Recollida de memòria brossa

Hash autor: 80608

Facultat d'Informàtica de Barcelona - UPC
Llenguatges de programació

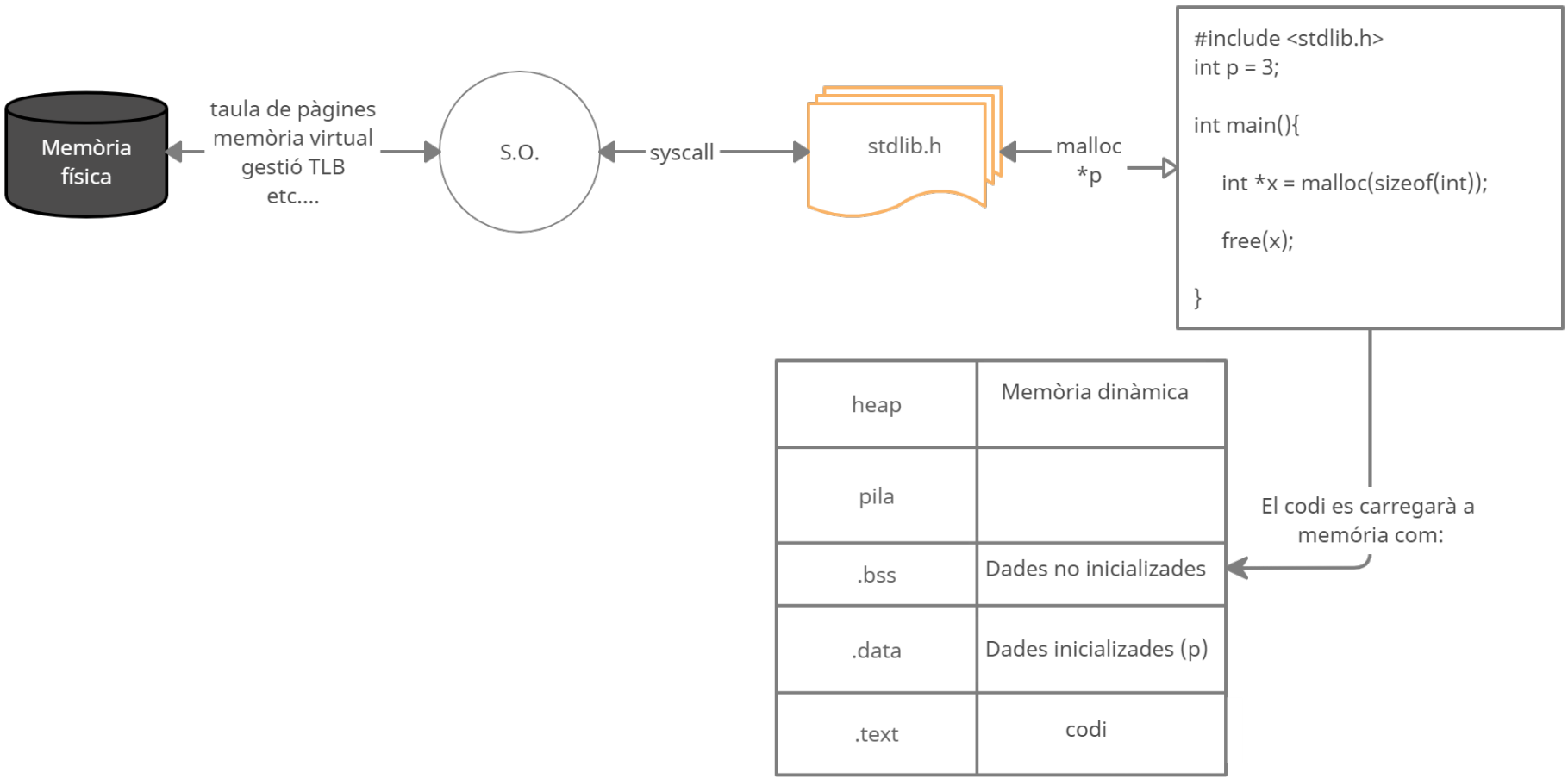
Quadrimestre de tardor, 2020

Índex

Introducció als recolector de memòria brossa

- Els *garbage collectors* són un mecanisme de gestió automàtica de la memòria dinàmica d'un programa.
- En llenguatges sense GC, és tasca del programador gestionar les reserves de memòria dinàmica.

