in

Ben Hutchings

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Ben Hutchings	February 12, 2023	

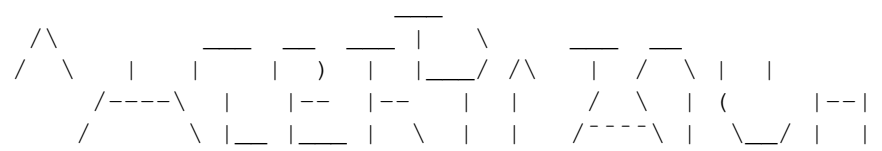
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 Contents



by Ben Hutchings version 1.1 26 November 1995

1.2 Legal Information

The contents of the 'AlertPatch' package are the files:

AlertPatch	AlertPatch.info
AlertPatch.guide	AlertPatch.guide.info

(i)~~~no charge is made for this above the costs of duplication, distribution and the media used,
(ii)~~all the aforementioned files are distributed together, and
(iii)~none of the aforementioned files are altered, except, if the distributor so wishes, by converting them into a compressed form from which they can be retrieved unaltered.

Software released under these conditions is often known as Freeware.

This software and documentation is provided 'as-is' without representation or warranty of any kind, either express or implied, including without limitation, any representations or endorsements regarding the use of, the results of, or the performance of the information, its appropriateness, accuracy, reliability, or currentness; the entire risk as to the use of this information is assumed by the user.

In no event shall I be liable for any damages, direct, indirect, incidental or consequential, resulting from any defect in the information, even if I have been advised of the possibility of such damages.

1.3 Introduction

'Guru' alerts are a occupational hazard for every programmer ↩ and many
other Amiga users besides. Not only is it very frustrating to see possibly hours of work fall into the bit bucket, but even more infuriatingly, the only explanation is some impenetrable hex code which only those with a serial debugger can fully interpret.

Until now! Admittedly, there are many programs around which can explain the last alert number, but hardly any which will reliably tell you which program was running at the moment the crash occurred. This is because the task information is usually corrupted by the time the computer has rebooted. AlertPatch is different. AlertPatch, as its name would suggest, actually replaces the system alerts with more informative reports, including a register dump for real Gurus.

Instead of

```
|                                                                        |
|    Software Failure.  Press left mouse button to continue.            |
|                                                                        |
|           Error: 8000 0003        Task: 002A4568                       |
|                                                                        |
--------------------------------------------------------------------------
```

you get something like

```
|                                                                        |
|    Software Failure.  Press left mouse button to continue.            |
|                                                                        |
| Error:   8000 0003                                                     |
|          CPU (68000) reported word access at odd address               |
|                                                                        |
| Task:    002A4568                                                      |
```

```
|          Background CLI                                              |
|                                                                      |
| Command: a_dodgy_program                                             |
|                                                                      |
| Registers:                                                           |
|     PC = ????????  SR = ????                                         |
|     D0 = 00003287  D1 = 43098764  D2 = 98327642  D3 = 00212F78       |
|     D4 = 00000000  D5 = 0000036A  D6 = 34089743  D7 = 0000000A       |
|     A0 = 00943643  A1 = 74364383  A2 = 00000000  A3 = 00032973       |
|     A4 = 43874343  A5 = 0028AB44  A6 = 000987BE  A7 = 002B9842       |
|                                                                      |
----------------------------------------------------------------------
```

(Unfortunately it is only possible to find the PC and SR registers for Recoverable Alerts. Also, there is currently no support for additional registers in the 68010+.)

Interested? Well go ahead and
install
it!

1.4 Installation

To install AlertPatch on your hard disk, simply drag the icon across to the appropriate place. You may, for instance, want to put it in your WBStartup drawer.

To install this guide file, simply drag its icon across to the appropriate place on your hard disk.

If you prefer to use the CLI, then you already know how to install it! You may want to consider adding AlertPatch to your s:user-startup file.

1.5 How to use AlertPatch

Normally you would have AlertPatch in your WBStartup drawer or as a command in your s:user-startup script. It is not currently possible to remove AlertPatch, although you can remove all reset handlers (including AlertPatch) using an anti-virus program such as BootX or a system control program such as Xoper. However, AlertPatch will never install more than one copy of itself.

If you do not include AlertPatch in your startup procedure, you can start it at any time by opening on its icon or running it from the CLI. It will automatically detach from the CLI so you do not need to use the Run command with it. AlertPatch will remain in memory until you remove all reset handlers or turn off the computer.

