

CONSTRUCCION DE UN MONTADOR DE ENLACE PARA
PROGRAMAS ESCRITOS EN PASCAL SECUENCIAL

N. TAKEDA

En los procedimientos standard de la mayor parte de sistemas, se incluye la posibilidad de catalogar subrutinas (propias del usuario o del sistema) en librerías. El posterior uso de un Montador de Enlaces facilita la tarea de crear programas complejos a partir de módulos y subrutinas compilados y probados -- por separado y posteriormente catalogados. En este trabajo se describe la implementación de dichas facilidades para el compilador PASCAL secuencial. Dicha implementación afecta a los procesos de compilación en el sentido de permitir definir referencias externas que son resueltas posteriormente por el programa Montador de Enlaces.

1. INTRODUCCION

La implementación del sistema operativo SOLO /2/ escrito por Brinch Hansen en lenguaje -- PASCAL Concurrente /6/ para una máquina virtual (e implementado por el propio Hansen sobre una máquina real PDP 11/45) implica escribir primeramente la interfase entre la máquina virtual y la máquina real de la que se dispone. El programa que sirve de interfase recibe el nombre de KERNEL y normalmente está escrito en el lenguaje ensamblador de la máquina real. En otras palabras, dado que el sistema operativo SOLO trabaja sobre una máquina virtual su implementación en cualquier sistema implica escribir el KERNEL para la máquina real en cuestión. En nuestro caso, la implementación del SOLO se hizo sobre una máquina real FACOM 230-25.

Tanto el propio sistema operativo SOLO como los compiladores PASCAL secuencial /1/, /3/, /4/, /5/ y PASCAL concurrente, además de los programas de aplicación, están escritos en PASCAL (concurrente para el S.O. SOLO y secuencial para compiladores y programas, respectivamente). El código de salida de los dos compiladores del sistema está escrito en lenguaje ensamblador de la máquina virtual (llamado P-code) y está diseñado para utilizar plenamente la arquitectura "pila"¹ de la máquina virtual. Si la máquina real sobre la que se implementa el sistema es una máquina "pila" (tal como las de la serie PDP 11) el rendimiento del sistema es el óptimo. En --

nuestro caso y como sea que el ordenador FACOM 230-25 no es una máquina "pila" el KERNEL es necesariamente más lento y complicado que en el caso anterior. Consiguientemente la lentitud es una característica constante en todas las operaciones del sistema.

Como es sabido, la posibilidad de dividir un programa en módulos funcionales (por ejemplo rutinas propias y/o del sistema) aumenta la flexibilidad de la programación por una parte y reduce los tiempos de compilación por otra al poder tener dichos módulos compilados y probados separadamente, en librerías. En nuestro caso, la poca velocidad del sistema hace especialmente crítico el hecho de -- compilar programas PASCAL, incluso los poco extensos.

Por otra parte aunque en el sistema SOLO pueden cargarse programas y utilizarlos conceptualmente como rutinas, esto se hace en tiempo de ejecución, es decir cargando el programa cada vez desde el disco y sobre la misma área. Nótese que el concepto de subrutina catalogada no existe en el SOLO. En realidad existen programas catalogados que pueden llamarse desde un punto de vista lógico como si fueran una subrutina. La diferencia es que estos programas llamados se cargan en la misma área que el programa que los llama, volviendo luego evidentemente éste último a la misma posición anterior. El "overhead" producido es por tanto crítico en programas que utilicen extensamente este tipo de fracciona

- N. Takeda, Analista de Sistemas al CCUPB. Av. Dr. Gregorio Marañón, s/n. Barcelona 28; i Prof. Assistant a la Universitat de Seikei-Japó.
- Article rebut el Setembre del 1978.

miento lógico.

Por todas estas razones, se decidió construir un Montador de Enlaces² que mejorara el rendimiento del sistema compaginando programas (usados como subrutinas funcionales) en tiempo de edición y montaje. Aunque aplicadas a un ordenador FACOM 230-25 las mejoras indicadas más arriba son completamente generales.

El estudio se divide en dos partes:

- Modificaciones en el compilador PASCAL secuencial para generar referencias externas.
- Programa Montador de Enlaces que trate y resuelva estas referencias externas.

Como convención en la notación,

- se llamará "programa" a la salida del compilador,
- se llamará "programa compaginado" a la salida del Montador de Enlaces,
- se llamará "programa principal" al programa que se especifica en primer lugar en el comando de lanzamiento del programa Montador de Enlaces,
- se llamará "rutina" en general tanto a las PROCEDURE como a las FUNCTION.

