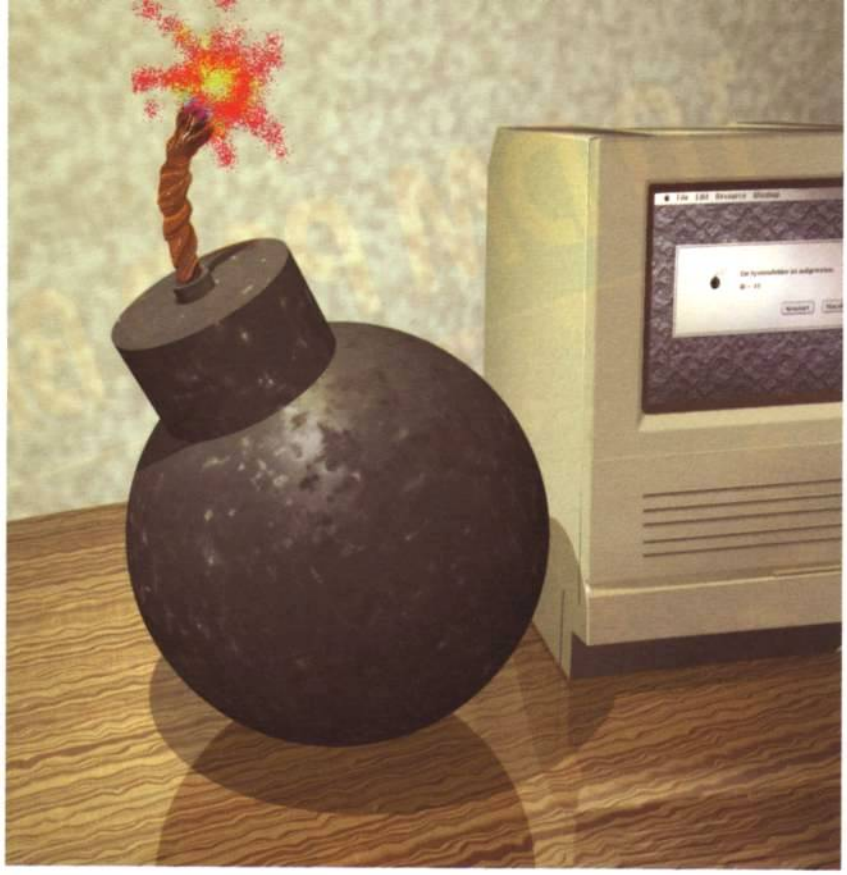

Debuggen mit dem MacsBug

Wer die Dialogbox „Ein Systemfehler ist aufgetreten" nicht mehr sehen kann, sollte sich Gedanken machen, einen Low-Level-Debugger zu installieren. Der verhindert zwar nicht den Absturz, kann aber helfen, die Ursache zu finden.

Ein Low-Level-Debugger wird auf dem Macintosh bei jedem Neustart automatisch geladen und ist stets im Hintergrund aktiv. Außer durch die Meldung „Debugger installiert." unterhalb des „Willkommen" beim Neustart bemerkt man den Debugger – hoffentlich – auch nicht. Er wird erst aktiv, wenn man ihn aufruft oder wenn der Macintosh abstürzt.

Ich gehe hier nicht weiter auf Assembler (68k bzw. PowerPC) ein, das grundlegende Wissen darüber setze ich einfach mal voraus ...

Jasiks Debugger

Auf dem Macintosh sind momentan zwei Low-Level-Debugger verfügbar. Der eine ist „The Debugger" von Steve Jasik, kurz: Jasiks Debugger. Ein kommerzieller Debugger, der die Geldbörse noch um viele hundert Mark erleichtert. Dafür erhält man auch die wohl schlechteste Anleitung in der Computerbranche ... Warum ist Jasiks Debugger trotzdem bei den Profis extrem erfolgreich? Nun, weil er – nachdem man sich in die extrem komplizierten Möglichkeiten eingearbeitet hat – nahezu perfekt ist und quasi jeden Programmierfehler finden kann – es gibt für jedes denkbare Problem Kommandos, um den Fehler einzukreisen.

