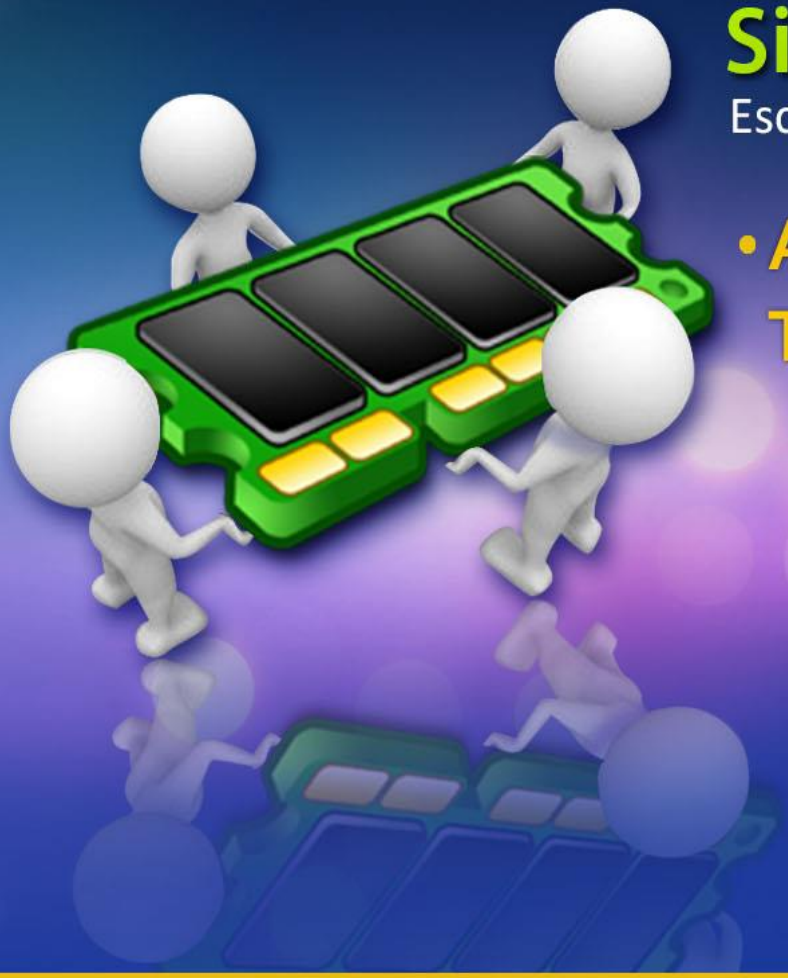


Sistemas Operativos

Escuela de Ingeniería Civil Informática

- Administración de Memoria
  Técnica de Segmentación

UNIVERSIDAD DEL BÍO-BÍO

LA SEGMENTACIÓN

- La **SEGMENTACIÓN** es una técnica creada en la Universidad de Harvard
- Es un mecanismo primitivo usado muy antiguamente
- Se usó en procesadores PDP-11 en las primeras implementaciones de Unix.
- Divide procesos en 4 segmentos: código, dato, pila y sistema.
- En la segmentación cada segmento ocupa un trozo de memoria real **contiguo**.

