

# Una aproximació al recollidor de memòria de la plataforma Mono (v1.1.15)

Aleix Badia i Bosch

juliol del 2006

## Resum

Arran de les problemàtiques d'integritat, seguretat, rendiment i complexitat associades a la gestió manual de la memòria, sorgeix l'interès per les tècniques de gestió automàtica i específicament pels recollidors de memòria.

L'objectiu de la xerrada és donar a conèixer les tècniques tradicionals utilitzades pels recollidors de memòria, realitzar una introducció bàsica a la implementació del recollidor de memòria Boehm-Demers-Weiser i descriure'n la utilització en el projecte Mono.

## 1 Introducció

La necessitat d'una gestió automàtica de la memòria per part dels entorns d'execució, sorgeix com a solució a les problemàtiques associades a la gestió manual:

- Space leak Desreferenciació d'objectes sense el previ alliberament de la memòria assignada.



Exemple:

```
#include <iostream>
class ObjBasic
{
private:
    ObjBasic *_next;
public:
    ObjBasic() { }
    ~ObjBasic() { }
```

```

        void next(ObjBasic *pitem) { _next = pitem; }
        ObjBasic *next() { return _next; }
};

int main(int argc, char *argv[])
{
    ObjBasic *a = new ObjBasic();
    ObjBasic *b = new ObjBasic();
    ObjBasic *c = new ObjBasic();

    a->next(b);
    b->next(c);

    ObjBasic *head = a;
    b->next((ObjBasic *)0);
    ObjBasic *it = head;
    while(it) {
        ObjBasic *current = it;
        it = it->next();
        delete(current);
    }

    return 0;
}

```

Tal i com es mostra en els resultats de l'anàlisi de l'ús de memòria <sup>1</sup>

```

ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 19 from 1)
malloc/free: in use at exit: 4 bytes in 1 blocks.
malloc/free: 3 allocs, 2 frees, 12 bytes allocated.
LEAK SUMMARY:
definitely lost: 4 bytes in 1 blocks.
possibly lost: 0 bytes in 0 blocks.
still reachable: 0 bytes in 0 blocks.
suppressed: 0 bytes in 0 blocks.

```

- Dangling pointer Referència a objectes prèviament alliberats.



Exemple:

```

#include <iostream>
class ObjBasic
{
private:
public:
    ObjBasic *_next;

    ObjBasic() { }
    ~ObjBasic() { }
}

```

<sup>1</sup>Anàlisi de l'ús de memòria via Valgrind <http://www.valgrind.org>

```

        void next(ObjBasic *pitem) { _next = pitem; }
        ObjBasic *next() { return _next; }
};

int main(int argc, char *argv[])
{
    ObjBasic *a = new ObjBasic();
    ObjBasic *b = new ObjBasic();
    ObjBasic *c = new ObjBasic();

    a->next(b);
    b->next(c);

    ObjBasic *head = a;

    delete(b);
    ObjBasic *it = head;
    while(it) {
        ObjBasic *current = it;
        it = it->next();
    }
    delete(a);
    delete(c);

    return 0;
}

```

Tal i com es mostra en els resultats de l'anàlisi de l'ús de memòria <sup>2</sup>

Electric Fence 2.1 Copyright (C) 1987-1998 Bruce Perens.

```

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread -1211120032 (LWP 3827)]
0x080488de in ObjBasic::next (this=0xb7bffffc) at ObjBasic.cpp:13
13      ObjBasic *next() { return _next; }

```

```

ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 19 from 1)
malloc/free: in use at exit: 0 bytes in 0 blocks.
malloc/free: 3 allocs, 3 frees, 12 bytes allocated.
For counts of detected errors, rerun with: -v

```

En el cas d'una implementació del mateix algorisme en un entorn amb gestió automàtica de memòria no es donen les problemàtiques descrites anteriorment:

```

namespace ObjBasicNamespace
{
    public class ObjBasic
    {
        private ObjBasic _next;
        public ObjBasic() {}
    }
}

