MuRedox

Thomas Richter

COLLABORATORS			
	TITLE : MuRedox		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	January 13, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MuRedox

1.1 MuRedox Guide

´### ,#. ,#### ,### ####’ #### ,#### ,###’ ####’ #### _____ ,#### ,###’ | | ####’ #### | ___ ___ | ___ ,#### ,###’ ---- | | | | | | |___ ####’ #### | | | | | | | ,#####. ,###’ . | |___| |___| |_ ___| ####’##. ,#### ,## ,#### ########’ # ,##’ ####’ `####’ `###’ ,#### ##### © 2001-2002 THOR - Software, ####’ Thomas Richter `##’

MuRedox Guide

Guide Version 1.03 MuRedox Version 40.3

Warning:

MuRedox requires a lot of "magic" to do its job, and hence have to alter the emulation process of those instructions of the 68040 and 68060 that are non-native and require interaction of the 68040 resp. 68060 library. It might not cooperate with all exising programs due to its very hacky nature. Before you play with MuRedox, make sure you read the installation and requirements chapters of this guide. Typically, noone except speed-maniacs need to run this program, and even then the expected effects are typically marginal.

© THOR-Software

Thomas Richter

Rühmkorffstraße 10A

12209 Berlin

Germany

EMail: thor@math.tu-berlin.de

1.2 The THOR-Software Licence

The THOR-Software Licence (v2, 24th June 1998)

This License applies to the computer programs known as "MuRedox" and the "MuRedox.guide". The "Program", below, refers to such program. The "Archive" refers to the package of distribution, as prepared by the author of the Program, Thomas Richter. Each licensee is addressed as "you".

The Program and the data in the archive are freely distributable under the restrictions stated below, but are also Copyright (c) Thomas Richter.

Distribution of the Program, the Archive and the data in the Archive by a commercial organization without written permission from the author to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities).

However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

(i) Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

(ii) Distributing the Program on a CD-ROM, provided that

a) the Archive is reproduced entirely and verbatim on such CD-ROM, including especially this licence agreement;

b) the CD-ROM is made available to the public for a nominal fee only,

c) a copy of the CD is made available to the author for free except for shipment costs, and

d) provided further that all information on such CD-ROM is re-distributable for non-commercial purposes without charge.

Redistribution of a modified version of the Archive, the Program or the contents of the Archive is prohibited in any way, by any organization, regardless whether commercial or non-commercial. Everything must be kept together, in original and unmodified form.

Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS", WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF YOU DO NOT ACCEPT THIS LICENCE, YOU MUST DELETE THE PROGRAM, THE ARCHIVE AND ALL DATA OF THIS ARCHIVE FROM YOUR STORAGE SYSTEM. YOU ACCEPT THIS LICENCE BY USING OR REDISTRIBUTING THE PROGRAM.

Thomas Richter

1.3 What's the MMU.library?

All "modern" Amiga computers come with a special hardware component called the "MMU" for short, "Memory Management Unit". The MMU is a very powerful piece of hardware that can be seen as a translator between the CPU that carries out the actual calculation, and the surrounding hardware: Memory and IO devices. Each external access of the CPU is filtered by the MMU, checked whether the memory region is available, write protected, can be hold in the CPU internal cache and more. The MMU can be told to translate the addresses as seen from the CPU to different addresses, hence it can be used to "re-map" parts of the memory without actually touching the memory itself.

A series of programs is available that make use of the MMU: First of all, it's needed by the operating system to tell the CPU not to hold "chip memory", used by the Amiga custom chips, in its cache; second, several tools re-map the Kickstart ROM to

faster 32Bit RAM by using the MMU to translate the ROM addresses - as seen from the CPU - to the RAM addresses where the image of the ROM is kept. Third, a number of debugging tools make use of it to detect accesses to physically unavailable memory regions, and hence to find bugs in programs; amongst them is the "Enforcer" by Michael Sinz. Fourth, the MMU can be used to create the illusion of "almost infinite memory", with so-called "virtual memory systems". Last but not least, a number of miscellaneous applications have been found for the MMU as well, for example for display drivers of emulators.

Unfortunately, the Amiga Os does not provide ANY interface to the MMU, everything boils down to hardware hacking and every program hacks the MMU table as it wishes. Needless to say this prevents program A from working nicely together with program B, Enforcer with FastROM or VMM, and other combinations have been impossible up to now.

THIS HAS TO CHANGE! There has to be a documented interface to the MMU that makes accesses transparent, easy and compatible. This is the goal of the "mmu.library". In one word, COMPATIBILITY.

Unfortunately, old programs won't use this library automatically, so things have to be rewritten. The "MuTools" are a collection of programs that take over the job of older applications that hit the hardware directly. The result are programs that operate hardware independent, without any CPU or MMU specific parts, no matter what kind of MMU is available, and programs that nicely co-exist with each other.

