Tema 5

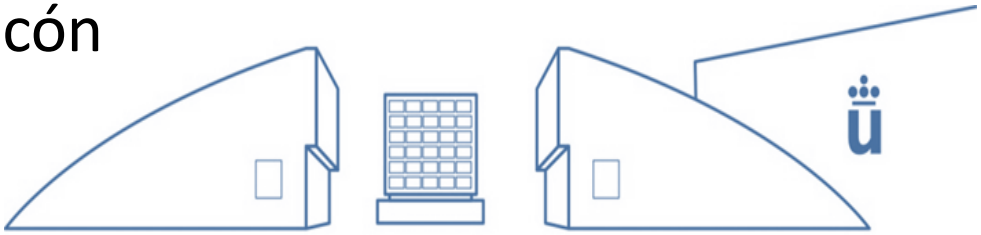
Gestión de memoria

Alberto Sánchez

Sofia Bayona

Luis Rincón

Universidad Rey Juan Carlos



Bibliografía

- **“Sistemas operativos: una visión aplicada”.** *Jesús Carretero, Felix García Carballeira, Pedro de Miguel y Fernando Pérez Costoya*
  - Capítulo 4: 4.1, 4.2, 4.4, 4.6, 4.7

Índice

- El S.O. multiplexa los recursos entre los procesos
  - Cada proceso cree que tiene una máquina para él solo.
    - Gestión de procesos: Reparto de procesador.
    - Gestión de memoria: Reparto de memoria.
- El gestor de memoria tiene dos facetas complementarias.
  - Servidor de memoria para los procesos.
  - Soporte a la memoria virtual.
- Los procesos no entienden del soporte físico del mapa de memoria, ya sea este soporte memoria principal o memoria virtual, solamente entienden de direcciones dentro del mapa de memoria del procesador.
  - Cuando el gestor de memoria asigna un marco de página a un proceso, no significa que el proceso vea más memoria. Simplemente establece un soporte físico más rápido a una zona de la memoria del proceso.
  - Existe hardware de apoyo para ayudar al servidor de memoria: **MMU** (unidad de manejo de memoria) encargado de detectar los problemas

Servidor de memoria de procesos

- Funciones del servidor de memoria
  - Ofrecer a cada proceso los recursos de memoria necesarios, dando soporte a las regiones necesarias.
  - Controlar qué direcciones de memoria están ocupadas y cuáles libre.
  - Controlar la memoria principal y de intercambio ocupada y libre.
  - Crear la imagen de los procesos a partir de los ficheros ejecutables.
  - Proporcionar protección entre procesos. Aislar los procesos.
  - Tratar los errores producidos en el acceso a memoria: detectados por el HW.
  - Permitir que los procesos puedan compartir memoria de forma controlada.
  - Recuperar la memoria cuando los procesos no la requieran.
  - Optimizar las prestaciones del sistema.
  - Proporcionar grandes espacios de memoria a los procesos.

Índice

Memoria del proceso

- Las necesidades de memoria de un proceso se resuelven a dos niveles:
  - Por el SO: Visión global o macroscópica consistente en regiones.
    - Regiones: grandes trozos de memoria contigua.
  - Por las bibliotecas del lenguaje usado para el programa: Visión de detalle o microscópica consistente en objetos (variables y estructuras de datos) dentro de las regiones.
    - Las bibliotecas del lenguaje utilizado en el desarrollo del programa gestionan el espacio disponible en la región de datos dinámicos.
    - Solamente llaman al SO cuando tienen que variar el tamaño de la región, o crear una nueva región.

Imagen de memoria

- Imagen de memoria: conjunto de regiones (o segmentos) de memoria asignados a un proceso.
- Características de una región
  - Es una zona contigua de direcciones de memoria definida por:
    - Una dirección de comienzo.
    - Un tamaño.
  - Fuente: lugar donde se almacena el valor inicial.
  - Puede ser compartida o privada.
  - Niveles de protección típicos: RWX.
  - Puede tener tamaño fijo o variable.

Regiones del proceso

- Las regiones más relevantes de la imagen de memoria del proceso son:
  - Código (texto): Contiene el código máquina del programa.
  - Datos, que se organiza en:
    - Datos con valor inicial: Variables globales inicializadas.
    - Datos sin valor inicial: Variables globales no inicializadas.
    - Datos creados dinámicamente o heap.
  - Pila: soporta los registros de activación de los procedimientos.
- La estructuración en regiones depende del diseño del SO.
  - Puede haber una sola región que englobe datos con valor inicial, datos sin valor inicial y datos creados dinámicamente.
  - O puede haber regiones separadas para distintos tipos de datos.

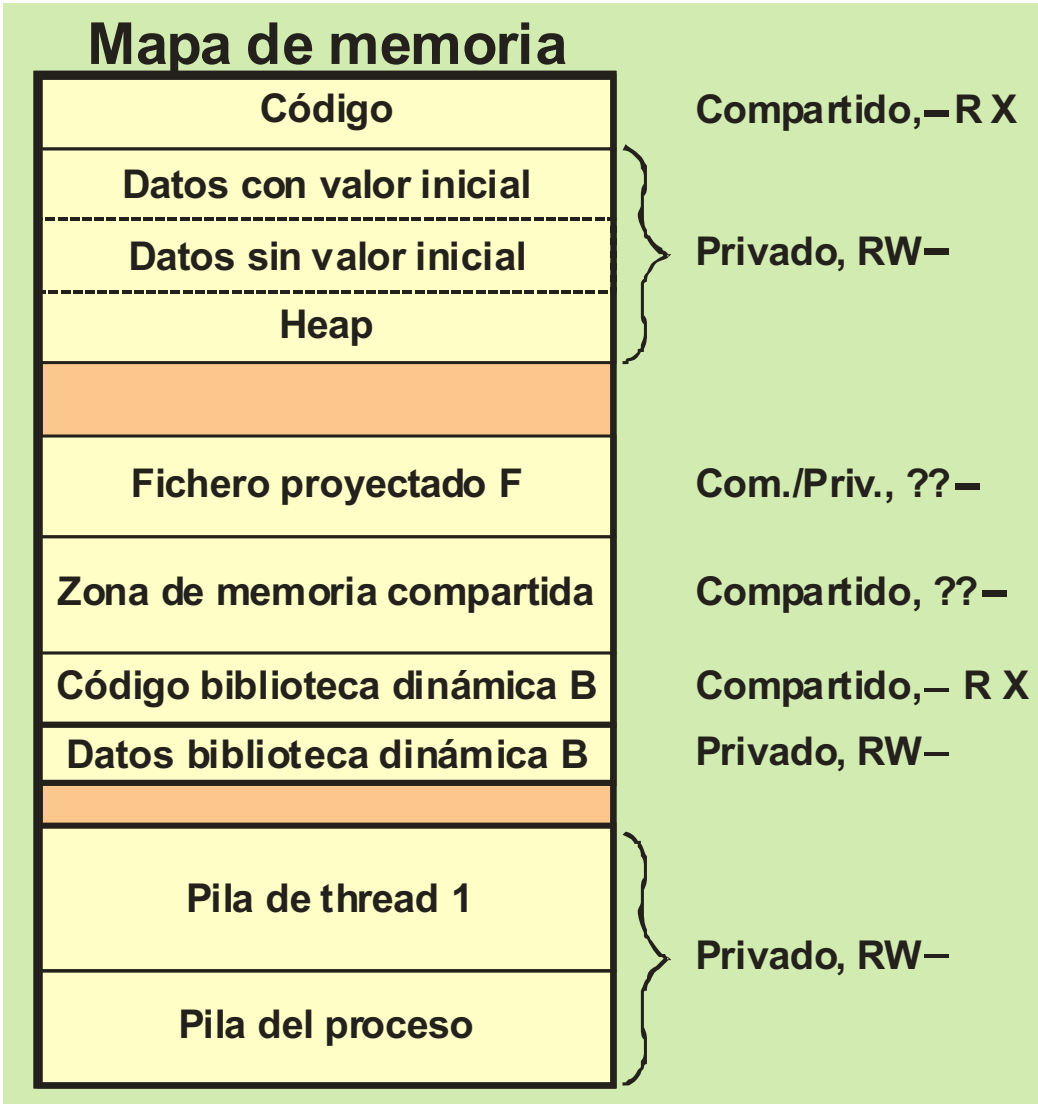
Características de las regiones

- Características típicas de las regiones creadas en el arranque del proceso:
  - Región de código.
    - Compartida, RX, tamaño fijo, fuente: el fichero ejecutable.
  - Región de datos con valor inicial.
    - Privada, RW, tamaño fijo, fuente: el fichero ejecutable.
  - Región de datos sin valor inicial.
    - Privada, RW, tamaño fijo, fuente: rellenar con 0.
  - Región de pila.
    - Privada, RW, tamaño variable, fuente: rellenar con 0.
    - Pila inicial (creada al arrancar el programa):
      - Variables de entorno.
      - Argumentos del programa.

Otras regiones de la imagen

- Región de Heap.
  - Soporte de memoria dinámica gestionada por el lenguaje (p.e. malloc en C).
  - Persistencia controlada por el programador.
  - Privada, RW, tamaño variable, fuente: rellenar con 0.
- Memoria compartida.
  - Región asociada a la zona de memoria compartida.
  - Compartida, tamaño variable, fuente: rellenar con 0.
  - Protección especificada en proyección.
- Fichero proyectado.
  - Región asociada a cada fichero proyectado.
  - Compartida, tamaño variable, fuente: el fichero proyectado.
  - Protección especificada en proyección.
- Biblioteca dinámica.
  - Regiones asociadas al código y datos de cada biblioteca dinámica.
- Pilas de threads.
  - Cada pila de thread corresponde con una región.
  - Mismas características que pila del proceso.

Mapa de memoria

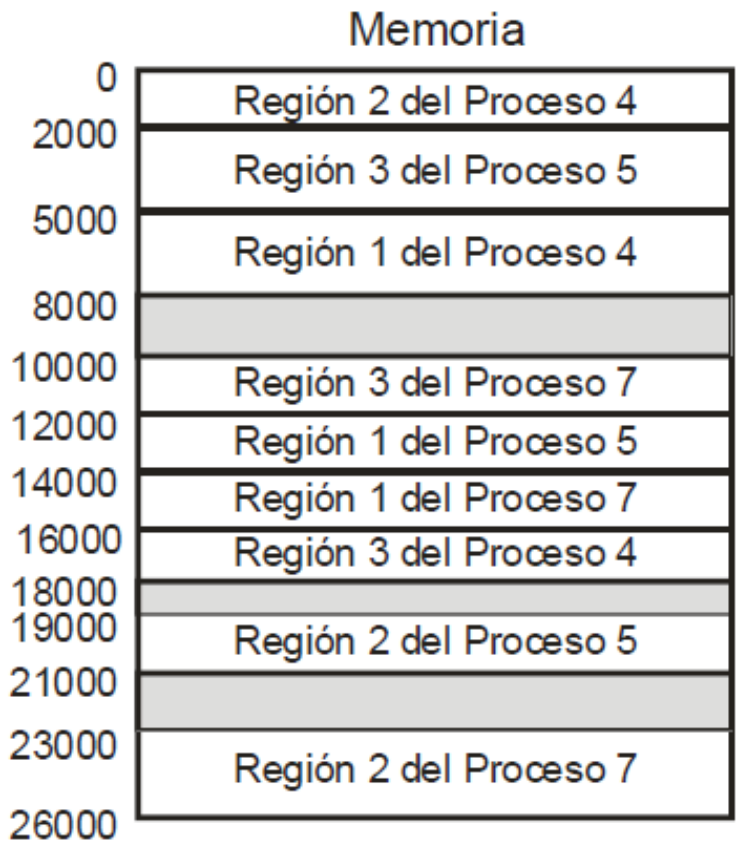


Esquemas de gestión de memoria

- En el nivel de procesos, existen varias formas de repartir la memoria principal disponible entre los procesos activos en el sistema:
  - Asignación contigua.
  - Segmentación.
  - Paginación.
  - Segmentación paginada.
- El esquema que más funcionalidad ofrece es la segmentación paginada.

