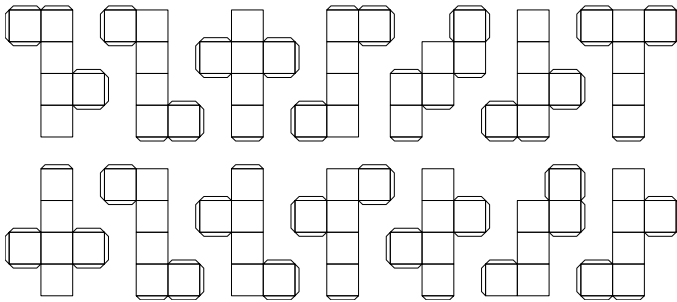


Das Aufspüren von Speicherlöchern in Windows-Programmen



Die Integration von mpatrol in VisualStudio
C++ 6.0



Version: \$Revision: 1.5 \$
Datum: \$Date: 2003/11/04 16:17:17 \$
URL: www.micromata.com
Autor: Bernd Kratz
✉ bernd.kratz@micromata.de

Inhaltsverzeichnis

1	Einleitung	3
1.1	Das Werkzeug mpatrol	3
1.1.1	Wie funktioniert mpatrol ?	4
2	Wie entstehen Speicherlöcher ?	5
2.1	Die Speicherlöcher auf dem Heap	5
2.2	Wie kommt das zustande ?	6
3	Einbinden von mpatrol in VisualStudio	9
3.1	Übersetzen von mpatrol	9
3.2	Installation von mpatrol	9
3.3	Erweitern des Include-Pfades	9
3.4	Einbinden der mpatrol-Bibliothek	9
3.5	Mehrfache Nennung von Symbolen erlauben beim Linken	10
3.6	Erweitern des Linker-Pfades	11
3.7	Einbinden von mpatrol.h in den Quelltext	13
4	Starten des Programmes mit mpatrol	14
5	Auswertung des Log-Files.	16
5.1	Die Fehlermeldungen von mpatrol.	16
5.2	Die leak table	17
6	Maßnahmen	18
7	Fragen und Anregungen	18

Abbildungsverzeichnis

1	Das Include-Verzeichnis einbinden.	10
2	Einbinden der mpatrol-Bibliothek.	11
3	Mehrfache Nennung von Symbolen erlauben beim Linken.	12
4	Erweitern des Linker-Suchpfades.	12

1 Einleitung

Die Programmiersprache C++ und Microsoft's Klassenbibliothek, die *Microsoft Foundation Classes (MFC)* sind mächtige und komplexe Werkzeuge. Die flexiblen und fein granulierbaren Möglichkeiten der Speicherverwaltung in C++ sind nicht Jedermanns Sache. In Verbindung mit den Klassen der MFC entstehen leicht Situationen, die zu Speicherlöchern führen.

Das Entwicklungswerkzeug *Visual Studio 6.0* bietet Unterstützung beim Finden von Speicherlöchern. Doch die Unterstützung ist sehr rudimentär. Neben kommerziellen Lösungen wie *Purify* oder *BoundChecker* gibt es die Open-Source-Lösung *mpatrol*. Dieses Dokument beschreibt die Integration und Benutzung von *mpatrol* in VisualStudio C++, Version 6.0. Die Entscheidung für VisualStudio C++ 6.0 fiel Bewußt, da sehr viele bestehende Projekte auf dieser Plattform entstanden sind und Visual Studio 6.0 noch weit verbreitet ist.

1.1 Das Werkzeug mpatrol

Das Werkzeug "*mpatrol*" dient zum Entdecken von Speicherlöchern und Bereichüberschreitungen in C/C++ - Programmen während der Laufzeit.

Mpatrol ist als Open-Source-Paket unter der GNU Library General Public License (GLPL) für alle gängigen Plattformen im Quelltext verfügbar.

Die Homepage von *mpatrol* ist: <http://www.cbmamiga.demon.co.uk/mpatrol/>. In deutscher Sprache findet man eine Zusammenfassung unter der URL: <http://www.c-handbuch.org/mpatrol.html>.

