i,j;

Estructura Cola de paginas ColaP;

booleano bandera <- falso;

Funcion ComprobarTabla(tablapaginacion)

booleano bandera <- falso;

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i][1] = V entonces

bandera <- verdadero;

finSi;

siguiente;

finLoop;

retornar bandera;

Inicio

Si (ComprobarTabla(tablapaginacion) == verdadero)

entonces

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i]->bitV-I = v entonces ( v

significa valido, se puede asignar)

tablapaginacion[i]<- paginaEntrante;

ColaP.push(pagina);

finSi;

siguiente;

finLoop;

sino

pagina paginaAUX <- ColaP.pop();

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i] == paginaAUX entonces

tablapaginacion[i]<- paginaEntrante\

finSi;

finSi;

fin



# Sustitución optima de paginas

variables

Estructura pagina paginaEntrante

instrucciones : tipo\_de\_elemento;

rotulo : numero\_de\_instrucciones;

bitV-I : caracter;

array de tipo pagina tablapaginacion;

entero i,j,mayor,posicion;

Estructura Cola de paginas ColaP;

Funcion ComprobarTabla(tablapaginacion)

booleano bandera <- falso;

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i][1] = V entonces

bandera <- verdadero;

finSi;

siguiente;

finLoop;

retornar bandera;

Inicio

Si (ComprobarTabla(tablapaginacion) == verdadero)

entonces

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i]->bitV-I = v entonces ( v

significa valido, se puede asignar)

tablapaginacion[i]<- paginaEntrante;

ColaP.push(pagina);

finSi;

siguiente;

finLoop;

sino

mayor <- tablapaginacion[0]->;

Desde i=0 hasta número de filas del arreglo

Si(tablapaginacion[i]->rotulo > mayor) entonces

mayor <- tablapaginacion[i]->rotulo;

posicion <- i;

finSi;

siguiente;

fin;

tablapaginacion[posicion] <- paginaEntrante;

finSi;

## Sustitucion de segunda oportunidad

variables

Estructura página paginaEntrante;

instrucciones: tipo\_de\_elemento;

bitReferencia : entero;

array de tipo página tablapaginacion;

entero i,j;

colaCircular de tipo paginas ColaMarcosC;

Funcion ComprobarTabla(tablapaginacion)

booleana bandera <- falso;

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i][1] = 0 entonces

bandera <- verdadero;

finSi;

siguiente;

finLoop;

retornar bandera;

Inicio

Si (ComprobarTabla(tablapaginacion) == verdadero)

entonces

Desde i=0 hasta número de filas del arreglo

Si tablapaginacion[i]->bitV-I = 0 entonces

tablapaginacion[i]<- paginaEntrante;

tablapaginacion[i]->bitReferencia =1;

ColaMarcosC.push(paginaEntrante);

finSi;

siguiente;

finLoop;

sino

Mientras (ColaMarcosC->bitReferencia != 0 )

pagina paginaAUX <- ColaMarcosC.pop();

Desde i=0 hasta número de filas del arreglo

Si tablapaginacion[i] == paginaAUX

entonces

tablapaginacion[i]<- paginaEntrante\

finSi;

finSi;

Sino

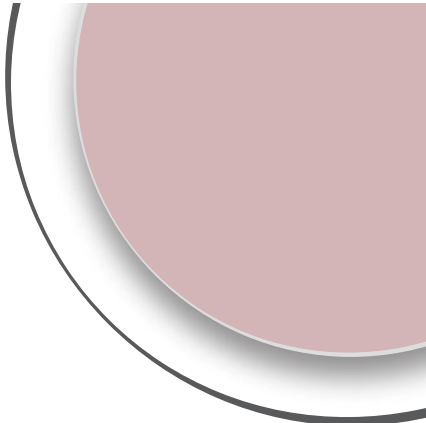
pagina paginaAUX <- ColaMarcosC.consulta();

Desde i=0 hasta número de filas del arreglo

Si tablapaginacion[i] == paginaAUX

entonces

tablapaginacion[i]->bitReferencia =0;



```
entonces
tablapaginacion[i]<- paginaEntrante\
finSi;
finSi;
Sino
pagina paginaAUX <- ColaMarcosC.consulta();
Desde i=0 hasta número de filas del arreglo
Si tablapaginacion[i] == paginaAUX
entonces
tablapaginacion[i]->bitReferencia =0;
finSi;
finSi;
ColaMarcosC<- ColaMarcosC->nextPagina;
finSi;
finLoop;
finSi;
Fin
```

## Sustitución de paginas LRU

Algoritmo LRU

variables

Estructura pagina paginaEntrante

instrucciones : tipo\_de\_elemento;

tiempo: entero;

bitV-I : caracter;

array de tipo pagina tablapaginacion;

entero i,j,mayor,posicion;

Estructura Cola de paginas ColaP;