LA SEGMENTACIÓN

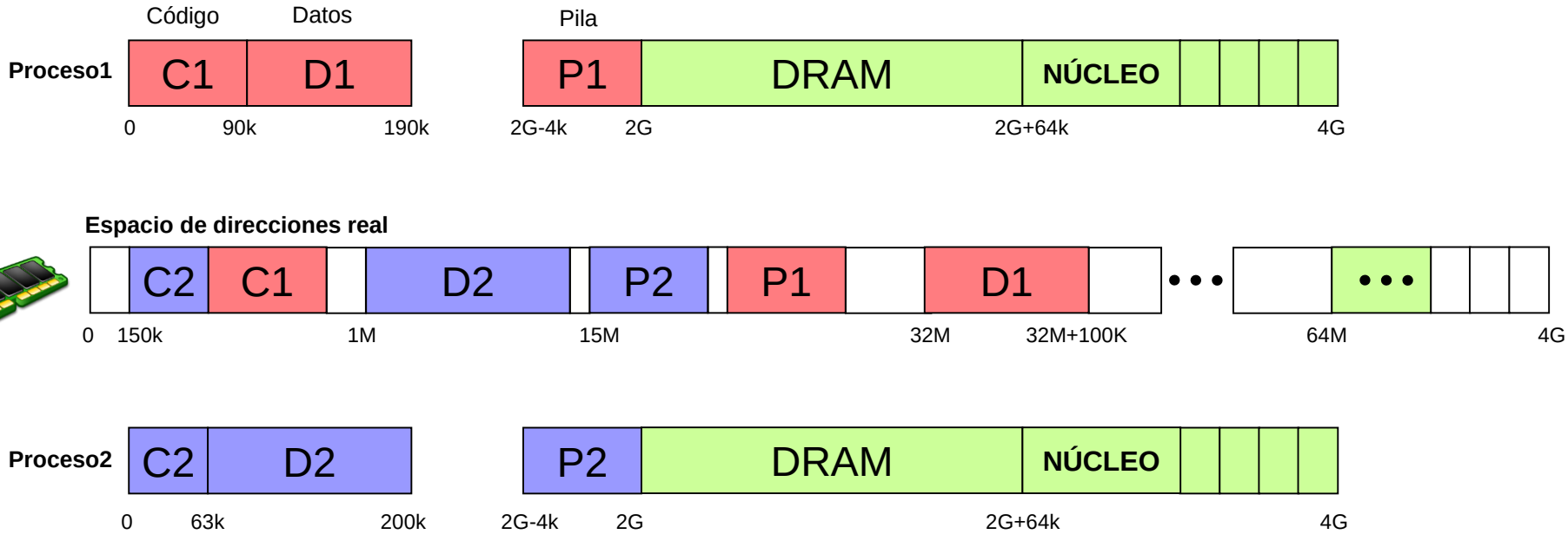


TABLA DE SEGMENTOS

- Se utiliza una **Tabla de Segmentos**, compuesta por aproximadamente 16 registros en la CPU.
- En ella se indica como traducir direcciones virtuales a direcciones reales para el proceso en ejecución.

TABLA DE SEGMENTOS

- Para cada uno de los 4 segmentos se indica:
  - Dirección virtual base (ejemplo: 63 KB para D2)
  - Dirección virtual límite (ejemplo: 200 KB para D2)
  - Desplazamiento es la diferencia entre base real y base virtual (1MB-63KB para D2).
  - Atributos (ejemplo: lectura y escritura para D2: +R, +W)

	Base	Límite	Desplazamiento	Atributos
Código	bc	lc	rc – bc	+R, +W
Datos	bd	ld	rd – bd	+R, +W
Pila	bp	lp	rp – bp	+R, +W
Sistema	2G	4G	0-2G	-R, -W

bc = base virtual código
rc = base real código
lc = límite virtual código
…

Ejemplo Dir. real para D2 = dirección virtual base D2 + desplazamiento D2 (rd D2 – bd D2)

