Tema 8
Gestión de la memoria en tiempo de ejecución.

✯ Organización de la memoria en tiempo de ejecución.

Cuando un programa se ejecuta sobre un sistema operativo existe un proceso previo llamado cargador que suministra al programa un bloque contiguo de memoria sobre el cual ha de ejecutarse. El programa resultante de la compilación debe organizarse de forma que haga uso de este bloque. Para ello el compilador incorpora al programa objeto el código necesario.

Las técnicas de gestión de la memoria durante la ejecución del programa difieren de unos lenguajes a otros, e incluso de unos compiladores a otros. En este capítulo se estudia la gestión de la memoria que se utiliza en lenguajes procedurales como FORTRAN, PASCAL, C, MODULA-2, etc. La gestión de la memoria en otro tipo de lenguajes (funcionales, lógicos, ...) es en general diferente de la organización que aquí se plantea.

Para lenguajes imperativos, los compiladores generan programas que tendrán en tiempo de ejecución una organización de la memoria similar (a grandes rasgos) a la que aparece en la figura 1.

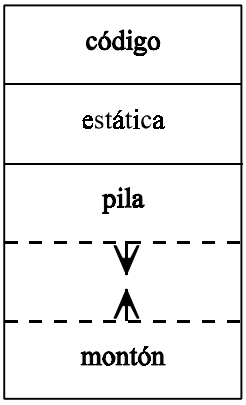

Fig. 1. Organización de la memoria en la ejecución

En este esquema se distinguen las secciones de:

- El Código
- La Memoria Estática.
- La Pila.
- El Montón.

El Código

Es la zona donde se almacenan las instrucciones del programa ejecutable en código máquina, y también el código correspondiente a los procedimientos y funciones que utiliza. Su tamaño puede fijarse en tiempo de compilación.

Algunos compiladores fragmentan el código del programa objeto usando "*overlays*". Estos "*overlays*" son secciones de código objeto que se almacenan en ficheros independientes y que se cargan en la memoria central (RAM) dinámicamente, es decir, durante la ejecución del programa. Los *overlays* de un programa se agrupan en zonas y módulos, cada uno de los cuales contiene un conjunto de funciones o procedimientos.

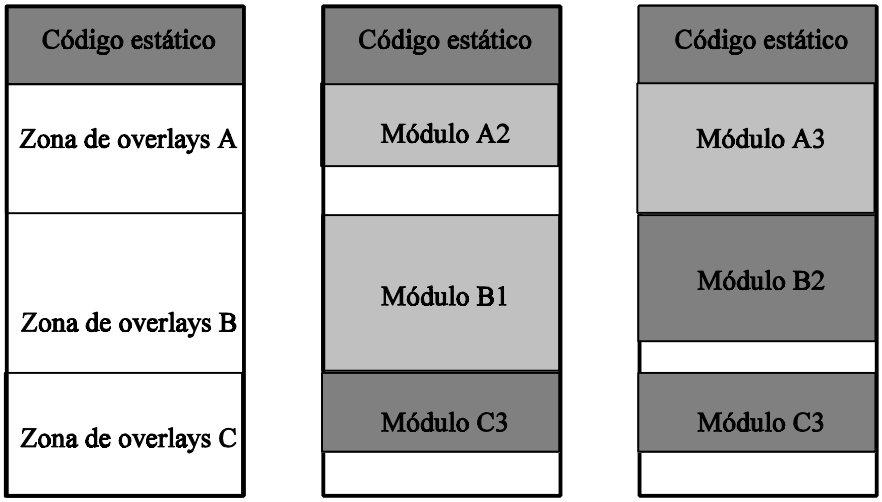

fig.2 Ejemplos de configuraciones de la zona de código utilizando "*overlays*"

Durante el tiempo de ejecución sólo uno de los módulos de cada uno de los *overlays* puede estar almacenado en memoria. El compilador reserva en la sección de código una zona contigua de memoria para cada *overlay*. El tamaño de esta zona debe ser igual al del mayor módulo que se cargue sobre ella. Es función del programador determinar cuantas zonas de *overlay* se definen, qué funciones y procedimientos se encapsulan en cada módulo, y cómo se organizan estos módulos para ocupar cada uno de los *overlays*. Una restricción a tener en cuenta es que las funciones de un módulo no deben hacer referencia a funciones de otro módulo del mismo *overlay*, ya que nunca estarán simultáneamente en memoria.

