palis

Hans Bühler and Codex Design Software

COLLABORATORS

	TITLE : palis		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Hans Bühler and Codex Design Software	February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 palis **1**

1.1 Palis V1.00 - The PatchLib solution... 1

1.2 Introduction to Palis... 1

1.3 Requirements & Installation 5

1.4 Bugs & Problems when using PALIS 6

1.5 ViewPalis - information for PALIS users 6

1.6 How to patch a library 7

1.7 PALIS/ViewPALIS source code available. 9

1.8 Copyrights 10

1.9 Alt F4 ! 10

1.10 Have a chat with me. 10

Chapter 1

palis

1.1 Palis V1.00 - The PatchLib solution...

```
                                 ·C·O·D·E·X· ·D·E·S·I·G·N· ·S·O·F·T·W·A·R·E·
                                  presents:

                                 PALIS V1.00
                            the patchlib solution
```

Introduction

Requirements & Installation

Bugs & Problems

ViewPALIS

How to patch a library

The source

Copyrights ?

Author

```
        written 14.10.1995 by Hans Bühler, Codex Design Software
This program might be freely distributed.
```

1.2 Introduction to Palis...

```
                                                INTRODUCTION

                            1. Who needs Palis ?
```

Everyone who has an Amiga possibly need this program. But this guide has been written for developpers (everybody who knows about problems concerning library-patching should pass this paragraph) because I assume that this program should only be distributed along with other programms that make use

of its features. On the other hand, if you think that you might need it, it won’t cause any trouble to you but tries to protect your system from some dangerous actions.
In that case I would suggest you simply install PALIS since it doesn’t use too much of your memory (~10K) but may help you not to crash your machine.
For people who know what I’m talking of: Palis allows you to remove any patch having been made to any of your system-libraries at any time. Up to now, AmigaOS could seriously be shot down by such actions. Here’s where PALIS comes in effect...

2. What’s the problem ?

In simplicity, PALIS tracks all patches made by exec/SetFunction() and will remember them. Then, a program tries to remove that patch, PALIS will ensure that former calls to the patched function will always work as they’re meant to.
Here is a little example (for all who don’t knwo what I’m talking of):

```
SETTING:
  Two programms: "first" is run at first,
                 "second" afterwards.

PLOT:
  1. Program "first" makes a patch to intuition.library/OpenWindow() (-204)
     It sets its function 'firstOpenWin()' there.
     This function does something and jumps into the original function,
     I gonna call it 'intOpenWin()' from now on.
  2. Program  "second"  makes another patch to OpenWindow() and installes
     its function 'secondOpenWin()'~there.
     This functions again does something cool and calls the old function
     => it calls 'firstOpenWin()'~since this function was set to
     intuition.library before by "first".

  3. Any programm opens a window.
     It calls 'secondOpenWin()' which calls 'firstOpenWin()' which calls
     'intOpenWin()'. Everything is fine since "first" and "second" are good
     programs and do not cause any trouble.
  4. Window is closed.

 Now, pay attention...

  5. User thinks that the "first" program isn't a good program and removes
     it.
     "first" has to remove its patch to intuition/OpenWindow() and sets ITS
     old pointer as returned from exec/SetFunction() to intuition.library.
     => now the original pointer is restored.
  6. User starts a program that opens a window and wants program "second"
     to make some stuff with that window.
     Unfortunately, "first" has re-installed the original 'intOpenWin()'.
     Therefore 'secondOpenWin()'~won't be called since 'intOpenWin()'~is
     now set to the original vector of intuition.library.
     The user may wonder why "second" doesn't do its job anymore.
  7. "Well,", user says, "it doesn't work... then I gonna quit that program
     "too."
     => Program "second" tries to de-install its patch... it sets ITS old
     function 'firstOpenWin()'~(which isn't there anymore since "first"
     has been removed times ago...).
     It installs 'firstOpenWin()'~to intuition.library because "second"
     doesn't know that the program "first" is dead.
  8. Now user opens a window... the program calls 'firstOpenWin()'...
     BUT THERE ISN'T ANY 'firstOpenWin()'~anymore... ;-(
     => BOOOOOOM

RESULT:
     Software failure
     Task held...
```

Of course there're are various tricks to avoid such crashes. Most programs will not install their own funcion pointer to the library but a pointer to a small piece of memory which looks like this (assembly):

```
           $00 UWORD jmp
           $02 APTR  func
```

where 'func' points to the function which the program wants to install. If it removes itself now, the 'func'~entry is set to the previous (old) function returned from exec/SetFunction(). That works but each time you install such a patch some memory (8 Bytes) will remain in memory unused. I doesn't bother about that 8 bytes but due to the handling of AllocMem() (AllocVec() uses 12 bytes) it will cause memory-fragmentation.
Other programs (like MagicMenu from Martin Korndörfer) send a request to the user allowing him to choose between simply deactivating the program or forcing the program to remove itself (what this means is been explained in my little story above ;^).