Introducció als recolector de memòria brossa



Problemes en la gestió de la memòria

1. Accés invàlid a memòria
2. Fuges de memòria
3. Reserva / alliberament incompatible
4. Alliberar regions lliures
5. Regions no inicializades
6. Accés a direccions incorrectes

Problemes en la gestió de la memòria

Accés invàlid a memòria

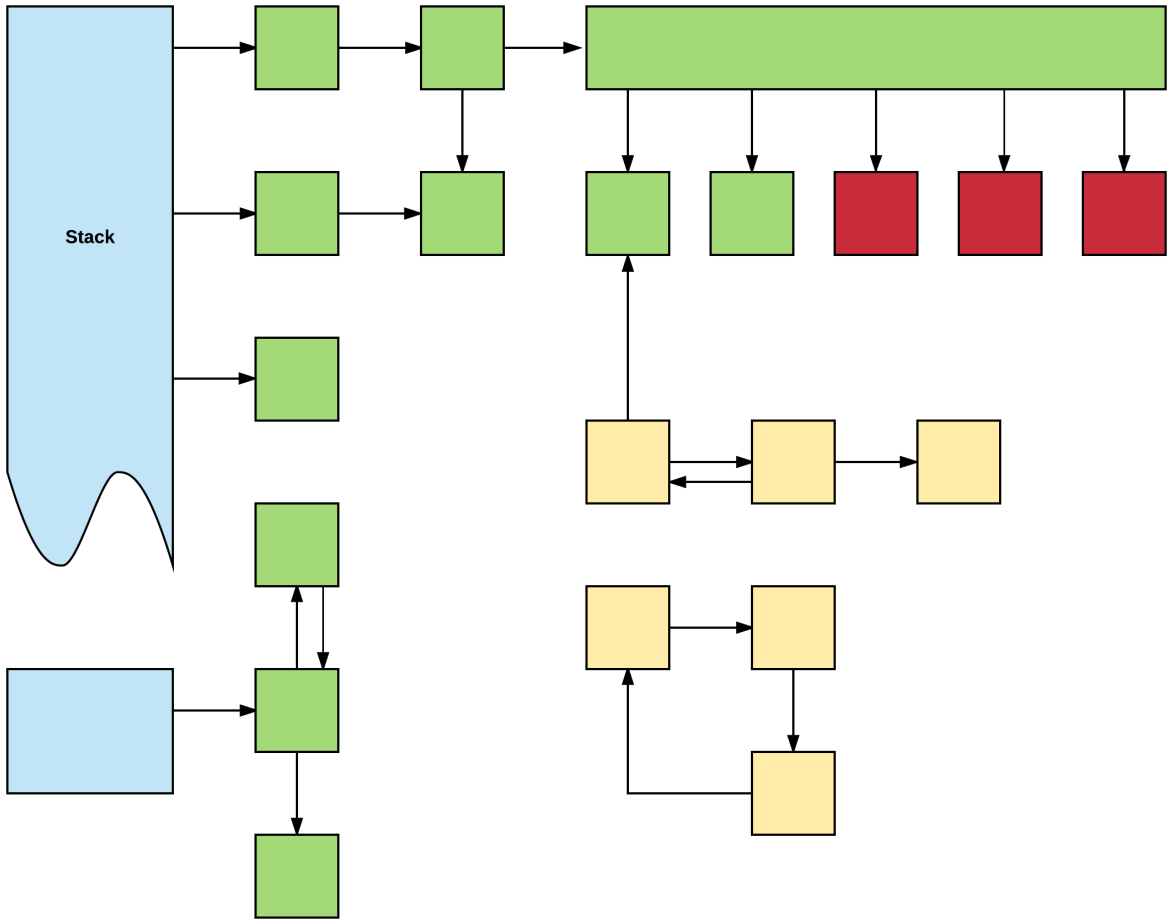
```
struct alfa{ int d1; int d2;};
char *cP =  (char*) malloc(sizeof(char));
free(cP);
strcpy(cP, "helloWorld");
```

Regions no inicializades

```
struct alfa{ int d1; int d2;};
struct alfa * aP = (struct alfa *) malloc(sizeof(struct alfa));
printf("@%x, valor: %i\n",&aP, *aP);//@61fe10 valor: 7213648
int resultat = aP->d1 + 24;
printf("resultat %i \n", resultat); //resultat 7213648
free(aP);
```

Principis dels *garbage collector*

Els recol·lector de memòria brossa són típics dels llenguatges O.O.