I hope other program authors choose to make use of the library in the future and provide powerful tools without the compatibility headache. The MuTools are just a tiny start, more has to follow.

1.4 What's the job of MuRedox?

MuRedox is a mmu.library speedup patch that replaces instructions that require emulation on a 68040 or 68060 by an emulating instruction sequence. MuRedox replaces therefore the similar tools "OxyPatcher" and "CyberPatcher".

The unimplemented instructions of the 68040 and 68060 - mainly FPU instructions - generate an exception and need to be emulated by the 68040 resp. 68060.library. This is the job of the so-called "FPSP routines" (floating point support package) within the CPU libraries. MuRedox detects these instructions as soon as they generate the emulator exceptions, runs a "just-in-time" compiler that generates a "stub replacement routine" for this specific instruction and patches the replacement routine into the running program. Hence, MuRedox replaces the overhead of the emulator trap on the next use of the same instruction sequence.

See also: Caveats of MuRedox.

1.5 Caveats when using MuRedox

MuRedox is pretty much an advanced hack and not a "system improver" patch. If your system requires optimal stability, the use of "MuRedox" is not recommended. The very same goes of course for the similar programs "OxyPatcher" and "CyberPatcher" for the very same reason, even though their authors might have "forgotten" to warn you about the caveats of their programs.

Top reasons why not to use this program:

It is a hack. MuRedox replaces program code on the fly, hoping that all will go well. This need not to be the case - especially commercial programs may keep a checksum over their code and may fail on purpose if their code gets altered. MuRedox will perform such code modifications, and it is beyond its ability to detect checksums within the patched program.

MuRedox will therefore not work for all programs - some incompatibilities should be expected.

MuRedox cannot be removed once it is run. The problem is that it cannot know whether the program code that got patched is still in use or not.

MuRedox increases the stack usage of the patched programs. This is because the stack the 68040 or 68060 emulator traps would have allocated from the system supervisor stack is now allocated from the program stack. Typically, MuRedox will require less than 400 bytes additional stack for the involved mathematical functions, much less for basic instructions. If MuRedox seems to cause program crashes, incrementing the program stack might help. See also the troubleshooting section.

If you need faster programs, you should rather:

Ask the vendor for a 68060 or 68040 specific release of the program that does not require the software emulated instructions of the 68040 resp. 68060. Typically, these versions will run faster than a 68020/68030 version with MuRedox, anyhow.

Remember: Programs are made fast by fast and smart algorithms, not by your favourite speedup-patch. MuRedox will give some speed impact, in realistic situations in the range of at most 10%. Specific benchmarks may show more dramatic improvements, but they typically test situations that are untypical in a real-life situation. Motorola choose less frequently used instructions for software emulation in first place, hence improvements should be expected to be marginal.

1.6 Requirements to run MuRedox

Since MuRedox patches unimplemented instructions of the 68040 or 68060, it requires:

At least an 68040 or an 68060. A 68030 or below implements all instructions anyhow and does not require emulator traps at all.

It also requires the mmu.library to setup the special memory mapping for its emulator jump table.

Furthermore, the fpsp.resource. This resource contains the program code that emulates most unimplemented math functions. This resource is made availavailable by the mmu.library-based 68040 and 68060.library, you don't get it as a separate file. To make the fpsp.resource available to the system, one of the mmu.library based processor-driver libraries must be loaded.

Therefore, installation of the "MuLib" CPU libraries is mandatory. Please see the "MMULib.lha" package on Aminet how to install them.

1.7 Installation of MuRedox

First, make sure that your system fulfills the requirements indicated in this guide. This means that you might have to install the mmu.library based CPU driver libraries - the 68040.library or the 68060.library. You find these in separate archives in the aminet. You might also have to install the "mmu.library" if you do not yet use it.

The "MuRedox" file within this archive can be copied to wherever you want to keep it. If you want to use it on a frequent basis, you should either run the program from the startup sequence or from the WBStartup drawer of your boot partition. It doesn't require any shell arguments.

NOTE: Once MuRedox is running, it cannot be removed anymore.

In case problems should arise with MuRedox, I would be glad to receive some feedback by the output of the "FPSPSnoop" program within this archive. Just copy it to wherever you want it. You should furthermore copy the "disassembler.library" to LIBS:, and should install the "Sashimi" program from the Aminet. It is not included in this archive. For further details how to use these programs, please see the the troubleshooting chapter.

1.8 Troubleshooting MuRedox: If things go wrong

In case of trouble:

...or if you think that MuRedox does not speedup your favourite program.

