Practica 7
Ligadores

Introducción

La función principal de los ligadores es la relocalización y resolución de nombres simbólicos en programas multimódulos. En esta práctica usaremos el ligador de Turbo C y analizaremos el formato de los archivos objeto que genera con el programa TDUMP.

Formato de los archivos objeto de DOS

Un archivo objeto es un conjunto de registros donde cada registro tiene el siguiente formato:

```
                        <------Record Length in Bytes----->
   1          2         <variable>              1
   Tipo       Longitud  Contenido               Checksum ó 0
   Reg        Reg       del Registro
```

El tipo de registro es un byte que identifica el tipo de registro (algunos tipos son: `THEADR o LHEADR`, `COMENT`, `SEGDEF` y `PUBDEF` entre otros). La longitud del registro es un campo de 2 bytes que proporciona la longitud en bytes del resto del registro. Finalmente el registro con un byte que contiene el checksum del registro o 0's. El programa TDUMP del Turbo C nos permite analizar archivos objeto y ejecutables.

Objetivo

El objetivo de esta práctica es conocer mejor el formato de un archivo objeto de DOS mediante un breve análisis de los archivos objeto y ejecutables que genera el compilador de Turbo C.

Actividades

1.1- Captura los siguientes archivos en C y compílalos con el compilador de Turbo C para generar el código en ensamblador y el código objeto de cada uno de ellos.

```
/* Archivo: a.c
  * Ilustra el uso de programas multimodulos */
#include<io.h>
#include<string.h>
void a(char *s)
{
    int i;
    for(i=0;i<10;i++)
     write(1,s,strlen(s));
}
```

---------- +++ ----------

```
/* Archivo: m.c
 * Ilustra el desarrollo de programas multimodulos */
extern void a(char *);
int main()
{
    static char texto[]="Hola Mundo desde programa multimodulo!\n";
    a(texto);
    return 0;
}
```

1.2.- Desde una ventana de DOS, genera el ensamblador y luego el objeto de estos 2 programas con los comandos

```
tcc -S a.c
tcc -S m.c
```

Tamaños: `a.asm: _____  m.asm: _____`

Ahora usando los comandos `more a.asm` y `more m.asm` observa el contenido de estos 2 archivos. Trata de ubicar las instrucciones de ensamblador a las que se convirtieron cada una de las instrucciones del programa en C.

1.3.- Ahora genera el código objeto de estos 2 programas con los comandos

```
tcc -c a.c
tcc -c m.c
```

Tamaños: `a.obj: _____  m.obj: _____`

2.- Análisis de los archivos objeto

Vamos ahora a usar el programa TDUMP para analizar brevemente el contenido de los archivos objeto.

Desde una ventana de DOS teclea el comando `tdump<Entrer>` que desplegará un texto de ayuda sobre este programa.

2.1.- ¿Qué función se activa con las siguientes opciones?

-a : _____
-h : _____
-o : _____

2.2.- ¿Que tipos de archivos puede manejar TDUMP?

i) _____
ii) _____
iii) _____
iv) _____
v) _____
vi) _____

2.3.- Ahora vamos a analizar cada uno de los archivos objeto con el comando TDUMP tecleando

```
tdump -o a.obj | more<enter>
```

que debe generar una salida parecida a lo siguiente:

```
Turbo Dump  Version 3.1 Copyright (c) 1988, 1992 Borland International
                      Display of File A.OBJ

000000 THEADR  a.c
000008 COMENT  Purge: Yes, List: Yes, Class: 0   (000h)
    Translator: TC86 Borland C++ 3.1
000023 COMENT  Purge: Yes, List: Yes, Class: 249 (0F9h)
    Debug information version 3.0
00002B COMENT  Purge: Yes, List: Yes, Class: 233 (0E9h)
    Dependency File: a.c                  10/12/105  01:28 pm
```

El número hexadecimal entre paréntesis al final de la primera línea de algunos registros es su identificador. Por ejemplo, el id (`000h`) nos indica que programa generó este archivo objeto, esto es quien fué el traductor de este

archivo. Los registros (0E9h) indican los archivos de dependencia, esto es, de que archivos se obtuvo el código fuente para este archivo objeto.

2.4.- Enlista los archivos de dependencia para el archivo objeto `a.obj`:

Archivo de dependencia 1: _____
Archivo de dependencia 2: _____
Archivo de dependencia 3: _____
Archivo de dependencia 4: _____
Archivo de dependencia 5: _____
Archivo de dependencia 6: _____

2.5.- El registro (0EAh) muestra parámetros de compilación. ¿Qué valores se muestran para estos parámetros?

```
Source Language: _____
Leading Underscores: _____
Memory Model: _____
```

Hay otras secciones que no tienen su identificador entre paréntesis. Entre ellas están las definiciones de segmentos, que en el caso de C siempre son 3: _TEXT, _DATA y _BSS mostrando su alineación, tipo (pública o privada), su clase ('CODE, 'DATA', 'BSS') y su longitud.

2.6.- ¿Cuáles son los parámetros para los segmentos de este archivo objeto?