Funcion ComprobarTabla(tablapaginacion)

booleano bandera <- falso;

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i][1] = V entonces

bandera <- verdadero;

finSi;

siguiente;

finLoop;

retornar bandera;

Inicio

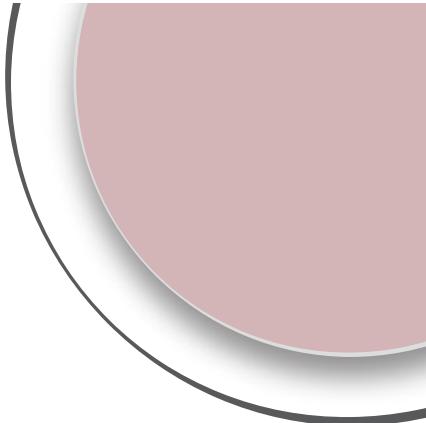
Si (ComprobarTabla(tablapaginacion) == verdadero)

entonces

Desde i=0 hasta numero de filas del arreglo

Si tablapaginacion[i]->bitV-I = v entonces (v  
significa valido, se puede asignar)



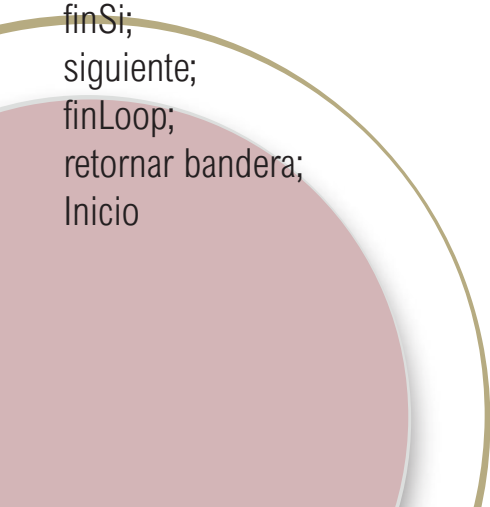


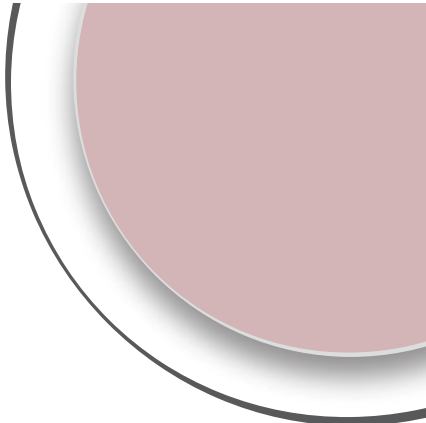
```
ColaP.push(pagina);
finSi;
siguiente;
finLoop;
sino
mayor <- tablapaginacion[0];
Desde i=0 hasta numero de filas del arreglo
mayor <- tablapaginacion[i];
Desde i=0 hasta numero de filas del arreglo
Si(tablapaginacion[i]->tiempo > mayor) entonces
mayor <- tablapaginacion[i]->tiempo;
posicion <- i;
finSi;
siguiente;
fin;
tablapaginacion[posicion] <- paginaEntrante;
finSi;
Fin
```

## Sustitución de páginas basada en contador

variables

Estructura pagina paginaEntrante  
instrucciones : tipo\_de\_elemento;  
Ccorrimientos: entero;  
bitV-I : caracter;  
array de tipo pagina tablapaginacion;  
entero i,j,menor,posicion;  
Estructura Cola de paginas ColaP;  
Funcion ComprobarTabla(tablapaginacion)  
booleano bandera <- falso;  
Desde i=0 hasta numero de filas del arreglo  
Si tablapaginacion[i][1] = V entonces  
bandera <- verdadero;  
finSi;  
siguiente;  
finLoop;  
retornar bandera;  
Inicio





```
Si (ComprobarTabla(tablapaginacion)== verdadero) entonces
Desde i=0 hasta numero de filas del arreglo
Si tablapaginacion[i]->bitV-I = v entonces (v
significa valido, se puede asignar)
tablapaginacion[i]->Ccorrimientos<-
tablapaginacion[i]->Ccorrimientos>> 1;
tablapaginacion[i]<- paginaEntrante;
finSi;
siguiente;
finLoop;
sino
Desde i=0 hasta numero de filas del arreglo
Si tablapaginacion[i]==paginaEntrante entonces
tablapaginacion[i]->Ccorrimientos<-
tablapaginacion[i]->Ccorrimientos>> 1;
Sino
menor <- tablapaginacion[0]->Ccorrimiento;
Desde i=0 hasta numero de filas del arreglo
Si(tablapaginacion[i]->Ccorrimiento <
menor) entonces
menor <- tablapaginacion[i]-
>Ccorrimiento;
posicion <- i;
finSi;
siguiente;
```

