Programación de Sistemas

Yadier Betancourt Martínez

7.1)

Symbol	.symbtab entry?	Symbol type	Module where defined	Section
buf	Yes	extern	m.o	.data
bufp0	Yes	global	swap.o	.data
bufp1	Yes	global	swap.o	COMMON
swap	Yes	global	swap.o	.text
temp	No	____	____	____

- buf está definido en m.o y declarado como externo. Este símbolo se asignan a la sección .data(se almacenan variables globales inicializadas).
- bufp0, swap y bufp1 están definidos en swap.o. Todos están declarados como globales. bufp1 se asigna a la sección COMMON, mientras que swap y bufp0 se asignan a las secciones .text(almacena el código del programa) y .data, respectivamente.
- temp es una variable local por lo que no posee entrada en la tabla de símbolos.

7.2)
A.

```
// Módulo 1
int main()
{
}
// Módulo 2
int main;
int p2()
{
}
```

a) REF(main.1) → DEF(main.1)
b) REF(main.2) → DEF(main.1)

En este caso, el enlazador elige el símbolo fuerte definido en el Módulo 1 sobre el símbolo débil definido en el Módulo 2.

B.

```
// Módulo 1
void main()
{
}
// Módulo 2
int main = 1;
int p2()
{
}
```

a) error
b) error

En este caso, cada módulo define un símbolo fuerte, lo que resulta en un error de enlace.

C.

```
// Módulo 1
int x;
void main()
{
}
// Módulo 2
double x = 1.0;
int p2()
{
}
```

a) REF(x.1) → DEF(x.2)
b) REF(x.2) → DEF(x.2)

En este caso, el enlazador elige el símbolo fuerte definido en el Módulo 2 sobre el símbolo débil definido en el Módulo 1.

7.3)
A.

```
gcc p.o libx.a
```

En este caso, el archivo objeto p.o depende de la biblioteca estática libx.a.

B.

```
gcc p.o libx.a liby.a
```

En este caso, el archivo objeto p depende de la biblioteca estática libx.a, que a su vez depende de la biblioteca estática liby.a.

C.

```
gcc p.o libx.a liby.a libx.a
```

En este caso, el archivo objeto p.o depende de la biblioteca estática libx.a, que a su vez depende de la biblioteca estática liby.a. Además, la biblioteca estática liby.a depende de la biblioteca estática libx.a, que ya se ha incluido en la línea de comando. La primera vez que se incluye libx.a, se resuelven todas las referencias de símbolos que dependen de libx.a. Luego, se incluye liby.a para resolver las referencias de símbolos que dependen de liby.a. Finalmente, se incluye libx.a nuevamente para resolver las referencias de símbolos que dependen de libx.a y que no se resolvieron en la primera inclusión.

5.4)

A. En la versión menos optimizada del código, el registro `%xmm0` se utiliza como un valor temporal en cada iteración del bucle. Es decir, en cada iteración, se lee el valor de *dest*, se realiza la operación de multiplicación y se guarda el resultado en *dest* nuevamente. En cambio, en la versión optimizada, el registro `%xmm0` se utiliza para acumular el producto de los elementos del vector a lo largo de todas las iteraciones del bucle. Esto es similar a tener una variable "acumulador" que guarda el resultado parcial en cada paso, lo que permite evitar lecturas y escrituras innecesarias en *dest*.

B. La versión optimizada implementa correctamente el código C de *combine3*, incluso cuando hay aliasing de memoria entre *dest* y el vector de datos. El aliasing de memoria ocurre cuando dos punteros apuntan a la misma ubicación de memoria, lo que puede causar comportamientos inesperados en el programa. Sin embargo, en este caso, la optimización no afecta el comportamiento del programa, ya que el valor de *dest* se actualiza correctamente en cada iteración del bucle.

C. La optimización preserva el comportamiento deseado del programa porque, con la excepción de la primera iteración, el valor leído de *dest* al comienzo de cada iteración será el mismo valor que se escribió en este registro al final de la iteración anterior. Por lo tanto, la instrucción de combinación puede simplemente utilizar el valor ya presente en `%xmm0` al comienzo del bucle. Esto evita la necesidad de leer y escribir en *dest* en cada iteración, lo que mejora el rendimiento del programa.

5.5)

A. La función realiza *2n* multiplicaciones y *n* adiciones.

B. Podemos ver que el cálculo que limita el rendimiento aquí es la repetición de la expresión `xpwr = x * xpwr`. Esto requiere una multiplicación de punto flotante (5 ciclos de reloj), y el cálculo para una iteración no puede comenzar hasta que se haya completado el de la iteración anterior. La actualización de `result` solo requiere una adición de punto flotante (3 ciclos de reloj) entre iteraciones sucesivas.