

Recolección de Basura en C

Edinson Sánchez

Kevin Filella

Adrian Aguilar

5 de febrero de 2014

Índice

1. Introducción	3
2. Métodos	3
2.1. Boehm-Demers-Weiser Garbage Collector	3
2.2. Metodo 2	3
2.3. Metodo 3	4
3. Ventajas y desventajas	4
3.1. Performance	4
3.2. Desarrollo	4
3.3. Confiabilidad	4
3.4. Debugging	4
4. Conclusiones	4

1. Introducción

En lenguajes de programación, un recolector de basura tiene como objetivo el de gestionar la memoria de un programa informático. La memoria debe ser gestionada de tal forma que se puedan reservar espacios en memoria para su uso, se puedan liberar espacios en memoria anteriormente reservados, se puedan compactar espacios de memoria libres y consecutivos, y se pueda llevar cuenta de los espacios libres y utilizados en la memoria.

Como es de conocimiento para muchos, el lenguaje C no incorpora un método de gestión de memoria automático, como lo hacen lenguajes de programación como Java, C Sharp, entre otros. Esto brinda a los programadores un set de ventajas y desventajas a la hora de programar.

En este documento, discutiremos sobre varios métodos populares que han sido creados, en forma de librerías, para brindarle al lenguaje C esta muy importante característica. Asimismo, discutiremos sobre las ventajas y desventajas de tener un recolector de basura automático y gestionar la memoria manualmente.

2. Métodos

2.1. Boehm-Demers-Weiser Garbage Collector

El método de recolección de basura de la librería Boehm GC consiste en dos fases; primero, se realiza un escaneo de toda la memoria activa para marcar bloques en desuso; después, la fase de barrido se hace cargo de mover todos los bloques marcados (de la primera fase) a la lista de bloques libres. Estas dos fases pueden ser, y usualmente son, ejecutadas aparte para mejorar el tiempo de respuesta de la librería.

El algoritmo del Boehm GC es generacional, ya que se enfoca en buscar memoria libre en los bloques más nuevos. Esto se basa en la idea de que los bloques más viejos tienen mayor tiempo de uso; por consecuencia, los bloques más recientes tienen un tiempo de uso reducido y deben ser liberados con mayor frecuencia. Este algoritmo también es conservativo, ya que también se debe encargarse de asumir cuales variables son punteros a datos dinámicos y cuales simplemente se ven de esa forma.

La librería Boehm GC viene en forma estática o dinámica, es fácil de instalar, y su uso es permitido en los lenguajes C y C++. Para utilizar la librería en programas en C que ya utilizan malloc(), calloc(), realloc(), o free(), simplemente se deben redefinir las funciones como se ve a continuación:

```
#define calloc(n,x) GC_malloc((n)*(x))
#define realloc(p,x) GC_realloc((p),(x))
#define free(x) (x) = NULL
```

2.2. Metodo 2

explicar metodo 2 aqui

2.3. Metodo 3

explicar metodo 3 aqui

3. Ventajas y desventajas

Antes de entrar en detalle acerca de las ventajas y desventajas de implementar un algoritmo de recolección de basura en C, es necesario entender que no todo programa en C se comporta de igual manera. Muchas de las ventajas y desventajas detalladas a continuación pueden o no cumplirse para cierto programa. Hay varios factores que influyen en esto: la técnica de gestión manual de memoria (malloc/free), la estructura de datos, la complejidad del programa, entre otros. Para los siguientes puntos, se tomara en cuenta el uso de la librería más común para recolección de basura en C, la librería Boehm-Demers-Weiser Garbage Collector o Boehm GC.

3.1. Performance

Los efectos de un garbage collector, como el Boehm GC, en un programa en C pueden traer efectos variados en cuanto a la performance. Para programas pequeños que utilizan poca memoria, la implementación de un GC es probablemente innecesaria, ya que el GC requiere de un cierto espacio libre en el heap para no tener que recoger basura después de cada asignación en memoria. Ya que algunos garbage collectors expanden el heap en espacios relativamente grandes, es probable que para estos programas, parte de esa memoria nunca sea utilizada. Esto no es del todo cierto para programas con estructuras más grandes y complejas; ya que se requiere de más memoria en el heap para las asignaciones, la memoria separada por el garbage collector será utilizada más frecuente y eficientemente.

3.2. Desarrollo

Es clara la ventaja que brinda la implementación de un garbage collector en un programa en C. Ya que C no incorpora un recolector por default, es necesario manejar la memoria manualmente. Esto significa que el tiempo que tome gestionar la memoria manualmente será proporcional al tamaño y complejidad del programa. Se estima que un 30 % hasta 40 % del esfuerzo en el desarrollo de un programa extenso se enfoca solo en la gestión de memoria. Al utilizar un garbage collector, se reduce drásticamente el tiempo de desarrollo.

3.3. Confiabilidad

3.4. Debugging

4. Conclusiones

conclusiones aqui