El tiempo de ejecución de un programa con *overlays* es mayor, puesto que durante la ejecución del programa es necesario cargar cada módulo cuando se realiza una llamada a alguna de las funciones que incluye. También es tarea del programador diseñar la estructura de *overlays* de manera que se minimice el número de estas operaciones. La técnica de *overlays* se utiliza cuando el programa a compilar es muy grande en relación con la disponibilidad de memoria del sistema, o bien si se desea obtener programas de menor tamaño.

La Memoria Estática.

La forma más fácil de almacenar el contenido de una variable en memoria en tiempo de ejecución es en memoria estática o permanente a lo largo de toda la ejecución del programa. No todos los objetos (variables) pueden ser almacenados estáticamente. Para que un objeto pueda ser almacenado en memoria estática su tamaño (número de bytes necesarios para su almacenamiento) ha de ser conocido en tiempo de compilación. Como consecuencia de esta condición no podrán almacenarse en memoria estática:

- Los objetos correspondientes a procedimientos o funciones recursivas, ya que en tiempo de compilación no se sabe el número de variables que serán necesarias.

- Las estructuras dinámicas de datos tales como listas, árboles, etc. ya que el número de elementos que la forman no es conocido hasta que el programa se ejecuta.

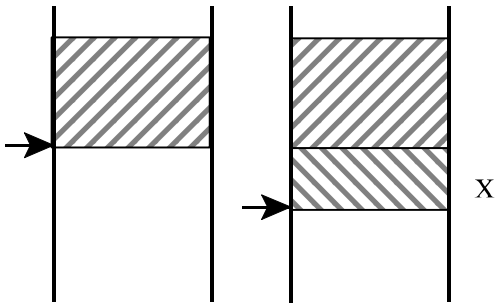

fig 3. Alojamiento en memoria de un objeto X

Las técnicas de asignación de memoria estática son sencillas. A partir de una posición señalada por un puntero de referencia se aloja el objeto X, y se avanza el puntero tantos bytes como sean necesarios para almacenar el objeto X. La asignación de memoria puede hacerse en tiempo de compilación y los objetos están vigentes desde que comienza la ejecución del programa hasta que termina.

En los lenguajes que permiten la existencia de subprogramas, y siempre que todos los objetos de estos subprogramas puedan almacenarse estáticamente (por ejemplo en FORTRAN-IV) se aloja en la memoria estática un registro de activación correspondiente a cada uno de los subprogramas. Estos registros de activación contendrán las variables locales, parámetros formales y valor devuelto por la función, tal como indica la fig. 4.b.

La Memoria Estática.

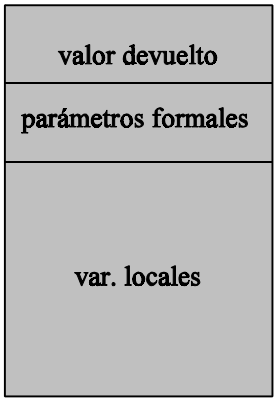

fig 4a. Registro de activación

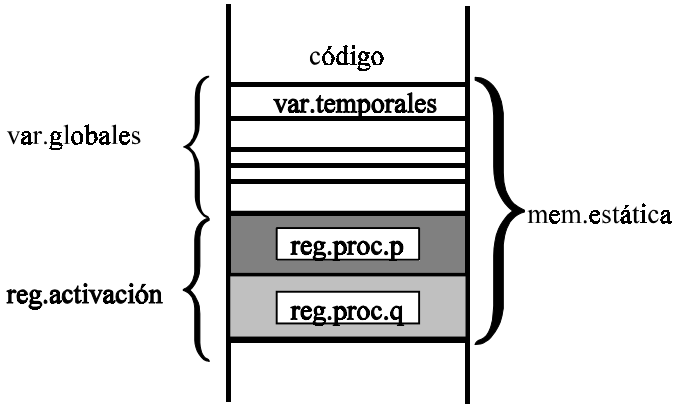

fig 4b. Estructura de la memoria estática en FORTRAN-IV

Dentro de cada registro de activación las variables locales se organizan secuencialmente. Existe un solo registro de activación para cada procedimiento y por tanto no están permitidas las llamadas recursivas. El proceso que se sigue cuando un procedimiento p llama a otro q es el siguiente:

- p evalúa los parámetros de llamada, en caso de que se trate de expresiones complejas, usando para ello una zona de memoria temporal para el almacenamiento intermedio. Por ejemplo, si la llamada a q es q((3*5)+(2*2),7) las operaciones previas a la llamada propiamente dicha en código máquina han de realizarse sobre alguna zona de memoria temporal. (En algún momento debe haber una zona de memoria que contenga el valor intermedio 15, y el valor intermedio 4 para sumarlos a continuación). En caso de utilización de memoria estática ésta zona de temporales puede ser común a todo el programa, ya que su tamaño puede deducirse en tiempo de compilación.
- q inicializa sus variables y comienza su ejecución.