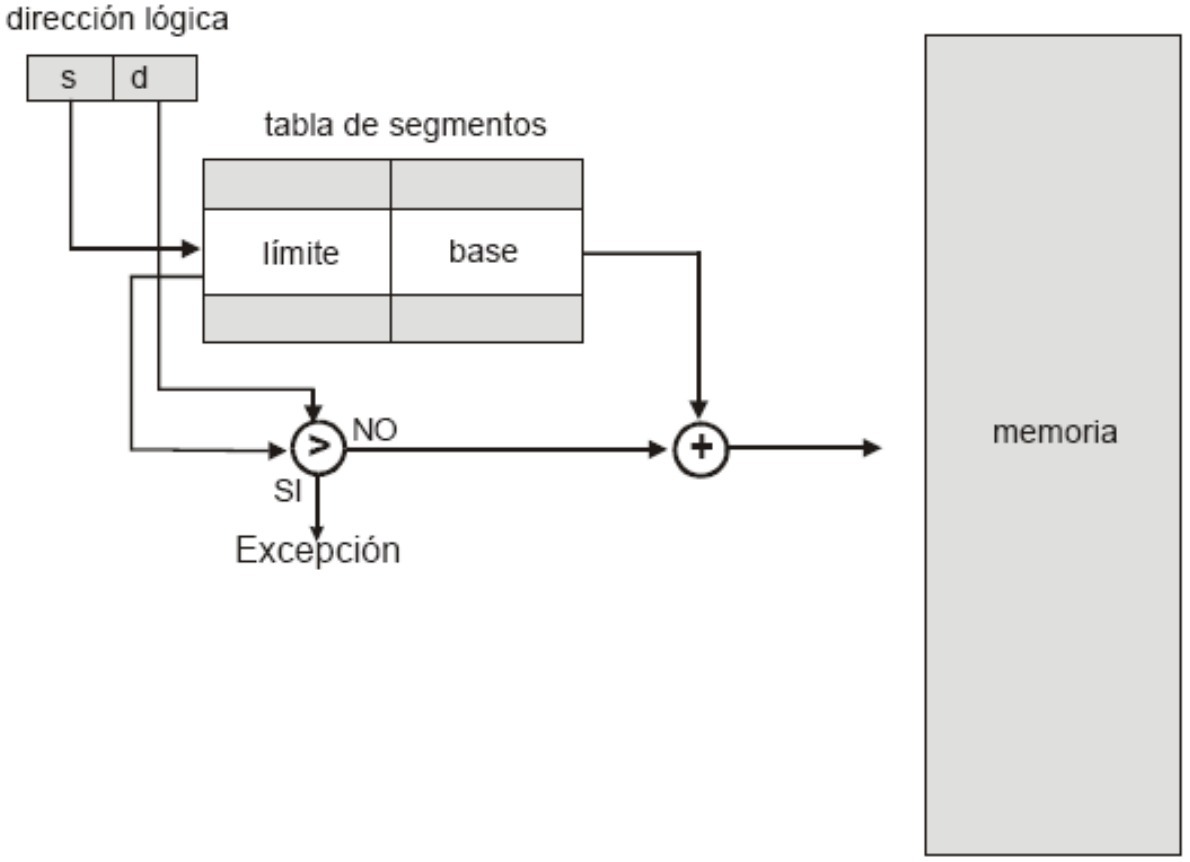
Segmentación de memoria

- Cada región del proceso corresponde con un segmento.
- Los segmentos de un proceso pueden estar separados.
- Cada segmento se almacena de forma contigua.
- El BCP contiene una tabla de segmentos.
- Información por cada segmento:
  - Dirección de carga.
  - Tamaño del segmento.
  - Bits de protección (RWX).
  - Bit de validez.
- Este sistema está obsoleto.



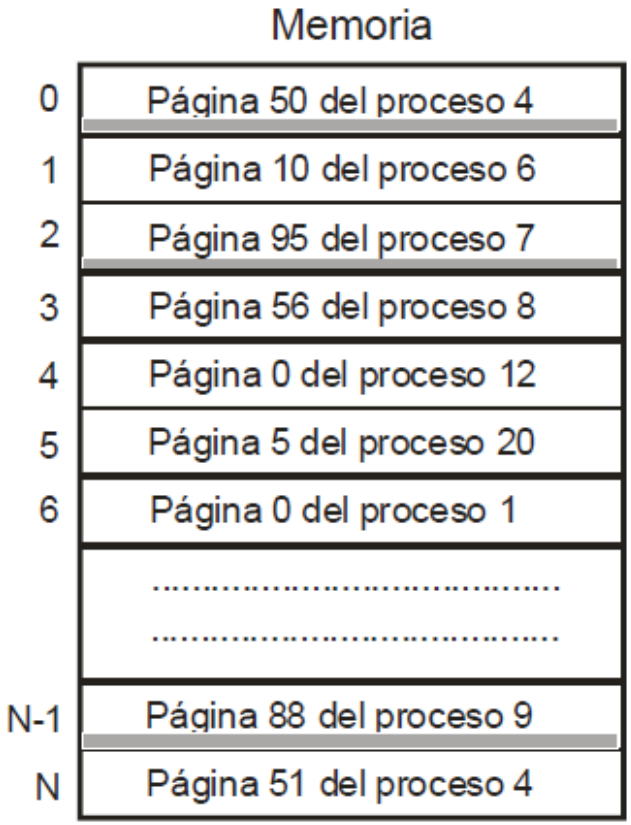
Segmentación de memoria

- Traducción de direcciones lógicas a físicas: realizada por la MMU.
  - La tabla de segmentos se copia en la MMU al realizar el cambio de contexto.



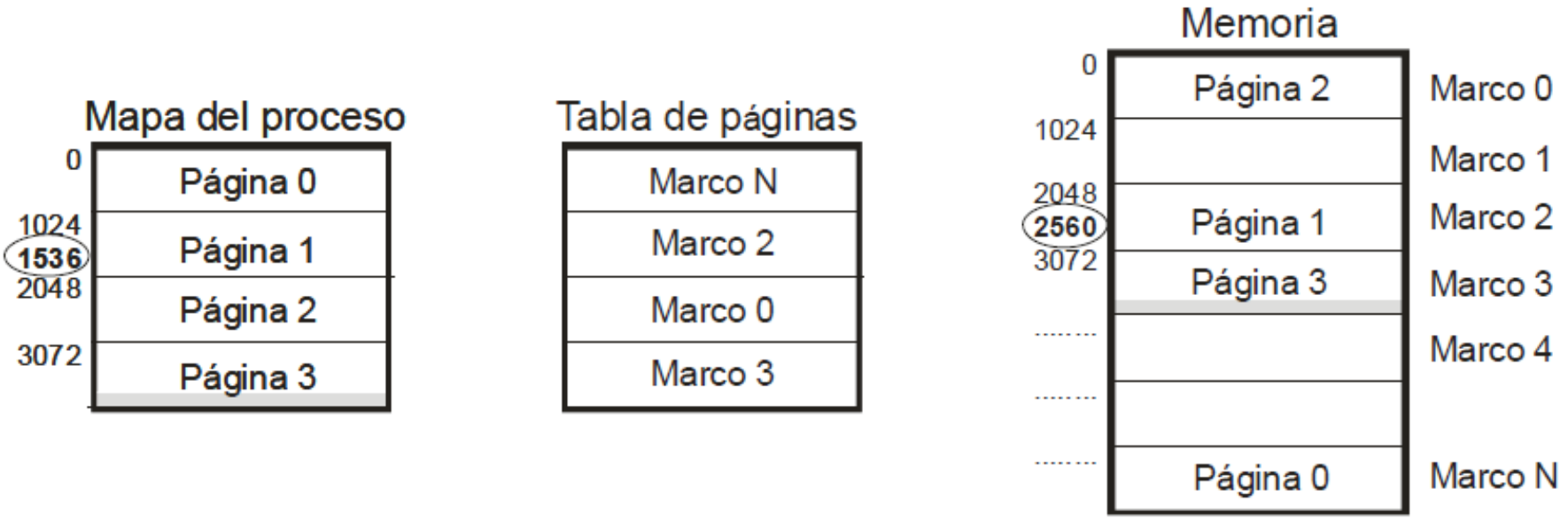
Paginación de memoria

- La memoria principal está dividida en huecos de igual tamaño denominados **marcos de página**.
- Mapa de memoria de un proceso: dividido en **páginas**.
  - Tamaño de una página = tamaño de un marco de página.
- A un proceso se le asigna un conjunto de marcos de página que no tienen por qué ser contiguos.
- Cada proceso cuenta con una **tabla de páginas**.
- Información por cada página:
  - Número del marco de página que le corresponde.
  - Bits de protección (RWX).
  - Bit de validez.
  - Etc.



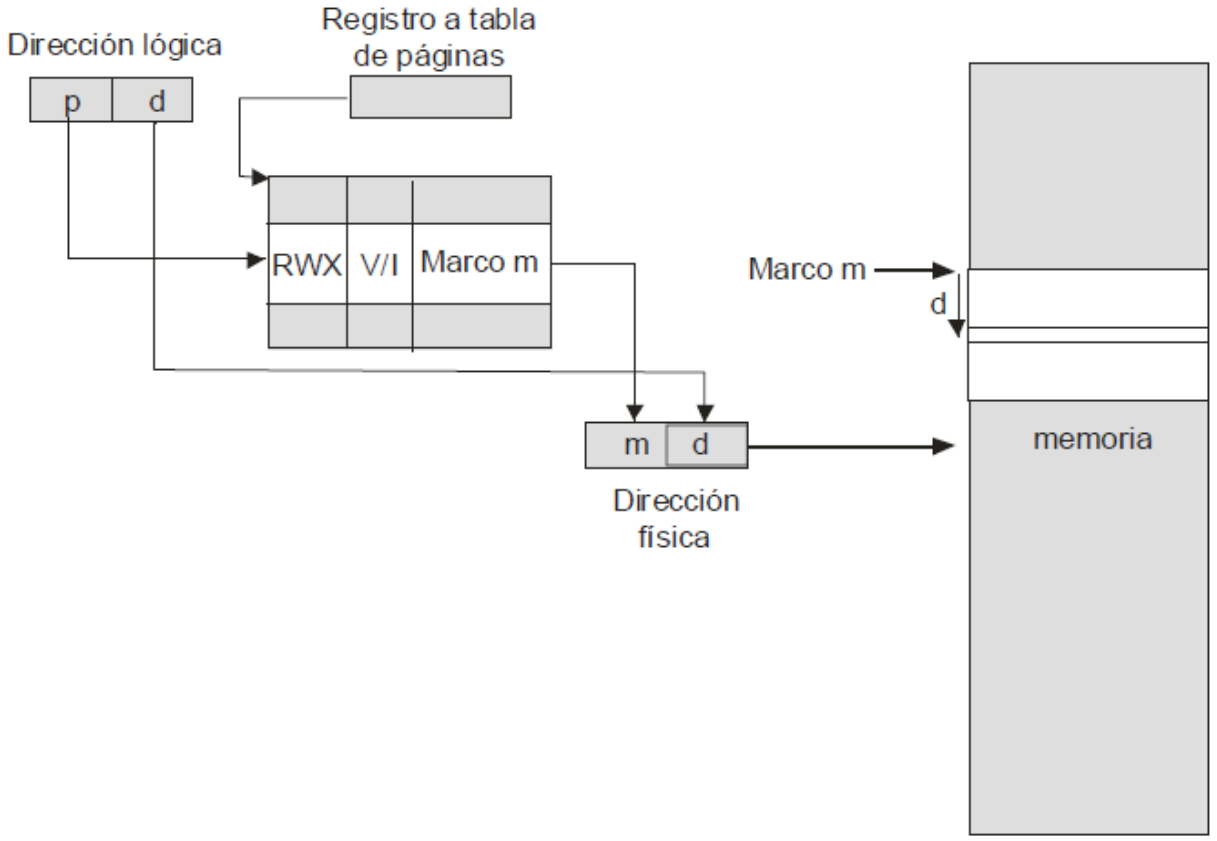
Paginación de memoria

- Las tablas de páginas de los procesos están en memoria principal (en espacio del S.O.).
- El BCP contiene un puntero a la tabla de páginas del proceso.