So erlaubt er sogar das Debuggen von allen denkbaren Programmen (Systemerweiterungen, Code-Fragmente, usw.). Nicht nur in Assembler, sondern sogar im Quellcode! Dabei kann er High-Level-Debugger, wie z.B. den Debugger vom Metrowerks CodeWarrior auch direkt ersetzen.

Seine Möglichkeiten, per Speicherschutz (auf 68030 CPUs per MMU) Fehler zu finden sind ungeschlagen. Selbst im PowerPC-Code können Watchpoints gesetzt werden, da eine komplette Software-Emulation des PowerPC vorhanden ist.

Ferner ist der Support nahezu perfekt: EMail-Anfragen über das Internet sind i.d.R. in weniger als 3-4 Stunden beantwortet. Updates kommen regelmäßig via Internet.

Kurzum: für jedermann der perfekte Debugger, wenn man die Zeit für die Einarbeitung und das nötige Kleingeld hat. Ich gehe deswegen im Folgenden nicht weiter auf Jasiks Debugger ein.

MacsBug

Für den kleinen Geldbeutel gibt es den MacsBug von Apple. Immer wieder an neue Rechner angepaßt, gibt es jetzt die Version 6.5.2, die endlich auch das Debuggen von PowerPC-Programmen ermöglicht. Ältere Versionen konnten nur 68k-Programme debuggen. Der Preis für den MacsBug liegt bei 0 DM – er ist Freeware.

Den MacsBug gibt es quasi, seitdem es Apple-Macintosh-Computer gibt. Ob er bereits vorher auf der Lisa verfügbar war, ist mir nicht bekannt. Der Name MacsBug stammt übrigens nicht von „Macintosh", sondern er ist eine Abkürzung für „Motorola advanced computer systems debugger".

Die Installation ist problemlos: einfach das File „MacsBug" in den Systemordner ziehen und den Rechner neu starten. Nun sollte sich der Debugger in der Willkommen-Dialogbox bemerkbar machen. Ansonsten sollte sich der Rechner in seinem Verhalten nicht ändern – außer, daß man weniger freies RAM als vorher hat.

Aufruf des Debuggers

Es gibt drei Möglichkeiten, einen Low-Level-Debugger aufzurufen:

1. durch einen Programmabsturz
Unplanmäßig, aber nicht zu ändern. Bei eigenen Programmen kann man nun nach der Ursache für den Absturz suchen. Bei fremden Programmen kann man i.d.R. nicht viel mehr, als sie abzuschließen oder gar den ganzen Rechner neuzustarten.

2. durch direkten Aufruf
Um den Debugger manuell aufzurufen, muß man den „Programmierschalter" betätigen. Dieser Taster löst einen NMI aus, welcher den Debugger veran-

laßt, sich zu melden. Ältere Macs bzw. große Macs haben noch so einen Schalter vorne, an der Seite oder hinten am Gerät – Näheres sollte in der Anleitung stehen. Teilweise – bei PowerBooks – braucht man eine Büroklammer, um den Taster zu betätigen.

Neuere Macs erlauben den Aufruf des Debuggers mit der Tastenkombination Command + PowerOn. Ab System 7.5.1 bekommt man leider manchmal eine Dialogbox mit der Frage, ob man den Rechner neustarten oder abschalten will – diese muß man dann abbrechen und es noch einmal probieren: Command festhalten und auf PowerOn hämmern ... Bei der Gelegenheit: mit Command + Control + PowerOn kann man einen Reset bei solchen Rechnern auslösen. Dies ist aber nur im äußersten Notfall sinnvoll, da sonst Daten auf der Festplatte beschädigt werden können!

Falls keiner der obigen Aufrufe möglich ist, kann man sich die Freeware Systemerweiterung „Programmer's Key" besorgen, welche den MacsBug per Tastatur aufrufbar macht.