Dado que las variables están permanentemente en memoria es fácil implementar la propiedad de que conserven o no su contenido para cada nueva llamada.

La Pila

La aparición de lenguajes con estructura de bloque trajo consigo la necesidad de técnicas de alojamiento en memoria más flexibles, que pudieran adaptarse a las demandas de memoria durante la ejecución del programa. En estos lenguajes, cada vez que comienza la ejecución de un procedimiento se crea un registro de activación para contener los objetos necesarios para su ejecución, eliminandolo una vez terminada ésta.

Dado que los bloques o procedimientos están organizados jerárquicamente, los distintos registros de activación asociados a cada bloque deberán colocarse en una pila en la que entrarán cuando comience la ejecución del bloque y saldrán al terminar el mismo. (Fig. 5b) La estructura de los registros de activación varía de unos lenguajes a otros, e incluso de unos compiladores a otros. Este es uno de los problemas por los que a veces resulta difícil enlazar los códigos generados por dos compiladores diferentes. En general, los registros de activación de los procedimientos suelen tener algunos de los campos que pueden verse en la fig. 5a.

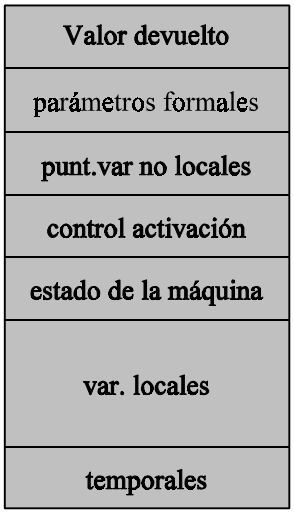

fig.5a. Registro de activación

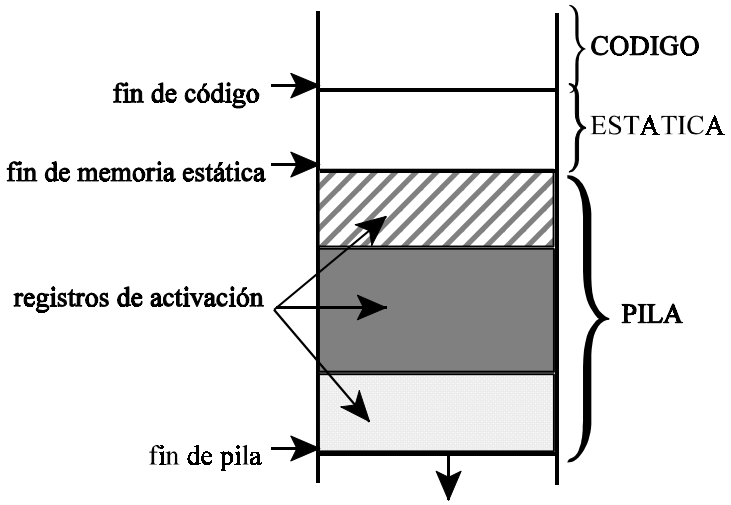

fig 5b. Estructura de la pila

El puntero de control de activación guarda el valor que tenía el puntero de la cima de la pila antes de que entrase en ella el nuevo registro, de esta forma una vez que se desee desalojarlo puede restituirse el puntero de la pila a su posición original. Es decir, es el puntero que se usa para la implementación de la estructura de datos “Pila” del compilador.

En la zona correspondiente al *estado de la máquina* se almacena el contenido que hubiera en los registros de la máquina antes de comenzar la ejecución del procedimiento. Estos valores deberán ser repuestos al finalizar la ejecución del procedimiento. El código encargado de realizar la copia del estado de la máquina es común para todos los procedimientos.

El puntero a las variables no locales permite el acceso a las variables declaradas en otros procedimientos. Normalmente no es necesario usar este campo puesto que puede

conseguirse lo mismo con el puntero de control de activación, sólo tiene especial sentido cuando se utilizan procedimientos recursivos.

La Pila

Al igual que en el alojamiento estático los registros de activación contendrán el espacio correspondiente a los parámetros formales (variables que aparecen en la cabecera) y las variables locales, (las que se definen dentro del bloque o procedimiento) así como una zona para almacenar el valor devuelto por la función y una zona de valores temporales para el cálculo de expresiones.