```

---

<sup>2</sup>Anàlisi de l'ús de memòria via Electric Fence <http://perens.com/FreeSoftware/> i Valgrind <http://www.valgrind.org>

```

        public ObjBasic next {
            get { return _next; }
            set { _next = value; }
        }
    }

    public class ObjBasicProgram
    {
        static void Main(string[] args)
        {
            ObjBasic a = new ObjBasic();
            ObjBasic b = new ObjBasic();
            ObjBasic c = new ObjBasic();

            a.next = b;
            b.next = c;

            ObjBasic head = a;
            b.next = null;
            ObjBasic it = head;
            while(it != null) {
                it = it.next;
            }
        }
    }
}

```

ERROR SUMMARY: 1489 errors from 51 contexts (suppressed: 39 from 2)  
 malloc/free: in use at exit: 516,533 bytes in 726 blocks.  
 malloc/free: 5,123 allocs, 4,397 frees, 2,525,817 bytes allocated.  
 LEAK SUMMARY:  
 definitely lost: 2,129 bytes in 249 blocks.  
 indirectly lost: 120 bytes in 10 blocks.  
 possibly lost: 72 bytes in 1 blocks.  
 still reachable: 514,212 bytes in 466 blocks.  
 suppressed: 0 bytes in 0 blocks.

*NOTA: Els resultats de l'anàlisi d'ús de memòria de l'exemple anterior amb gestió automàtica de memòria reconeix errors no directament associats a la gestió de memòria per part del programa implementat*

En aquest document es realitzarà una primera introducció als diferents algorismes de recollida de memòria i les tècniques addicionals associades. Seguidament es descriurà superficialment el funcionament intern del recollidor de memòria Boehm-Demers-Weiser i la corresponent interfície de programació. Finalment s'introduirà l'ús que fa del recollidor el projecte Mono i el conjunt de classes i eines disponibles per a la seva gestió i anàlisi.

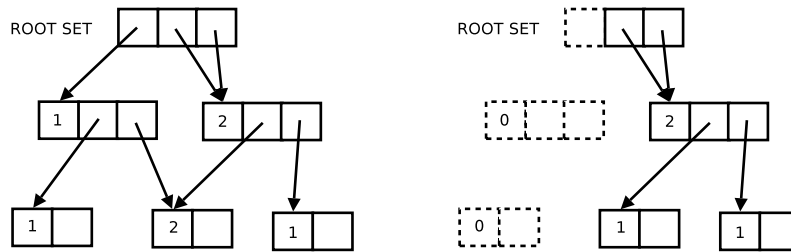
## 2 Recollida de memòria

La tècnica de recollida s'utilitza en la gestió automàtica de la memòria, i es basa en diferents algorismes cerca i alliberament la memòria dels objectes no accessibles [4] [5].

## 2.1 Algorismes de recoll·lecció de memòria

- Reference counting. El procés de recoll·lecció es basa en realitzar el comptatge de les referències a un objecte mitjançant un comptador associat. En instanciar una classe, el comptador de l'objecte a memòria s'inicialitza a 1. Cada vegada que un objecte hi fa referència, el comptador s'incrementa en una unitat; en cas de suprimir-se la referència, el comptador es decremента una unitat.

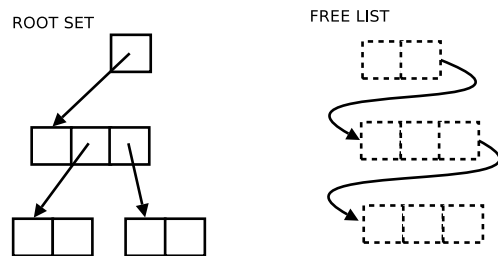
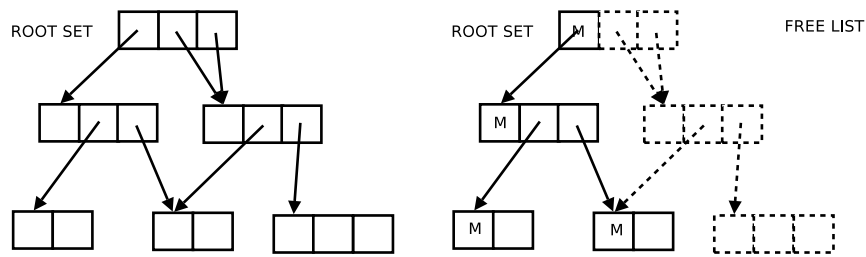
Si el valor del comptador de referències és 0, indicarà que cap altre objecte hi fa referència i que esdevindrà inaccessible; a partir d'aquí el recoll·lector de memòria podria alliberar la memòria assignada a l'objecte. En produir-se l'alliberament de l'objecte, es decrementarien els comptadors associats als objectes als quals fa referència, desencadenant un alliberament en cascada.