Paginación de memoria

- Traducción de direcciones lógicas a físicas: a través de la tabla de páginas.
  - El puntero a la tabla de páginas se copia en la MMU durante el cambio de contexto.
  - Para acelerar la traducción, la MMU cuenta con una TLB que es una pequeña caché de la tabla de páginas.

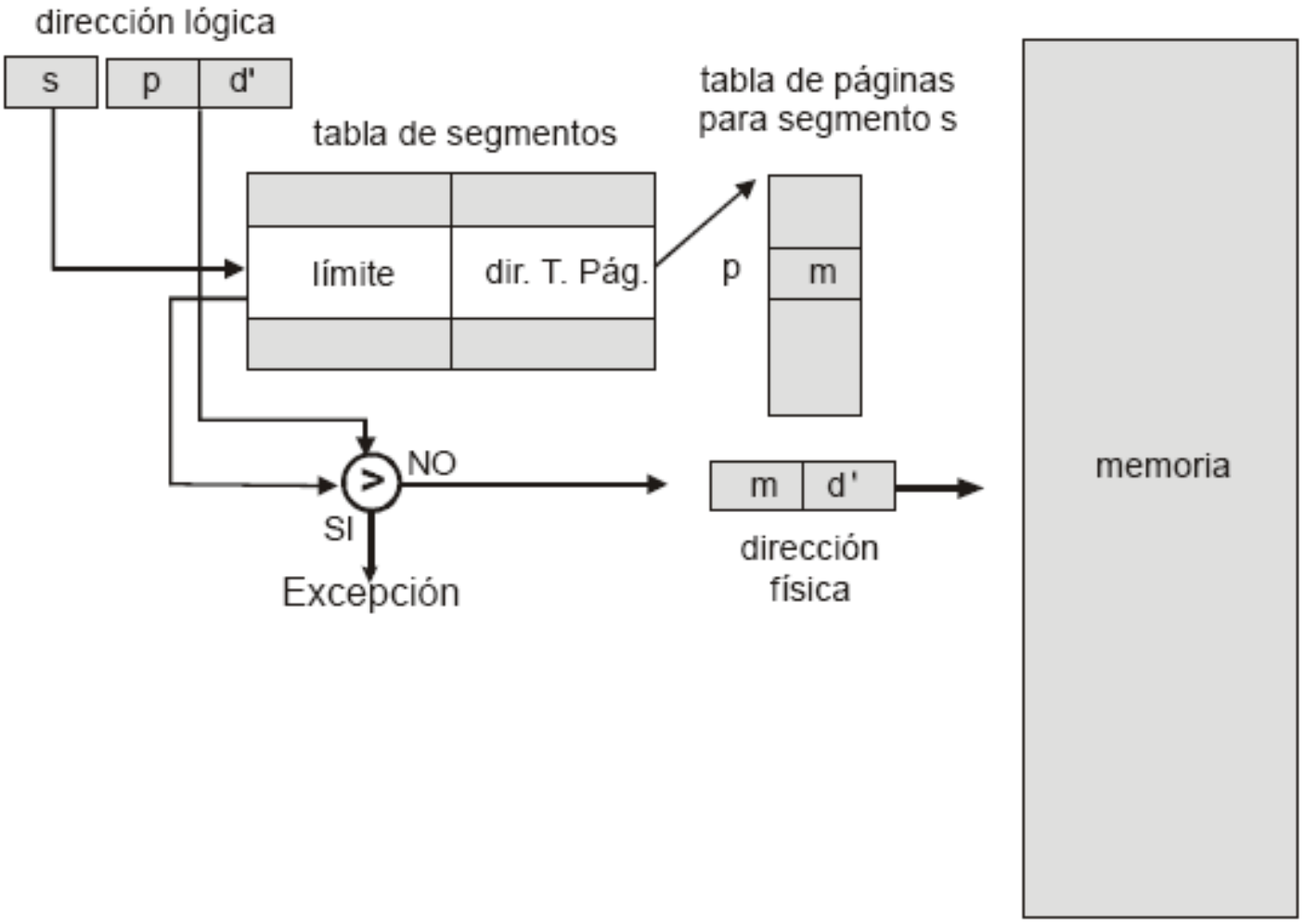


Segmentación paginada

- Combina las dos técnicas anteriores.
- Un proceso está formado por una serie de segmentos.
- Cada segmento ocupa uno o varios marcos de página, que no tienen por qué estar almacenados en memoria principal de forma contigua.
- Por cada proceso tenemos una tabla de segmentos (en el BCP).
- Dos modalidades:
  - Segmentación paginada local:
    - Hay un espacio lógico de direcciones por proceso.
    - A cada segmento se le asocia una tabla de páginas.
    - El puntero a cada tabla está en la propia tabla de segmentos.
  - Segmentación paginada global:
    - Hay un espacio único de direcciones para todos los procesos.
    - Hay una única tabla de páginas para todos los procesos.

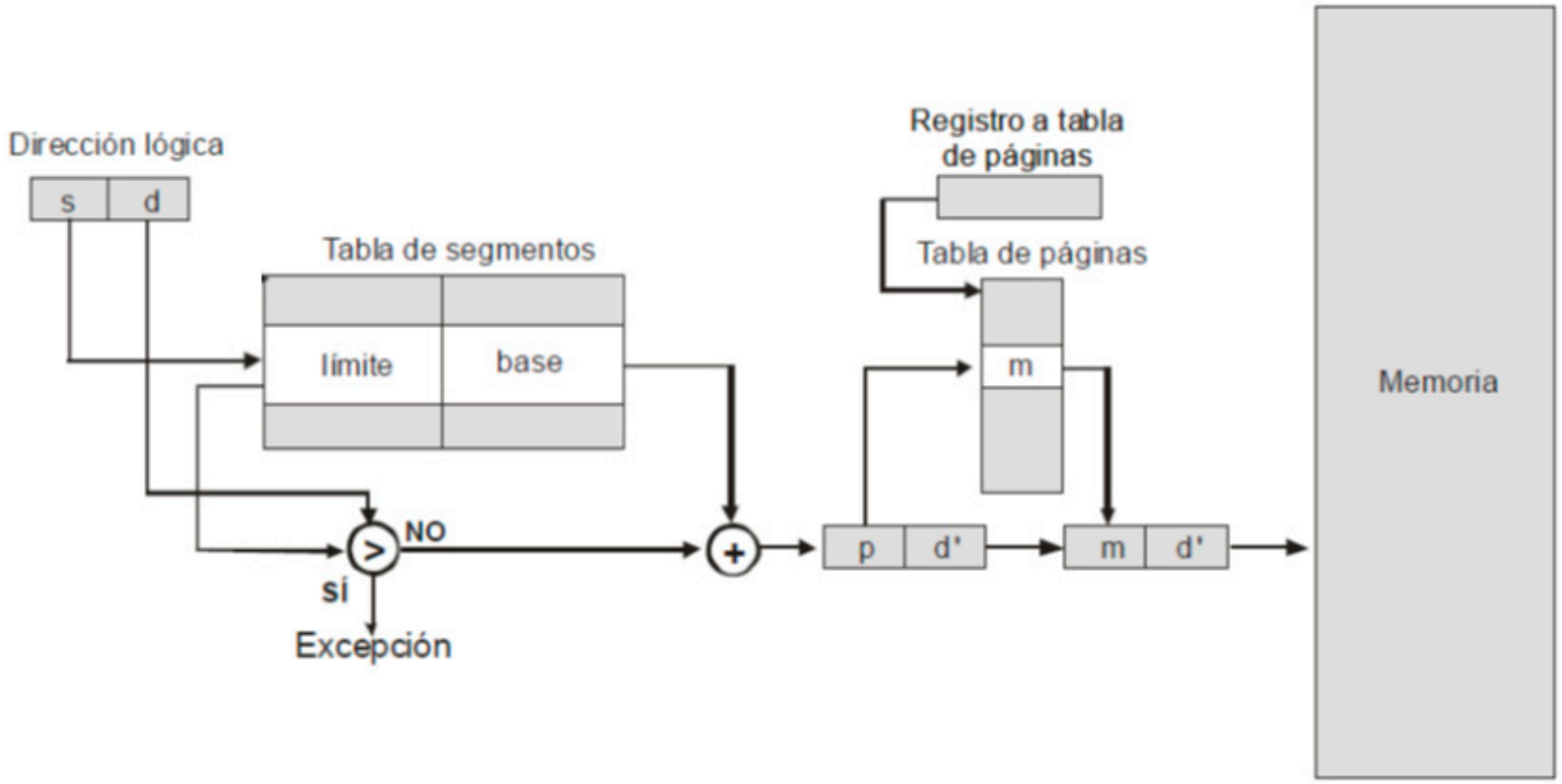
Segmentación paginada local

- Traducción de direcciones lógicas a físicas en segmentación paginada local.



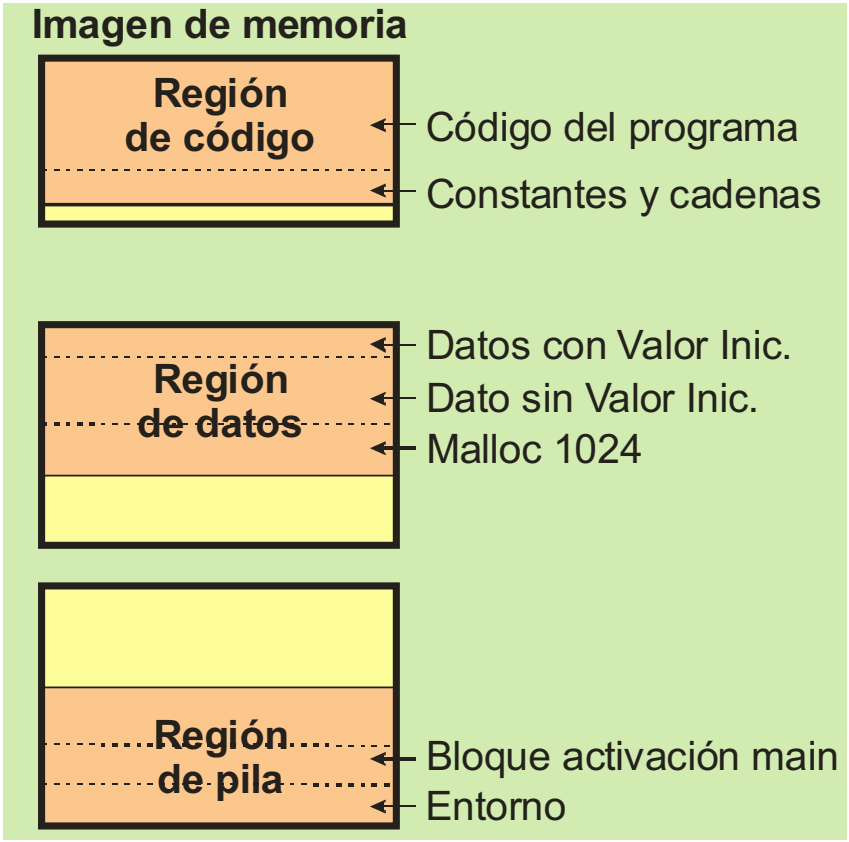
Segmentación paginada global

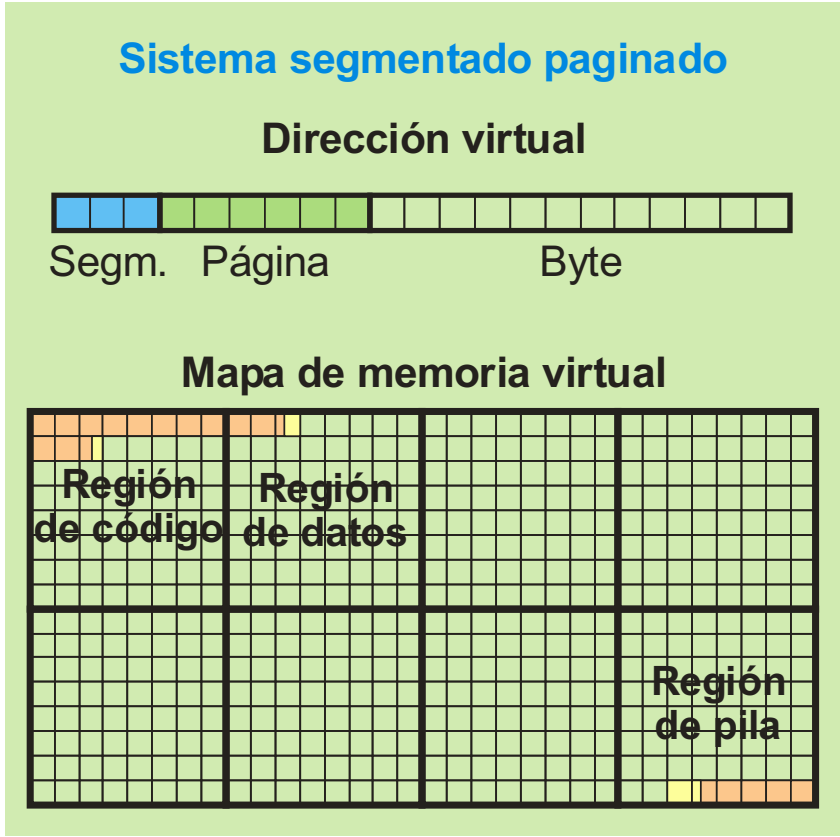
- Traducción de direcciones lógicas a físicas en segmentación paginada global.