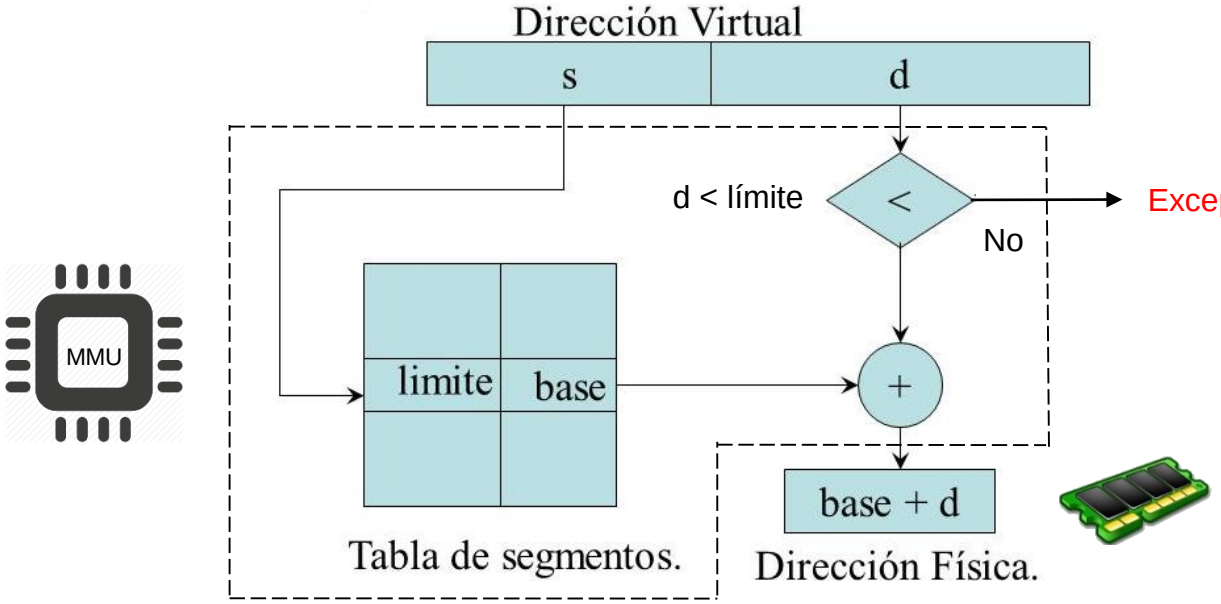
TRADUCCIÓN EN LA SEGMENTACIÓN

- Cuando un proceso accede a una ubicación de la memoria, la dirección virtual se traduce de la siguiente manera:

La MMU (memory management unit) busca un segmento **X**, tal que:

*base(**X**) ≤ dirección virtual < límite(**X**)*

→ dirección real: *dirección virtual base(**X**) + desplazamiento(**X**)*



TRADUCCIÓN EN LA SEGMENTACIÓN

- Tiempo requerido: 1 ciclo de reloj

- Esta tabla del procesador sólo almacena los valores para el proceso en ejecución y en el descriptor de proceso (en memoria) se almacena la tabla de segmentos cuando no está en ejecución.

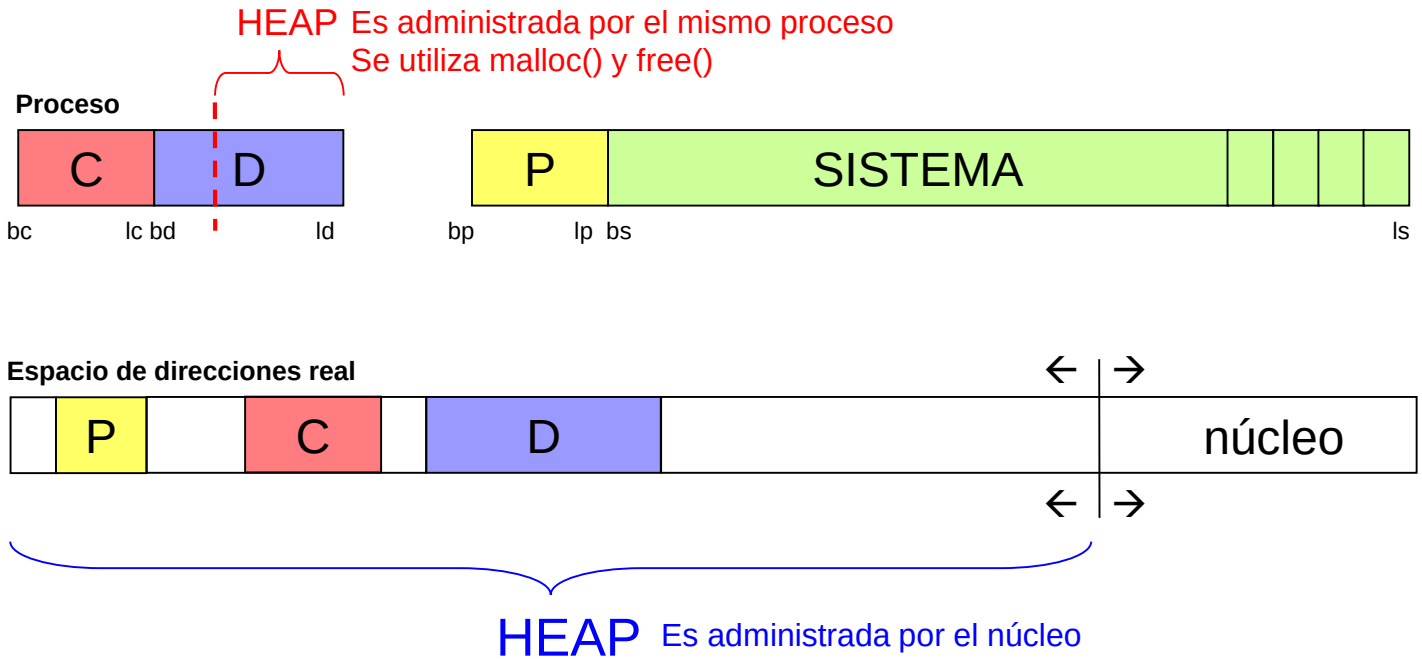
- En un cambio de contexto, estos valores se copian en la tabla de segmentos del procesador y se cargan con la tabla de segmentos del nuevo proceso.

