Tema 9. Organización de la Memoria en Tiempo de Ejecución

Francisco José Ribadas Pena

PROCESADORES DE LENGUAJES
4º Informática

ribadas@uvigo.es

20 de mayo de 2006

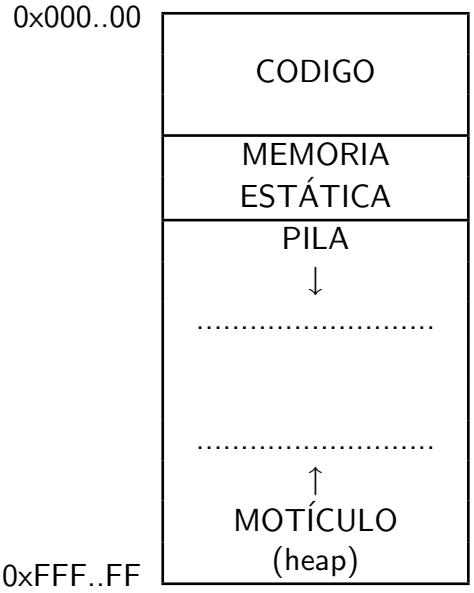
9.1 Disposición de los Programas en Memoria

Sist. Operativo (cargador/loader) suministra espacio sobre el que se ejecutará el programa

S.O. + Compilador determinan organización del espacio asignado al programa objeto

$\longrightarrow$ Compilador debe incorporar código adicional al programa objeto para gestinarlo

Organización típica



CODIGO: Almacena instrucciones en codigo máquina del programa ejecutable

- incluye el código de funciones y procedimientos
- su tamaño puede fijarse en tiempo de compilación

MEM. ESTATICA: Datos (variables) cuyo tamaño se conoce en tiempo de compilación

- Variables globales + Literales (constantes)(enteros, reales, strings, ...)
- Variables estáticas (var. locales cuyo valor se mantiene entre llamadas a procedimientos) (`static` en C)
- Otros: direcciones reservadas (por el S.O.), código y datos estáticos cargados desde librerías o módulos precompilados

PILA Zona dinámica.

Mantiene *registros de activación* de los procedimientos que se han llamado

- *Reg. activación*: continen var. locales del procedimiento + info. de control adicional
- Pila crece (en llamadas a procedim.) y decrece (al retornar)

MOTÍCULO Zona dinámica.

- Generalmente: comienza en la dirección más alta y crece "hacia abajo"
- Guarda datos cuyo tamaño varía en tiempo de ejcución o que no se pueden mantener en mem. estática ni en pila
- Espacio puede ser asignado y desasignado en cualquier momento

9.2 Asignación de Memoria

La forma en que el compilador organiza la memoria depende de las características del lenguaje:

- Soporte funciones recursivas
- Posibilidad de referenciar nombres no locales
- Modos de paso de parámetros
- Admitir procedimientos como parámetros y/o valor devuelto
- Posibilidad de asignar/desasignar memória dinámicamente

9.2.1 Asinación Estática

Modo de almacenamiento más sencillo

$\longrightarrow$ Usado en primeros compiladores, ej.: FORTRAN

Disponer los datos en memoria durante compilación

- No es necesario añadir código de gestión adicional
- Tamaño de variables debe poder conocerse en tiempo de compilación
- Dirección de las variables (globales y locales) es la misma durante toda la ejecución $\longrightarrow$ Fue calculada durante compilación

Si admite procedimientos:

- Se reserva en mem. estática espacio para un registro de activación de cada procedimiento
- Guardará: parámetros, var. locales (y temporales), y valor devuelto

No admite:

- Procedimientos recursivos
  $\longrightarrow$ imposible determinar en compilación el n^{o} de registros de activación necesarios
- Estructuras dinámicas (listas, árboles, colas,...)
  $\longrightarrow$ se desconoce el num. de elementos en tiempo de compilación

En compiladores modernos se usa asignación estática sólo para código, vars. globales y estáticas, constantes, etc...

9.2.2 Asignación Mediante Pila

Se dispone de una pila donde almacenar los registros de activación de los procedimientos llamados

$\longrightarrow$ Permite proced. recursivos: se pueden añadir nuevos RAs mientras haya espacio

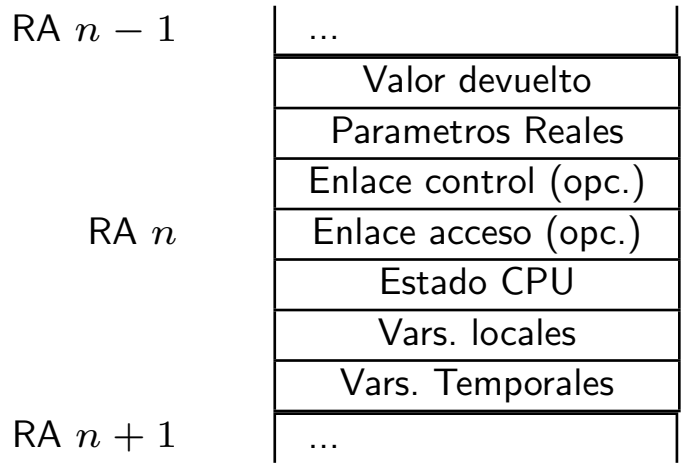
Registros de Activación (RA)