Organización de las regiones

- En un sistema con memoria virtual el espacio de cada región es un múltiplo del tamaño de página. Parte de ese espacio estará desaprovechado.





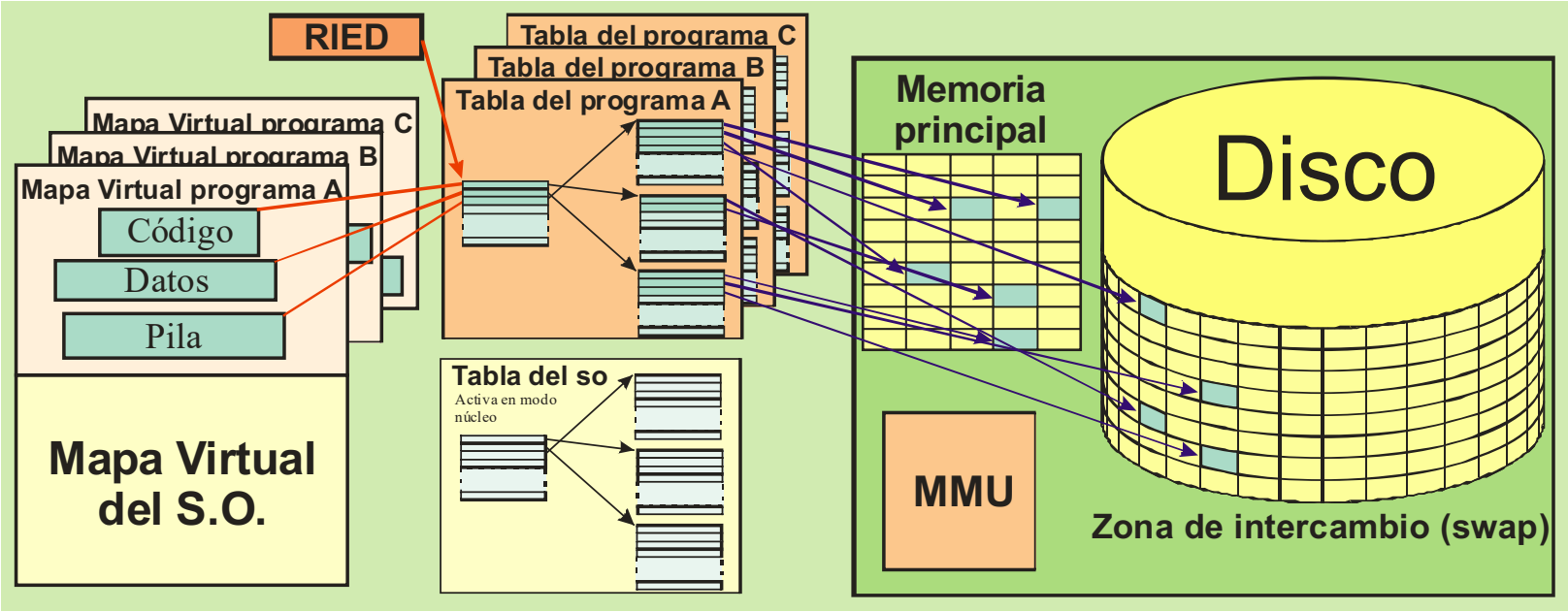
Soporte físico a la Memoria Virtual

Metainformación

- Tablas de página, almacenadas en memoria principal.

Información

- Páginas almacenadas en la zona de intercambio del disco.
- Páginas almacenadas en marcos de página de la memoria principal.
- En algunos SO las páginas que se copian a memoria se liberan del disco, para su reutilización. En otros se mantiene la copia del disco.



Gestión del espacio de swap

- **Preasignación** de swap: cada página virtual asignada tiene un espació fijo en el swap.
  - Ventaja: se detecta fácilmente si el espacio en el swap se agota.
  - Pega: se usa más memoria secundaria.

- **Sin preasignación** de swap: una página virtual asignada o está en el swap o está en un marco de memoria. En reemplazo hay que copiar la página a swap.
  - Sólo se reserva espacio para una página en el swap cuando se la expulsa de memoria principal por primera vez.
  - Si el swap se agota, hay que abortar algún proceso.

Páginas de un proceso

- **Conjunto residente de un proceso**: conjunto de páginas del proceso que en un momento dado se encuentran ubicadas en marcos de página de la memoria principal.

- **Conjunto de trabajo de un proceso**: conjunto de páginas en las que se encuentran los datos y las instrucciones que el proceso está utilizando.

Reemplazamiento de páginas

- Políticas de reemplazamiento de páginas de un proceso:
  - **Reemplazamiento local**: la nueva página se asigna a un marco libre o un marco ya asociado al proceso.
  - **Reemplazamiento global**: ante un fallo de página, la nueva página se puede asignar a un marco cualquiera (propio, ajeno o libre).

- Algoritmos de reemplazamiento:
  - FIFO.
  - Reloj (o algoritmo de segunda oportunidad)
    - Se basa en una cola FIFO.
    - Utiliza el bit de referencia.
    - Si al seleccionar una página ésta tiene activo el bit de referencia, se pone al final de la cola y se borra dicho bit.
    - Si seleccionamos una página con el bit de referencia a 0, ésta página se reemplaza por la nueva.
  - LRU.

Asignación de páginas

- Política de asignación de páginas a un proceso:
  - Asignación fija: cada proceso dispone de un número fijo de marcos de página.
  - Asignación dinámica: el número de marcos de página asignados a un proceso puede variar en función de sus necesidades.
- Política de asignación + política de reemplazamiento:
  - Asignación fija: sólo tiene sentido utilizar reemplazamiento local.
  - Asignación dinámica + reemplazamiento local: el proceso aumenta o disminuye su conjunto residente según su comportamiento.
  - Asignación dinámica + reemplazamiento global: los procesos se quitan páginas entre ellos.

Creación de regiones

- El gestor de memoria crea las regiones de memoria cuando:
  - se crea un proceso (fork),
  - se cambia el programa del proceso (exec),
  - o se solicita una nueva región.
- La creación de una región requiere:
  - Definir la ubicación de la región en el espacio virtual.
  - Asignarle, en su caso, espacio de swap como soporte físico.
    - Las regiones compartidas comparten el espacio físico
  - En su caso, dar contenido al soporte físico.
  - **Crear la tabla de páginas**.

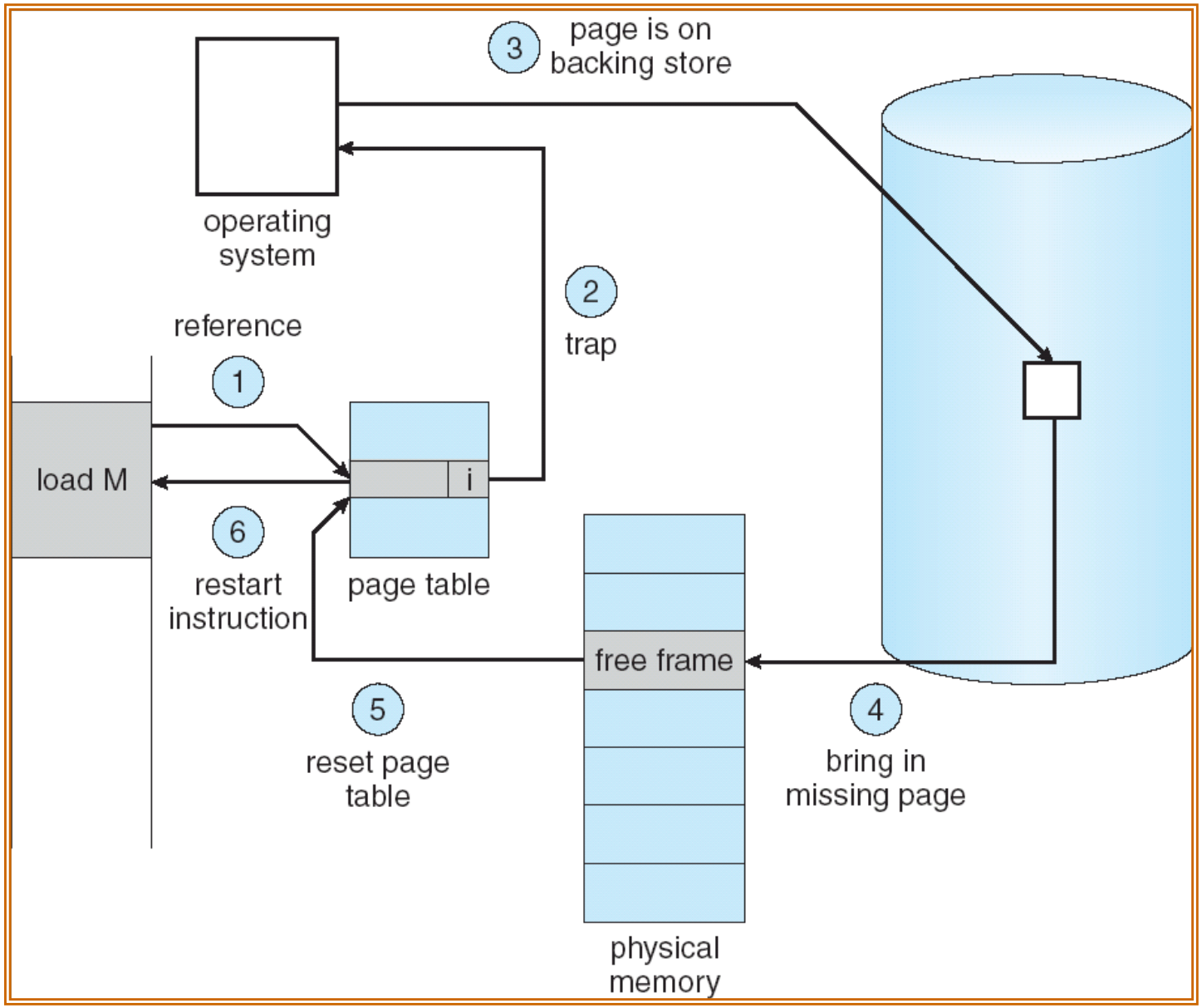
Fallo de página

- Cuando se produce un fallo de página la MMU genera una interrupción que es tratada por el gestor de memoria que:
  - Selecciona un marco de página.
  - En caso necesario libera el marco de página, devolviendo su contenido a swap.
  - Lee del disco la página requerida y la transfiere al marco seleccionado.
  - Actualiza la tabla de páginas con la nueva metainformación.
  - Bits de referenciada y modificada
    - Los actualiza la MMU (es la que trata los accesos a memoria).
    - Los utiliza el gestor de memoria para hacer el reemplazo.
- **El gestor de memoria crea y mantiene las tablas de páginas (menos los bits de referenciada y modificada).**

Fallo de página

- Cuando se produce un fallo de página, el gestor de memoria realiza las siguientes acciones:
  1. Consulta una tabla interna, normalmente dentro del PCB, para determinar si la referencia a memoria era válida o inválida. Si era inválida se termina el proceso (segmentation fault)
  2. Si el acceso era válido, busca un marco libre para alojar la página
  3. Trae la página desde disco al marco libre
  4. Marca la página como válida en la tabla de páginas
  5. Reinicia la instrucción interrumpida