Ús del GC a Java

A Java, les variables són apuntadors als objectes. Els objectes s'instancien amb la paraula reservada *item*.

```
import java.util.*;

public class myClass{
    int atb1, atb2;

    myClass(int a, int b){
        atb1 = a;
        atb2 = b;
    }
    int add(){return atb1+atb2;}

    public static void main(String []args){
        myClass instance = new myClass(1,2);
        myClass arrayOBJ[] = new myClass[2];
        List<myClass> l = new ArrayList<myClass>();
        System.out.println(instance.add());
        int a = arrayOBJ[1].add();  //java.lang.NullPointerException
        instance = null;
        System.out.println(instance.add());//java.lang.NullPointerException
    }
}
```

Ús del GC a Python

A Python, tot està representat per objectes, es poden eliminar del programa amb la crida a *del*.

```
>>> x = 23
>>> type(x)
<class int>
>>> hex(id(x))
'0x7ffc7c571960'
>>> del x
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Algoritmes de *garbage collection*

Tots els algoritmes per fer recol·lecció de memòria brossa tenen el següent esquema:

- Marcar objectes
- Eliminar objectes
- Compactar els objectes restants (si s'escau)

Existeixen diverses variacions d'aquest esquema, per diferents casos aplicacions.

Garbage collection a Java

Java té diverses implementacions, totes divideixen el *heap* en generacions. Tots els objectes tenen assignat una "edad".

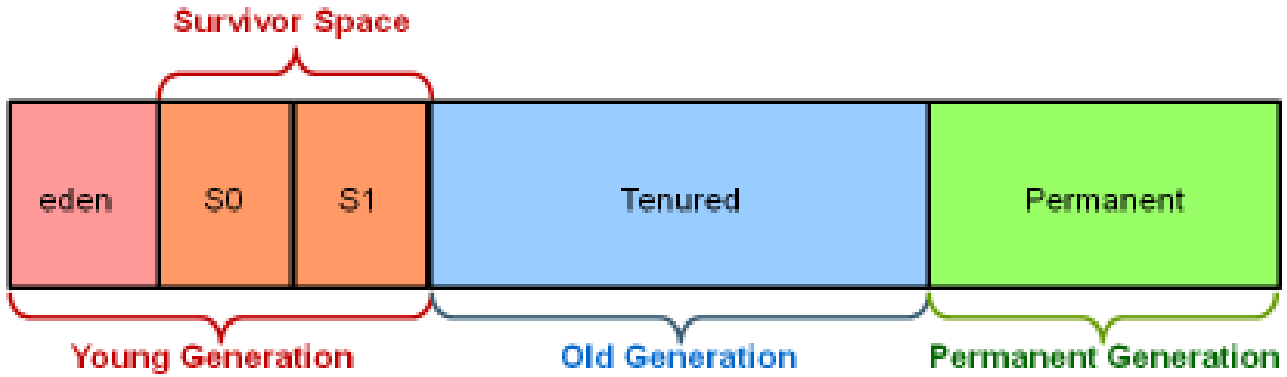


Figura: Imatge extreta de Java Garbage collection basics[6]

- *Eden*: Aquí s'assignen els nous objectes. Edad $= 1$.
- *Survivor space*: Objectes no eliminats. Edad $+= 1$.
- *Old generation*: Objectes supervivents. Edad $>$ Llindar.
- Permanent: Objectes permanents de la JVM

Garbage collection a Java

Java té diverses implementacions, totes divideixen el *heap* en generacions. Tots els objectes tenen assignat una "edad".

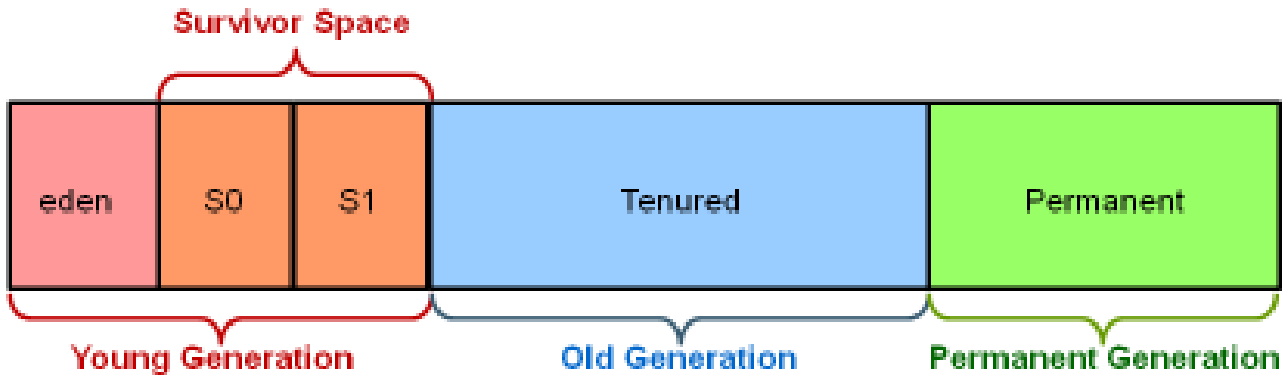


