snoopy

COLLABORATORS

	TITLE : snoopy	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 27, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

snoopy

1.1 Snoopy Version 1.4 - Help Guide

Snoopy V1.4 -- Monitors any Amiga function calls

This is I-WANT-TO-BE-FREE-WARE

1.2 INTRODUCTION

This tool enables you to monitor library function calls - of any library you wish. The idea of course came from SnoopDos by Eddy Carroll, but Snoopy is different in approach and purpose. Snoopy has no specific patches for specific functions - it is an all-purpose tool to monitor *ANY* library call in *ANY* system library.

Snoopy is driven by an input file (called a script file) which describes in detail which functions you want to monitor, and how this monitoring is to be done. This input file is a plain ASCII text that contains a simple but very readable syntax, and you can edit it with your favourite text editor.

Snoopy comes with some default scripts. One emulates SnoopDos behaviour, an other watches interesting intuition calls and a third one is used to track down some very important exec.library functions. These scripts will do the job for the causal users that need some kind of "advanced concepts" SnoopDos thing. Programmers however should read the script syntax, Snoopy will give you much "debugging power" (oh well ;-).

1.3 USAGE

Snoopy can be started from the CLI and from the Workbench. If you start it from the CLI (or a shell), the command line usage is :

Snoopy SCRIPT,TASKINFO/S,OUTPUT/K,PRI/N,SHOW/K/M

Note that you can start Snoopy without any arguments, see the defaults in the option description for what will happen in this case. Also note that starting with Version 1.4 Snoopy detaches itself from the CLI, so you don't need to type "RUN Snoopy". The Arguments are as follows:

SCRIPT - script filename

you can specify a Snoopy script file by giving its filename here. If this argument is not given, Snoopy will try to open "s:snoopy.script" - which hopefully exists. Snoopy cannot work if no scriptfile is given, so you either have to keep "s:snoopy.script" or provide your own scriptfile each time you start Snoopy. If snoopy doesn't file your scriptfile it will refuse to work because in this case there is nothing to work.

TASKINFO - caller task information

if you activate this switch on the command line, Snoopy will print out the address and the name of each task that called a patched function before the patch message itself. This helps you to determine who called which function in which particular order - a very usefull feature. TASKINFO is disabled by default.

OUTPUT - redirect output to this handle

you can give a filename to redirect snoopy output to. This might come in handy if you know that there'll be lots of calls or you want to examine the output later (or whatever). Just give the name of the file you want Snoopy to print its messages to. If this argument is not given, Snoopy will open its standard output window, which in turn will be "con:0/0/640/150/Snoopy".

PRI - set snooping priority

you may need to change the priority Snoopy is running at (some reasons for this are described in the THE PRI KEYWORD section). You can give the pri you want to use at this option, but it should probably be in the range -5 to +5 (I won't guarantee for any Snoopy function if running outside this priority range).

SHOW - explicitly show this task (multiple arguments)

you can tell Snoopy to watch only a series of tasks (found by their name). This comes in handy if e.g. you want to track down memory allocations by a certain task (most likely the prg you're debugging). Note that this option automatically disables snooping of any other task [i.e. any not-to-be-SHOWn-task].

Once Snoopy has started you will see the calls in either the output window or whatever output handle you have specified. You quit Snoopy by sending CTRL-C to it (if you have redirected your data with the OUTPUT option, use the CLI command BREAK to do this). Also, you can press CTRL-D to disable and CTRL-E to enable output, which means that you can pause Snoopy for some time. You can use CTRL-F to toggle the TaskInfo feature on or off.

If you start Snoopy from the Workbench you can add three ToolTypes :

SCRIPT - same as commandline SCRIPT argument

OUTPUT - same as commandline OUTPUT argument

PRI - same as commandline PRI argument

Note that Snoopy does NOT take these into consideration if started from the CLI.