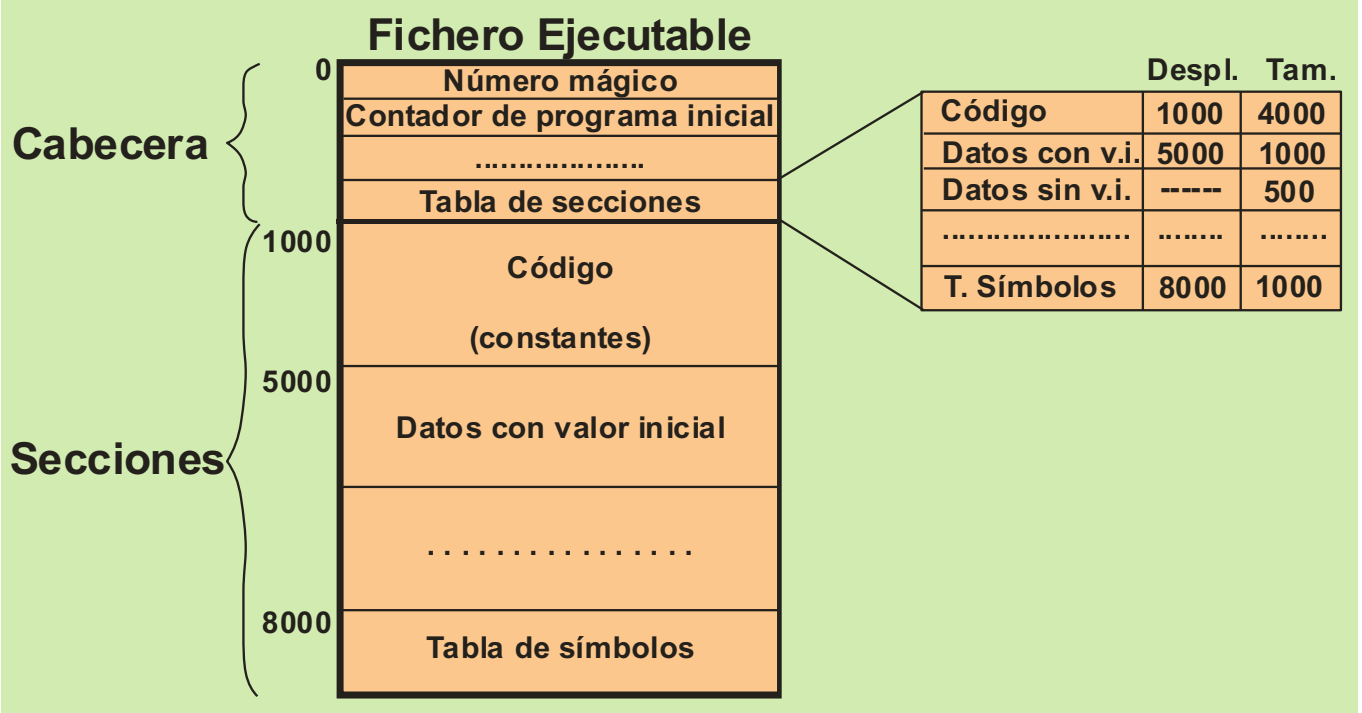
Fallo de página



Índice

Fichero ejecutable

- El fichero ejecutable contiene toda la información para crear un proceso.
- Existen distintos formatos
  - P.e. Executable and Linkable Format (ELF).
- Estructura del ejecutable: Cabecera y conjunto de secciones.
- Cabecera:
  - Número mágico.
  - Registros (PC inicial).
  - Tabla de regiones.



Secciones del ejecutable

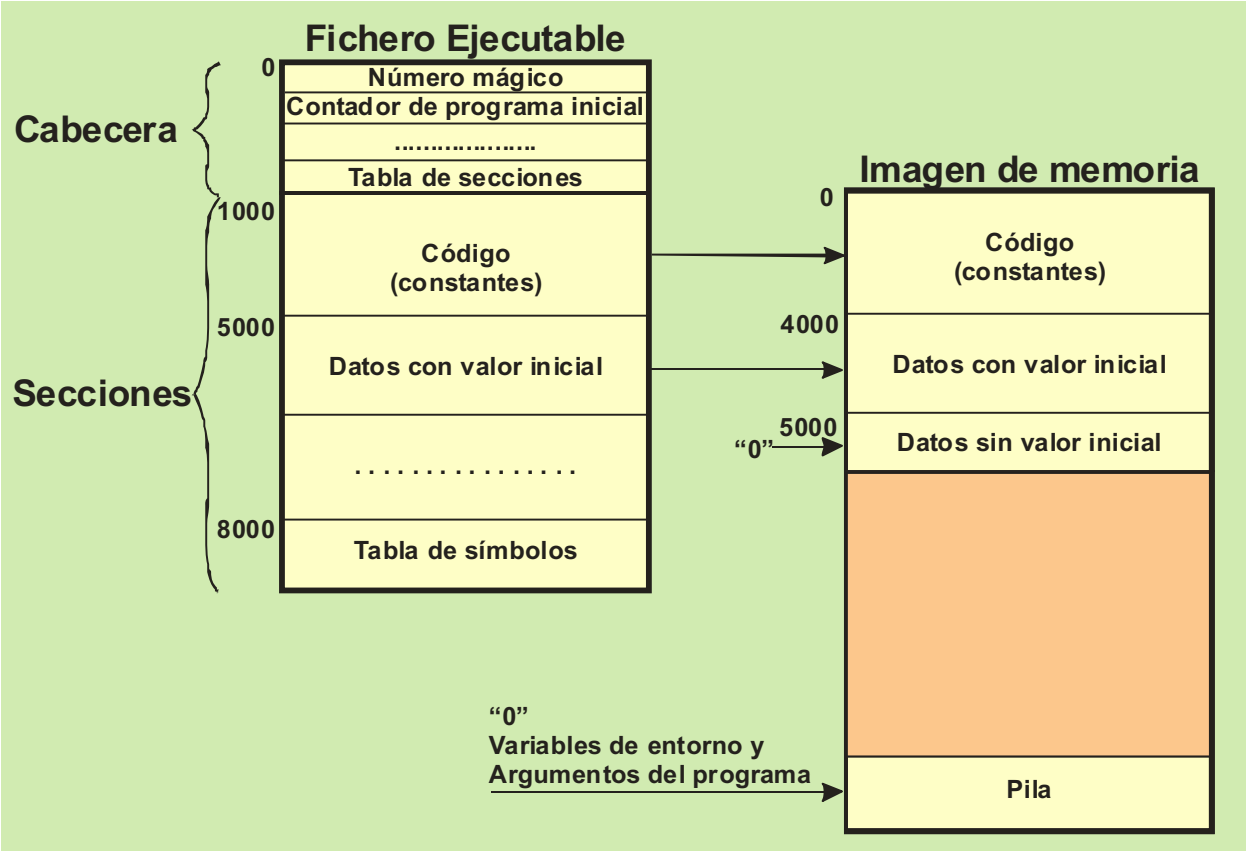
- Secciones principales
  - Sección de código (texto): Contiene código del programa. Suele incluir también las constantes del programa y cadenas de caracteres.
  - Sección de datos con valor inicial: Variables globales inicializadas.
- No existen secciones para
  - Datos estáticos sin valor inicial: Variables globales no inicializadas.
    - Aparece en tabla de secciones pero no se almacena en el ejecutable.
  - Variables locales.
  - Pila.

Variables globales vs. locales

- Variables globales
  - Estáticas.
  - Se crean al iniciarse el programa.
  - Existen durante toda la ejecución del mismo.
  - Dirección fija en memoria y en ejecutable.
- Variables locales y parámetros
  - Dinámicas. Creadas dentro del RAR (Registro de Activación de Rutina).
  - Se crean al invocar una función. Su valor inicial se establece por el propio código.
  - Se destruyen al retornar.
  - La dirección de la variable es relativa al RAR (p.ej. en pila).
    - Recursividad: varias instancias de una variable, cada una en una dirección distinta.

Crear imagen desde ejecutable

- El gestor de memoria se encarga de crear la imagen del proceso a partir de un fichero ejecutable. Esta operación la realiza `exec()`



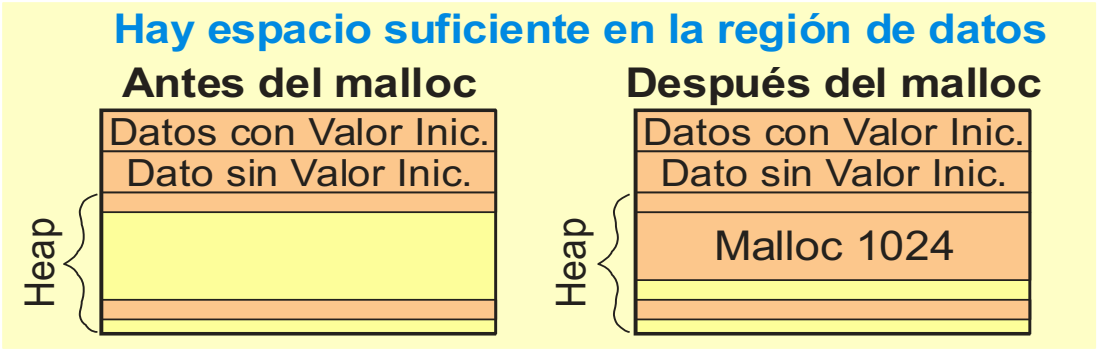
Índice

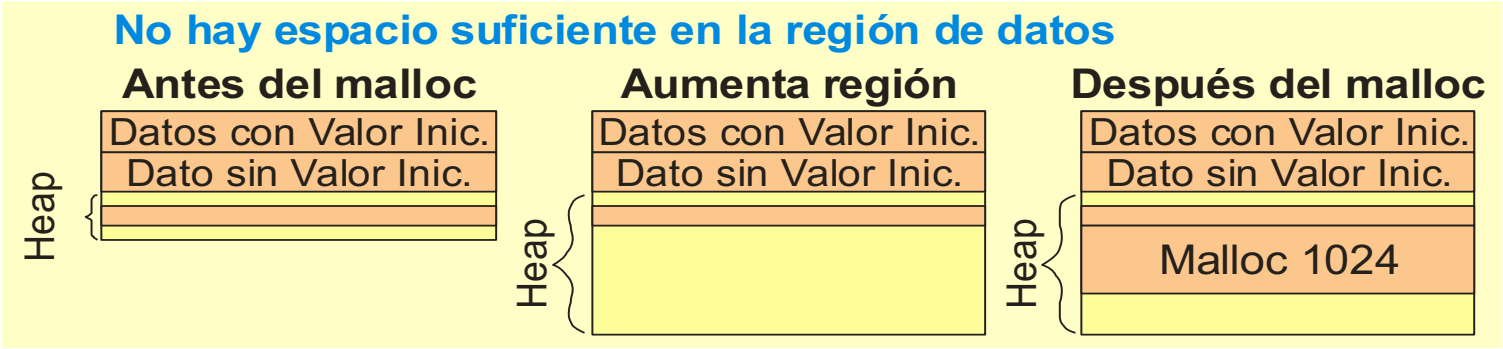
Uso de memoria dinámica

- Distintos lenguajes tienen distintas primitivas para solicitar y liberar memoria dinámica.
  - Lenguaje C
    - Solicitar memoria: `p = malloc(size); calloc(nelem, size); realloc(p, size);`
    - Liberar memoria: `free(p);`
  - C++
    - Solicitar memoria: `p = new datatype[size];`
    - Liberar memoria: `delete[ ] p;`
  - Java
    - Solicitar memoria: `p = new datatype[size];`
    - Liberar memoria: se hace automáticamente por el "recolector de basura" (garbage collector).
- La memoria asignada se maneja a través de la referencia, puntero o manejador devuelto por la función de solicitar memoria.

Malloc

- Malloc no es un servicio del SO sino una función del lenguaje.
- Si la memoria solicitada en el malloc no cabe en la región, es necesario solicitar al SO una ampliación de la región de datos (servicio Unix: `brk`).
- La memoria obtenida sobrevive al retorno de la función donde se ubicó. Hay que liberarla explícitamente (free()).