MuRedox is a pretty tricky program, it might still be that this release contains some bugs. The same goes for situations where you think that MuRedox "misses" some instructions that might slowdown your favourite application. For tracing down these problems, you need the "FPSPSnoop" program and the "disassembler.library" within this archive, and possibly a second computer and a null-modem cable or the "Sashimi" program. The latter can be found in the Aminet in a separate archive and is not included.

Install the disassembler.library into LIBS:, keep Sashimi and FPSPSnoop whereever you want. Run Sashimi first. Output of FPSPSnoop will then appear in the Sashimi window. If you do not run Sashimi, the FPSPSnoop log will be written out over the serial port at 9600 baud, 8 bits, one stop bit, no parity.

Usage of FPSPSnoop is pretty simple: Run it from the shell without arguments.

The order how programs are run determinates the logic of what is "snooped" by FPSPSnoop:

Run FPSPSnoop first, MuRedox later: Then FPSPnoop will only snoop instructions that are not handled by MuRedox. This might be useful to detect cases where MuRedox misses some instructions. Use this sequence in case you believe that MuRedox does not provide emulations for instructions that are important for your application, sent me the output.

Run MuRedox first, FPSPSnoop (and Sashimi) later: This will snoop all instructions that will be emulated before they are replaced by MuRedox. This is useful when you suspect that one of the emulation routines of MuRedox is broken and your system crashes if MuRedox is run.

Another popular reason why programs might crash is because they run out of stack as MuRedox increases the stack usage of the patched programs. This is because the stack the 68040 or 68060 emulator traps would have allocated from the system supervisor stack is now allocated from the program stack. Typically, MuRedox will require less than 400 bytes additional stack for the involved mathematical functions, much less for basic instructions. If MuRedox seems to cause program crashes, incrementing the program stack might help as well.

In either case, I would need the output of FPSPSnoop to fix any problem!

1.9 Command line options and tooltypes

MuRedox can be started either from the workbench or from the shell. In the first case, it reads its arguments from the "tooltypes" of its icon; you may alter these settings by selecting the "MuRedox" icon and choosing "Information..." from the workbench "Icon" menu. In the second case, the arguments are taken from the command line. No matter how the program is run, the arguments are identically.

NOTE: MuRedox cannot be removed once it is running. This is intentional as MuRedox has no control over which programs got altered and whether these alterations are still active.

MuMapRom EMULATORSIZE=EMUSIZ/N,SHOWFILLLEVEL/S,SHOWPATCHEDINSTRS/S

EMULATORSIZE

Size in K bytes of the output buffer for the just-in-time compiler. Emulated instruction sequences go into this buffer. If this buffer overruns, MuRedox will no longer be able to replace further emulated instructions.

Defaults to 64K which has shown to be more than sufficient for normal operation; typically not more than 16K are required.

SHOWFILLLEVEL

Displays the fill level of the emulator output buffer in percent. This requires that MuRedox has been run before already.

SHOWPATCHEDINSTRS

Shows a disassembly of the database of all patched instructions for the curious. Hence, the output presents the collection of all instruction words MuRedox was able to replace by a emulation routine.

When started from the workbench, MuRedox knows one additional tooltype, namely:

WINDOW=<path>

where <path> is a file name path where the program should print its output. This should be a console window specification, i.e. something like

CON:0/0/640/100/MuRedox

This argument defaults to NIL:, i.e. all outputs will be thrown away.

1.10 Internals: The Advanced Hacker's Guide To MuRedox

This chapter contains some implementation details of MuRedox the advanced hacker might find interesting.

Thanks to its "just in time compiler", MuRedox is able to replace almost all instructions that would cause emulator traps. MuRedox does not come with a fixed set of stub-routines that are just patched in, it creates these routines as soon as the problem is detected. Therefore, it is maybe easier to say what MuRedox does not emulate rather than what it does...

Integer instructions that get replaced:

mulu 64 bit variant, all addressing modes, 68060 only, native on 68040 muls 64 bit variant, all addressing modes, 68060 only, native on 68040 divu 64 bit variant, all addressing modes, 68060 only, native on 68040 divs 64 bit variant, all addressing modes, 68060 only, native on 68040 movep all addressing modes, 68060 only, native on 68040 cmp2 all addressing modes, 68060 only, native on 68040

Integer instructions that do not get patched:

chk2 This instruction is especially useful for debugging as it will cause a "trap" if a condition to be checked is not fulfilled. Replacing this instruction may confuse an active debugger. Except that I haven't seen any program that makes use of it. This is a native instruction on the 68040. cas (with odd addresses only) Not useable on amiga hardware due to the unsupported read-modify-write cycle that might break DMA access. This is a native instruction on the 68040. cas2 Not usable on Amiga hardware due to the unsupported read-modify-write cycle. This is a native instruction on the 68040.