- Optimización: se cargan sólo 3 segmentos ya que el segmento sistema no cambia.

ADMINISTRACIÓN DE SEGMENTOS EN EL HEAP

- Los bloques de memoria libre deben mantenerse identificables para su asignación en cualquier momento.

- La memoria se debe administrar para que no se asigne el mismo bloque dos veces.



ADMINISTRACIÓN DE SEGMENTOS EN EL HEAP

- Los trozos de memoria se organizan comúnmente como una lista enlazada.

- Algunas estrategias empleadas para administrar los trozos de memorias son:

  - **First Fit**
    - Realiza una búsqueda de bloques disponibles en la memoria
    - Asigna el primer bloque disponible que sea igual o mayor que el tamaño solicitado
    - Es en promedio mejor pero si se construye una lista circular

  - **Best Fit**
    - Realiza una búsqueda de bloques disponibles en la memoria
    - Asigna el bloque que calce más exactamente con el tamaño de memoria solicitado
    - Tiende a producir trozos pequeños dispersos de memoria libre

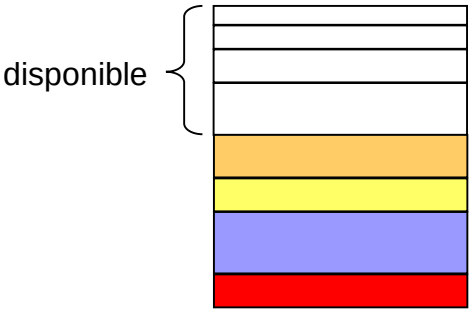
- Ambas estrategias producen finalmente **FRAGMENTACION**!!

¿CÓMO SOLUCIONAR LA FRAGMENTACIÓN?

- Visualmente la fragmentación se puede ver así en la memoria:

Se puede dar que la suma de los trozos es suficiente pero ninguno de ellos es del tamaño que se solicita.

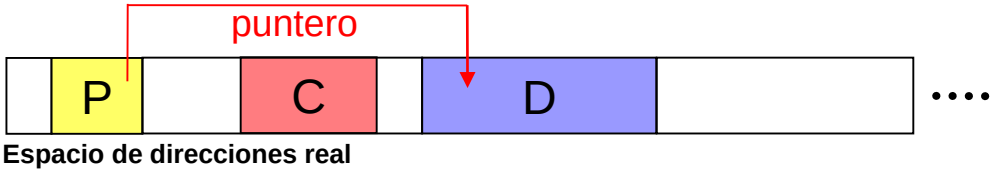
- Solución a este problema: usar **COMPACTACIÓN**



Las direcciones virtuales no cambian.