Das "*mpatrol*"-Paket besteht im wesentlichen aus dem Programm "*mpatrol*", den dazu gehörigen Header-Dateien, Bibliotheken und dem sehr detaillierten Handbuch (mehr als 200 Seiten).

Das komplette Paket liegt im Quelltext vor und muß auf der entsprechenden Zielplattform übersetzt werden. Für den C++-Commandline-Compiler von VisualStudio gibt es vorbereitetes *NMakefile*, welches geringfügig angepasst werden muß.

Die Merkmale von *mpatrol* sind:

- Es zeigt ein Abbild des Stacks bei einem Fehler an.
- Es findet alle denkbaren Fehler auf dem Heap. Fehler auf dem Stack werden nicht gefunden.
- Es bietet eine erstklassige Dokumentation.
- Datei- und Zeileninformationen werden mit ausgegeben.
- Es ist kompatibel zu anderen Tools wie *dmalloc*, *dbmalloc*, *insure* und *purify*.

- Es wird ständig weiter entwickelt.
- Der Autor, Mr. Graeme, bietet Support bei Problemen.

1.1.1 Wie funktioniert mpatrol ?

Mpatrol ersetzt die bekannten Operatoren und Funktionen zum Anfordern und Freigeben von Speicher durch eigene Routinen. Diese Routinen "merken" sich, welche Speicherbereiche angefordert und wieder freigegeben werden.

Zur Laufzeit eines Programmes werden alle Speicheranforderungen und Freigaben erfasst und protokolliert. Am Programmende werden alle Stellen des Programmes, deren Speichermanagement fragwürdig erscheint, in der Log-Datei inkl. Stacktrace ausgegeben. In der Regel sind das Stellen, an denen zu wenig oder kein Speicher freigegeben wird oder es sind Zeiger, die versuchen auf fremde Speicherbereiche zuzugreifen.

Um mpatrol zu benutzen, muss in ein zentrale Stelle des zu untersuchenden Programmes die Header-Datei `mpatrol.h` eingebunden werden. Anschliessend muß das zu untersuchende Programm gegen die mpatrol-Bibliothek gelinkt und dann unter der Kontrolle von mpatrol gestartet werden. In Kapitel [4](#) auf Seite [14](#) ist eine kleine Batch-Datei, die ein Programm unter mpatrol mit den geeigneten Optionen startet.

2 Wie entstehen Speicherlöcher ?

Speicherlöcher können auf verschiedenste Arten entstehen. Eine ausführliche Abhandlung über das Thema würde den Rahmen dieses Dokumentes sprengen. Das Handbuch zu mpatrol enthält eine sehr gute Abhandlung zu diesem Thema und ist jedem Leser wärmstens empfohlen. Das Handbuch findet man unter: <http://www.cbmamiga.demon.co.uk/mpatrol/files/mpatrol.pdf>

2.1 Die Speicherlöcher auf dem Heap

Bei der Auswertung der Log-Dateien von mpatrol bin ich in vielen den Quelldateien auf eine bestimmte Kategorie von Speicherlöchern gestoßen: Es handelt sich dabei um sogenannte "Heap-Speicherlöcher". Das Programm "CKPalArtikelTest.cpp" in Kapitel 2.2 auf Seite 7 enthält ein Vertreter für diese Kategorie von Speicherlöchern. Dabei wird Speicher auf dem Heap alloziert und nur teilweise freigegeben.

Das Programm "CKPalArtikelTest.cpp" fordert in einer for-Schleife, die zweimal durchlaufen wird, Speicher für ein Array mit 5 Elementen der Klasse MyClass an. Der Speicher wird mit

```
MyClass* h = new MyClass[5];
```

angefordert und mit

```
delete h;
```

wieder freigegeben. Die Instanzierung bzw. die De-Instanzierung eines Objektes der Klasse MyClass wird im Konstruktor bzw. Dekonstruktor mitprotokolliert. Das Protokollieren erfolgt durch Erhöhen bzw. Runterzählen der statischen Variablen `_instanz`. Am Ende des Programmes sollte die Variable `_instanz` den Wert 0 haben.

