Instituto Tecnológico de Costa Rica

Introducción a los gráficos por computadora

Profesor: Francisco Torres Rojas

Estudiante: Steven Josué Retana Cedeño

2017144537

Tarea Corta 2

"Capítulo 1 - Foley (Ejercicios)"

I Semestre 2021

20 de Febrero, 2021

1 Problemas - "The assembly language level"

1) Para un programa determinado, el 2% del código representa el 50% del tiempo de ejecución. Compare las siguientes 3 estrategias respecto al tiempo de programación y al tiempo de ejecución. Suponga que se necesitarían 100 meses-hombre para escribirlo

a. Programa entero en C.
b. Programa entero en ensamblador.
b. Primero todo en C, luego el 2% del código clave reescrito en ensamblador

En la primera estrategia, escribir el progama completo en C, se puede tener un mucho mejor tiempo de programación debido a las facilidades que ofrece un lenguaje de alto nivel y las soluciones tan amplias existentes, pero su desventaja en este caso es que puede que algunas partes del código se vuelvan lentas y en especial afectar a ese 2% del código que representa al 50% de su ejecución por lo que podría traernos problemas en tiempo y más esa fracción de la mitad del tiempo total.

Utilizando el segundo enfoque, se podrían tener mejores tiempos de ejecución debido a la optimización y el nivel en que se puede escribir en ensamblador, pero esto trae una contaparte muy grande y es sacrificar de una gran manera el tiempo de programación que se le debe dedicar al programa, esta estrategia para este ejercicio, desde mi punto de vista, no debería ser viable debido a que sería perder mucho tiempo de programación para tener una ganancia en ese 2% del código mencionado, lo cual creo que no merece la pena para este caso

Finalmente escribiendo ese 2% en ensamblador y el resto en C, siento que se podría tener una gran ganancia en los 2 aspectos, tanto tiempo de programación como tiempo de ejecución, al hacer sólo esta pequeña parte

en ensamblador y optimizar la mitad del código del problema, no se pierde mucho tiempo programando y se obtiene un buen margen de ganancia en cuanto a este ejercicio.

6) ¿Cuál es la diferencia entre una instrucción y una pseudoinstrucción

En el ensamblador, las instrucciones se refieren a las instrucciones directas referente a la máquina como tal y su manipulación, como en el caso de las instrucciones MOV, ADD, entre otras, las cuales se interpretan directamente para ser parte del lenguaje objeto generado y no del lenguaje ensamblador como tal. Las pseudoinstrucciones son instrucciones para el ensamblador mismo, es decir, aquellas que se usan dentro del propio ensamblador para realizar procesos dentro del ensamblaje del código fuente, estas son representadas por palabras como SEGMENT, ALIGN, END, EQU, y muchas otras para manejas instrucciones a nivel del ensamblador mismo y no el lenguaje máquina como en el caso de las anteriores, también son llamadas "directivas de ensamblador."

16) Los programas a menudo enlazan múltiples DLL. ¿No sería más eficiente poner todos los procedimientos en una gran DLL y luego vincularla?

Considero que no sería eficiente debido a que se tendrían muchos procedimientos sin utilizar a pesar de ser un gran DLL y aparte existiría mucho consumo de recursos por múltiples aplicaciones al intentar usar esos procesos, esto debido a que cuando un programa necesita un procedimiento en específico busca ciertas DLL, las enlaza y utiliza y luego se desliga de ellas en el momento que no las utilize, cada vez que la DLL se enlaza ocupa cierto espacio de memoria y disco, por lo que si hay una gran librería general

para múltiples programas entonces existirían una gran cantidad de procesos "abiertos" por todos los programas que necesitan acceso a ella en lugar de cada proceso acceder sólo a lo necesario y de esta manera ganar eficiencia reduciendo espacio en disco y memoria.

17) ¿Se puede asignar una DLL a los espacios de direcciones virtuales de 2 procesos en diferentes direcciones virtuales? Si es así, ¿Qué problemas surgen? ¿Pueden resolverse? Si no, ¿qué se puede hacer para eliminarlos?

Según el capítulo, a través del linking implícito se guarda alguna información del DLL en un archivo llamado "import library", esta es vinculada al programa en ejecución. Mediante eso también, son mapeados esos linkings a los espacios de direcciones virtuales. Si se tienen espacios de 2 procesos, se mapean varias veces estos procesos según los procesos virtuales que se requieren lo que puede llevar a un problema de eficiencia en los recursos al tener procesos duplicados.

22) Un linker lee 5 módulos, cuyas longitudes son 200,800,600,500 y 700 palabras respectivamente. Si se cargan en ese orden, ¿cuáles son las constantes de reubicación?

De acuerdo al capítulo, la constante de reubicación equivale a la dirección de inicio del módulo, por lo que para el caso de estos de 5 módulos, si iniciaran a partir de la dirección 100 por ejemplo, se tendrían las siguientes constantes de reubicación:

Módulo	Longitud	Dirección de inicio
A	200	100
B	800	300
C	600	1100
D	500	1600
E	700	1700

Cada una de ellas de las constantes corresponden a las direcciones de inicio suponiendo que se empieza en 100, por lo que en cada módulo se le suma a la dirección de inicio, la cantidad de palabras leídas para dar como resultado las constantes de reubicación.