3. And Palis...?

Here's where PALIS comes into effect: PALIS will in fact do nothing else like I explained above: It will catch each exec/SetFunction() call and will not install the function which is to be installed but a 'dummy' function as described above, automatically.
But it has a great advantage: Due beeing able to recognize _each_ exec/SetFunction() call it will note if a program wants to remove its patch using exec/SetFunction(). Moreover it can detect whether the entry (the dummy function block above) is needed anymore.
Again the same story as above during PALIS is active (PALIS jobs are marked by '>'):

```
SETTING:
  Two programms: "first" is run at first,
                 "second" afterwards.

PLOT:
  1. Program "first" makes a patch to intuition.library/OpenWindow() (-204)
     It sets its function 'firstOpenWin()' there using exec/SetFunction()
     which points to a PALIS-function.
     > PALIS will recognize that this is a true patch (no attempt to remove
     > a previously installed function) because it tracks all functions that
     > have been made to intution/OpenWindow().
     > Hence it will generate a little dummy-function (see above - again)
     > which will be installed to intuition/OpenWindow() which will simply
     > call 'firstOpenWin()'.
     > Moreover PALIS will store the result from the original
     > exec/SetFunction().
     > This old funtion 'intOpenWin()'~will be returned to "first" thus it
     > can execute the original function if it needs to do that.
```

```
    The new function does something and jumps into the original function,
    I gonna call it 'intOpenWin()' from now on.
 2. Program "second" makes another patch to OpenWindow() and installes its
    function 'secondOpenWin()'~there.
    > Again, PALIS notes that it is a new patch.
    > 'secondOpenWin()'~will be installed as 'firstOpenWin()'~has been.
    > Because the function currently set to intuition/OpenWindow() is _not_
    > 'firstOpenWin()'~but a dummy function from PALIS this functionaddress
    > will be returned to "second".
    This functions again does something cool and calls the old function
    => it calls the dummyfunction installed by PALIS which calls
    'firstOpenWin()' since this function was set to intuition.library
    before by "first".

 3. Any programm opens a window.
    > It calls a dummyfunction which calls 'secondOpenWin()'.
    > 'secondOpenWin()'~does something and calls a dummyfunction which calls
    > 'firstOpenWin(). 'firstOpenWin()' calls the original 'intOpenWin()'.
    => Everything is fine since "first" and "second" are good
    programs and do not cause any trouble.
 4. Window is closed.

Now, pay attention...

 5. User thinks that the "first" program isn't a good program and removes
    it.
    "first" has to remove its patch to intuition/OpenWindow() and sets ITS
    old pointer as returned from exec/SetFunction() to intuition.library.
    > PALIS finds that old pointer in its internal lists thus finds out
    > that "first" wants to remove its function from the library.
    > Additionally, PALIS notes that there's another patch which had been
    > installed after "first" made its patch.
    > Therefore PALIS won't do anything with intuition/OpenWindow() but
    > replaces the 'func'~pointer in the dummyfunction by the function
    > which had been replaced by "first". This is 'intOpenWin()'.
 6. User starts a program that opens a window and wants program "second"
    to make some stuff with that window.
    > Since the dummyfunction for 'secondOpenWin()'~is still set to
    > intuition/OpenWindow() this dummy-function is been called which
    > jumps right into 'secondOpenWin()'. 'secondOpenWin()'~does its work
    > and calls the function from which it assumes to be the original
    > function. Actually, this function is the dummyfunction for the "first"
    > patch. It jumps in there but the dummyfunction does nothing else than
    > jumping into the original 'intOpenWin()'... luckily !
    The user may wonder why "second" did its job well.
 7. "Well,", user says, "fine, it works ... but..." and removes "second"
    from his system.
    > "second" calls exec/SetFunction() with ITS old function (which is
    > the "first" dummyfunction)
    > PALIS notes that "second" wants to remove its patch. Moreover
    > it recognizes that no further patches are followed by that one.
    > That means that the "second" dummyfunction won't be needed anymore.
    > Additionally, the "first" dummyfunction is not needed !
    > PALIS will install the original function 'intOpenWin()'~to
    > intition/OpenWindow() and will free all memory having been used to
    > track all these patches.
    > => no more memory is spent for 'dead'~dummyfunctions.
```

```
  8. Now user opens a window... the program calls 'intOpenWin()'...
     => Everything is fine !

RESULT:
     A new window.
```

You may take into account that PALIS will _not_ use much more memory than needed, actually. Since it's a very small program you should use it to prevent your system from beeing crashed by such things as described above.