Hier ist die Ausgabe des Programmes:

Datei: CKPalArtikelTest_out.txt

```
1 constructor myclass, _instanz=0
2 constructor myclass, _instanz=1
3 constructor myclass, _instanz=2
4 constructor myclass, _instanz=3
5 constructor myclass, _instanz=4
6 naechster Schleifendurchlauf...
7 destructor myclass, _instanz=4
8 constructor myclass, _instanz=4
9 constructor myclass, _instanz=5
```

```
10 constructor myclass, _instanz=6
11 constructor myclass, _instanz=7
12 constructor myclass, _instanz=8
13 naechster Schleifendurchlauf...
14 destructor myclass, _instanz=8
15 _instanz=8
```

Wie man sieht, hat die Variable `_instanz` den Wert 8. Das bedeutet, daß der Speicher für acht Objekt-Instanzen der Klasse `MyClass` nicht freigegeben worden sind.

2.2 Wie kommt das zustande ?

Der Speicher für das Array wurde mit

```
MyClass* h = new MyClass[5];
```

angefordert und mit

```
delete h;
```

wieder freigegeben. Dabei wird nur das erste Element des Arrays freigegeben. Die restlichen 4 Elemente des Arrays werden mit ihrem belegten Speicher zur Laufzeit nicht freigegeben.

Korrigiert man die Zeile, in der der Speicher freigegeben wird, folgender maßen:

```
delete[] h;
```

ist alles in Ordnung. Dadurch wird der Destruktor für alle Objekte des Arrays aufgerufen und der entsprechende Speicher wird freigegeben.

Datei: CKPalArtikelTest.cpp

```
1 // Dieses Program besitzt ein Speicherloch
2 // bei der Erzeugung eines Arrays
3 // der Klasse MyClass. Es zeigt, wo
4 // und wann die Elemente in dem Array
5 // erzeugt und wieder gelöscht werden.
6 //
7 // 2003 (c) Micromata Objects GmbH
8 //
9 // $Id: CKPalArtikelTest.cpp,v 1.1 2003/09/12 14:18:35 ben Exp $
10 //
11 //
12
13 #include <iostream.h>
14 #include <Windows.h>
15 #include <Winbase.h>
16 #include "mpatrol.h"
17
18 // Laenge des Datenpuffers.
19 const int cbSTRLEN=512;
20
21 // Zaehlt die Instanzen der Klasse.
22 static int _instanz = 0;
23
24 // Die Beispielklasse
25 class MyClass {
26 public:
27 MyClass() { cout << "constructor myclass, _instanz=" << _instanz++ << endl; }
28 ~MyClass() { cout << "destructor myclass, _instanz=" << --_instanz << endl; }
29 char _data[cbSTRLEN];
30 };
31
32 int main(int argc, char* argv[])
33 {
34 // Die Schleife erzeugt zwei Arrays der Klasse MyClass.
35 for(int i=0; i< 2; i++) {
36 MyClass* h = new MyClass[5];
37 cout << "naechster Schleifendurchlauf..." << endl;
38 sleep(1);
39 // Achtung: Heap-Speicherloch !!!
40 // Es wird nur das erste Element in dem Array freigegeben.
41 delete h;
42 }
```

```
43 // Ausgabe der Instanzvariablen.  
44 cout << "_instanz=" << _instanz << endl;  
45 return 0;  
46 }  
47
```

3 Einbinden von mpatrol in VisualStudio

Um mpatrol in VisualStudio 6.0 verwenden zu können, sind folgende Einstellungen in VisualStudio notwendig:

3.1 Übersetzen von mpatrol

Das komplette Paket liegt im Quelltext vor und muß auf der entsprechenden Zielplattform übersetzt werden. Für den Commandline-Compiler von VisualStudio gibt es vorbereitetes NMakefile, welches, wie bereits erwähnt, noch geringfügig angepasst werden muß.

