Gemeinsamer Memory Manager ohne ShareMem

In unserer täglichen Arbeit mit Delphi benutzen wir Strings, dynamische Arrays und Objekte, ohne uns tiefergehende Gedanken über den Mechanismus zu machen, der hinter deren reibungsloser Verwendung steckt. Hierbei geht es mir nicht so sehr um die Verwaltung von dynamischen Speicher an sich, sondern um die automatische Speicherverwaltung. So ist bei Objekten zwar noch recht offensichtlich, wann der Memory Manager von uns benötigt wird – nämlich beim Instanziieren und Zerstören –, bei dynamischen Arrays wird es jedoch schon wesentlich komplexer.

Erhalten wir beispielsweise von einer Funktion einen String als Rückgabewert, so tritt hier die automatische Speicherverwaltung gleich mehrfach auf den Plan. So muss in der Funktion selbst Speicher für den String alloziert werden; um die anschließende Freigabe des Strings wiederum muss sich der aufrufende Code kümmern. Denn würde die Funktion selbst für die Freigabe sorgen, so würde ja kein zulässiger String zurückgegeben werden. Um dies alles kümmert sich der automatische Memory Manager, ohne den Programmierer in seiner Freiheit einzuschränken oder zu behelligen.

Doch hierbei gibt es ein Problem: Binden wir in unsere Anwendung eine selbstgeschriebene DLL ein, so verfügen beide Module – Anwendung und DLL – über zwei verschiedene Memory Manager, da die dynamische Speicherverwaltung in den Tiefen der Modulinitialisierung erzeugt wird. Würden wir also obige Funktion aus der DLL importieren, so würde der Memory Manager der DLL Speicher für den String allozieren, und der Memory Manager der Hauptanwendung für die anschließende Freigabe sorgen. Da der Speicherbereich des Strings aber nicht zum Zuständigkeitsbereich der Speicherverwaltung der Hauptanwendung gehört, erzeugen wir im besten Fall ein Speicherloch, am häufigsten jedoch erhalten wir eine Zugriffsverletzung.

Borland bietet hierfür eine Lösung in Form der Unit ShareMem an, die sowohl in Hauptanwendung als auch DLL eingebunden werden muss. Diese Unit sorgt dafür, dass Anwendung und DLL nicht getrennte Memory Manager verwenden, sondern auf eine gemeinsame Speicherverwaltung zurückgreifen. Dazu wird die bei Delphi mitgelieferte borlndmm.dll benötigt, welche den gemeinsamen Memory Manager enthält.

Es geht jedoch auch wesentlich einfacher, und zwar mit der Unit PECommonMM (auf Heft-CD), die – ähnlich wie bei ShareMem – sowohl in die Uses-Klausel der Anwendung als auch in die der DLL aufgenommen werden muss. Im Unterschied zu ShareMem wird für die gemeinsame Speicherverwaltung jedoch keine zusätzliche DLL benötigt, wie der folgende Auszug aus PECommonMM zeigt:

```
Function GetApplicationMM: TMemoryManager;
  Stdcall;
begin
  GetMemoryManager(Result);
end;

Exports
  GetApplicationMM Name 'getapplicationmm';

Type
  TGetApplicationMMProc = Function:
    TMemoryManager; Stdcall;

Var
  OldMM: TMemoryManager;
  GetMM: TGetApplicationMMProc;

Initialization
  If IsLibrary Then
  Begin
    GetMemoryManager(OldMM);
    GetMM := GetProcAddress(
      GetModuleHandle(NIL),
      GetApplicationMMProcName);
    If Assigned(@GetMM) Then
    Try
      SetMemoryManager(GetMM);
    Except
      SetMemoryManager(OldMM);
    End;
  End;
Finalization
  If IsLibrary Then
    SetMemoryManager(OldMM);
End.
```

Das hier aufgezeigte Verfahren ist eigentlich recht simpel: Handelt es sich um eine Anwendung, so exportiert sie ihren Memory Manager mit Hilfe der Funktion "GetApplicationMM", handelt es sich hingegen um eine DLL (IsLibrary), so wird der von der Anwendung bereitgestellte Memory Manager importiert, und fortan als lokaler Memory Manager verwendet (SetMemoryManager). Dieses einfache Verfahren steht in seiner Funktionsfähigkeit der Verwendung von ShareMem in nichts nach, dafür können wir jedoch erfreulicherweise auf die Distribution von borlndmm.dll verzichten.

(Oliver Friedmann)