Índice

Necesidades de memoria del proceso

Los procesos necesitan memoria para almacenar el código y los datos.

```
int a;
int b = 5;
const float pi = 3.1416;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[]) {
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```

Tipos de memoria requeridos

1.- Código

2.- Datos declarados

- Estáticos
  - Constantes + cadenas
  - Con valor inicial
  - Sin valor inicial
- Dinámicos
  - Con valor inicial
  - Sin valor inicial

3.- Datos en bruto (para mem. compartida)

- Asignación de memoria

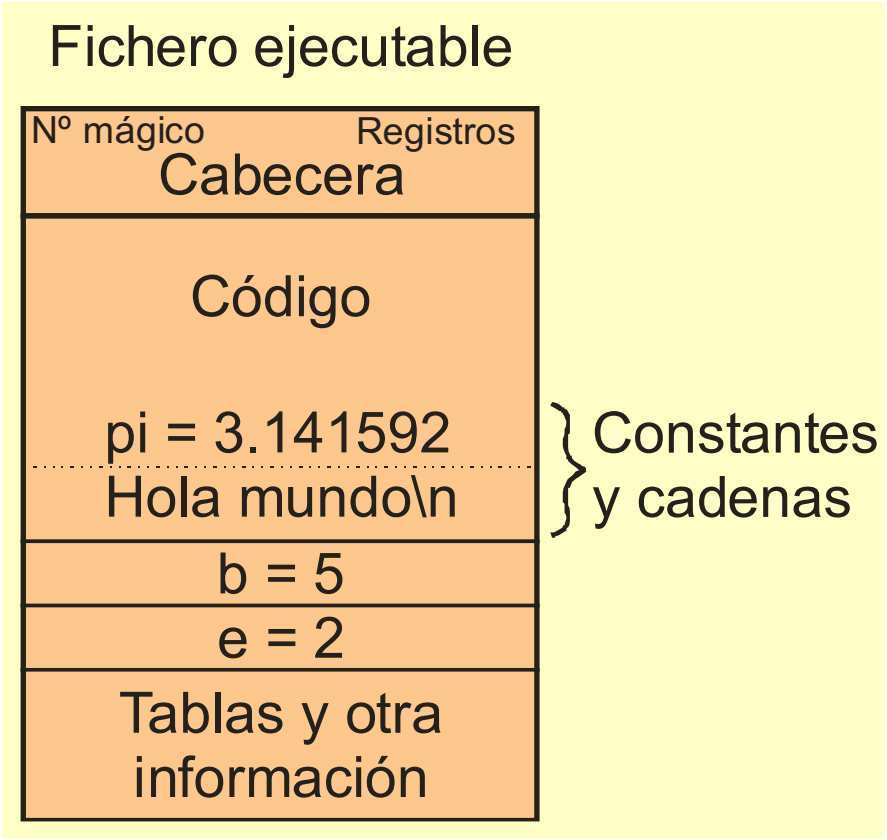
Fichero ejecutable

Las constantes y las cadenas de caracteres deben ser inmutables, por lo que se suelen asociar al código.

```
int a;
int b = 5;
const float pi = 3.141592;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[])
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```



Fichero ejecutable

Los datos estáticos y con valor inicial aparecen en la zona de datos del ejecutable.

```
int a;
int b = 5;
const float pi = 3.1416;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[])
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```

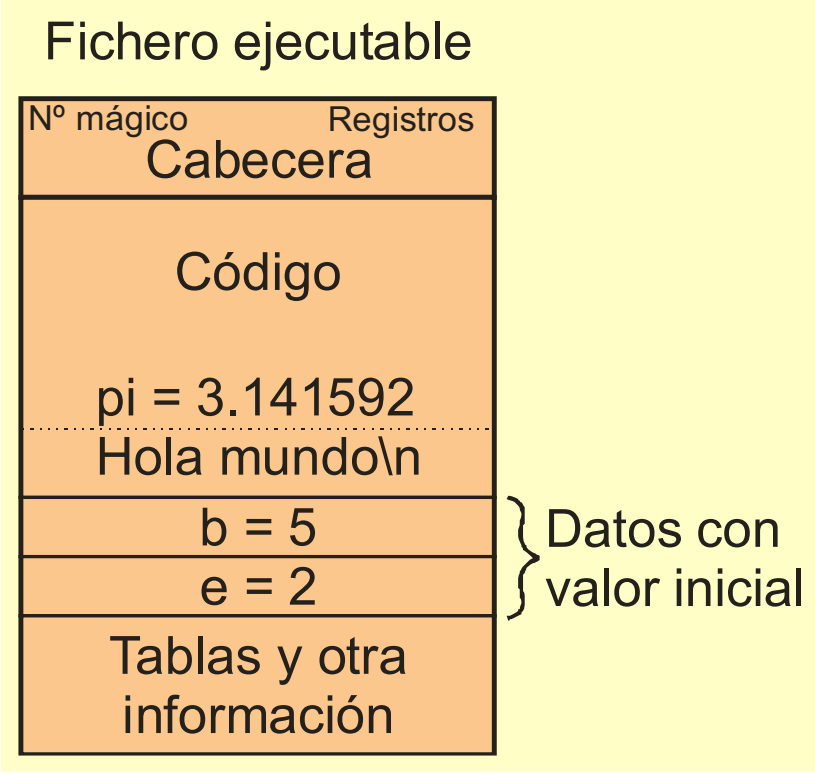


Imagen de memoria al principio del la ejecución

La imagen de memoria inicial incluye el código, los datos estáticos con y sin valor inicial, y la pila inicial.

```
int a;
int b = 5;
const float pi = 3.1416;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[]
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```

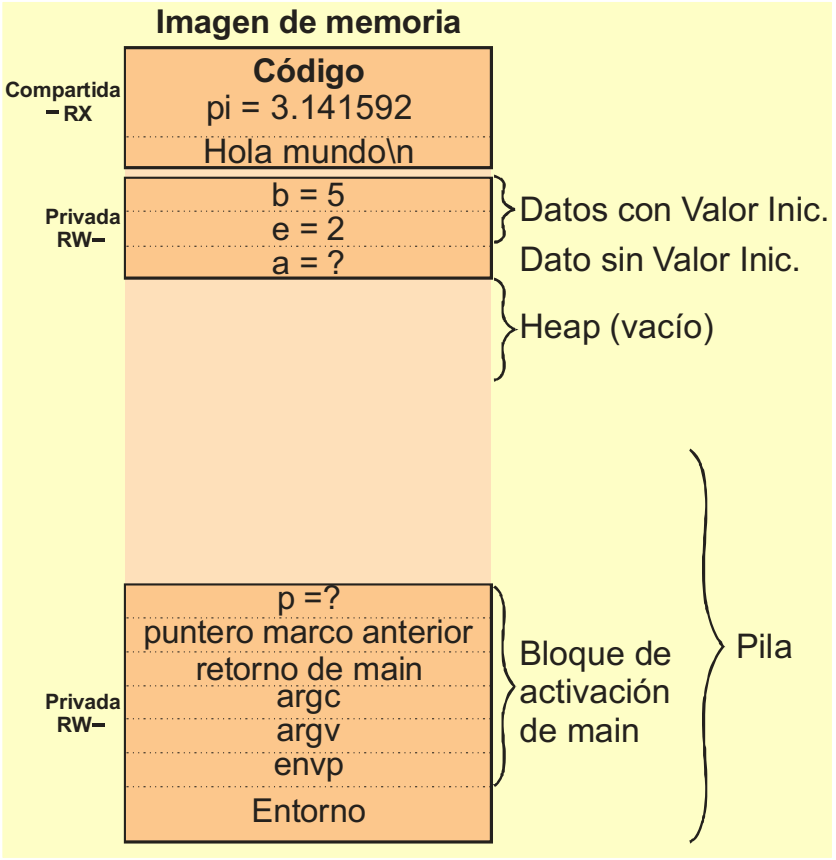


Imagen de memoria al ejecutar malloc

El malloc lo resuelve la biblioteca de C:

```
int a;
int b = 5;
const float pi = 3.1416;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[])
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```

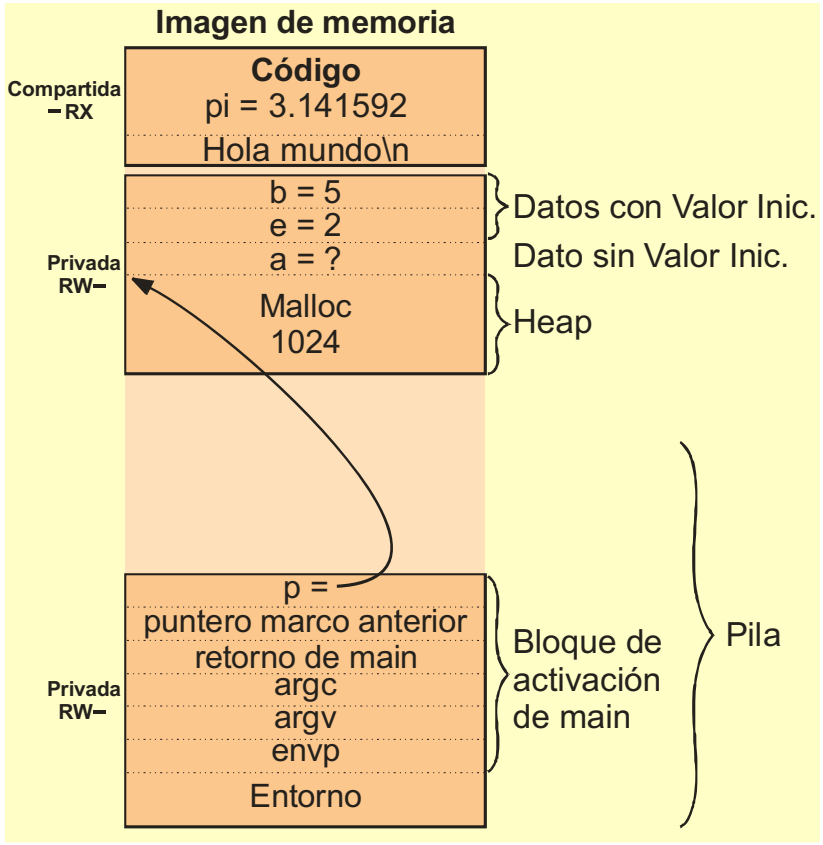


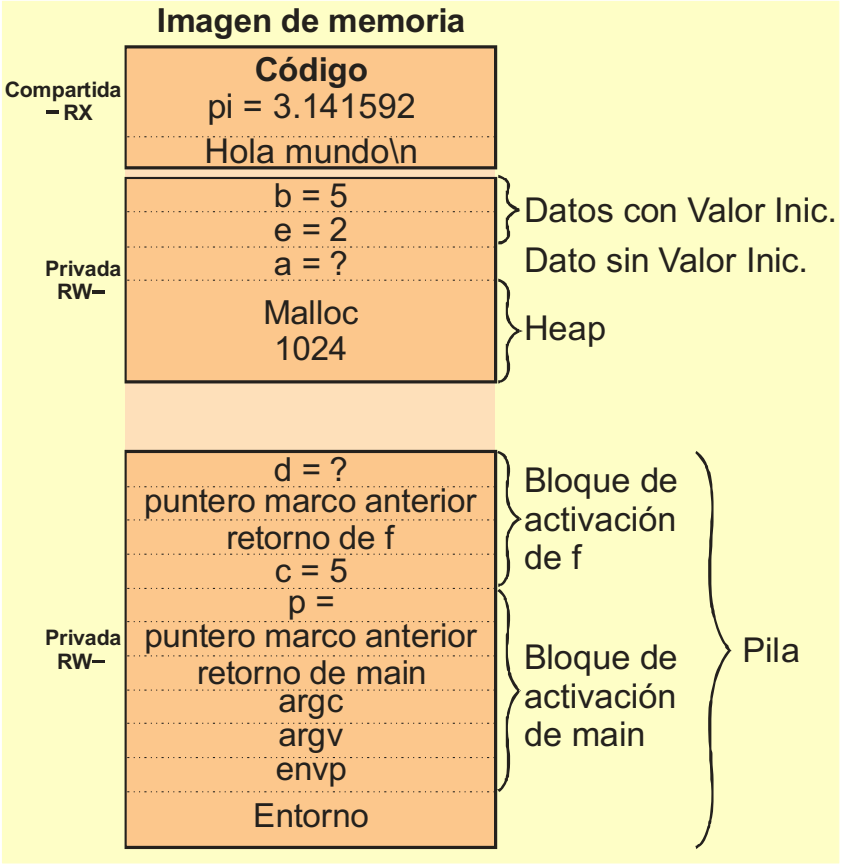
Imagen de memoria al llamar a una función

Al realizarse la llamada el programa crea el bloque de activación con los parámetros de invocación y las variables locales.

