

Tema 2.7. Subprogramas. Traducción

Pedro Javier Rodríguez Rodrigo, Víctor Cuadrado Juan

7 de mayo de 2014

Organización de la memoria

- Es posible acceder a los datos globales

Organización de la memoria

- Es posible acceder a los datos globales
- Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)

Organización de la memoria

- Es posible acceder a los datos globales
- Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- Dos Posibles organizaciones:

Organización de la memoria

- Es posible acceder a los datos globales
- Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- Dos Posibles organizaciones:
 - ① Enlaces estáticos

Organización de la memoria

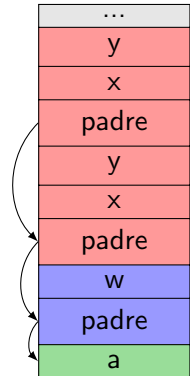
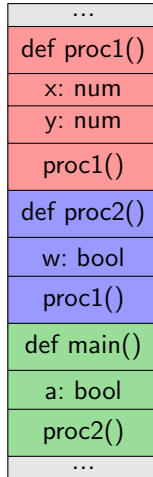
- Es posible acceder a los datos globales
- Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- Dos Posibles organizaciones:
 - 1 Enlaces estáticos
 - 2 Displays

Enlaces estáticos

- En el registro de activación se incluye un enlace al registro de activación del bloque padre (enlace estático)
- La memoria se organiza en forma de pila de registros de activación, enlazados a través de los enlaces estáticos

Enlaces estáticos: Ejemplo

```
proc proc1(){  
    x: num;  
    y: num;  
    proc1();  
}  
  
proc proc2(){  
    w: bool;  
    proc1();  
}  
  
main(){  
    a: bool;  
    proc2();  
}
```



Enlaces estáticos: Problemas

¿Qué problemas hay?

Enlaces estáticos: Problemas

¿Qué problemas hay?

- 1 La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado k niveles por encima, es necesario realizar k indirecciones hasta llegar al correspondiente registro de activación

Enlaces estáticos: Problemas

¿Qué problemas hay?

- 1 La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado k niveles por encima, es necesario realizar k indirecciones hasta llegar al correspondiente registro de activación
- 2 Hay que considerar la complejidad de generar código que gestione de manera adecuada los enlaces estáticos

Enlaces estáticos: Problemas

¿Qué problemas hay?

- 1 La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado k niveles por encima, es necesario realizar k indirecciones hasta llegar al correspondiente registro de activación
- 2 Hay que considerar la complejidad de generar código que gestione de manera adecuada los enlaces estáticos

Solución:

Almacenar los enlaces estáticos *fuera* de los registros de activación. La estructura que los almacena se llama **display**.

Display

- Secuencia de celdas consecutivas que apuntan a registros de activación

Display

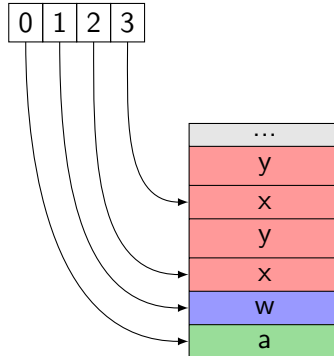
- Secuencia de celdas consecutivas que apuntan a registros de activación
- La celda i apunta al registro de activación que está siendo utilizado en el nivel de anidamiento i

Display

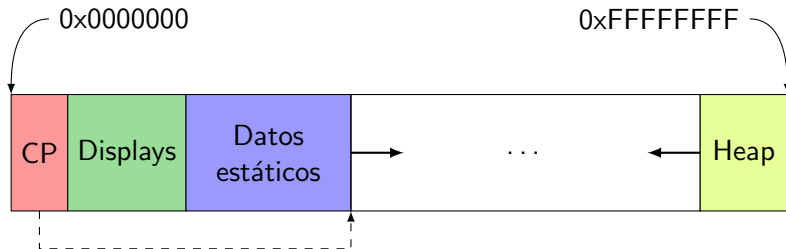
- Secuencia de celdas consecutivas que apuntan a registros de activación
- La celda i apunta al registro de activación que está siendo utilizado en el nivel de anidamiento i
- Esta estructura facilita el acceso a los datos globales: el enlace para un identificador declarado en un bloque que se encuentra a profundidad i estará en el registro de activación referido por la celda i del display (el *display* i a partir de ahora)