3. durch Aufruf vom Programm aus

Das MacOS unterstützt den expliziten Aufruf eines Debuggers. Der Aufruf lautet *Debugger()* bzw. *DebugStr()*. Bei letzterem Aufruf darf man noch einen Pascal-String mit übergeben, welcher im Debugger ausgegeben wird. Sehr praktisch, um die Fehlerstelle zu beschreiben, falls man mehrere Debugger-Aufrufe im Programm hat. Zudem kann man Kommandos für den MacsBug direkt mit übergeben und sofort beim Aufruf ausführen lassen!

Sobald der MacsBug aktiv ist, schaltet er auf eine eigene Bildschirmseite um. Der originale Bildschirm bleibt dadurch erhalten. Man kann vom Debugger aus jederzeit mit der ^-Taste (links oben auf der Tastatur) zwischen den Seiten hin- und herschalten.

Wenn man versucht, einen nicht installierten Debugger aufzurufen (z.B. mit Command-PowerOn), so landet man im ROM Debugger. Es geht eine Dialogbox auf, oben links erscheint ein „>" und ein Cursor – fertig. Dieser Debugger kann nicht wesentlich mehr, als einen Speicher-Dump anzeigen und mit „G" + Return den Programmlauf fortfahren zu lassen.

Aufbau vom MacsBug

Leider ist es mir nicht möglich, aus dem Low-Level Debugger eine Bildschirm-Hardcopy zu machen, so daß ich den Bildschirmaufbau vom MacsBug hier beschreiben muß ...

Der Bildschirm ist in vier Teile geteilt:

Am linken Rand befindet sich von unten nach oben eine Liste mit allen Registern, dem aktuell laufenden Programmnamen sowie einem Speicherauszug vom Stack-Bereich.

In der untersten Zeile blinkt der Cursor für die Eingabe. Alle MacsBug-Befehle werden dort eingegeben. MacsBug hat keine grafische Oberfläche mit Maus und Dialogboxen (wie der Jasiks Debugger).

Direkt über der Cursoreingabe sieht man einige Zeilen Programmcode. Die Zeile, an der man in den MacsBug gelagt ist und auf die der aktuelle Programmcounter (PC) zeigt, ist mit einem „*" markiert. „." vor einer solchen Zeile zeigen einen Breakpoint an.

Der restliche Bildschirm oberhalb dieser Zeilen enthält die Ausgaben der Eingabebefehle – quasi eine History, was man im Debugger gemacht hat. Man kann mit den Cursortasten hin- und herscrollen.

Die Eingabezeile

Alle Befehle des MacsBug bestehen aus minimal einem Buchstaben, meistens jedoch aus zwei bis drei kryptischen Buchstabenkombinationen. Der MacsBug ist über sogenannte *dcmds* jedoch nahezu um beliebige Befehle erweiterbar.

In der Eingabezeile kann man den Cursor mit den Cursortasten bewegen – man hätte es fast erahnen können ... Ferner kann man sich jederzeit mit Command-D die interne Symboltabelle abrufen (spart Tipparbeit), sowie mit Command-E bzw. -R die externen Symbole.

Ein Befehl kann keine oder mehrere Parameter haben, die teilweise optional, teilweise nötig sind. Alle Zahleneingaben werden als hexadezimale Zahlen aufgefaßt, wenn man sie nicht mit einem # vorneweg als Dezimalzahlen kennzeichnet. Parameter können natürlich auch Formeln sein, die neben den Grundrechenarten auch logische Operationen, Vergleiche und das Auslesen einer Speicherstelle (z.B. eines Langworts) ermöglichen um damit weiterzurechnen. Ebenso sind dort die Register der CPU erlaubt. Achtung: A0 wird stets als Register A0 aufgefaßt und nicht als Hexadezimalzahl A0! In einem solchen Fall kann man entweder ein „$" oder eine Null voranstellen.

Der MacsBug erkennt zudem Symbole im Programmcode und erlaubt ebenfalls die Eingabe dieser Symbole in Formeln. Auch kann man Traps (wie *GetNewDialog*) direkt eingeben – der Wert ist dann die Adresse der Routine, oder der Wert des Opcodes – je nach Wunsch.