- Se crea un RA cuando se activa (llama) un procedimiento
- Cuando se desactiva (retorno) se libera su espacio y se vuelve al RA del procedim. llamador
- Cada RA se referencia mediante su dir. de inicio
- Generalmente, un registro de CPU apunta al RA actual en la pila
- En cada momento SOLO UN RA está activo $\longrightarrow$ (el correspondiente al proced. actual)

Contenido

- Parámetros + variables locales
- info. de control para volver a procedim. llamador despues de la llamada

Estructura



Valor Devuelto + Params. Reales: Intercambio de datos entre proc. llamado y llamador

- Puede hacerse a través de registros CPU
- Llamador puede acceder sin conocer estructura del siguiente RA

Enlace Control + Enlace Acceso: Punteros que direccionan otros RAs

- Para acceso a vars. no locales (enlace de acceso)
  $\longrightarrow$ RA que comparten variables
  $\longrightarrow$ deben repetarse las reglas de visibilidad del lenguaje
- Para recuprrar RA anterior (enlace de control)

Estado CPU: Info. del estado de CPU antes de la llamada (necesaria para recuperar ejecución)

- contador de programa
- registros de estado
- flags, etc,...

Variables Locales: Espacio para guardar la variables definidas dentro del proc.

- NOTA: El *offset* que guarda la TDS del bloque será un desplazamiento en este área

Variables Temporales: Vars. temporales necesarias durante ejecución del proc.

- Normalmente, incluidas al generar CI.

En general los tamaños de cada campo se podrán determinar en tiempo de compilación.

Secuencias de Llamada y Retorno

Al generar código las llamadas a procedim. se implementan mediante la generación de *secuencias de llamada* y *secuencias de retorno*

- Incluyen el código adicional encargado de la gestión del los RAs
- Las tareas se reparten entre proc. llamador y proc. llamado
- Muy dependientes de la arquitectura
  - info. sobre estado CPU depende de la máquina
  - pueden existir instrucciones específicas para facilitar llamadas a proc.
  - pueden existir convenciones (de la CPU o del SO) que determinan como realizar estas secuencias

SEC. LLAMADA: Crea el RA del nuevo proc. y asigna sus valores (params. y estado)

Proc. llamador:

- **1:** Crea nuevo RA, guarda en él el valor del puntero al RA actual y establece ese puntero hacia el nuevo RA
- **2:** Evalúa parámetros reales y los guarda en el nuevo RA
- **3:** Calcula dir. de la siguiente instrucción a ejecutar despues de llamada y la guarda en nuevo RA
- **4:** Salto a la primera instrucción del proc. llamado

Proc. llamado:

- **5:** Guarda valores registros CPU + info. de estado
- **6:** Inicializa vars. locales e inicia ejecución

SEC. RETORNO: Restablece estado de la máquina para que el proc. llamador continúe su ejecución

Proc. llamado:

- **1:** Coloca el valor devuelto al inicio de su RA
- **2:** Restablece estado CPU. El puntero a la pila de RAs vuelve a apuntar al RA llamador

Se recupera la ejecución del proc. llamador:

- Directamente, al restaurar el contador de programa
- Cargando explícitamente la dir. de retorno guardada en el RA

Proc. llamador:

- **3:** Continúa su ejecución
  $\longrightarrow$ el valor devuelto aún está en la dir. a continuación de su RA

9.2.3 Asignación por Montículos

Almacena objetos cuyo tamaño puede variar en tiempo de ejecución
$\longrightarrow$ No es posible ubicarlos en mem. estática o pila

MONTICULO: Área de mem. de tamaño variable que no se ve afectada por la activación/desactivación de RAs

Operaciones básicas

- Asignacion: se demanda un bloque contíguo para un objeto de un tamaño dado
  $\longrightarrow$ el resultado es un puntero a la zona reservada
- Desasignación: se indica que ya no es necesario conservar un objeto y que su espacio puede liberarse
- Ambas operaciones pueden ser
  - Implícitas: invocadas por el programador
    $\longrightarrow$ programador proporciona el código para reservar y liberar
    $\longrightarrow$ común en lenguajes imperativos: C, Pascal,...
  - Explícitas: invocadas por el sistema
    $\longrightarrow$ compilador incluye el códgio objeto para la gestión automática del montículo
    $\longrightarrow$ lenguajes declarativos (lógicos y funcionales), Java, ...

Asignación de Espacio

- Es posible solicitar y asignar trozos de tamaño variable
- Necesidad de mantener control de los bloques libres desasignados
- Técnicas: BEST_FIT, WORSE_FIT, FIRST_FIT,...
- Pueden ser necesarias operaciones de compactación
  - se agrupan todos los bloques libres en un único bloque
  - operación muy costosa

Desasignación de Espacio Operación más compleja. Tres opciones:

- Sin desasignación: se ignora la libreración de espacio
  - no se reutiliza el espacio liberado, se asigna espacio nuevo hasta que se agota
  - aceptable si los bloques asignados se usan durante todo el programa o si se usa mem. virtual
  - el programador puede codificar sus propias funciones de gestión de espacio
- Desasignación explícita: liberación de espacio es responsabilidad del programador
  - programador dispone de comandos de liberación: `free` C, `dispose` Pascal, ...
  - Problema: referencias nulas
- Desasignación implícita: recuperación automática del espacio no usado (*garbage collector*)
  - compilador incluye en el programa código adicional para el control automático de los bloques que ya no se usan