Display: Ejemplo

```
proc proc1(){  
    x: num;  
    y: num;  
    proc1();  
}  
proc proc2(){  
    w: bool;  
    proc1();  
}  
main(){  
    a: bool;  
    proc2();  
}
```



Memoria de un programa I



Memoria de un programa II

Las primeras celdas de la memoria se destinarán a mantener la información de estado necesaria para gestionar adecuadamente la pila de registros de activación:

- Registro *CP*: Contendrá siempre la dirección de la **última celda ocupada** por la pila de registros de activación (cuando la pila esté vacía, el valor de *CP* será la dirección de la celda anterior -la última celda ocupada por el display-)

Memoria de un programa II

Las primeras celdas de la memoria se destinarán a mantener la información de estado necesaria para gestionar adecuadamente la pila de registros de activación:

- Registro *CP*: Contendrá siempre la dirección de la **última celda ocupada** por la pila de registros de activación (cuando la pila esté vacía, el valor de *CP* será la dirección de la celda anterior -la última celda ocupada por el display-)
- *Display*

Estructura de los Registros de activación

dibujo del registro de activación

Inicio

- Se fija el *display 0* a la primera celda de datos estáticos

dibujo

Inicio

- Se fija el *display 0* a la primera celda de datos estáticos
- Se fija el *CP* a la posición de la última celda del display (la última celda ocupada)

dibujo

Inicio

- Se fija el *display 0* a la primera celda de datos estáticos
- Se fija el *CP* a la posición de la última celda del display (la última celda ocupada)
- Con ello se consigue un esquema homogéneo de direccionamiento de datos estáticos y de datos en los registros de activación

dibujo

Inicio

```
1  fun inicio(numNiveles,tamDatos) devuelve
2      // fijamos display 0 a la 1a celda de datos estaticos:
3      apila(numNiveles+2)          ||// +2: CP, display 0
4      desapila-dir(1)              ||
5      // fijamos CP a la ultima celda de datos estaticos:
6      apila(1+numNiveles+tamDatos) ||// +1: display 0
7      desapila-dir(0)
8  ffun
9  cons longInicio = 4
```

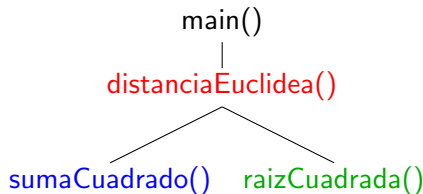
Ejemplo: inicio(2,5)

Dibujo del movimiento del cp

Ejemplo de invocación

```
tipo tpar= rec x:num; y:num;
proc distanciaEuclidea(p1:tpar, p2:tpar, var res
:num)
  a: num; b: num;
  proc sumacuadrado(a:num, b:num, var r:num)
    a:=a*a;
    b:=b*b;
    r:=a+b;
  proc raizcuadrada(var n:num)
    ...
  &
  a:=p1.x-p2.x;
  b:=p1.y-p2.y;
  sumacuadrado(a, b, res);
  raizcuadrada(res);
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5;
par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,resultado);
```

¿Cual es el máximo nivel de anidamiento para éste programa?



¿Cual es el máximo nivel de anidamiento para éste programa? 2

Esquema de la traducción

Dibujo de cómo queda la traducción de subprogramas y el main, y el salto con ir-a.