- Mark-sweep. El procés de recoll·lecció es divideix en dues etapes:
  1. Marking. Recorre i marca la totalitat dels objectes accessibles a partir de les referències dels objectes del rootset.<sup>3</sup>
  2. Sweep. Recorre la totalitat dels objectes del heap cercant els objectes que no han estat marcats en l'etapa anterior, objectes no accessibles que s'afegeixen a la llista de fragments de memòria disponibles.

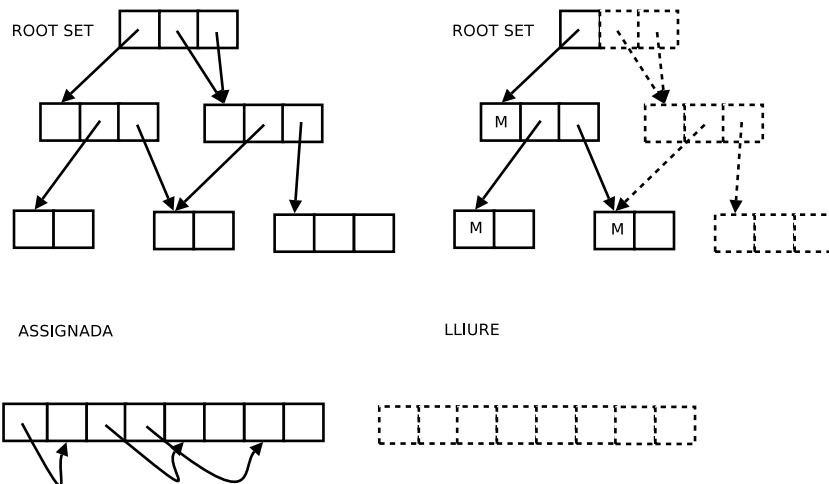
---

<sup>3</sup>Conjunt d'elements origen en la recerca de referències a objectes accessibles

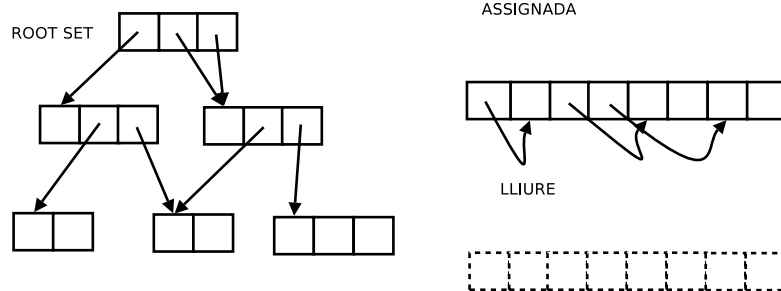


- Mark-compact. El procés de recollecció comparteix la primera etapa de l'algorisme mark-sweep.

En la segona etapa, l'algorisme mou tots els fragments de memòria dels objectes referenciats actualitzant les referències des d'altres objectes, generant una zona compacta d'objectes adjacents i accessibles, i una zona compacta de memòria lliure.



- Copying. El procés de recollecció es realitza en una única etapa. Es recorren la totalitat dels objectes a partir del rootset, movent tots els fragments de memòria dels objectes referenciats i actualitzant les referències des d'altres objectes, generant una zona compacta d'objectes adjacents i accessibles, deixant la resta de la memòria com una zona de memòria lliure compacta.