La compactación es a nivel de memoria real, por lo tanto, lo único que varía es el campo desplazamiento de la tabla de segmentos.

Al ajustar el desplazamiento, se ajustan automáticamente los punteros.



POTENCIAL DE LA SEGMENTACIÓN

- Algunas características que se pueden implementar fácilmente en Unix mediante la segmentación, son:

- **Extensión automática de la pila (en demanda):**

  - Cuando ocurre un desborde de pila, se produce una interrupción y se verifica si la dirección cae lejos del borde de la pila (ej: más allá de los 4M – tamaño máximo).

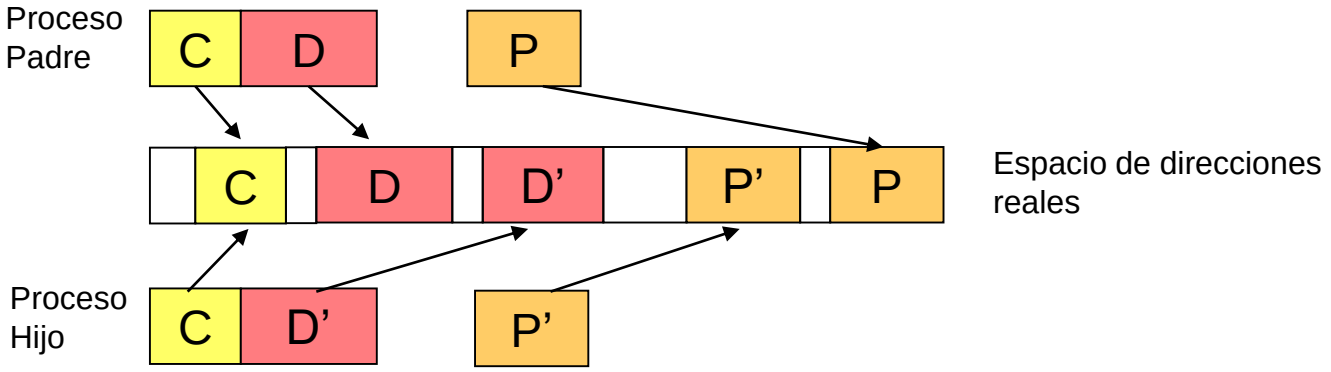
  - Si la dirección sobrepasa por poco al borde, se le asigna más memoria de la siguiente manera:
    - Se solicita un segmento de pila más grande.
    - Se copia el contenido a ese nuevo segmento
    - Se corrige el desplazamiento en la tabla de segmentos
    - Se retoma la ejecución

POTENCIAL DE LA SEGMENTACIÓN

- **Extensión explícita del segmento de datos:**
  - Esto se hace a nivel de proceso (malloc y free), en unix se pide más espacio para el segmento de datos con la llamada al sistema **sbrk(nbytes)** (el usuario debe ejecutar esta instrucción, por lo cual no es transparente).
  - El resto es similar a la pila.

POTENCIAL DE LA SEGMENTACIÓN

- **Implementación de fork:**
  - fork es una llamada al sistema de unix/linux
  - crea un nuevo proceso clon del padre. El padre recibe el PID del hijo y el hijo recibe cero.



  - En Unix, tanto el padre como el hijo tienen el mismo código.
  - En el caso de los Unix que permiten modificar el código, este se marca como sólo lectura y luego se duplica cuando se quiere escribir en él.

POTENCIAL DE LA SEGMENTACIÓN

- **Problema de Implementación de fork:**
  - Es ineficiente porque el 99% de los casos de uso es una secuencia fork/exec que descarta de inmediato los segmentos duplicados.
  - Ejemplo: en el shell, cada vez que se ejecuta un comando se hace un fork. Luego, es decir, pocas instrucciones después se hace exec y luego se descartan los segmentos.
  - Si el proceso es grande → el proceso al duplicarse consume mucho tiempo y se inhiben las interrupciones (por mucho tiempo).
  - Con esto se podría implementar Unix pero era ineficiente por la implementación de fork.