2. MODIFICACIONES EN EL COMPILADOR

Si el usuario desea utilizar la facilidad de un Montador de Enlaces debe declarar símbolos externos y símbolos globales (EXTERN SYMBOL y GLOBAL SYMBOL).

Estas declaraciones deben escribirse después de la sentencia PROGRAM, sirviendo solamente para indicar que son variables o rutinas a enlazar:

```
PROGRAM P(ARG:ARGLIST);
GLOBAL PROCEDURE ADD(A:INTEGER);
GLOBAL VAR R:REAL;
EXTERN PROCEDURE WRITE(I:INTEGER,
    VAR NUM:IDENTIFIER);
```


```
VAR R:REAL;
```

```
-----  
-----  
PROCEDURE ADD(A:INTEGER);  
-----  
-----
```

Los significados de GLOBAL y EXTERN son los siguientes:

GLOBAL: Las variables o rutinas declaradas como GLOBAL se definen en el programa en que son declaradas.

Otro programa puede hacer referencia a estas variables o rutinas.

Debido a las reglas de ámbito de las variables en PASCAL, las variables definidas GLOBAL han de declararse en la rutina principal del programa principal.

EXTERN: Las variables o rutinas declaradas EXTERN no se definen en el programa en que son declaradas. Sirven para indicar que la mencionada variable o rutina está definida (mediante un GLOBAL) en otro programa.

En ningún caso puede declararse un programa como GLOBAL o EXTERN.

A continuación se describen brevemente las modificaciones efectuadas en los distintos pasos del compilador.

Cada uno de los siete pasos de compilación genera código objeto sobre el cual trabaja el siguiente paso. La principal modificación en el compilador PASCAL secuencial de Brinch Hansen es debida a la necesidad de generar un registro con la información sobre los atributos GLOBAL o EXTERN aplicados a variables y rutinas:

```
TYPE LINKRECORD7=RECORD;
    KEY: (EXTERN,GLOBAL);
    SYMBOL: ARRAY(.1..12.) OF CHAR;
    MODE: (VARIABLE,ROUTINE);
    DISPLACEMENT: INTEGER;
END;
```

En lo sucesivo nos referiremos a "registro" al hablar de la información del tipo LINKRE-

CORD que generaremos junto con el código objeto.

PASO1: Sus principales tareas son:

- cambio de las constantes numéricas a números binarios,
- cambio de los símbolos de operación a códigos internos de operación,
- construcción de la tabla de identificadores y asignación de códigos numéricos.

La implementación es la siguiente para cada EXTERN o GLOBAL encontrado:

- crear un código para los símbolos EXTERN y GLOBAL,
- guardar el identificador al cual se le aplica el atributo EXTERN o GLOBAL para el paso 2 del compilador, creando un registro del tipo LINKRECORD1,

```
TYPE LINKRECORD1=RECORD;  
    KEY: (EXTERN,GLOBAL);  
    SYMBOL: ARRAY(.1..12.) OF CHAR;  
END;
```

- continuar la compilación de la sentencia en curso.

Nótese que para la variable R del -- ejemplo anterior, se generará un registro LINKRECORD1 correspondiente a la declaración GLOBAL VAR R:REAL, y posteriormente el código objeto normal correspondiente al proceso de la declaración VAR R:REAL.

PASO2: Sus tareas son:

- análisis sintáctico.

La implementación es la siguiente:

- pasar los registros LINKRECORD1 al paso 3 de la compilación.

PASO3: Sus tareas son:

- análisis del ámbito de definición de las variables.

La implementación es la siguiente:

- tratar las sentencias GLOBAL aplicadas a rutinas como si fueran sentencias FORWARD,
- tratar las sentencias EXTERN aplicadas a rutinas como si fueran sentencias de rutinas definidas en el PREFIX,
- tratar las sentencias EXTERN aplicadas a variables como sentencias normales de variables,
- tratar las sentencias GLOBAL aplicadas a variables reduciendo la información sobre el tipo de atributos de la variable, ya que estos atributos se encuentran repetidos en el código objeto generado en los pasos anteriores,
- pasar los registros LINKRECORD1 al paso 4 de la compilación.

PASO4: Sus tareas son:

- finalizar el análisis de las declaraciones,
- decidir los desplazamientos de las variables,
- eliminar el paso de información concerniente a las declaraciones al paso 5 de compilación.