1.6 The History of AlertPatch

version 1.1, 26 November 1995

·~Now works when run from the Workbench. It was crashing before. Very sorry about this.

version 1.0, 12 November 1995

First public release.

1.7 Explanation of Some Common Alerts

Note to programmers: Where I have used the term 'object' it is not meant in the precise sense associated with object-oriented programming!

In my experience, the ten most common alerts are these:

Error code	Subsystem	Reason	Section
0100 0009	Exec	memory already free at FreeMem	3
0100 000C	Exec	MemList failed sanity check	3
8000 0003	CPU	word access at odd address	1
8000 0004	CPU	illegal instruction	2
8000 0005	CPU	division by zero	4
8000 0009	CPU	illegal $A instruction	2
8000 000A	CPU	illegal $F instruction	2
8100 0005	Exec	corrupt MemList	3
8100 0009	Exec	memory already free at FreeMem	3
8700 0004	AmigaDOS	reception of unexpected packet	5

And here are some in-depth explanations which will hopefully go some way to alleviating your frustration... Some notes apply only if the program at fault is your own.

1. Address error - 8000 0003

The message for this is 'word access at odd address'. What it means (technically) is that the processor has generated a memory address which it needs to fetch a word or long word from, and this turns out to be an odd number. The 68000 processor is unable to fetch any word or longword data from an odd address, and none of the 680x0 processors are able to read program code from an odd address.

There are three likely possible causes:

(a)~The program is intended for use only on 68020 or better CPUs (usually this would be stated in its documentation). PROGRAMMERS: Make sure your compiler isn't optimising for a processor you don't have!

(b)~Some of the program's data or structures are not properly aligned. PROGRAMMERS: Check the following three possibilities.
(i)~~~If it is an assembler program, then you have probably used a DC.B/DS.B directive which declares an odd number of bytes, then gone on to assemble word data or code. To fix this, insert the directive 'EVEN' after the problematic directive. Alternatively,

you may be using an odd-numbered displacement. If you have used Commodore's structure-defining macros, don't forget that you may need to include padding bytes in your structure definition.
(ii)~~If you are using NorthC, then you have run into a nasty bug in the compiler. This usually happens if you define and initialise a string array followed by another definition. My solution is to add the text '\0' at the end of the string. This simply pads the text to an even address without actually affecting the string's contents.
(iii)~Make sure you aren't using a compiler option which optimises for a processor you don't have...!
If none of these apply, this probably isn't the cause.

(c)~The program tried to allocate a resource or create an object, and was unsuccessful, but didn't check for this possibility and so proceeded to attempt access to the nonexistent object; or it attempted to access an object which had ceased to exist. Then it read a pointer supposedly contained in the object, which happened to be odd (because it wasn't really a pointer) and used it... PROGRAMMERS: Double-check your resource (de-)allocation, window opening/closing etc. If resources can be freed at various points in a program, it is often necessary to zero the resource pointer immediately afterwards. Then future checks will correctly identify the resource as unallocated. Unfortunately, these bugs can sometimes be very subtle, but then you probably realised that already...

2. Instruction errors - 8000 0004, 8000 000A, 8000 000B

These errors are all caused by the CPU reading in an instruction which it does not understand - which generally means that it isn't really supposed to be an instruction at all. The separate error codes for $A and $F exist because opcodes of the form $Axxx and $Fxxx were unused in the 68000 and are intended to be gradually used up in later processors. The exception handlers for these instructions can attempt to implement new instructions in software instead of just crashing. There are four likely possible causes:

(a)~The program is intended for use on a better CPU than that which you have. For example, the 68020 processor has many extra instructions which won't be recognised by a 68000, so a program intended for 68020s will crash with this error (if it doesn't check and politely fail). PROGRAMMERS: Make sure you aren't using a compiler option which optimises for a processor you don't have, or an assembler instruction which isn't supported by your processor.

(b)~The program has used a pointer to data as a code address. PROGRAMMERS: Try checking any hook, interrupt or patch addresses you are passing to functions or storing in structures.

(c)~A system patch, interrupt or hook has been installed, and the memory containing the code which handles this has been freed and re-used. PROGRAMMERS: Take a good look at your cleanup/exit code, and make sure any references to your code which are held by other programs have been removed before your program terminates (unless you have made special provisions).