```
int a;
int b = 5;
const float pi = 3.1416;

void f(int c) {
        int d;
        static int e = 2;
        d = 3;
        b = d + 5;
        .......
        return;
}

int main (int argc, char *argv[])
        char *p;
        p = malloc (1024);
        f(b);
        .......
        free (p);
        printf ("Hola mundo\n");
        return 0;
}
```



Parámetros

- Los parámetros de un procedimiento constituyen los primeros elementos de su bloque de activación.
- El programa llamante, al ir construyendo dicho bloque de activación, mete en la pila los valores asignados a los parámetros (en orden inverso: primero el último).
  - Una vez pasado el control al procedimiento, éste es el único que accede a dichas posiciones de la pila.
- Observamos que este mecanismo sirve para parámetros de entrada con paso por valor. Cuando se pasa la dirección de un dato, en vez del valor de un dato, se está haciendo un paso por referencia.
- Para los parámetros de salida la técnica a emplear es pasar al procedimiento, un espacio o variable suficiente para el tipo de datos de retorno en la pila. De esta forma, el procedimiento llamado puede acceder al dato del llamante, y modificarlo.
  - El valor de la función es un parámetro de salida que se pasa por valor.

Heap vs. Bloque de activación


```
struct s {int d; char a[2000];};
char * fun(int c) {
  struct s *p;
  p = malloc(sizeof (struct s));
  .......
  return p->a;
```

La función puede devolver p->a (la zona de memoria sobrevive el retorno de f)

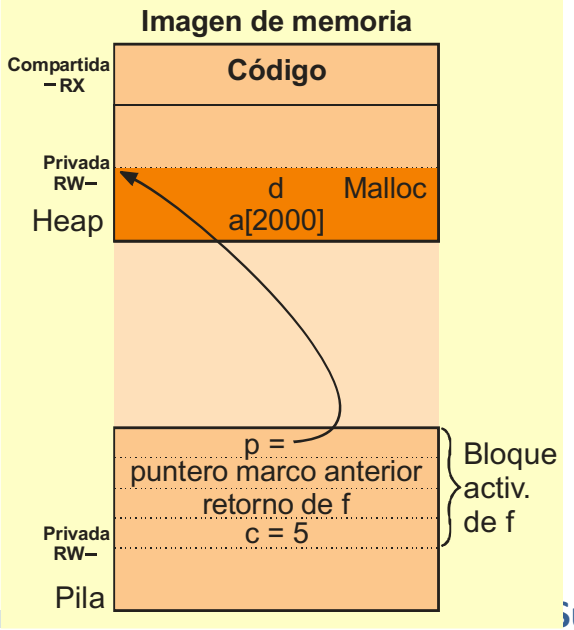
Idóneo para funciones que crean estructuras dinámicas (listas, árboles)

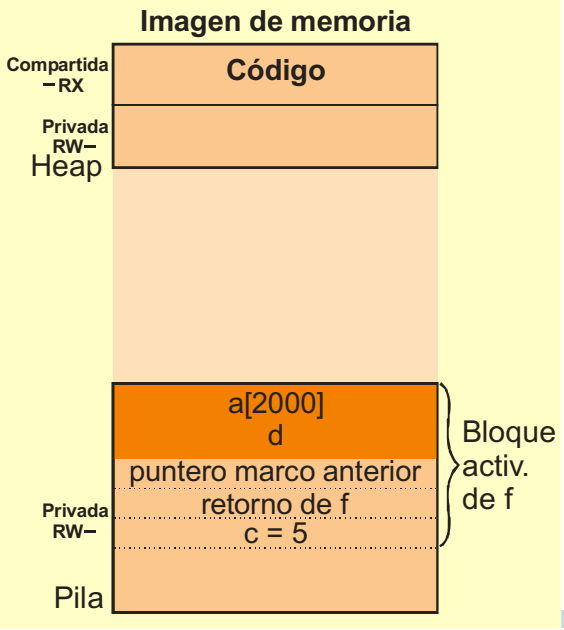
```
struct s {int d; char a[2000];};
char * fun(int c) {
   struct s p;
   .......

   // no poner return p.a;
```

Mucho menor coste computacional que el malloc o el new.

La zona de memoria se recupera en el retorno de f (no devolver dirección p.a).





- Un proceso puede tener varios procesos ligeros o hilos.
- Las regiones de memoria son del proceso no del hilo.
  - Todos los hilos tienen los mismos derechos sobre todas las regiones de memoria del proceso.
  - Por ejemplo, un proceso ligero puede leer y escribir en la pila de otro proceso ligero. Por ejemplo, un hilo puede crear otro hilo y pasarle como parámetro una referencia a una variable local suya.
- Coutilización de memoria
  - Mediante variables globales.
  - Mediante variables/estructuras dinámicas pasadas como parámetros a los hilos.
  - Es necesario utilizar mecanismos de sincronización para evitar los problemas de acceso concurrente a las variables coutilizadas.

Índice

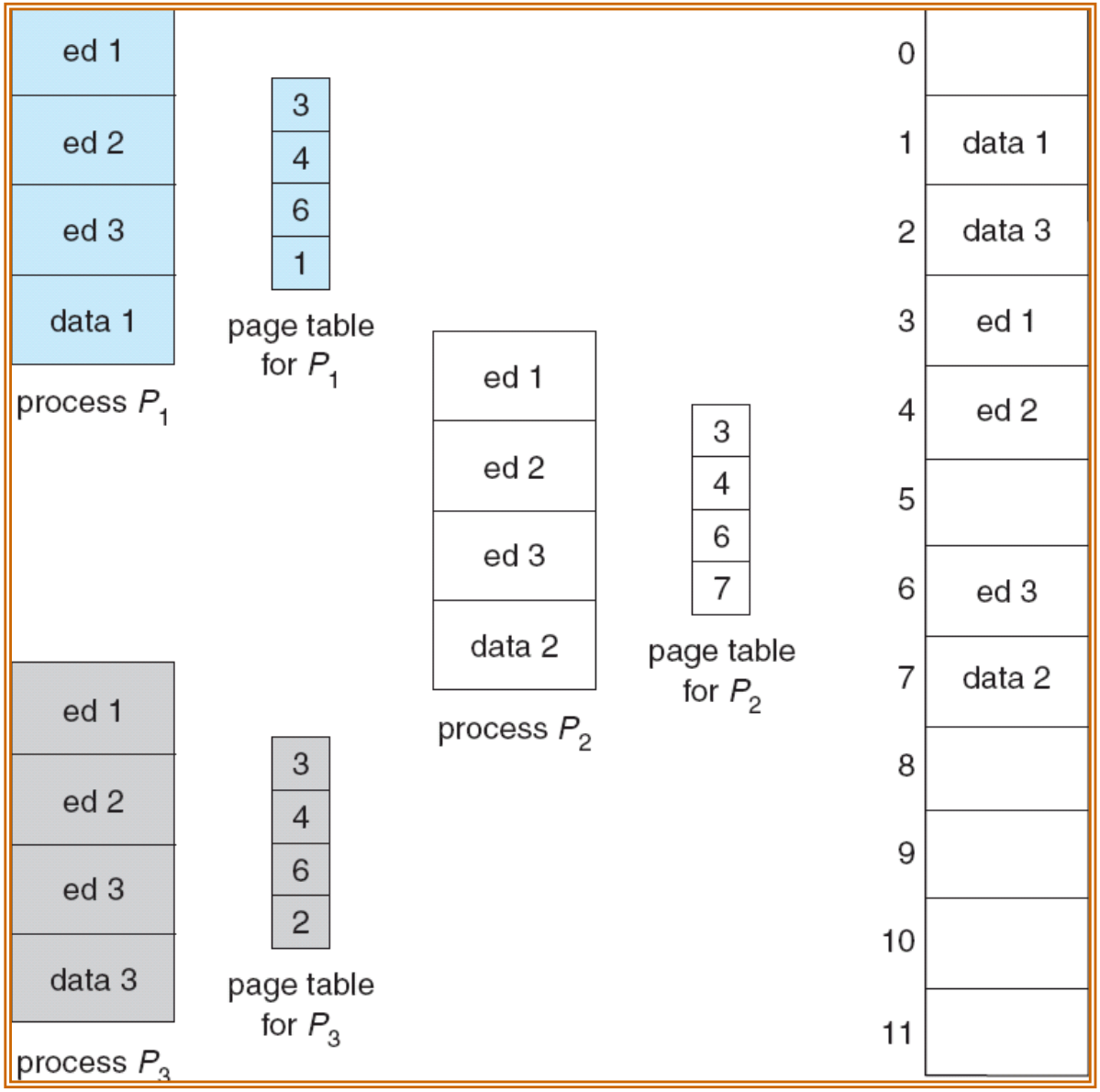
Memoria compartida

- Se rompe el concepto de independencia de los procesos.
- Se permite que unas regiones -del mismo tamaño- de dos o más procesos utilicen el mismo soporte físico y, por tanto, compartan la misma información.
- Se hace bajo control del S.O. y se exige que los procesos lo soliciten expresamente. (las regiones de código se suelen compartir de forma implícita, es decir, sin solicitárselo al SO).
- Beneficios:
  - Procesos que ejecutan el mismo programa comparten el código.
  - Mecanismo muy rápido de comunicación entre procesos.
- Utilización
  - Los procesos han de ponerse de acuerdo para escribir y leer de esa memoria común de forma ordenada.

Memoria compartida

- El mecanismo de paginación permite que varios procesos traduzcan sus direcciones lógicas al mismo marco de página
  - Las regiones pueden estar en direcciones virtuales distintas en cada proceso.
  - Se comparte la tabla de páginas (o por lo menos apuntan a las mismas páginas de swap y marcos de memoria principal).
- Esto es útil para crear regiones de memoria compartida entre procesos
- También es útil para compartir código entre procesos (el código debe ser *reentrante*)

Memoria compartida

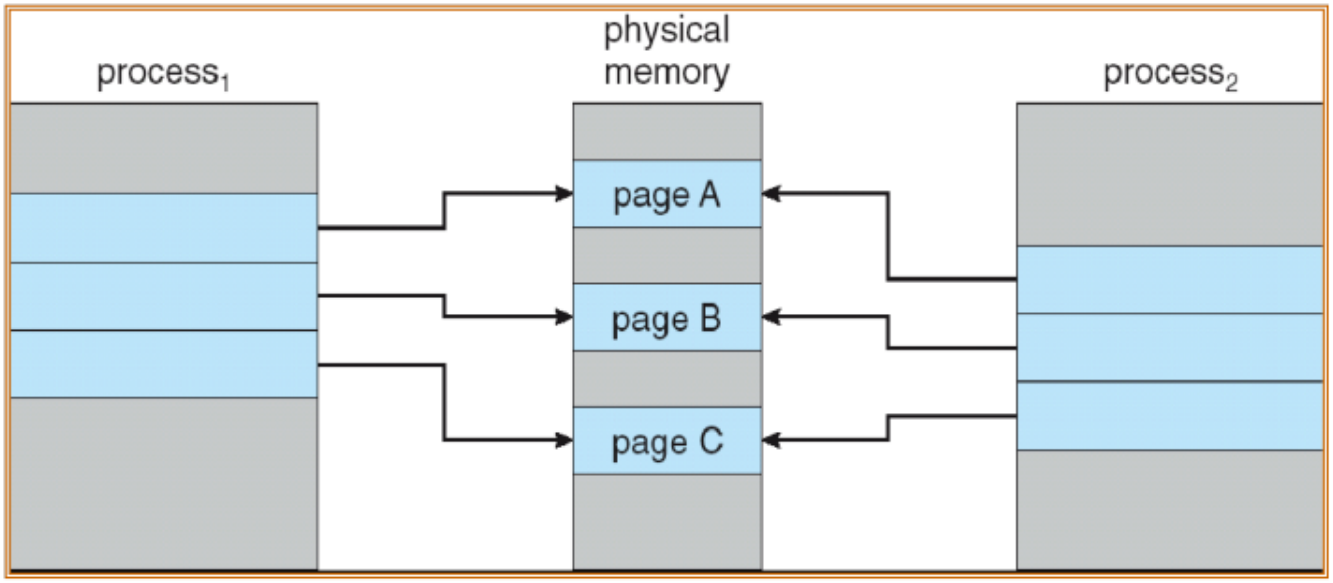


Memoria compartida – fork() – exec()

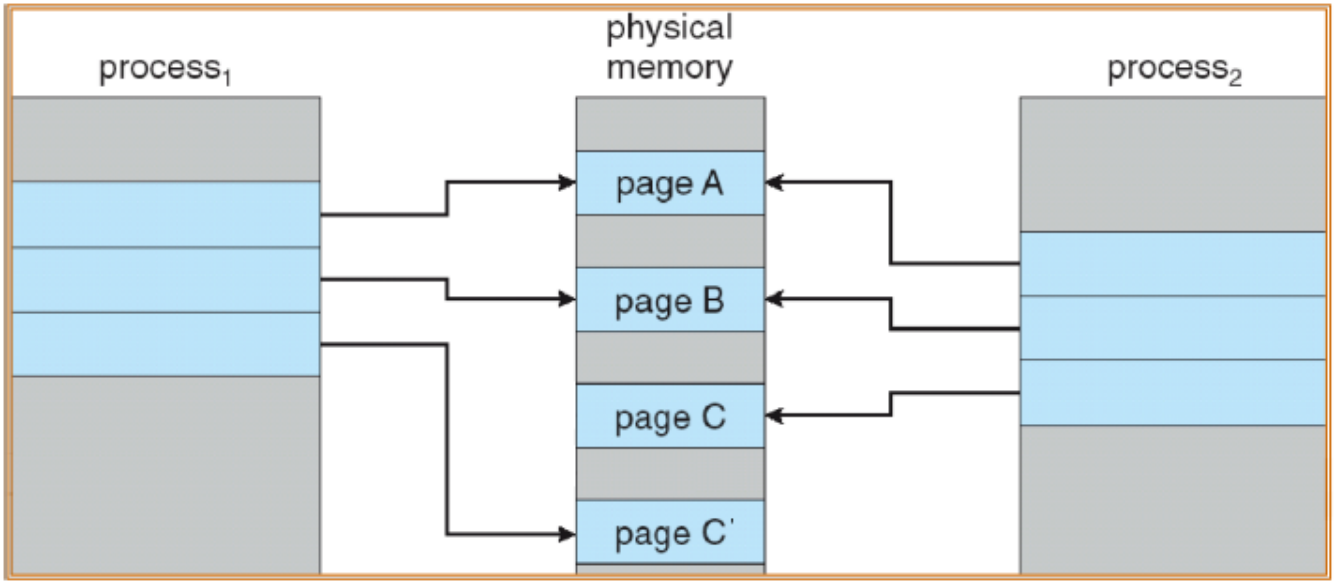
- Al crear un proceso nuevo con fork(), se duplican las páginas de memoria del proceso padre.
  - ¿Si justo después se hace un exec()?
  - Se sustituye la imagen de memoria del proceso por la del nuevo proceso lanzado con exec()
  - Todo el proceso de copia de páginas del proceso padre habrá sido en balde
- Para evitar esta copia innecesaria se utiliza CopyOnWrite:
  - Cuando se crea un proceso con fork() el padre y el hijo comparten sus páginas
  - Si cualquiera de ellos escribe en una página, se crea la copia de esa página

Memoria compartida – fork() – exec()

Después del fork() :



El proceso 1 escribe en la página C :



Índice

Servicios de gestión de memoria

- Aumento del tamaño de una región
  - `size = brk (addr);`
    - Coloca , si es posible, el fin del heap en la dirección addr.
    - Supone que aumenta o disminuye el heap del segmento de datos.
    - Lo utiliza la biblioteca del lenguaje que maneja memoria dinámica.
  - La región de pila crece automáticamente según la necesidad.
- Creación y destrucción de regiones
  - En UNIX se realiza mediante las primitivas para compartir memoria y para proyectar ficheros en memoria.
  - Uso de memoria compartida (`shm_open, shmget, shmat ..`)
  - Proyección de ficheros (`mmap`)
  - Montaje explícito de biblioteca dinámica (`dlopen`)

Servicios de memoria compartida

- `int shm_open(char *name, int oflag, mode_t mode)`: Crea un objeto de memoria a compartir entre procesos
- `int close(int fd)`: Cierre. El objeto persiste hasta que es cerrado por el último proceso.
- `int shm_unlink(const char *name)`: Borra una zona de memoria compartida.
- `int shmget(key_t key, size_t size, int shmflg)`: Crea un objeto de memoria a compartir entre procesos
- `void *shmat(int shmid, const void *shmaddr, int shmflg)`: asociar segmento de memoria compartida al espacio de direcciones del proceso
- `int shmdt(const void *shmaddr)`: desasociar segmento