Diese Arbeit ist bereits geschehen. Dieser [Link \(mpatrol_win32vs60.zip\)](#) führt zu einer kompilierte Versionen von mpatrol.

3.2 Installation von mpatrol

Die Installation gestaltet sich recht einfach:

Entweder erweitert man den Pfad um die Verzeichnisse "*mpatrol\lib*" und "*mpatrol\bin*" oder man kopiert die DLL's aus dem Verzeichnis "*mpatrol\lib*" in das Verzeichnis "*\WINNT\System32*" auf dem Systemlaufwerk. Zum Beispiel:

```
cd mpatrol\lib
copy *.dll c:\WINNT\System32
```

3.3 Erweitern des Include-Pfades

Damit der Compiler die Header-Datei "*mpatrol.h*" findet, muss der Include-Pfad erweitert werden. Dazu wählt man aus dem Menu "Projekt->Einstellungen" den Reiter "C/C++ Option Kategorie=Präprozessor" aus und trägt in das Eingabefeld "*Zusätzliche Include-Verzeichnisse*" das Include-Verzeichnis von mpatrol ein, z.B: "*.\mpatrol\inc*". Ein Beispiel findet man in [Abbildung 1](#) auf der nächsten Seite.

3.4 Einbinden der mpatrol-Bibliothek

Um die mpatrol-Bibliothek einzubinden, wählt man unter "Projekt->Einstellungen" den Reiter "Linker, Option Kategorie=Allgemein" aus. In das Eingabefeld "*Objekt/Bibliothek-Module*" wird die mpatrol-Bibliothek eingetragen, "*mpatrolmt.lib*". Die mpatrol-Bibliothek

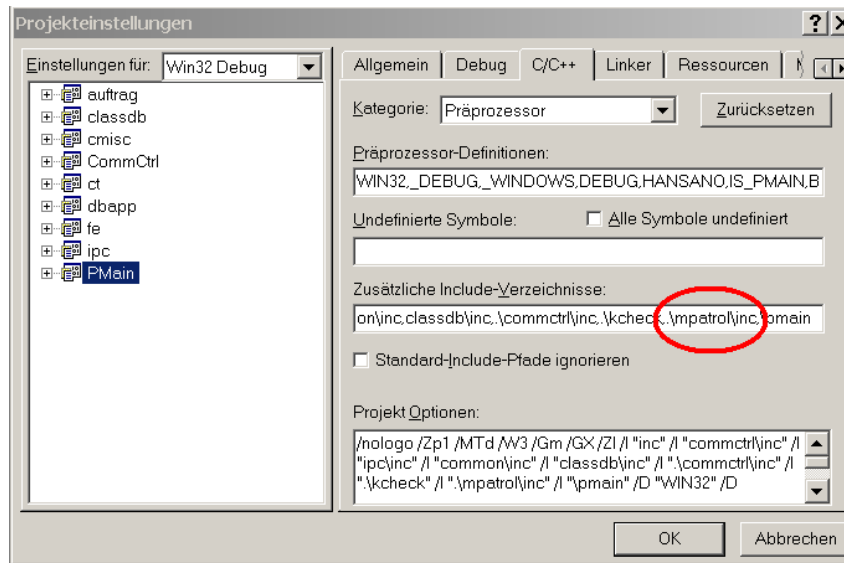


Abbildung 1: Einbinden des Include-Verzeichnis von mpatrol.

gibt es für *Multi-Threaded* und *Single-Threaded*-Programme. Bei *Multi-Threaded*-Programmen, muß die *Multi-Threaded* Version der *mpatrol*-Bibliothek verwendet werden. Sie heißt "*mpatrolmt.lib*". Im Zweifelsfall sollte man die *Multi-Threaded* Version verwenden.