Figura: Imatge extreta de Java Garbage collection basics[6]

- Minor GC: S'executa al omplir l'*eden*.
- Major GC: S'executa al omplir la *old generation*.

Garbage collection a Java

A Java s'implementen a sobre aquestes particions del *heap* diverses alternatives, totes elles de tipus generacionals:

Serial

Únic *thread* per minor i major GC. Mètode esborrar i compactar.

Paral·lel

Múltiples threads per el minor GC i opcionable al major.

Concurrent Mark Sweep

Funciona concurrentment l'execucció del programa amb els major GC. Busca minimitzar les pauses d'execucció del programa.

Garbage collection a Java

Desde Java JDK7, el CMS s'ha substituït per el nou algorisme G1. És un canvi de paradigma, no s'utilitza un *garbage collector* generacional.

- Dividir el *heap* en regions iguals.
- Marcar els objectes de forma paral·lela.
- S'eliminen primer els objects de les regions amb menys objectes (més espai de cop).

Molt més rendiment que el CMS.

Alternatives als *garbage collector*

Assignació per frame
Consisteix en reservar memòria i descart-la sencera en un cert esdeveniment.
Cal trobar aquest 'esdeveniment' i acotar la mida dels objectes.

Pool d'objectes
Reutilitzar els objectes, cal tenir una estructura de dades per veure quins estan ocupats. Cal acotar el nombre màxim d'objectes.

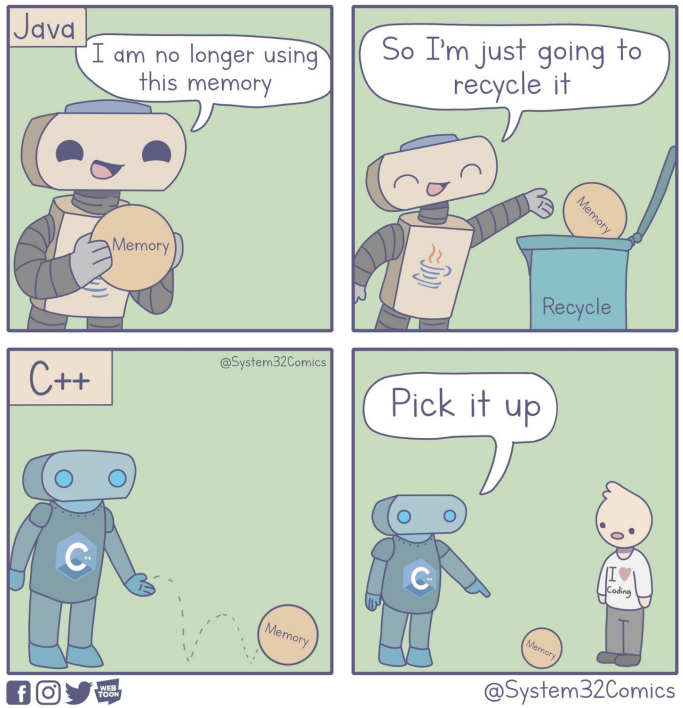
Reserva directa a la pila
En alguns casos particulars, es pot reservar directament a la pila, afegint moltes restriccions als programes.

Conclusions

L'automatització de la gestió del *heap* té coses a favor i en contra:

- El rendiment i el temps de resposta baixa.
- Es facilita la programació.

Tenim diverses versión i configuracions de *garbage collectors* a dins dels llenguatges.



Recollida de memòria brossa

Hash autor: 80608

Facultat d'Informàtica de Barcelona - UPC
Llenguatges de programació

Quadrimestre de tardor, 2020