Zwei sehr praktische Abkürzungen sind der „.", welcher für die zuletzt angesprochene Adresse bei vorherigen Befehlen steht. Man kann also mit „SM" eine Speicherstelle ändern und mit „DM ." sich diese Speicherstelle sofort wieder ansehen.

Eine weitere Abkürzung ist der „:", welcher auf den Anfang des aktuellen Unterprogramms zeigt. Er ist nur dann gültig, wenn das Programm eine Symboltabelle hat.

Stellt man einem Wert ein „.b" oder ein „.w" an, so wird der Wert als Byte oder Wort interpretiert.

Flußkontrolle

Der wichtigste Befehl lautet einfach „G". Hiermit kann man das Programm am PC (oder einer anderen Adresse) fortfahren lassen. Dies ist die übliche Methode, den Debugger wieder zu verlassen.

Mit S bzw. Command-S kann man im Einzelschrittmodus voranschreiten. Dabei wird in Unterprogramme und Traps verzweigt. Man sieht also jeden ausgeführten Opcode.

Mit T bzw. Command-T kann man ebenfalls im Einzelschrittmodus voranschreiten, jedoch werden Unterprogramme und Traps übersprungen, d.h. als ganzes ausgeführt. Eine schnellere Methode, um bekannte Routinen zu übergehen.

Breakpoints

Ebenfalls sehr wichtig ist der Befehl „BR", welcher einen Breakpoint auf eine bestimmte Adresse setzen kann. Nach dem Setzen des Breakpoint läßt man das Programm mit „G" laufen und wenn der Breakpoint erreicht wird, landet man wieder im Debugger.

Der Breakpoint kann zudem mit einem Ausdruck belegt werden, so daß der Breakpoint nur dann in den MacsBug verzweigt, wenn eine Nebenbedingung erfüllt ist. Zudem kann man bei Erreichen des Breakpoints automatisch ein paar MacsBug Befehle ausführen lassen.

Mit „BRC" kann man einen oder alle Breakpoints wieder löschen. Mit „BRD" kann man sich die aktuellen Breakpoints anzeigen lassen.

Line-A Traps

Bei den 68k-Macintosh-Rechnern laufen ja alle Betriebssystemfunktionen über den Line-A-Vektor der CPU. Verschiedene Line-A-Opcodes stehen dabei für verschiedene MacOS-Funktionen. MacsBug bietet mit mit dem Befehl „ATB" nun die Möglichkeit an, beim Aufruf einer bestimmten MacOS Funktion in den Debugger zu springen. Mit „ATBA" werden nur MacOS-Funktionen überwacht, die vom aktuellen Programm aus aufgerufen werden.

Mit „ATC" kann man diesen Breakpoint wieder löschen, mit „ATD" sich die Tabelle anzeigen lassen.

Diese nette Funktion funktioniert aber nicht bei native Code, da dieser keine Line-A-Opcodes enthält!

Disassembler

Ebenfalls sehr wichtig ist der eingebaute Disassembler. Mit „IL" kann man sich eine Routine in Assemblercode ansehen. Es gibt noch einige weitere Abwandlungen, die man sich in der eingebauten Hilfe vom MacsBug ansehen kann.

Heaps

Der MacsBug bietet zudem die Möglichkeit sich die Speicherbelegung eines Programms anzusehen (mit „HZ", „HD" bzw. „HT") und zu überprüfen, ob sie noch intakt ist (mit „HC"). Zudem kann man mit Heapscramble („HS") den Heap automatisch bei allen MacOS-Funktionen neu mischen lassen, die dies eventuell tun könnten! Gerade solche Fehler (nicht gelockte Handles) sind sonst schwer zu finden!

Stack

Mit „SC6" bzw. „SC7" kann man sich den Aufrufstack ansehen. Dabei kommt der MacsBug auch mit Mode-Switchen zwischen verschiedenen Routinen zurecht! Das ist die Funktion um herauszubekommen, wo ein Programm abgestürzt ist und wie es dahin gekommen ist.