Bei *Single-Threaded* Programmen kann man die *Single-Thread*-Version der *mpatrol*-Bibliothek "*mpatrol.lib*" verwenden.

Abbildung 2 auf der nächsten Seite dient als Beispiel.

Achtung:

Die Reihenfolge der Bibliotheken spielt eine Rolle. Daher sollte die *mpatrol*-Bibliothek möglichst an letzter Stelle stehen. So wird gewährleistet, das der Linker alle anderen Neudefinitionen von Operatoren und Funktionen wie *new*, *delete*, *malloc()*, etc. ignoriert und die Symbole aus dieser Bibliothek benutzt werden.

3.5 Mehrfache Nennung von Symbolen erlauben beim Linken

Die Microsoft Foundation Classes (MFC) stellen für Debugging-Zwecke eine eigene, rudimentäre Speicherverfolgung zur Verfügung. Diese Speicherverfolgung wird im Debugging-Modus von VisualStudio automatisch benutzt und überlädt die bekannten Operatoren und Funktionen zum Anfordern und Freigeben von Speicher. Damit die mehrfache Nennung vom Symbolen beim Linken erlaubt wird, muß man die Checkbox "*Dateiausabe erzwingen*" aktivie-

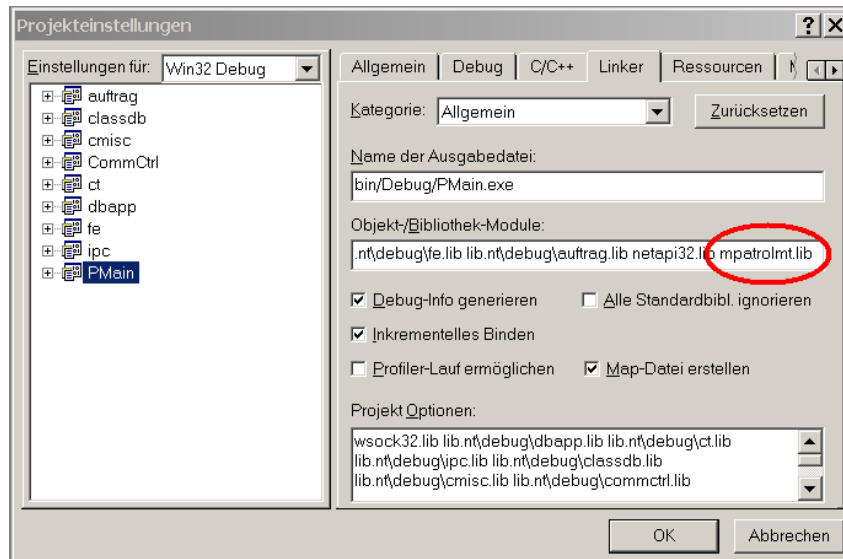


Abbildung 2: Das Einbinden der mpatrol-Bibliothek. Da die Reihenfolge der Bibliotheken eine Rolle spielt, sollte die mpatrol-Bibliothek möglichst an letzter Stelle stehen.

ren. Sie befindet sich unter: "Projekt->Einstellungen", Reiter "*Linker->Kategorie=Anpassen*". Ein Beispiel findet man in [Abbildung 3](#) auf der nächsten Seite.

3.6 Erweitern des Linker-Pfades

Analog zu dem Include-Pfad, muß der Suchpfad des Linkers für Bibliotheken erweitert werden. Unter "Projekt->Einstellungen", Reiter "Linker, Option *Kategorie=Eingabe*" trägt man:

- in das Eingabefeld "*Objekt/Bibliothek-Module*" die mpatrol-Bibliothek ein, "mpatrolmt.lib".
Achtung:

Die Reihenfolge der Bibliotheken spielt eine Rolle. Die mpatrol-Bibliothek sollte möglichst an letzter Stelle stehen. So wird gewährleistet, das der Linker alle vorhergehenden Definitionen und Symbole von Operatoren und Funktionen wie `new`, `delete`, `malloc()`, etc. verwirft. Es werden dann die Symbole aus der mpatrol-Bibliothek benutzt.

