How to Find Whether a Pointer in Linked list is Corrupted in C programming

May 7, 2017

Finding out whether a linked list in C programming includes a corrupted pointer or not is a bit tricky and difficult but not impossible.

Below are the methods to find out whether a linked list in C programming includes a corrupted pointer or not.

Consistent Checks

Try to perform consistent checks in order to find out whether the linked list is having the information (or figures) that you inserted into it. Do not wait for the effect of corrupt linked list rather ascertain and repair bugs as soon as they corrupt the linked list.

Pointer Value

Instantly after refreshing the memory pointed by the pointer, ensure that the pointer value is set to NULL as it will assist in debugging. Moreover, using reference counts, ascertain the number of objects pointing to an object, if required.

Use of a Good Debugger

In order to know how the data structures are being corrupted and to discover the problem, make use of a good debugger. Memory profilers like Electric fence and purify and debuggers like ddd on Linux help you to discover heap corruption easily.

Magic Value

It is also advisable to introduce a magic value in your node structures. At the time of new node allocation initialize the magic value before each and every access, make sure that the node structure that the pointer points contains the valid magic value. However your program will crash if an unreadable data block is pointed out by the pointer. For this purpose, call API VirtualQuery() on windows before reading and ascertain that only readable data is pointed out by the pointer.

De-referencing and Validation

Discovering a corrupt pointer is a bit difficult, therefore start by ascertaining the value of the pointer to the subsequent element before de-referencing it and make sure to facilitate that it is a valid heap address to avoid segmentation error. Thereafter ascertain that a valid linked list node element is being pointed out by the pointer. Now, the pointer shall be dereferenced into a list of class. After doing so, validate (ascertaining the validity) the int and “next” pointer.

If the int and “next” pointer are also correct and good then it is assured that the preceding node was also good otherwise vica-versa.

Traverse the link with debugger on and check for value of node->link and see whether the address is accessible or not.