Speicher

Natürlich kann MacsBug auch Speicher einfach nur als Hexdump darstellen. Dazu dienen die Funktionen um „DM". Mit „SM" kann man Speicher verändern.

Register

Mit „TD" kann man sich die CPU Register anzeigen lassen – und zwar nicht nur die paar am linken Rand, sondern wirklich alle. Mit „TF" kann man sich die Register der FPU, mit „TM" die der PMMU anzeigen lassen.

Man kann Register einfach ändern, indem man ihnen mit der Zuweisung „=" einen neuen Wert zuweist. Beispiel: PC=PC+2 um den PC um 2 Byte weiterzusetzen.

Macros

MacsBug erlaubt es häufig benutze Befehlsfolgen mit Macros abzukürzen.

So kann man mit „MC test 'il pc'" bei Eingabe von „test" den Befehl „il PC" ausführen lassen.

Diverses

Ebenfalls wichtig sind die Funktionen um den Rechner nach einem „harten" Crash gegenenfalle neuzustarten. Üblicherweise probiert man nach einem Absturz eines fremden Programmes zuerst einmal die Funktion „ES" (aktuelles Programm beenden), falls dies nicht klappt so probiert man „RS" (Restart), falls auch dies nicht klappt „RB" (Reboot).

„ES" entspricht dem Command-Option-Escape von System 7. „RS" versucht die Platten zu unmounten, bevor der Rechner neu gestartet wird. Dies ist sicherer, damit evtl. Änderungen auch wirklich weggeschrieben werden können. „RB" startet den Rechner nämlich direkt neu – ein echter Reset.

Nach einem deftigen Crash kann es passieren, daß der MacsBug beschädigt wurde und keine der drei Möglichkeiten mehr funktioniert. Nun, dann kommt man nicht umhinn den Reset-Taster zu betätigen.

Mit der Funktion „HOW" kann man jederzeit abfragen, warum man sich eigentlich im Debugger befindet. Es wird z.B. nocheinmal die Absturzursache ausgegeben.

Ebenfalls wichtig ist die „LOG" Funktion, welche das Protokollieren der Debuggerausgaben in eine Datei ermöglichen. In Verbindung mit *DebugStr()* kann man einen automatischen Crashreport bei Betatestern erzeugen – ohne das diese Ahnung vom MacsBug haben brauchen!

Eine ganz ganz wichtige Funktion ist HELP oder „?". Hiermit bekommt man nämlich die Kurzanleitung zum MacsBug ausgegeben. Die entspricht im wesentlichen meiner Zusammenfassung hier, ist aber doch noch um einiges ausführlicher.

dcmds

Der MacsBug kann mit dcmds um eigene oder fremde Funktionen ergänzt werden. Von solchen nützlichen Funktionen gibt es inzwischen eine ganze Menge. So kann man sich Informationen über alle Laufwerke, geladenenen Resourcen, Things, installierte VBL-Interrupts, laufende Prozesse, alle vorhandenen Gestaltwerte, alle offenen Dateien und alle installierten Device-Treiber ausgeben lassen.

Es gibt auch dcmds für die Analyse von AppleEvents und anderen Betriebssystemstrukturen.

Ferner kann man mit dem *dcmd* „Leaks" sehr schön Speicherlöcher finden, die immer dann auftreten, wenn man beim Aufruf einer Routine etwas Bufferspeicher alloziert, diesen beim Verlassen aber nicht wieder freigibt.

Diese Fehler sind normalerweise sehr sehr schwierig zu finden und bewirken bei ansonsten fehlerfreien Programmen einen unerklärlichen ständigen Speicherverlust.

Absturz, was tun?

Vor dem Absturz sollte man daran denken, unbedingt im Compiler die Symboltabelle an- und alle Optimierungen abzuschalten. Ersteres macht das Debuggen leichter, letzteres macht es schwieriger. Nur wenn man Fehler hat, die nur im optimierten Code auftreten, sollte man diese anschalten. Solche Fehler sind – dank guter Compiler – aber sehr selten.