- in das Eingabefeld "*zusätzlicher Bibliothekspfad*" das Verzeichnis für die mpatrol-Bibliotheken ein, ".\mpatrol\lib" ein.

Ein Beispiel findet man in [Abbildung 4](#) auf der nächsten Seite.

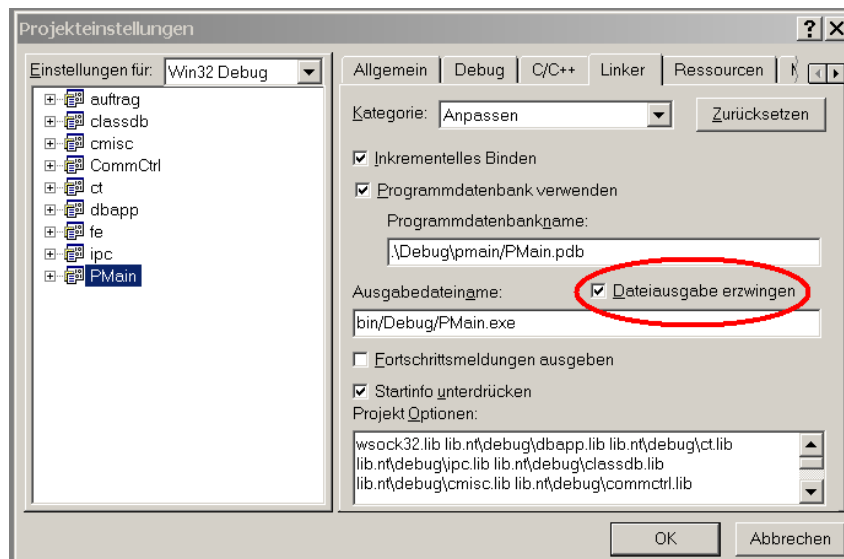


Abbildung 3: Mehrfache Nennung von Symbolen erlauben beim Linken.

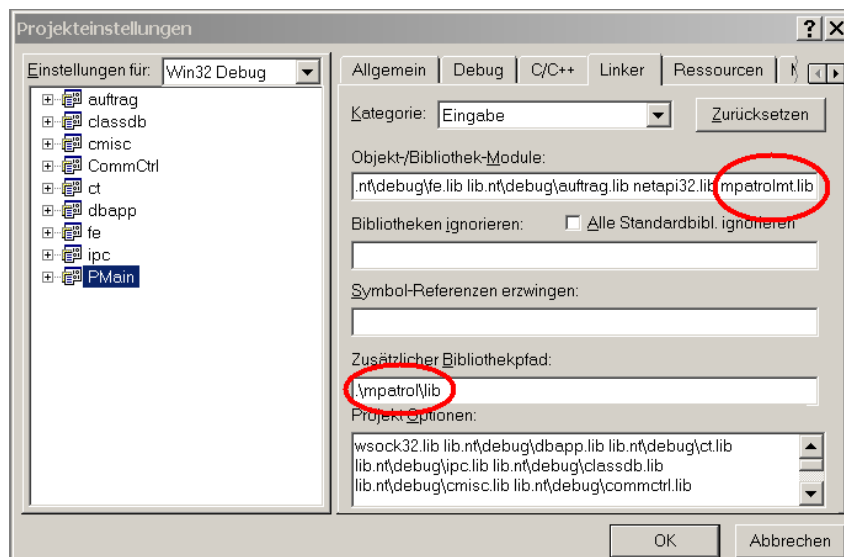


Abbildung 4: Erweitern des Linker-Suchpfades.

3.7 Einbinden von mpatrol.h in den Quelltext

An mindestens einer zentralen Stelle im Programm sollte die Header-Datei `mpatrol.h` von `mpatrol` eingebunden werden. Dies geschieht mit:

```
1 #define MP_NOPLUSPLUS 1
2 #include "mpatrol.h"
```

Um Konflikte bei der Überladung der MFC internen Speicherverfolgung zu vermeiden, empfiehlt es sich, die Header-Datei `mpatrol.h` als letzte Header-Datei einzubinden.