## 2.2 Tècniques addicionals

L'execució d'alguns dels algorismes anteriors poden estar condicionats pels factors següents:

1. Interacció amb el programa en execució
  - Stop-the-world. El procés de recol·lecció de memòria atura l'execució del programa inhabilitant la possibilitat que s'assignin objectes, s'alliberin objectes o s'en modifiquin referències
  - Incremental. El procés de recol·lecció de memòria s'intercala amb intervals d'execució del programa, obligant a la sincronització entre el cicle de recol·lecció i el d'execució del programa
2. Gestió de les referències dels objectes
  - Precís. El recol·lector pot identificar la totalitat de les referències a d'altres objectes
  - Conservatiu. El recol·lector no pot identificar les referències i ha de d'analitzar la totalitat de la memòria cercant patrons de referències
3. Evolució temporal de la recol·lecció
  - Generacional. Es mantenen diferents agrupacions d'objectes anomenades "generacions" en funció de la proximitat temporal en l'assignació. Els nous objectes s'assignen en les generacions recents i en una primera instància es promocionen en la recol·lecció. En l'evolució temporal, els objectes es promocionen a agrupacions d'objectes de menys proximitat temporal, podent-se aplicar diferents criteris de recol·lecció en cada una de les agrupacions.

## 3 El recol·lector de memòria Boehm-Demers-Weiser

El recol·lector de memòria conservatiu Boehm-Demers-Weiser es basa en l'algorisme Mark-Sweep [3]. La gestió dels diferents objectes assignats pel

recol·lector està condicionada pe tipus d'objecte:

- NORMAL L'objecte pot contenir referències a d'altres objectes
- ATOMIC L'objecte no conté referències a d'altres objectes
- TYPED El fragment de memòria conté una descripció del tipus de l'objecte, on es defineixen les referències a d'altres objectes

### 3.1 Assignació

El recol·lector de memòria realitza la peticions de memòria del sistema a través de les funcions malloc, sbrk, mmap.

L'assignació de memòria a un objecte estarà condicionada per la quantitat de memòria sol·licitada. En cas de tractar-se d'un objecte de mida superior a  $HBLKSIZE/2$ , on  $HBLKSIZE$  correspon a la mida del blocs utilitzats pel recol·lector, es produeix un arrodoniment al següent  $HBLKSIZE$  i una sol·licitud d'assignació del bloc corresponent. En el cas de d'un objecte de mida inferior a  $HBLKSIZE/2$  es farà una aproximació a partir d'una taula predefinida de mides i a continuació es cercarà a través de la llista d'objectes alliberats d'aquell tipus cercant fragments prèviament allibertats; en cas de no trobar-se s'iniciarà la sol·licitud d'assignació del bloc corresponent.

En cas que la quantitat de memòria total sol·licitada des de l'última recol·lecció sigui inferior al heap dividit per una variable modificable en funció de l'espai i el temps de recol·lecció, s'intentarà expandir el heap, en cas contrari s'iniciarà la recol·lecció.

### 3.2 Mark

El marcatge dels objectes accessibles s'iniciarà a partir del rootset definit per:

- Registres
- Stack
- Fragments de memòria estàtic

El marcadore disposa d'una stack amb les referències i descripció dels fragments de memòria accessibles però pendents d'anàlisi. En la inicialització de l'stack s'inclouran els objectes continguts en el rootset, serà a partir de l'anàlisi d'aquests elements que s'evolucionarà a l'anàlisi de tots els objectes accessibles. El marcadore disposa d'una stack amb les referències i descripció dels fragments de memòria accessibles però pendents d'anàlisi. En la inicialització de l'stack s'inclouran els objectes continguts en el rootset, serà a partir de l'anàlisi d'aquests elements que s'evolucionarà a l'anàlisi de tots els objectes accessibles.

En funció de la interacció amb el programa en execució, el marcatge es durà a terme processant la totalitat dels elements i referències (stop-the-world) o de manera incremental processant porcions (incremental).

En el cas dels elements de l'stack amb tipus específic s'analitzaran a partir de la descripció continguda en el propi fragment de memòria.