BUT: Beeing a programmer, you don't have to do to anything else than calling SetFunction() once to install your patch and twice to remove it~!!! Therefore PALIS will also work with programs that are not designed to cooperate with PALIS.

Click here for further information about that...

4. Everything fine ?

Well, I think PALIS helps to solve the problem described above. There's another kind of problem that PALIS cannot solve for you:
Is there any task still using your function ?
That means: If you savely remove a patch from any library-function it's not sure whether there might be ONE program that is still running your function ! For this, there's no reliable solution.

1.3 Requirements & Installation

SYSTEM REQUIREMENTS

Palis

Palis needs Kickstart V2.04 or higher to run.

ViewPalis

ViewPalis (an additional program) needs the following libraries to run:
icon.library V37+, diskfont.library V37+
reqtools.library (c)Nico François will be used if available.

INSTALLATION

- Copy Palis somewhere in your path.
- Copy ViewPalis somewhere in your path.
- Make sure that Palis will be started _before_ any other programs that patch libraries (execpt those that you won't remove ever - like Setpatch etc.)
- Copy this poor guide somewhere you want.

I may add an installation script for further versions but it's a fairly simple job to do for all of us !

1.4 Bugs & Problems when using PALIS

BUGS

I think that PALIS itself will do its job without any bugs. Due its a simple program and I was using it for three month now I expect it wouldn't have problems anymore. Please note that PALIS won't do any patch-tracking when it runs out of memory. Then it returns to the original system-behaviour.

PROBLEMS

Alien software

There could occur several problems with alien software having not prepared to work with PALIS:

a) A program uses its own dummy-function to avoid conflicts with its patches. That means that this memory won't be freed and on the other hand that PALIS will never remove its own dummy-function while assuming that the program hasn't tried to quit yet.
Therefore some memory will be wasted.

b) Some programs check the current library-vector before they remove their patches. They check whether these vectors are unlike their own function-addresses and will reject to quit if they think so. They will of course NOT find their own funtion-addresses their due PALIS has put its very own functions in there...

PALIS problems

c) A program may cause trouble if it relies of PALIS having been installed. Programmers should _always_ check whether PALIS is active when they want to install a patch (and make appropiate steps... a warning etc.).
See
example code
for such things.

d) PALIS will _not_ save your patches of other things than beeing overwritten and stuff. Note that _you_ are still responsible for the cases that other programs do actually execute your function while you are about to remove it.
See
example code
for such things.

e) You are not allowed to quit PALIS !!!!!!

f) Making patches to dos/Delay(), exec/Obtain[Attempt]Semaphore(), exec/Obtain[Attempt]SemaphoreShared, exec/ReleaseSemaphore(), could be dangerous since PALIS will use them itself. Check out whether it works.

1.5 ViewPalis - information for PALIS users

```
                                VIEWPALIS

 I  added  this  little  external  program  to the archive using it you may
determine how PALIS works or what it had done for you that far.

                                   GUI

Here’s the GUI (designed using GadToolsBox V2.0c ©Jaba Development):

  +-+------------------------------+-+-+
  +·| ViewPALIS V1.00 hotkey = xxx | | | <= title. hotkey might be adjusted
  +-+------------------------------+-+-+    using tooltype CX_HOTKEY=xxx
  |                                    |
  |           Current patches:         |
  |                                    |
  | +------------------------------+-+ |
  | +                              | | |
  | +                              | | | <= list of currently known patches.
  | +                              | | |    patches that have been removed
  | +                              | | |    and which are still simulated
  | +                              | | |    are marked <removed>
  | +                              |#| |
  | +                              |#| |
  | +                              |#| |
  | +                              |#| |
  | +------------------------------+-+ |
  |                                    |
  |   [Hide] [Update][About] [Close]   | <= action gadgets...
  |                                    |
  +------------------------------------+

 Hide:   Closes the window but keeps ViewPALIS active.
 Update: Since ViewPalis just takes a copy of PALIS internal data,
         you may want to "update" the list.
 About:  ;^)
 Close:  Ends up ViewPALIS

                                TOOLTYPES

                   These tooltypes are known to ViewPALIS:

 CX_POPUP:  Open window when ViewPALIS is been started.
 CX_HOTKEY: Hotkey to re-open the gui if ViewPALIS is "hidden"
            (Default: "lalt lshift p").
 CX_PRI:    Priority to load ViewPALIS (when loading workbench)
            and pri of commodities job.
 WINX,WINY: Last window position (will automatically be saved for you).
 DONOTWAIT: Workbench shouldn’t wait for ViewPALIS having been finished
            (You cannot disable that ;^)
```