Manejo de la activación y desactivación I

dibujo con la explicación de ir-ind

Manejo de la activación y desactivación II

orden de la ejecución del código generado

Prellamada

Asociada con la invocación $p(e_1, \dots, e_k)$:

1

dibujo de lo que aparece al hacer los pasos

Prellamada

Asociada con la invocación $p(e_1, \dots, e_k)$:

1

2

dibujo de lo que aparece al hacer los pasos

Prellamada

Asociada con la invocación $p(e_1, \dots, e_k)$:

- 1
- 2
- 3

dibujo de lo que aparece al hacer los pasos

Prellamada: Ejemplo

```
1  fun apila-ret(ret) devuelve
2    // calcular CP+1:
3    apila-dir(0)           ||
4    apila(1)               ||
5    suma                   ||
6    // guardar dir retorno:
7    apila(ret)             ||
8    desapila-ind           ||
9  ffun
10 cons longApilaRet = 5
```

Ejemplo: `apila-ret(0x527)` , siendo 0x527 el nº de instrucción.

Dibujo

Prólogo I

Asociado con el prodecimiento *procp*(...):

1

dibujo de lo que aparece al hacer los pasos

Prólogo I

Asociado con el prodecimiento *procp*(...):

- 1
- 2

dibujo de lo que aparece al hacer los pasos

Prólogo I

Asociado con el prodecimiento *procp(...)*:

- 1
- 2
- 3

dibujo de lo que aparece al hacer los pasos

Prólogo II

```
1  fun prologo(nivel,tamlocales) devuelve
2    // salvar display antiguo:
3    apila-dir(0)           ||
4    apila(2)               ||
5    suma                   ||
6    apila-dir(1+nivel)    ||
7    desapila-ind          ||
8    // fijar el display actual:
9    apila-dir(0)           ||
10   apila(3)               ||
11   suma                   ||
12   desapila-dir(1+nivel) ||
13   // reservar espacio para datos locales:
14   apila-dir(0)           ||
15   apila(tamlocales+2)    ||
16   suma                   ||
17   desapila-dir(0)
18  ffun
19  cons longPrologo = 13
```

Prólogo: Ejemplo I

```
1  fun prologo(nivel,tamlocales) devuelve
2      // salvar display antiguo:
3      apila-dir(0)      ||
4      apila(2)          || // dir. retorno, antiguo display
5      suma              ||
6      apila-dir(1+nivel) || // +1: saltar al display 0
7      desapila-ind      ||
8      // fijar el display actual:
9      apila-dir(0)      ||
10     apila(3)           ||
11     suma               ||
12     desapila-dir(1+nivel) ||
13     // reservar espacio para datos locales:
14     apila-dir(0)       ||
15     apila(tamlocales+2) ||
16     suma               ||
17     desapila-dir(0)
18  ffun
```

Ejemplo: prologo(1,7)
Dibujo

Prólogo: Ejemplo II

```
1  fun prologo(nivel,tamlocales) devuelve
2  // salvar display antiguo:
3  apila-dir(0)      ||
4  apila(2)          || // dir. retorno, antiguo display
5  suma              ||
6  apila-dir(1+nivel) || // +1: saltar al display 0
7  desapila-ind      ||
8  // fijar el display actual:
9  apila-dir(0)      ||
10 apila(3)           ||
11 suma              ||
12 desapila-dir(1+nivel) ||
13 // reservar espacio para datos locales:
14 apila-dir(0)       ||
15 apila(tamlocales+2) ||
16 suma              ||
17 desapila-dir(0)
18 ffun
```

Ejemplo: prologo(1,7)
Dibujo

Prólogo: Ejemplo III

```
1  fun prologo(nivel,tamlocales) devuelve
2  // salvar display antiguo:
3  apila-dir(0)      ||
4  apila(2)          || // dir. retorno, antiguo display
5  suma              ||
6  apila-dir(1+nivel) || // +1: saltar al display 0
7  desapila-ind      ||
8  // fijar el display actual:
9  apila-dir(0)      ||
10 apila(3)           ||
11 suma              ||
12 desapila-dir(1+nivel) ||
13 // reservar espacio para datos locales:
14 apila-dir(0)       || // Mem[1+nivel] = Pila[Cima]
15 apila(tamlocales+2) || // +2: dir. retorno, antiguo display
16 suma              ||
17 desapila-dir(0)
18 ffun
```