En el cas dels elements de l'stack sense tipus específic s'haurà d'analitzar la totalitat del fragment cercant patrons de referència a d'altres objectes. L'anàlisi dels fragments es farà a partir dels criteris següents:

- Que estigui en els límits que defineixen el heap
- En el cas de fragments  $< \text{HBLKSIZE}/2$  es comprova l'adreça a la qual es fa referència a la taula de punters dels blocs  $\text{HBLKSIZE}$ , els quals poden disposar de descriptors del tipus d'objectes que contenen; en el cas  $> \text{HBLKSIZE}/2$  es cerca la referència a l'inici del fragments. En ambdós casos es facilita la identificació de l'objecte.

### 3.3 Sweep

Un cop finalitzat el marcatge dels objectes de l'stack, els objectes  $> \text{HBLKSIZE}/2$  es retornen a la llista d'objectes alliberats. En el cas dels objectes  $< \text{HBLKSIZE}/2$  es comprova si tots els objectes que conté estan alliberats i es pot retornar tot el bloc a la llista d'alliberats o si estan referenciats i no es poden alliberar. Aquests blocs es podran alliberar quan en una assignació es trobi una llista lliure per a un objecte d'aquell tipus i mida.

### 3.4 Finalization

Un cop finalitzada l'etapa mark, el recol·lector recorre la llista d'objectes marcats com a finalitzables i no accessibles. Situa cada un dels objectes a l'stack de mark marcant tots els objectes als quals referencia per evitar que s'alliberen o finalitzin abans d'executar-se el procés de finalització del pare.

Qualsevol objecte que no hagi quedat pel procediment anterior s'afegirà a la cua d'objectes dels quals s'executarà la finalització.

## 4 Interfície C

Les funcions més utilitzades de la interfície C del recol·lector de memòria Boehm-Demers-Weiser

```
void * GC_MALLOC(size_t nbytes) // assigna nbytes de memòria
// i els allibera quan l'objecte associat ja no està referenciat.

void * GC_MALLOC_ATOMIC(size_t nbytes) // assigna nbytes de memòria
// i els allibera quan l'objecte associat ja no està referenciat.

void * GC_GCJ_MALLOC(size_t lb, void * ptr_to_descr)
```

```

// assigna nbytes de memòria.
void GC_INIT(void) // Inicialitza el recollidor de memòria
void GC_collect(void) // Força un cicle de recollició

```

## 5 El recollidor de memòria Boehm-Demers-Weiser i Mono

La implementació del conjunt d'eines compatibles amb la plataforma .NET, segons especificacions de ECMA, que realitza el projecte Mono, utilitza el recollidor de memòria Boehm-Demers-Weiser de manera predeterminada:

```

using System;
namespace ObjBasicNamespace
{
    public class ObjBasic
    {
        private int _attribute;

        public ObjBasic(int pAttribute) {
            _attribute = pAttribute;
        }
        public int attribute {
            get { return _attribute; }
            set { _attribute = value; }
        }
    }

    public class ObjBasicProgram
    {
        static void Main(string[] args)
        {
            for (int i=0;i<10000;i++) {
                ObjBasic o = new ObjBasic(i);
                System.Console.WriteLine(o.attribute);
            }
        }
    }
}

```

El codi ensamblador de la instància de la classe ObjBasic és

```

...
IL_0007: ldloc.0
IL_0008: newobj instance void class
ObjBasicNamespace.ObjBasic::.ctor(int32)
IL_000d: stloc.1
IL_000e: ldloc.1
IL_000f: callvirt instance int32 class
ObjBasicNamespace.ObjBasic::get_attribute()

```

```

IL_0014:  call void class
[mscorlib]System.Console::WriteLine(int32)
IL_0019:  ldloc.0
IL_001a:  ldc.i4.1
IL_001b:  add
IL_001c:  stloc.0
IL_001d:  ldloc.0
IL_001e:  ldc.i4 10000
IL_0023:  blt IL_0007
...

```

Segons les especificacions de l'estàndard ECMA-335 [2], la implementació de la instrucció `newobj` hauria de:

The `newobj` instruction allocates a new instance of the class associated with `ctor` and initializes all the fields in the new instance to 0 (of the proper type) or null as appropriate. It then calls the constructor with the given arguments along with the newly created instance. After the constructor has been called, the now initialized object reference is pushed on the stack.