La implementación es:

- formación del registro LINKRECORD4.

```
TYPE LINKRECORD4 = RECORD;  
    KEY: (EXTERN,GLOBAL);  
    SYMBOL: ARRAY(.1..12.) OF CHAR;  
    MODE: (VARIABLE,ROUTINE);  
    DISPLACEMENT: INTEGER;  
END;
```

Nótese que la aparición del registro LINKRECORD1 para una variable GLOBAL en el flujo de entrada al paso 4 es anterior a la aparición del código objeto generado para la misma variable. Dado que el paso 4 resuelve los desplazamientos de las variables declaradas, dicho desplazamiento se obtendrá con posterioridad a la generación de LINKRECORD4. Por tanto, para completar LINKRECORD4 deberemos ir hacia atrás para localizarlo y completarlo con el desplazamiento calculado por el proceso general del paso 4. El sistema de localizar hacia atrás el re--

gistro LINKRECORD4 correspondiente se implementa a base de salvar su posición física.

PASO5: Sus tareas son:

- análisis semántico de la parte ejecutable del programa.

La implementación es:

- pasar LINKRECORD4 al siguiente paso de compilación.

PASO6: Sus tareas son:

- selección de los códigos de operación,
- construcción de la tabla de localizaciones de puntos de entrada a rutinas, puntos de control de secuencia, etc., para el cálculo de desplazamientos en el paso 7, tomando como origen el principio del programa.

La implementación es:

Para cada referencia a una rutina o variable declarada con el atributo EXTERN, se crea una entrada del tipo EXTRTABLE:

```
TYPE EXTRTABLE = RECORD;  
  PAGE -NUMBER: INTEGER;  
  WORD -NUMBER: INTEGER;  
  IDENTIFICATION -NUM: INTEGER;  
  LOCATION: INTEGER;  
END;
```

con la información siguiente:

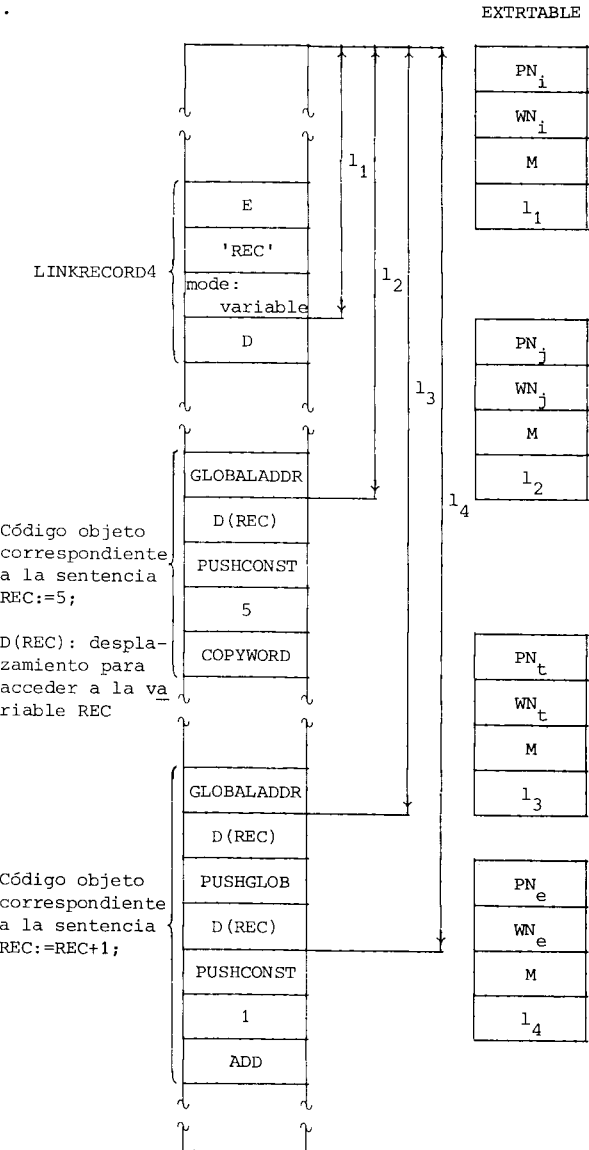
PAGE -NUMBER: contienen la dirección física de la palabra código objeto que el paso 7 modificará.

IDENTIFICATION -NUM: número de identificación de la rutina o variable.

LOCATION: dirección lógica relativa al principio del programa de la palabra de código objeto que el paso 7 usará.