(d)~The program is attempting to use more stack space than is available -

and so is over-running into code space. If it is not your program, check the documentation for any information on stack requirements. To change the stack space allocated for it, you must use either the Workbench 'Information' function or the CLI 'Stack' command (depending on which the program is started from) - see your Commodore manual for usage information. PROGRAMMERS: Check how much data you are putting on the stack. In C programs, all variables declared within a function which aren't of storage type 'static' or 'register' will be stored on the stack, so perhaps you should consider using dynamic memory allocation for some of this data, or enabling your compiler's dynamic stack allocation (it does have it, doesn't it? :-). Also, recursive function calling and some system calls can be real stack-killers.

3. Memory errors - 0100 0009, 0100 000C, 8100 0005, 8100 0009

The first and fourth of these codes give the message 'memory already free at FreeMem'. What this actually means is that the program tried to free memory (having finished using some memory space) which was already marked as free. So the program quite possibly had been using this memory to store its data during the time in which it had been marked as free and hence liable to be re-used by other programs! There are two likely possible causes of this:

(a)~The memory was freed once, and the pointer which the program used to access this memory was left as it was. Then the program did a little tidying up, and decided to free the memory which it no longer needed. But... the memory was already free! PROGRAMMERS: Track your memory (de-)allocation carefully. If your memory allocations are very complex, you should try clearing all memory pointers after the memory is freed, just as you should with pointers to any other resource or dynamic object.

(b)~The program tried to free memory using a pointer which was either not really a pointer or else was a pointer to data which were part of a larger block of memory (e.g. the program tried to free part of itself). PROGRAMMERS: PLEASE recheck those flamin' pointers AGAIN!

The other two codes occur if the system's list of free memory is somehow corrupt. The first is the real serious one, and it seems to be the only error guaranteed to crash the system every #%&$ing time. There are three likely possible causes:

(c)~An attempt was made to free memory which was already free - see (a) and (b).

(d)~A program has gone out of control and written junk all over memory, including the free memory list. This could result from attempting to use a larger block of memory than it has allocated for itself, or else it is using as a pointer something which isn't a pointer (does this sound familiar?). PROGRAMMERS: You should be ashamed of yourself! Seriously though, this can occur if you over-run the stack - see 2(d).

(e)~A program has attempted to free part of a memory block, rather than freeing the whole lot at once. PROGRAMMERS: Note that although the system usually won't complain if you try this, apparently in some cases it will crash.

(f)~A program has continued to use memory that it has freed. PROGRAMMERS: Don't free memory while you still have pointers to it!

The second code (0100 000C) seems to be a less serious case of the first error. I don't know why this error usually appears as a Recoverable Alert. I have on one occasion when this alert popped up found I had 200MB of free Chip RAM - hmm... Maybe this error results from wierd bugs in the system memory routines... but I doubt it. In fact, I have no idea why some corruption causes this whereas other corruption brings down the system with error 8100 0005. Sorry.

4. Division by zero - 8000 0005

Strangely enough, it is impossible to calculate the result of division by zero, since it is undefined... PROGRAMMERS: You should add code to your own program to deal with this ever-present problem.

5. Unexpected packet - 8700 0004

This usually means that the program is intended to be used from the CLI only, and has been started from the Workbench. There is a simple work-around, which is to use Workbench's 'Information' function on the program's icon, and add the tooltype 'CLI' to it. Then, whenever you open the icon, you will get a command-line requester (just as if you were starting a tool icon revealed by Show>All Icons). PROGRAMMERS: Use different startup code or compiler options - refer to your documentation for specifics.

1.8 Acknowledgements

Thanks to all those people out there who wrote Guru guides!

Thanks not a lot to those lamers whose fault it is I see alerts so flamin' often! (I'm excluding myself because the programs of mine which crash are generally pre-release versions.)

Thanks to ESCOM and Amiga Technologies for bringing the Amiga back to the street!

Thanks to Carl Sassenrath for creating Exec and the Guru in the first place.

Thanks not a lot to Commodore for removing the Guru in 1990!

1.9 How and Why to Contact Me

Please write to me if you have any comments, suggestions, bug reports or programming hints to make.

Snail mail:
Ben Hutchings
43 Harrison Close

Reigate
Surrey RH2 7HS
ENGLAND

E-mail (Internet): benjamin.hutchings@worc.ox.ac.uk

For up-to-date information on AlertPatch and other software by me see my web page: http://sable.ox.ac.uk/~worc0223/freeware/