```
#define SHMSZ     27
int main(){
    ...
     int shmid;
     char *shm, *s;

     shmid = shmget(5678, SHMSZ, IPC_CREAT | 0666);
     shm = shmat(shmid, NULL, 0);
     s = shm;
     for (c = 'a'; c <= 'z'; c++)
         *s++ = c;
     *s = NULL;
     /* Esperamos hasta que el otro proceso cambie el primer
caracter de nuestra memoria a '*'  indicando que se ha leido */
     while (*shm != '*')
         sleep(1);
}
```

Cliente

```
#define SHMSZ     27
int main(){
    ...
     int shmid;
     char *shm, *s;

     shmid = shmget(5678, SHMSZ, 0666);
     shm = shmat(shmid, NULL, 0);
     s = shm;
     for (s = shm; *s != NULL; s++)
         putchar(*s);
     putchar('\n');
     /* Cambiamos el primer caracter a '*' Para indicar que lo
hemos leido */
     *shm = '*';
}
```

Proyección de ficheros

- Establecer una proyección entre el espacio de direcciones de un proceso y un descriptor de fichero u objeto de memoria compartida.

```
void * mmap(void *addr, size_t len, int prot,
            int flags, int fildes, off_t off);
```

  - *addr* si se quiere especificar donde empieza
  - *len* especifica el número de bytes a proyectar.
  - *prot* el tipo de acceso (lectura, escritura o ejecución).
    - PROT_READ, PROT_WRITE, PROT_EXEC o PROT_NONE.
  - *flags* especifica información sobre el manejo de los datos proyectados (compartidos, privado, etc.).
    - MAP_SHARED, MAP_PRIVATE, ...
  - *fildes* representa el descriptor de fichero del fichero o descriptor del objeto de memoria a proyectar.
  - *off* desplazamiento dentro del fichero a partir del cual se realiza la proyección.
- Desproyectar parte del espacio de direcciones de un proceso comenzando en la dirección *addr*. El tamaño de la región a desproyectar es *len*.
  - `void munmap(void *addr, size_t len);`

Acceso a fichero con proyección

- Ejemplo: Cuántas veces aparece un determinado carácter en un fichero.

```
int main(int  argc, char **argv) {
    ....................................
    caracter = *argv[1];          /* Carácter a buscar */
    fd=open(argv[2], O_RDONLY));/* Abre fichero */
    fstat(fd, &bstat);            /* Obtiene atributs del fichero */
    /* Se proyecta el fichero */
    org=mmap(NULL, bstat.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
    close(fd);                 /* Se cierra el descriptor del fichero */

    p=org;
    contador = 0;
    for (i=0; i<bstat.st_size; i++, p++)       /* Bucle de acceso */
        if (*p==caracter) contador++;

    munmap(org, bstat.st_size);        /* Se elimina la proyección */
    printf("%d\n", contador);
    return 0;
}
```

Copia de fichero con proyección

```
int main(int  argc, char **argv) {
    .....................................
    fdo=open(argv[1], O_RDONLY); /* Abre el fichero origen*/
    fdd=open(argv[2], O_CREAT|O_TRUNC|O_RDWR, 0640); /* Crea destino */
    fstat(fdo, &bstat);          /* Obtienes atributos del fichero origen */
    ftruncate(fdd, bstat.st_size); /* Establece long destino igual a origen.*/

    /* Se proyectan en memoria ambos ficheros */
    org=mmap(NULL, bstat.st_size, PROT_READ, MAP_PRIVATE, fdo, 0);
    dst=mmap(NULL, bstat.st_size, PROT_WRITE, MAP_SHARED, fdd, 0);

   close(fdo); close(fdd);    /* Se cierran los descriptores de fichero */

    p=org; q=dst;
    for (i=0; i<bstat.st_size; i++)  /* Bucle de copia */
        *q++= *p++;

    /* Se eliminan las proyecciones */
    munmap(org, bstat.st_size); munmap(dst, bstat.st_size);
    return 0;
}
```

Índice

Protección

- Monoprogramación: Hay que proteger al S.O.
- Multiprogramación: Proteger al S.O. y a los procesos entre sí.
- **Es necesario validar todas las direcciones que genere el programa.**
  - La detección o supervisión debe realizarla el hardware del procesador.
    - El HW comprueba que la dirección y el tipo de acceso son correctos.
  - El tratamiento de la infracción lo hace el SO (p.e. aumenta la pila o envía una señal al proceso que normalmente lo mata).
- **Rellenar con 0:**
  - Un posible problema de seguridad aparece cuando se le suministra a un proceso un soporte físico de memoria (ya sea una página en el espacio swap o un marco de página) y éste no tiene valor inicial.
  - Si no se rellena a 0 por el SO el proceso podrá leer el contenido que dejó en ese soporte físico el proceso que lo utilizó anteriormente, pudiendo recuperar información valiosa del mismo.

Errores de acceso a memoria

- Errores y avisos detectados por el HW y tratados por el SO:
  - Error: Acceso a una dirección fuera de las regiones asignadas.
    - Acción del SO: Enviar una señal al proceso, que suele matarlo (Segmentation fault)
  - Error: Acceso a una dirección correcta pero con un tipo de acceso no permitido.
    - Acción del SO: Enviar una señal al proceso, que suele matarlo.
  - Aviso: Desbordamiento de pila.
    - Acción del SO: Incrementar la región de pila. Si no es posible el incremento, enviar una señal al proceso para matarlo.
  - Aviso: Fallo de página.
    - Acción del SO: Realizar la migración de la página fallida y actualizar la tabla de páginas afectada.

Errores en el manejo de memoria dinámica

- Usarla sin necesidad. Es mucho más costosa que variables locales
- Pérdidas o goteras de memoria.
  - Se produce cuando el programa crea datos dinámicamente y no libera la memoria ocupada cuando esos datos ya no interesan.
  - Hay que prestar especial atención a la liberación de datos obsoletos.
- Intento de acceso a un dato obsoleto eliminado.
  - Se produce cuando accedemos a los datos mediante una referencia (p.e. un puntero). Al eliminar el dato no se elimina la referencia, por lo que se puede intentar su utilización.
  - Si el dato obsoleto pertenece a una región o trozo de sección eliminada, el HW detectará el fallo y el SO enviará una señal al proceso, que suele matarlo.
  - En caso contrario se accede a basura, generándose un resultado inesperado, lo que suele producir un error difícil de diagnosticar.
- Intento de acceso a un dato no creado.
  - Se produce cuando se utiliza la referencia (puntero) antes de proyectar el dato.

Errores en el manejo de memoria dinámica

- Desbordamiento de un dato múltiple (índice fuera de rango).
  - Se produce cuando se accede a un dato múltiple a través de una referencia (p.e. un puntero) a la que se asigna un valor mayor que el tamaño real del dato.
  - Si el acceso se produce dentro de la región se obtendrá un resultado inesperado, difícil de diagnosticar. En caso contrario el HW detectará un error de acceso.
- Uso de definiciones de datos no compatibles en memoria compartida.
  - Los distintos programas que ejecutan los procesos que comparten memoria han de tener definiciones compatibles. Al actualizar estas definiciones es fácil olvidarse de actualizar alguno de los programas.