Por ejemplo:

```
EXTERN VAR REC:RECORD;  
.  
.  
REC:=5;  
.  
REC:=REC+1;  
.  
.
```



PASO7: Sus tareas son:

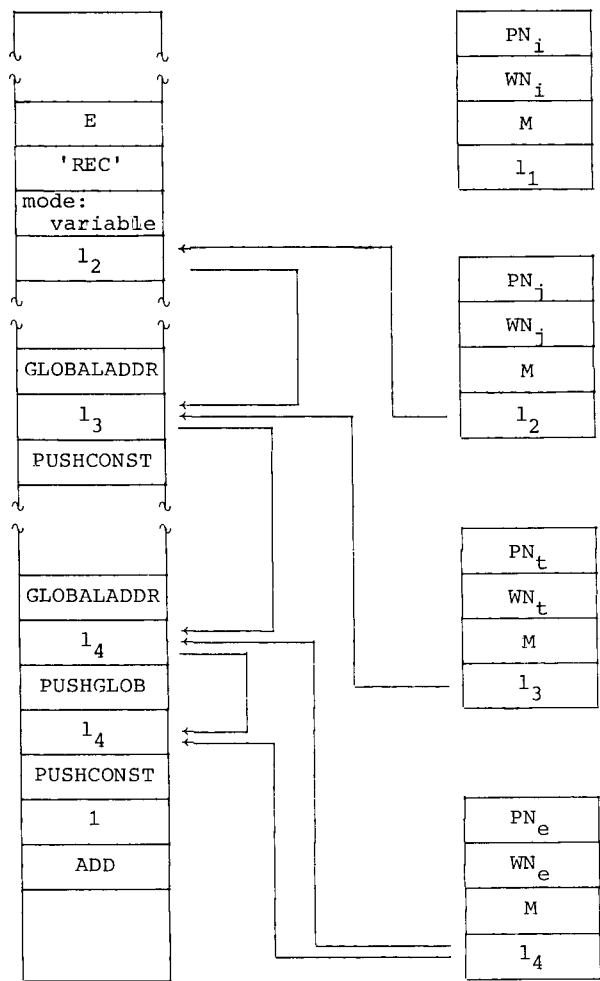
- ensamblaje del código objeto,
- cálculo de los desplazamientos de los puntos de control a partir de la tabla del paso 6.

La implementación es:

- completar el registro tipo LINKRECORD4 con el desplazamiento correcto, convirtiéndose en el registro tipo LINKRECORD7.

Para las rutinas GLOBAL, la variable DISPLACEMENT contiene la localización (relativa al principio del programa) de su punto de entrada.

Para las rutinas y variables EXTERN, ya que no podemos saber el desplazamiento en tiempo de compilación, deberemos dar información para el Montador de Enlaces sobre dónde se hace referencia a estas variables. Para ello se utilizan las entradas EXTRTABLE en el siguiente proceso:



En definitiva se pasa (en cada referencia a la variable o rutina) la localización de la variable o rutina siguiente (con igual IDENTIFICATION- $\rightarrow$ NUM=M). Para la última referencia se pone la localización de ella misma.

Para las variables EXTERN, se deben generar las operaciones PUSHGLOB y --GLOBADDR para acceder a la variable --definida en la rutina principal del --

programa principal.

El número de declaraciones EXTERN y --GLOBAL se añaden al código objeto.

Como ejemplo de todo el proceso, se indica --el resultado de la compilación para el siguiente programa:

```
PROGRAM P (ARG:ARGLIST);
  GLOBAL PROCEDURE ADD(A:INTEGER);
  GLOBAL VAR R:REAL;
  EXTERN PROCEDURE WRITE(I:INTEGER);
  VAR R:REAL;
  PROCEDURE ADD(A:INTEGER);
  .
  .
  .
  BEGIN
  END
```

PROGRAM LENGTH	CODE LENGTH	STACK LENGTH	VARIABLE LENGTH
JUMP	MAIN PROGRAM ³	(number of extern & global dec)	global
A	D	D	---
---	---	---	---
---	---	---	---
mode: PROCEDURE	displacement: 104	global	R
---	---	---	---
---	---	---	---
---	---	---	mode: VARIABLE
displacement: -1	extern	W	R
I	T	E	---
---	---	---	---
---	---	mode: PROCEDURE	displacement: 210
ENTER for procedure ADD	---	---	---

El nombre del nuevo compilador PASCAL es NEW PASCAL y por tanto una compilación debe llamarse:

```
NEWPASCAL (SOURCE,DESTINATION,  
          OBJECT:IDENTIFIER)
```

3. MONTADOR DE ENLACES

3.1 General

El trabajo del Montador de Enlaces es el de compaginar varios programas compilados por el compilador NEWPASCAL. Se ejecuta el Montador de Enlaces con la sentencia:

```
LIED (NEWPRG,PRGMAIN,PRGSUB1,PRGSUB2,...  
      ...,PRGSUB8)
```

donde NEWPRG es el nombre del programa completo (una vez compaginados el PRGMAIN y los PRGSUB1,...,PRGSUB8).

Con este comando se empieza la ejecución del compaginador LIED.