Zuerst sollte man sich den Hauptbildschirm nochmals ansehen. Häufig kann man schon erkennen, warum das Programm abgestürzt sein könnte.

Im zweiten Schritt sollte man mit WH nachsehen, wo man sich befindet. Z.B. sagt einem der MacsBug, daß man sich gerade im ROM befindet.

Als nächstes guck man sich mit SC6 (falls dies nicht klappt mit SC7) den Aufrufstack an. Hier kann man erkennen, wie man zu dieser Absturzstelle gelangt ist. Hat man eine Symboltabelle, sollte sie Sache nun schon recht klar werden.

Jetzt guckt man sich mit IP die Stelle vor und um den aktuellen PC (unsere Absturzstelle) an. Falls die Stelle vor dem PC nicht aussagekräftig genug ist, kann man mit IL PC-80 weiter vorne anfangen sich den Code anzusehen. Hierbei kann es praktisch sein, sich die Ausgabe mit „LOG" zu protokollieren um sie bei Bedarf später mit der Disassembler-Ausgabe des Compilers zu vergleichen. So kann man die Stelle dann auch ohne Symboltabelle (im Notfall) einkreisen. Zumindest Think C und Metrowerks CodeWarrior erlauben das Disassemblieren von Quelltexten.

Im Anschluß guckt man sich nochmal den Heap (mit „HT") an um zu sehen, ob dieser beschädigt ist. Wenn ja, dann ist ein Pointer wohl Amok gelaufen und hat die Strukturen zerstört. Solche Fehler sind üblicherweise sehr unangenehm und nicht leicht zu finden!

Auch sollte man „DW memErr" und „DW resErr" eingeben. Ersterer zeigt den letzten Fehlercode des Memory Managers an, letzterer zeigt den letzten Fehler des Resource Managers an. Ist einer der Werte nicht 0, so hat man wahrscheinlich einen Fehlercode nicht ausgewertet.

Falls man mitten im ROM einen Absturz hat, so sollte man sich den Aufruf (Sprung ins ROM) ansehen. Die häufigste Ursache für einen Absturz im ROM sind fehlerhafte Parameter!

Vor dem Verlassen des MacsBug nicht vergessen mit „LOG" das Logfile zu schließen! Verlassen kann man den Debugger, wie bereits oben beschrieben. Fortfahren mit „G PC+2" o.ä. sollte man nur, wenn die Fehlerursache völlig klar war und im Geiste schon behoben ist ...

Weitere Informationen

Als erstes sollte man sich die Hilfetexte im MacsBug durchlesen. Wem das nicht reicht, der kann sich das „MacsBug Reference and Debugging Guide" von Addison-Wesley (ISBN 0-201-56767-9, Preis $24.95) besorgen. Auf etwas über 400 Seiten wird nicht nur der MacsBug besprochen, sondern auch einiges zum Speicheraufbau des Macintosh, sowie typische Probleme. Leider ist das Buch noch für den MacsBug 6.2 gedacht – ohne PowerPC Erweiterungen.

Natürlich sollte man 68k-Assembler – bei Bedarf auch PowerPC-Assembler – zumindest lesen können um einen Fehler einzukreisen. Literatur hierzu gibt es inzwische ja reichlich und für jeden Geschmack. Eine sehr gute Hilfe sind hier wieder einmal die Disassembler Ausgaben der Compiler. Man kann prima sehen, was für Assembler-Code aus welchem Programmcode erzeugt wird!

Schlußwort

Ich weiß, daß diese extrem kompakte Einführung sehr trocken und auch viel zu kurz geworden ist. Aber der Platz erlaubt nicht mehr und ich hoffe doch, daß zumindest klar geworden ist, wofür der MacsBug gut ist, und was man alles mit ihm machen kann. Viel Spaß bei der Fehlersuche ...

MFR