Bei der anschliessenden Neuübersetzung des Projektes erscheinen einige Warnungen, daß Operatoren wie `new`, `delete`, etc. bereits definiert sind und von `mpatrol.h` neu definiert werden. Diese Warnungen können ignoriert werden.

4 Starten des Programmes mit mpatrol

Nachdem das Programm und alle benutzten Bibliotheken, wie *"auftrag.lib, classdb.lib, cmisc.lib, CommCtrl.lib, etc."* erfolgreich neu übersetzt worden sind, kann man das Programm unter der Kontrolle von mpatrol starten.

Um den Start des Programmes mit mpatrol zu vereinfachen, gibt es das Skript *"runme.bat"*. Das Skript ist in dem [Zip-File \(mpatrol_win32vs60.zip\)](#) enthalten. Hier ist der Quelltext:

Datei: runme.bat

```
del ..\log\pmllog*. *
mpatrol --use-debug --show-unfreed --leak-table --threads -l mpatrol.log .\pmain.exe
```

Das Skript löscht die alten Log-Dateien und startet das Programm, das als Parameter übergeben wird mit den entsprechenden Optionen von mpatrol:

Option	Bedeutung
-use-debug	Schaltet den Stacktrace bei der Ausgabe ein.
-show-unfreed	Gibt alle Stellen aus, an denen der Speicher nicht freigegeben wird.
-leak-table	Gibt eine Tabelle der Speicherlöcher aus.
-threads	Schalter für Multi-Threaded-Programme.
-l mpatrol.log	Die Log-Datei heißt mpatrol.log.

Alle Optionen bekommt man mit dem Befehl `mpatrol --help` angezeigt. Nähere Informationen über die verschiedenen Optionen und Schalter findet der interessierte Leser im Handbuch zu mpatrol.

Achtung:

- Jede Speicheranforderung und Freigabe wird protokolliert. Die Programmausführung verlangsamt sich dadurch sehr stark. Auf einem Pentium 3, mit 1.13 Ghz und 512 MB. Ram dauerte die Initialisierungsphase eines untersuchten Programmes ca. 8 Minuten !
Bei diesen Programm wurden 100 Schleifendurchläufen durchgeführt. Ein Schleifendurchlauf dauerte ohne die Speicherverfolgung ca. 2-3 Sekunden. Mit aktivierter Speicherverfolgung dauerte ein Test ca. 1 Stunde !
- Das Log-File von mpatrol wächst sehr schnell. Es hat sich für Testzwecke als nützlich erwiesen, daß sich das untersuchende Programm nach 10 - 100 Schleifendurchläufen in automatisch beendet.
So erhält man schnell einen Überblick, wo Speicher alloziert und nicht wieder freigegeben wird.

Das Log-File von mpatrol wird dabei ca. 200 MB. groß.

5 Auswertung des Log-Files.

Der Aufbau des Log-Files wird in dem Handbuch von mpatrol detailliert beschrieben. Der geneigte Leser sei auf dieses Dokument verwiesen.

5.1 Die Fehlermeldungen von mpatrol.

Den Hauptteil der Log-Datei machen die *Fehlermeldungen* von mpatrol aus.

Bei den *Fehlermeldungen* handelt sich um Stellen im Quellcode, mit fragwürdigem Speicher-
management. In der Regel werden Programmstellen angezeigt, wo Speicher angefordert oder
freigegeben wird, und mpatrol eine Differenz zwischen der angeforderten und der freigegebenen
Größe entdeckt hat.