Ejemplo: prologo(1,7)
Dibujo

Epílogo I

Asociado con el prodecimiento *procp(...)*:

1

dibujo de lo que aparece al hacer los pasos

Epílogo I

Asociado con el prodecimiento *procp(...)*:

1

2

dibujo de lo que aparece al hacer los pasos

Epílogo I

Asociado con el prodecimiento *procp(...)*:

- 1
- 2
- 3

dibujo de lo que aparece al hacer los pasos

Epílogo II

```
1  fun epilogo(nivel) devuelve
2    // apilar la dir. retorno:
3    apila-dir(1+nivel)    ||
4    apila(2)              ||
5    resta                 ||
6    apila-ind             ||
7    // liberar espacio (mover CP):
8    apila-dir(1+nivel)    ||
9    apila(3)              ||
10   resta                 ||
11   copia                 ||
12   desapila-dir(0)       ||
13   // recuperar antiguo display:
14   apila(2)              ||
15   suma                  ||
16   apila-ind             ||
17   desapila-dir(1+nivel)
18 ffun
19 cons longEpilogo = 13
```

Epílogo: Ejemplo I

```
1  fun epilogo(nivel) devuelve
2  // apilar la dir. retorno:
3  apila-dir(1+nivel)  ||
4  apila(2)           ||
5  resta              ||
6  apila-ind          ||
7  // liberar espacio (mover CP):
8  apila-dir(1+nivel)  ||
9  apila(3)           ||
10 resta              ||
11 copia              ||
12 desapila-dir(0)     ||
13 // recuperar antiguo display:
14 apila(2)             ||
15 suma                ||
16 apila-ind           ||
17 desapila-dir(1+nivel)
18 ffun
```

Ejemplo: epilogo(1)
Dibujo

Epílogo: Ejemplo II

```
1  fun epilogo(nivel) devuelve
2      // apilar la dir. retorno:
3      apila-dir(1+nivel)    ||
4      apila(2)              ||
5      resta                  ||
6      apila-ind              ||
7      // liberar espacio (mover CP):
8      apila-dir(1+nivel)    ||
9      apila(3)              ||
10     resta                  ||
11     copia                  || // si no se usa copia, hay que mover CP lo ultimo
12     desapila-dir(0)        ||
13     // recuperar antiguo display:
14     apila(2)                ||
15     suma                    ||
16     apila-ind               ||
17     desapila-dir(1+nivel)
18  ffun
```

Ejemplo: epilogo(1)

¿Es la mejor forma de implementar el epilogo, con mover el CP en 2º lugar?

Dibujo

Epílogo: Ejemplo III

```
1  fun epilogo(nivel) devuelve
2  // apilar la dir. retorno:
3  apila-dir(1+nivel)  ||
4  apila(2)           ||
5  resta              ||
6  apila-ind          ||
7  // liberar espacio (mover CP):
8  apila-dir(1+nivel)  ||
9  apila(3)           ||
10 resta              ||
11 copia              ||
12 desapila-dir(0)     ||
13 // recuperar antiguo display:
14 apila(2)            ||
15 suma               ||
16 apila-ind          ||
17 desapila-dir(1+nivel)
18 ffun
```

Ejemplo: epilogo(1)
Dibujo

Postllamada

- No es necesaria en nuestra implementación, ya que el epílogo recupera el estado anterior a la invocación.

dibujo de la cima de la pila
dibujo de la memoria

Postllamada

- No es necesaria en nuestra implementación, ya que el epílogo recupera el estado anterior a la invocación.
- Hemos terminado con la dirección (indirecta) de retorno en la cima de la pila

dibujo de la cima de la pila

dibujo de la memoria

Resumen

repetir dibujo del manejo de la activacion y la desactivacion