Garbage collection

Processes are provided with memory.

Objects carry *pointers* to local objects
and *references* to remote objects.

A root object can be created in memory;

objects are always accessed by navigating from a root object.

Aim of garbage collection: To reclaim inaccessible objects.

Three operations by processes to build or delete a reference:

- *Creation*: The object owner sends a pointer to another process.
- *Duplication*: A process that isn't object owner sends a reference to another process.
- *Deletion*: The reference is deleted at its process.

Garbage collection

Processes are provided with memory.

Objects carry *pointers* to local objects and *references* to remote objects.

A root object can be created in memory;

objects are always accessed by navigating from a root object.

Aim of garbage collection: To reclaim inaccessible objects.

Three operations by processes to build or delete a reference:

- *Creation*: The object owner sends a pointer to another process.
- *Duplication*: A process that isn't object owner sends a reference to another process.
- *Deletion*: The reference is deleted at its process.

Reference counting

Reference counting tracks the number of references to an object.

If it drops to zero, and there are no pointers, the object is garbage.

Advantage: Can be performed at run-time.

Drawback: Can't reclaim *cyclic* garbage.

Reference counting

Reference counting tracks the number of references to an object.

If it drops to zero, and there are no pointers, the object is garbage.

Advantage: Can be performed at run-time.

Drawback: Can't reclaim *cyclic* garbage.

+ owner of an object needs to be notified upon duplication of a reference (and needs to ack notification)

Indirect reference counting

A tree is maintained for each object, with the object at the root, and the references to this object as the other nodes in the tree.

Each object maintains a counter how many references to it have been *created*.

Each reference is supplied with a counter how many times it has been *duplicated*.

References keep track of their parent in the tree, where they were duplicated or created from.

Indirect reference counting

If a process receives a reference, but already holds a reference to or owns this object, it sends back a decrement.

When a duplicated (or created) reference has been deleted, and its counter is zero, a decrement is sent to the process it was duplicated from (or to the object owner).

When the counter of the object becomes zero, and there are no pointers to it, the object can be reclaimed.

Weighted reference counting

Each object carries a total weight (equal to the weights of all references to the object), and a partial weight.

When a reference is created, the partial weight of the object is divided over the object and the reference.

When a reference is duplicated, the weight of the reference is divided over itself and the copy.

When a reference is deleted, the object owner is notified, and the weight of the deleted reference is subtracted from the total weight of the object.

If the total weight of the object becomes equal to its partial weight, and there are no pointers to the object, it can be reclaimed.

Weighted reference counting

Each object carries a total weight (equal to the weights of all references to the object), and a partial weight.

When a reference is created, the partial weight of the object is divided over the object and the reference.

When a reference is duplicated, the weight of the reference is divided over itself and the copy.

When a reference is deleted, the object owner is notified, and the weight of the deleted reference is subtracted from the total weight of the object.

If the total weight of the object becomes equal to its partial weight, and there are no pointers to the object, it can be reclaimed.

Weighted reference counting

Each object carries a total weight (equal to the weights of all references to the object), and a partial weight.

When a reference is created, the partial weight of the object is divided over the object and the reference.

When a reference is duplicated, the weight of the reference is divided over itself and the copy.

When a reference is deleted, the object owner is notified, and the weight of the deleted reference is subtracted from the total weight of the object.

If the total weight of the object becomes equal to its partial weight, and there are no pointers to the object, it can be reclaimed.

The previous algorithms are very similar to termination detection algorithms:

Indirect *reference counting* $\Rightarrow$ Dijkstra-Scholten *termination detection*.

Weighted *reference counting* $\Rightarrow$ weight-throwing *termination detection*.

This is not by chance

Garbage collection $\Rightarrow$ termination detection

Garbage collection algorithms can be *transformed* into (existing and new) termination detection algorithms.

Given a basic algorithm.

Let each process p host one artificial root object O_p.

There is also a special non-root object Z.

Initially, only initiators p hold a reference from O_p to Z.

Each basic message carries a duplication of the Z-reference.

When a process becomes passive, it deletes its Z-reference.

The basic algorithm is terminated if and only if Z is garbage.

Garbage collection $\Rightarrow$ termination detection

Garbage collection algorithms can be *transformed* into (existing and new) termination detection algorithms.

Given a basic algorithm.

Let each process p host one artificial root object O_p.

There is also a special non-root object Z.

Initially, only initiators p hold a reference from O_p to Z.

Each basic message carries a duplication of the Z-reference.

When a process becomes passive, it deletes its Z-reference.

The basic algorithm is terminated if and only if Z is garbage.

Garbage collection $\Rightarrow$ termination detection

Garbage collection algorithms can be *transformed* into (existing and new) termination detection algorithms.

Given a basic algorithm.

Let each process p host one artificial root object O_p.

There is also a special non-root object Z.

Initially, only initiators p hold a reference from O_p to Z.

Each basic message carries a duplication of the Z-reference.

When a process becomes passive, it deletes its Z-reference.

The basic algorithm is terminated if and only if Z is garbage.

Garbage collection $\Rightarrow$ termination detection

Garbage collection algorithms can be *transformed* into (existing and new) termination detection algorithms.

Given a basic algorithm.

Let each process p host one artificial root object O_p.

There is also a special non-root object Z.

Initially, only initiators p hold a reference from O_p to Z.

Each basic message carries a duplication of the Z-reference.

When a process becomes passive, it deletes its Z-reference.

The basic algorithm is terminated if and only if Z is garbage.