Datei: mpatrol.log

```

1  FREE: operator delete (0x01140004) [1244|-|c:\mpatrol\tests\ckpalartikeltest\ckpalartike
2  0x004014F5 std::bad_alloc::bad_alloc+165 at c:\mpatrol\inc\mpatrol.h:1091
3  0x004018D8 MyClass::'scalar deleting destructor'+56
4  0x0040119C main+252 at c:\mpatrol\tests\ckpalartikeltest\ckpalartikeltest.cpp:40
5  0x004045DC mainCRTStartup+252 at crt0.c:206
6  0x77E9847C ProcessIdToSessionId+381
7
8  ERROR: [MISMAT]: operator delete: 0x01140004 does not match allocation of 0x01140000
9  0x01140000 (2564 bytes) {operator new[:1:0} [1244|-|c:\mpatrol\tests\ckpalartikeltest\c
10 0x00401277 main+471 at c:\mpatrol\inc\mpatrol.h:1062
11 0x004010F0 main+80 at c:\mpatrol\tests\ckpalartikeltest\ckpalartikeltest.cpp:36
12 0x004045DC mainCRTStartup+252 at crt0.c:206
13 0x77E9847C ProcessIdToSessionId+381

```

In unserem Beispiel hat mpatrol eine Differenz bei der Größe des freigegebenen Speichers zu
der Größe des angeforderten Speichers festgestellt.

```
ERROR: [MISMAT]: operator delete: 0x01140004 does not match allocation of 0x01140000
```

Die Fehlermeldung in Zeile 8 bezieht sich auf den Speicher, der in Zeile 1 freigegeben wurde.
Die Größe des Speichers, die in Zeile 1 freigegeben wurde, entspricht nicht der Größe des
Speicher, der in Zeile 9 mit `new[]` angefordert wurde.

Dies ist ein Indiz für ein Speicherloch !

Es lohnt sich, diese Fehlermeldungen zu überprüfen und zu beheben.

6 Maßnahmen

In der Praxis war die folgende Vorgehensweise sehr erfolgreich:

- In das zu untersuchende Programm wird mpatrol integriert. Es werden Testläufe durchgeführt und ausgewertet.
- Alle Quelltext-Stellen mit fragwürdigem Speichermanagement werden gekennzeichnet. Dabei hat sich die Kennzeichnung mit Hilfe des @todo-Tag von Doxygen bewährt. Bei der Kennzeichnung wird, wenn möglich, die Art des Speicherlochs gekennzeichnet. Es wurde dabei folgende Markierung benutzt:

```
// @todo potentielles Speicherloch bei new char[...]
```

Bei der Erstellung des Aufrufbaums der Klassen mit dem Werkzeug [Doxygen](http://www.doxygen.org) (<http://www.doxygen.org>) werden die so markierten Stellen erfasst und in der TODO-Tabelle dokumentiert.

- Jede dieser Stellen muß überprüft werden, wo und wann der angeforderte Speicher wieder freigegeben werden kann.

In der Regel muß an einigen Stellen sensibler Code geändert werden. Um evtl. Nebeneffekte zu minimieren, müssen die geänderten sensiblen Stellen sorgfältig und ausgiebig getestet werden.

Für diese Tests eignen sich Unit-Tests in der Regel am besten.

Die beseitigten Stellen werden ebenfalls mit Tags gekennzeichnet, so daß man diese Stellen später leicht identifizieren kann.

Falls bis dato noch kein Versionsmanagement mit *CVS* oder *Visual Source Safe* eingeführt worden ist, so sei dies in diesem Schritt empfohlen.

- Nach der Beseitigung der Speicherlöcher muß die Applikation getestet werden. Verlaufen diese Tests positiv, so wird erneut ein Test mit mpatrol durchgeführt.

7 Fragen und Anregungen

Falls Sie Fragen oder Anregungen zum Thema *Speicherlöcher und deren Beseitigung*, haben, haben Sie keine Scheu sich an uns zu wenden. Sie erreichen uns unter:

Micromata Objects GmbH

Marie-Calm-Str. 3

D-34131 Kassel

Tel: 0561 / 316 79 30

Fax: 0561 / 316 79 3 11

Email: Software-Sanierung  softwaresanierung@micromata.de