Segmento	Nombre	Alineación	Tipo	Clase	Longitud
SEGDEF 1 :	_____	_____	_____	_____	_____
SEGDEF 2 :	_____	_____	_____	_____	_____
SEGDEF 3 :	_____	_____	_____	_____	_____

2.7.- En las sección EXTDEF aparecen los nombres de las funciones externas que se invocan en este archivo objeto (esto es, las funciones que este módulo utiliza pero que están definidas en otro módulo). ¿Cúantas y cuáles son para el archivo `a.obj`?
Referencia externa 1: _____
Referencia externa 2: _____
Referencia externa 3: _____

2.8.- En la sección PUBDEF se enlistan las funciones definidas como públicas en este módulo (esto es, las funciones definidas en este módulo y que pueden ser invocadas desde otro módulo externo). ¿Cuál es la función pública de este módulo? _____

Posteriormente, en la sección `LEDATA segment _TEXT` aparece el desplazamiento (offset) y la longitud del código en lenguaje máquina de este módulo, seguido de la sección `FIXUPP` que indica en que localidades el código debe ser parchado.

2.9.- Datos de la sección `LEDATA segment _TEXT`
Offset: _____ Longitud: _____

2.10.- Datos de la sección FIXUPP
Primer fixup : _____ Modo: _____ Loc: _____
Segundo fixup : _____ Modo: _____ Loc: _____

Si cuentan (en hexadecimal) los números asociados a los Fixup se darán cuenta que apuntan a grupos de 0's (en el

caso del Fixup 00e, apunta a los ceros que siguen a E8 (E8 00 00), que es el código máquina de la instrucción CALL dir_procedimiento, esto es, se debe parchar la dirección de un procedimiento externo a llamar. Y el otro FixUp (018) apunta a los 0's que siguen a otro E8 (E8 00 00), la llamada a otro procedimiento externo.
2.11.- ¿Cuáles son estos procedimientos externos? _____
_____

3.- Análisis del archivo m.obj
Aplicando lo visto en el inciso anterior, vamos ahora a utilizar el programa TDUMP para analizar el objeto m.obj usando el comando

```
tdump -o m.obj | more<enter>
```

3.1.- ¿Cuáles son los archivos de dependencia des este módulo?
Archivo de dependencia 1: _____
Archivo de dependencia 2: _____

3.2.- El registro (0EAh) muestra parámetros de compilación. ¿Qué valores se muestran para estos parámetros?

```
Source Language: _____
Leading Underscores: _____
Memory Model: _____
```

3.3.- ¿Cuáles son los parámetros para los segmentos de este archivo objeto?

Segmento	Nombre	Alineación	Tipo	Clase	Longitud
SEGDEF 1 :	_____	_____	_____	_____	_____
SEGDEF 2 :	_____	_____	_____	_____	_____
SEGDEF 3 :	_____	_____	_____	_____	_____

3.4.- ¿Cuáles son las funciones externas que se invocan en este archivo objeto (sección EXTDEF)?
Referencia externa 1: _____
Referencia externa 2: _____

3.5.- ¿Cuál es la función pública de este módulo (sección PUBDEF)?
_____

3.5.- Este archivo tiene dos secciones LEDATA (segment _TEXT y segment _DATA). ¿porqué crees que a.obj no tiene seccción _DATA y m.obj sí la tiene?
_____
_____
_____

3.6.- Datos de la sección LEDATA segment _TEXT
Offset: _____ Longitud: _____

3.7.- Datos de la sección FIXUPP de segment _TEXT
Primer fixup : _____ Modo: _____ Loc: _____
Segundo fixup : _____ Modo: _____ Loc: _____

3.8.- ¿A qué instrucciones se aplican los parches que se indican en este módulo?
Primera instrucción: _____
Segunda instrucción: _____

Nota: la instrucción `E8 00 00` es el código máquina de la instrucción `MOV AX, DatoInmediato`

3.9.- Datos de la sección `LEDATA segment _DATA`
Offset: _____ Longitud: _____

3.10.- Datos de la sección `FIXUPP de segment _DATA`
Primer fixup : _____ Modo: _____ Loc: _____
Segundo fixup : _____ Modo: _____ Loc: _____

4.- Creación y análisis del ejecutable `a.exe`
Desde una ventana de DOS vamos a crear el ejecutable a.exe tecleando:

```
tcc a.obj m.obj<enter>
```

que debe generar el ejecutable `e.exe`

4.1.-Tamaño del archivo `a.exe:` _____

4.2.- Usando el comando `tdump a.exe | more` despliega información del archivo ejecutable. Menciona cuál es esa información:

Tamaño DOS del archivo: _____
Tamaño de la imagen de carga: _____
Conteo de entradas en la tabla de relocalización: _____
Tamaño del encabezado (en párrafos): _____
Requerimiento mínimo de memoria: _____
Requerimiento máximo de memoria: _____
Suma de verificación del archivo: _____
Número de overlay: _____
SS:SP inicial _____
CS:IP inicial _____
Ubicación de las relocalizaciones:
_____ _____ _____

4.3.- ¿Porqué crees que el tamaño del archivo ejecutable es mayor que la imagen de carga?
_____
_____

5.- Comentarios y conclusiones
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____