Integer addressing modes:

All supported except (a7)+ as target and -(a7) as source. Both addressing modes break the logic of a stack and won't work in a multitasking environment either. The 68040 and 68060 emulator core doesn't support them either.

FPU general instructions:

The following are handled by forwarding the emulation to the fpsp.resource:

facos all addressing modes fasin all addressing modes fatan all addressing modes fatanh all addressing modes fcos all addressing modes fcosh all addressing modes fetox all addressing modes fetoxm1 all addressing modes fgetexp all addressing modes fgetman all addressing modes fint all addressing modes, 68040 only, native on 68060 fintrz all addressing modes, 68040 only, native on 68060 flog10 all addressing modes flog2 all addressing modes flogn all addressing modes flognp1 all addressing modes fmod all addressing modes frem all addressing modes fscale all addressing modes fsin all addressing modes fsincos all addressing modes fsinh all addressing modes ftan all addressing modes ftanh all addressing modes ftentox all addressing modes ftwotox all addressing modes

The following are handled with special emulation routines within MuRedox:

fmovecr all constants, with special versions for 0,1 and 10. fdbcc all addressing modes, 68060 only, native on 68040 fscc all addressing modes, 68060 only, native on 68060

Supported floating point addressing modes:

All including #immediate.x(extended precision) for which emulation routines are generated as well for the 68060, and excluding "packed decimal".

Instructions that do not get patched over:

ftrapcc 68060 only, native on 68040 Mainly useful for debugging, this instruction causes an exception if the indicated FPU condition is met. A debugger might depend on this specific instruction, therefore this is left alone.

fmovem.x with dynamic register list (in CPU registers), 68060 only, native on 68040.

fmovem.l multiple control registers with immediate operand and more than one control register, 68060 only, native on 68040.

I haven't seen any program using these instructions. If you need them emulated, prove me that they get used by running FPSPSnoop.

Addressing modes that are not replaced by emulation code:

#immed.p (packed decimal) and any other packed decimal operation.

Since these instructions are slow even on a 68881/82 CPU, they are hardly ever used in speed-critical situations. If you really need them emulated, please tell me the program that uses them.

1.11 Frequently Asked Questions: Did you check these?

This chapter presents some frequently asked questions for typical problems that may show up if you run MuRedox.

Q: When I try to run MuRedox, I always get the error message "MuRedox failed : requires the fpsp.resource", but I installed your 68040 or 68060.library which should contain this resource.

A: Some boards come with the 68040 resp. 68060.library in ROM such that a disk-based version is never loaded. To run MuRedox on these boards, the ROM based CPU driver library must be disabled and a proper MMU-Configuration has to be setup. Please follow the MMU.guide of the full mmu.library distribution how to do this.

Q: I tried FPSPSnoop and I found that MuRedox does not patch some instructions that cause "Unimplemented FPU data type" exceptions. Furthermore, I see here some "Overflow" or "Underflow" exceptions, but the programs I'm using work fine. Should I be worried about this?

A: No, this is normal, and MuRedox cannot patch the instructions because it is not the FPU-instruction that causes this exception, but rather its arguments. The problem is that the 68060 requires software support for so-called "denormalized" numbers which are near the precision limit of the selected FPU data type. Whenever the 68060 FPU encounters a denormalized number as input argument of an otherwise well supported data type, an exception is raised and support of the 68060.library is required. Similarly, if the FPU generates a result that would not fit into the destination data type without denormalizing, an "overflow" or "underflow" exception is raised to get support from the 68060.library. Since it is not the instruction that requires help, but its arguments that are out of bounds, MuRedox need not to patch the instruction. Typically, the FPU will not encounter any problem when executing this instruction except for very rare cases when the arguments are near the limit of the precision. Patching an instruction by an emulation sequence that would only speed up exceptional cases, but would slow down the generic ones doesn't make much sense.

Q: You write that the emulator buffer of MuRedox has a fixed size and cannot grow while MuRedox is operating. Doesn't this mean that MuRedox runs out of free space provided I run it long enough?

A: No. The point is that MuRedox re-uses the emulation routine for an unimplemented instruction once it learned how to emulate this specific instruction. This means that sooner or later the MuRedox output buffer will saturate as no new unknown instructions must be learned - and all required emulation routines have been generated. In fact, the amount of different instructions MuRedox can learn is limited anyhow by technical reasons beyond reach of MuRedox by 5461 - the same limit applies to similar patch programs as well. This means that an emulator buffer of about 512K or more will, in fact, never overrun.

1.12 History

Release 40.1:

This is the first official release.

Release 40.2:

Added the SHOWFILLEVEL option.

Release 40.3:

Added the SHOWPATCHEDINSTRS switch.

Release 40.4:

Updated the MOVEP emulation a bit.