Para dos módulos o procedimientos diferentes, los registros de activación tendrán tamaños diferentes. Este tamaño por lo general es conocido en tiempo de compilación ya que se dispone de información suficiente sobre el tamaño de los objetos que lo componen. En ciertos casos esto no es así como por ejemplo ocurre en C cuando se utilizan arrays de dimensión indefinida. En estos casos el registro de activación debe incluir una zona de desbordamiento al final cuyo tamaño no se fija en tiempo de compilación sino solo cuando realmente llega a ejecutarse el procedimiento. Esto complica un poco la gestión de la memoria, por lo que algunos compiladores de bajo coste suprimen esta facilidad.

El procedimiento de gestión de la pila cuando un procedimiento q llama a otro procedimiento p, se desarrolla en dos fases, la primera de ellas corresponde al código que se incluye en le procedimiento q antes de transferir el control a p, y la segunda, al código que debe incluirse al principio de p para que se ejecute cuando reciba el control.

- El procedimiento autor de la llamada (q) evalúa las expresiones de la llamada, utilizando para ello su zona de variables temporales, y copia el resultado en la zona correspondiente a los parámetros formales del procedimiento que recibe la llamada.
- El autor de la llamada (q) coloca el puntero de control del procedimiento al que llama(p) de forma que apunte al final de la pila y transfiere el control al procedimiento al que llama (p).
- El receptor de la llamada (p) salva el estado de la máquina antes de comenzar su ejecución usando para ello la zona correspondiente de su registro de activación.
- El receptor de la llamada (p) inicializa sus variables y comienza su ejecución.

Al terminar la ejecución del procedimiento llamado (p) se desaloja su registro de activación procediendo también en dos fases. La primera se implementa mediante instrucciones al final del procedimiento que acaba de terminar su ejecución (p), y la segunda en el procedimiento que hizo la llamada tras recobrar el control:

- El procedimiento saliente (p) antes de finalizar su ejecución coloca el valor de retorno al principio de su registro de activación.
- Usando la información contenida en su registro el procedimiento que finaliza (p) restaura el estado de la máquina y coloca el puntero de final de pila en la posición en la que estaba originalmente.
- El procedimiento que realizó la llamada (q) copia el valor devuelto por el procedimiento

al que llamó (p) dentro de su propio registro de activación (de q).

La fig. 6b. Muestra el estado de la figura (durante el tiempo de ejecución cuando se alcanza la instrucción señalada en el código de la fig. 6a:

La Pila

```
función q (... //param.  formales q // ...)
{
     // var. locales de q
      .....
     x = p (23 + y * z .....)
}
función p (... //param.  formales p // ...)
{
     // var. locales de p
      .....
       p (.....);
      .....
}
```