1.6 How to patch a library

```
                         SOLUTIONS FOR A SMALL PLANET
```

```
 Well, here’s a small code from which I assume to be as safe as possible:

        :
        :
        :

struct Semaphore sem;                     // semaphore for function

/*
   Setting a function
   ------------------
*/

APTR SetFunc(struct Library *lib,         // library you want to patch
             WORD offset,                 // offset
             APTR newFunc,                // your new function
             struct SignalSemaphore *sem) // address of an unused signalsem.
{
   APTR old;

   if(!( FindSemaphore(PALIS_SEMAPHORE_NAME) ))
   {
     ... palis is not been loaded...
     ... take appropiate steps ...
   }

   InitSemaphore(sem);
   old = SetFunction(lib,offset,newFunc);
   return old;
}

/*
   Removing a function
   -------------------
*/

void RemFunc(struct Library *lib,         // library you want to unpatch
             WORD offset,                 // offset
             APTR oldFunc,                // old function from SetFunc()
             struct SignalSemaphore *sem) // address of signalsem.
{
   SetFunction(lib,offset,oldFunc);

   CacheClearU();                         // clear program cache !
   ObtainSemaphore(sem);                  // wait till function finished
   Delay(1);                              // wait till rts worked..
}

/*
   New function protowork
   ----------------------
*/

ULONG NewFunc(...)                        // your newfunction should look
{                                         // like this...
   ULONG ret;
   ObtainSemaphoreShared(sem);

   ret = ... walk around... jump into old...

   ReleaseSemaphore(sem);

   return ret;
}

        :
        :
        :

 This work-around will install functions for you !
```

1.7 PALIS/ViewPALIS source code available.

```
                 You  may  want to have a look at the source of PALIS/ViewPALIS.   ↩
                    Note that
any  access  to internal data structures is forbidden execpt when following
the  rules  defined  in  PALIS.h.   However, I don’t think that you need to
access PALIS itself.  Keep your fingers off !

src/
  PALIS.h         - Include for programs that do want to access PALIS.

  Include.h       - Includes used by PALIS (little more since I always copy
                    the same file...;^)
  pl.h            - Include for PALIS.exe

  Basic.c         - Basics as requesters and stuff.
  Com.c           - commodities stuff.
  Main.c          - Mainloop (actually doesn’t do much)
  pl.c            - Initialisation object.
  SetMan.c        - PALIS installation/work/etc.
  SCOPTIONS       - Options for SAS/C

src/vpl/
  plView.h        - include.
  PalisViewGUI.c  - src from GadToolsBox; fixed using gtp 1.07
  PalisViewGUI.h  - include from GTB
  basic.c         - requesters and stuff.
  Main.c          - mainloop.
  Com.c           - commodities stuff.
  Gui.c           - managing the gui.
  Lists.c         - list work.
  plView.c        - main.
  ttype.c         - argument parsing (object not yet included; might be
                    copied from the author if needed).


                See copyright notes !
```

1.8 Copyrights

```
                Disclaimer

 The  author  cannot be held liable for the suitability or accuracy of this
manual  and/or  the  program(s)  it  describes.  Any  damage  directly  or
indirectly caused by the use or misuse of this manual and/or the program it
describes is the sole responsibility of the user her/him self.

 Copyrights

 PALIS V1.00 & ViewPALIS V1.00 have been written by
               Hans Bühler
                for common
 use. This is real freeware. Do what you want.
```

1.9 Alt F4 !

```
  Was sagt ein Intel-Entwickler zu Neujahr 1996 ?
  - Schönes neues 1995,9997889998899989999999.....
  oder:
  - was ist kooperatives Multitasking ?`
    antwort: scheiße.
```

1.10 Have a chat with me.

```
                                  .
                                  .
                                  .
                                  .
                                  :
                                  :
                                  :
                                  :
                                  :
                                  :
                          ........:
                          :
      h a n s b u e h l e r : c o d e x d e s i g n s o f t w a r e
                          :
                          :.....................
                                               :
                          [email]              :
         codex@stern.mathematik.hu-berlin.de (Hans Bühler)
                            [&]                :
           codex@kadewe.artcom.de (Christian Würdemann)
                                               :
                          [smail]              :
                        Hans Bühler            :
                Kirchstr. 22 - 10557 Berlin 21
                          Germany              :
                                               :
```

```
                                     .............:
                                     :
                                     :
                                     :
                                     :
                                     :
                                     .
                                     .
                                     .
```