POTENCIAL DE LA SEGMENTACIÓN

- **Problema de Implementación de fork:**
  - Es ineficiente porque el 99% de los casos de uso es una secuencia fork/exec que descarta de inmediato los segmentos duplicados.
  - Ejemplo:

```
if ( fork()==0 ) { /*proceso hijo */
   execlp ("ls","ls","-l","/usr/include",0);
   printf ("Si ves esto, no se pudo ejecutar el programa\n");
   exit(1);
}
else { /* proceso padre */
   int rc;
   wait(&rc);
}
```

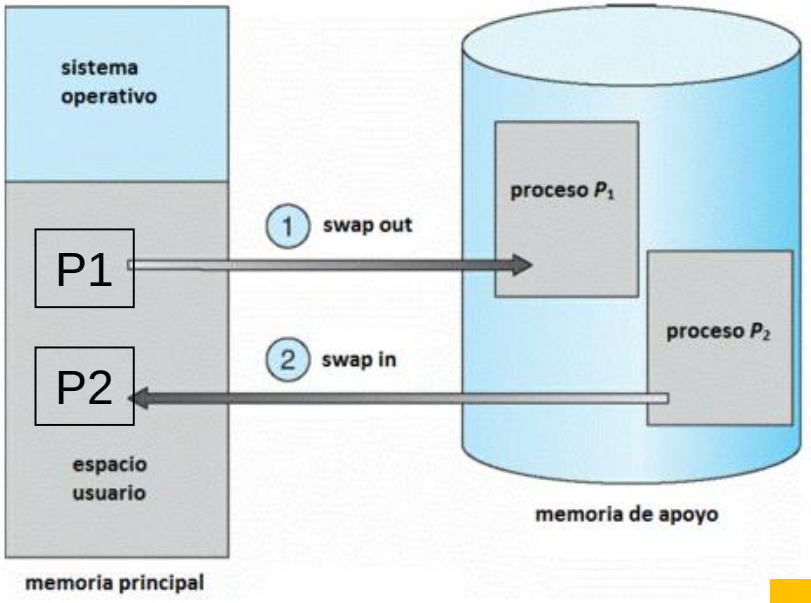
```
$> ls -l /usr/include
```

int **execlp**(const char ***file***, const char ***arg0***, ..., const char ***argn***, char * /*NULL*/);

- const char ***file***: ruta al programa en disco
- const char ***arg0***, ..., const char ***argn***: parámetros pasados al programa (igual que cuando se invoca un programa desde el intérprete de comandos
- char * /*NULL*/: NULL para indicar fin de los parámetros, por portabilidad

POTENCIAL DE LA SEGMENTACIÓN

- **Swapping:**
  - Cuando se acaba la memoria se llevan los procesos completos a disco.
  - Código, pila y datos pero el descriptor de proceso permanece en memoria, el estado del proceso ahora dice “SWAP”.
  - En la tabla de segmentos se coloca ahora el Nº de bloque del disco en el cual se almacenó el proceso.
  - El segmento de sistema no se puede llevar a disco porque es compartido y siempre residente en memoria.
  - Quien decide es el scheduler de mediano plazo (tiene un tiempo más prolongado que las tajadas → permanencia en disco).



POTENCIAL DE LA SEGMENTACIÓN

- Este scheduler decide que proceso llevar a disco. Administra la memoria, es decir, que los procesos quepan en memoria.

- Cuando decide llevar un proceso a disco lo lleva por lo menos 10 segundos. Eso se puede apreciar por ejemplo, en el retraso en la respuesta al presionar una tecla mientras el proceso está en disco por falta de memoria.

POTENCIAL DE LA SEGMENTACIÓN

- Desventaja de la segmentación:
  - fork ineficiente
  - la compactación introduce una pausa
  - **el tamaño de un proceso no puede exceder el tamaño de la memoria real**.

¿Cómo solucionar estos problemas?