fig 6a. Código fuente

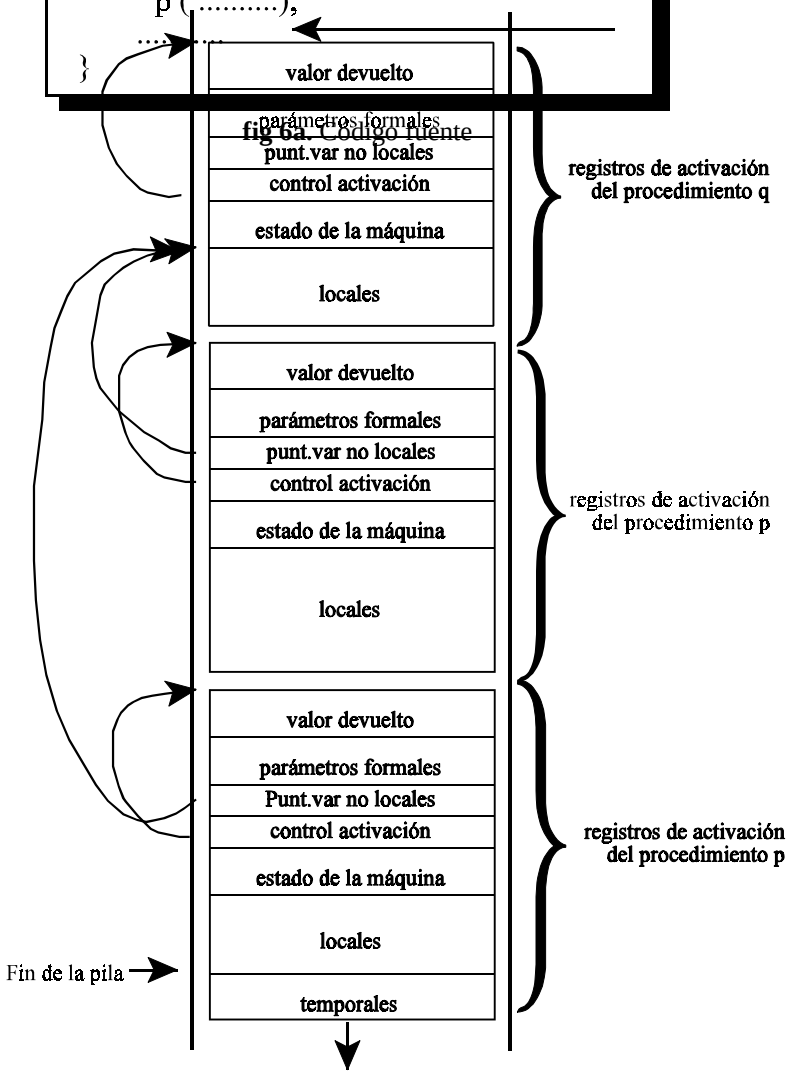

fig.6b. Registros de activación en la pila durante la ejecución del programa

La Pila

Dentro de un procedimiento, las variables locales se referencian siempre como direcciones relativas al comienzo del registro de activación, o bien al comienzo de la zona de variables locales. Por tanto, cuando sea necesario acceder desde un procedimiento a variables definidas de otros procedimientos cuyo ámbito sea accesible, será necesario proveer dinámicamente la dirección de comienzo de las variables de ese procedimiento. Para ello se utiliza dentro del registro de activación el puntero de enlace a variables no locales. Este puntero, señalará al comienzo de las variables locales del procedimiento inmediatamente superior. El puntero de enlace en una posición fija con respecto al comienzo de sus variables locales. Cuando los procedimientos se llaman a sí mismos recursivamente, el ámbito de las variables impide por lo general que una activación modifique las variables locales de otra activación del mismo procedimiento, por lo que en estos casos el procedimiento inmediato superior será, a efectos de enlace, el que originó la primera activación, tal como puede apreciarse en la figura 6b.

El montón

Cuando el tamaño de un objeto a colocar en memoria puede variar en tiempo de ejecución, no es posible su ubicación en memoria estática, ni tan siquiera en la pila. Son ejemplos de este tipo de objetos las listas, los árboles, las cadenas de caracteres de longitud variable, etc. Para manejar este tipo de objetos el compilador debe disponer de un área de memoria de tamaño variable y que no se vea afectada por la activación o desactivación de procedimientos. Este trozo de memoria se llama montón (traducción literal del termino ingles *heap* que se utiliza habitualmente en la literatura técnica). En aquellos lenguajes de alto novel que requieran el uso del montón, el compilador debe incorporar al programa objeto el código correspondiente a la gestión del montón. Las operaciones básicas que se realizan sobre el montón son:

- Alojamiento: Se demanda un bloque contiguo para poder almacenar un objeto de un cierto tamaño.
- Desalojo: Se indica que ya no es necesario conservar un objeto, y que por lo tanto, la memoria que ocupa debe quedar libre para ser reutilizada en caso necesario por otros objetos.

Según sea el programador o el propio sistema el que las invoque, estas operaciones pueden ser explícitas o implícitas respectivamente. En caso de alojamiento explícito el programador incluye en el código fuente una instrucción que demanda una cierta cantidad de memoria para la ubicación de un dato (por ejemplo en PASCAL mediante la instrucción *new*, fig. 7a; en C mediante *malloc*, fig 7b; etc.). La cantidad de memoria requerida puede ser calculada por el compilador en función del tipo correspondiente al objeto que se desea alojar, o bien puede ser especificado directamente por el programador. El resultado de la función de alojamiento es por lo general un puntero a un trozo contiguo de memoria dentro del montón que puede usarse para almacenar el valor del objeto. Los lenguajes de programación imperativos utilizan por lo general alojamiento y desalojo explícitos. Por el contrario los lenguajes lógicos y funcionales evitan al programador la tarea de manejar directamente los punteros realizando las operaciones implícitamente.

```
var
      pt : ^real;
begin
      :
      new (pt);
      :
      free(pt);
      :
end;
```

fig. 7a. PASCAL

```
double *pt;
main( )
{
   :
   pt = (double*) malloc (sizeof (double));
   :
   free ( pt );
   :
}
```

fig. 7b. C

La gestión del montón requiere técnicas adecuadas que optimicen el espacio que se ocupa o el tiempo de acceso, o bien ambos factores.