En el cas de la plataforma Mono, la implementació de la instrucció analitzarà el tipus d'objecte en funció de les referències que contingui a d'altres objectes tal i com s'ha comentat anteriorment, i assignarà memòria a través de les interfícies en C del recollidor de memòria Boehm-Demers-Weiser.

## 5.1 Paràmetres d'execució del recollidor de memòria

El recollidor de memòria Boehm-Demers-Weiser permet definir paràmetres d'execució mitjançant variables d'entorn, alguns d'aquests paràmetres són:

- `GC_INITIAL_HEAP_SIZE=<n>` Mida inicial del heap
- `GC_MAXIMUM_HEAP_SIZE=<n>` Mida màxima del heap recollit
- `GC_PRINT_STATS` Habilita l'enregistrament de les tasques
- `GC_PRINT_ADDRESS_MAP` Volca `/proc/self/maps`
- `GC_NPROCS=<n>` Nombre de processadors que pot utilitzar
- `GC_MARKERS=<n>` Nombre de fils que pot utilitzar en l'etapa de marcat

- GC\_IGNORE\_GCJ\_INFO Ignora els descriptors del tipus d'objecte (traçat conservatiu)
- GC\_ENABLE\_INCREMENTAL Habilita la recoll·lecció incremental
- GC\_PAUSE\_TIME\_TARGET Especifica l'interval de pausa en recoll·lecció incremental
- GC\_FIND\_LEAK Habilita la detecció de leaks de memòria
- GC\_DONT\_GC Inhabilita la recoll·lecció de memòria

## 5.2 GC class

La classe System.GC de Mono [1] permet gestionar mecanismes del recoll·lector de memòria de l'entorn d'execució.

```

void Collect ()
// Força un cicle de recoll·lecció
int GetGeneration ()
// No implementat en el GC Boehm–Demers–Weiser
long GetTotalMemory ()
// Pot forçar un cicle de recoll·lecció i retorna la memòria
//ocupada
void KeepAlive(object obj)
// Crea una referència a l'objecte
void ReRegisterForFinalize(object obj)
// Afegeix l'objecte a la llista d'objectes que precisen de
//finalizer
void SuppressFinalize(object obj)
// Indica al GC que no ha d'executar el mètode Object.Finalize de
//l'objecte
void WaitForPendingFinalizers ()
// Atura temporalment l'execució del fil actual fins a la
//finalització del conjunt d'objectes pendents de finalització

```

## 5.3 Heap-buddy profiler

Mono disposa de diferents punts de seguiment en l'entorn d'execució que permeten desenvolupar aplicacions de traçat i anàlisi de l'execució de Mono.

El profiler heap-buddy analitza:

- Cicles de recoll·lecció i redimensionament del heap

```

SUMMARY
Filename: outfile
Allocated Bytes: 1,5M
Allocated Objects: 51880
GCs: 14
Resizes: 5

```

Final heap size: 468k

Distinct Types: 60

Backtraces: 380

- Nombre i tipus d'objectes instanciats i quantitat de memòria consumida

Type	Num	Total	AvSz	AvAge	BT
ObjBasicNamespace.ObjBasic	10000	117k	12,0	0,0	1

- Traça de les assignacions de memòria

## 6 Conclusions

A través d'aquest document s'ha intentat realitzar una primera aproximació al recol·lector de memòria utilitzat actualment pel projecte Mono i el conjunt d'eines disponibles per definir i analitzar el seu comportament.

## Referències

- [1] Mono projecte documentation. <http://www.go-mono.com/docs/>.
- [2] Standard ecma-335 common language infrastructure(cli). <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [3] Hans Boehm. Conservative gc algorithmic overview. [http://www.hpl.hp.com/personal/Hans\\_Boehm/gc/gcdescr.html](http://www.hpl.hp.com/personal/Hans_Boehm/gc/gcdescr.html).
- [4] Rafael Lins Richard Jones. *Garbage Collection: algorithms for automatic dynamic memory management*. John Wiley & Sons, 1996.
- [5] Paul R. Wilson. Uniprocessor garbage collection techniques. In *Proc. Int. Workshop on Memory Management*, number 637, Saint-Malo (France), 1992. Springer-Verlag.