Primeramente el LIED recupera todos los programas especificados y coloca cada uno de ellos a frontera de página.

3.2 Proceso de EXTERN

Seguidamente se deben dar las direcciones correctas a las variables y rutinas declaradas EXTERN. Para ello se toman los registros --LINKRECORD7 del código objeto y al encontrar una variable o rutina EXTERN, se inspeccionan los demás programas en busca de una variable o rutina declarada GLOBAL con el mismo nombre. Una vez encontrada se puede iniciar ya el proceso de enlace.

Siguiendo ahora la cadena preparada por el paso 7, podemos ir obteniendo el desplazamiento correcto para cada variable o rutina, modificando adecuadamente el código objeto con el nuevo desplazamiento.

3.3 Proceso de constantes

El código objeto PASCAL consta de dos partes:

código y constantes.

La parte de constantes debe colocarse después de la parte del código. Por tanto, al conectar varios programas, las partes de constantes de cada uno deben ser agrupadas al final. De acuerdo con esto, los operandos de la instrucción CONSTA deben modificarse.

3.4 Proceso final

Al unir varios programas deberemos modificar también las palabras del código objeto:

```
PROGRAM _LENGTH, CODE _LENGTH, STACK _LENGTH,  
VARIABLE _LENGTH
```

El programa FILE (standard de Brinch Hansen) se usa para catalogar el programa ya compaginado en un archivo en disco, preparado para ser ejecutado.

4. CONCLUSION

El compilador NEWPASCAL y el montador LIED se han implementado y probado sobre un ordenador FACOM 230-25.

Para comparar el compilador SPASCAL secuencial standard con el compilador NEWPASCAL se han pasado programas de prueba.

Ex. 1.: Compilación SPASCAL

sentencias fuente: 514 sentencias
(prefix: 120 sent).
código objeto: 21 páginas
tiempo de compilación: 18' 50"

Compilación NEWPASCAL
programa dividido en dos partes:

Main: sentencias fuente: 365 sentencias
(prefix: 124 sent).
código objeto: 8 páginas
tiempo de compilación: 9' 50"

Sub1: sentencias fuente: 292 sentencias
(prefix: 124 sent).
código objeto: 13 páginas
tiempo de compilación: 12' 35"

Tiempo total para la compilación en:
NEWPASCAL: 22' 25".

La mayor lentitud del compilador NEWPASCAL es debida al proceso de EXTERN y GLOBAL así como al doble proceso del PREFIX. A pesar de ello, los tiempos totales de compilación obtenidos hasta poner a punto el programa se redujeron al partirlo en dos y usar el nuevo compilador.

En una primera versión el programa LIED compaginaba los distintos programas cargando todas sus páginas de código objeto en memoria al mismo tiempo. La limitación en cuanto al número total de páginas de código objeto del programa compaginado era de 22 páginas. En una posterior versión, se ejecuta el proceso sobre una sola página de código objeto a la vez, utilizando la técnica de intercambio de memoria³. La limitación en este caso es de 40 páginas, número máximo de páginas de código objeto que el sistema SOLO permite ejecutar.

Con la nueva versión del compilador, el tiempo de compaginación de dos programas de 3 páginas cada uno fué de 22 segundos. Este tiempo aumentó a 23 segundos con la segunda versión. Para un programa de 8 y 15 páginas compaginadas, el LIED tardó 1' 09" con su segunda versión.

5. RECONOCIMIENTOS

Al Prof. Dr. Martín Vergés Trias, Director del Centro de Cálculo de la Universidad Politécnica de Barcelona por auspiciar la realización de este trabajo.

6. BIBLIOGRAFIA

- /1/ BRINCH HANSEN, P. "Sequential PASCAL REPORT". Calif. Inst. of Technology, July 1975.
- /2/ BRINCH HANSEN, P. "The Solo Operating -- System". Calif. Inst. of Technology, July, 1975.
- /3/ JENSEN, K. & WIRTH, N. "Pascal-User Manual and Report". Springer Verlag, New York 1974/75.
- /4/ WIRTH, N. "The programming language Pascal". Acta Informática, I, 1971. pp. 35-63.

- /5/ WIRTH, N. "Systematic Programming". Prentice-Hall, Englewood Cliffs, N.J. 1963.
- /6/ BRINCH HANSEN, P. "Concurrent PASCAL REPORT". Calif. Inst. of Technology, June 1975.

NOTAS

- 1. Se traduce por "pila" el término inglés "stack".
- 2. Se traduce por "Montador de Enlaces" el término inglés "linkage editor".
- 3. Se traduce por "intercambio de memoria" el término inglés "swapping".