1.4 SCRIPT FILE FORMAT

The script file is the most important thing about Snoopy - it gives you all the nice flexibility you need. A scriptfile is a plain ASCII text which basically contains five possible elements :

- a BASE specification

- a WATCH definition

- a HIDE name

- a SHOW name

- a PRI definition

Additionally, if you want to include a whole line as a comment, you should place an asterix '*' in the first column. This line then is ignored by the SnoopDos input parser. If you want to comment a line that also contains other data you can use the semicolon ';'. Everything following (and including) this char is ignored, also. For instance, the SnoopDos emulation scriptfile provided with this distribution looks something like

```
------------------------------------------------------------------------------
* SnoopDos emulation script for Snoopy
hide=Workbench ; forget about the big brother
..
; a library is opened for snooping by calling
; base=<alias>,<complete name>
base=dos,dos.library
; a function can be monitored by calling
; watch=<base>,<offset>,<regs>,<template>
watch=dos,-30,D1L/D2L/RD0L,Open( "%s", $%lx ) = $%lx
watch=dos,-36,D1L,Close( $%lx )
...
------------------------------------------------------------------------------
```

The following pages of this document describe what the statements above stand for....

1.5 THE BASE KEYWORD

You have to give Snoopy a list of all librarys you want to monitor, so that Snoopy can manage the function monitoring effectively. This is done by using the BASE keyword with the syntax

BASE=<alias>,<library name>[,<version>]

where <alias> is an abbreviation for the library you want to monitor. The <alias> is restricted to a maximum of 32 characters which should not pose too many problems since e.g. there are (and this is only a rough estimate) 7.1054532597E36 possible names you can build out of 32 characters ;-). Typically, this is the part of the name before the '.library' ('dos' for 'dos.library' and so on...). Note that <alias> can even be one of the Snoopy keywords such as "SHOW" and "PRI" (its magic, isn't it!). This <alias> is then used by the WATCH keyword to find the library base.

The <library name> is the complete name of the library you want to watch (which is directly transfered to OpenLibray()). You can even give a path (such as "code:debugging/libs/funny.library") if you need to do so. [see autodocs:exec.doc/OpenLibrary() for more details]

<version> is a specific version number you need. If <version> is 0 (which is the default) no version checking is done. This parameter is optional, you should only use it if you want to monitor functions specific to a certain library version (such as graphics.library/WriteChunkyPixels() [V40])

Some examples for this keyword :

base=dos,dos.library ; any version'll do

BASE=int,intuition.library

Base=MyLib,dh2:source/libtest/libs/test.library

base=gfx,graphics.library,40

1.6 THE WATCH KEYWORD

This is the most important keyword, since it specifies which function you want to watch, and how this should be done. The syntax is

WATCH=<base>,<offset>,<regs>,<template>

where <base> is a base alias given by a BASE keyword. Note that forward referencing is not possible, so you need to specify a BASE before you can use it in a WATCH.

The <offset> is the (signed) offset of the library function [decimal by default, use prefix $ for hex numbers], which should probably always be a negative number.

The <regs> part describes how the registers are to be examined before and after the call and the <template> describes what you want Snoopy to print to the screen. The concept is this : the <template> is a C-style format string like those you would use to call exec/RawDoFmt() or dos/VPrintf(). The order of the arguments to this <template> is specified by the <regs>-list, so that for each <template> format code the corresponding <register> format data is used. Now, the <registers> are given by a syntax that looks something like this

<reg 0>/<reg 1>/.../<reg n>

where <reg ?> is described as <[R][L](D|A|I)(0-7)>

where [R] says that this register is a return value rather than data before the function call, [L] says that the register contains 32-bit data (rather than 16-bit default) and [D0]-[A7] is the register you want. If you say I0 instead of A0 you will get register indirect (i.e. "(a0)" instead of "a0") Confused ? Well, I admit I cannot really find a more readable description of this, but maybe a look at a simple example will give you the Idea. First, some plain register descriptions :

D0 - data register d0, word-size data, pre-call (=function argument)

RD0 - data register d0, word-size data, post-call (=function returncode)

LD0 - data register d0, long-size data, pre-call

RLD0 - data register d0, long-size data, post-call

A0 - address register a0, word-size data, pre-call (=function argument)

RA0 - address register a0, word-size data, post-call (=function returncode)

LA0 - address register a0, long-size data, pre-call

RLA0 - address register a0, long-size data, post-call

I0 - address register (a0), word-size data, pre-call (=function argument)

RI0 - address register (a0), word-size data, post-call (=function returncode)

LI0 - address register (a0), long-size data, pre-call

RLI0 - address register (a0), long-size data, post-call

I7 - return address of function (address register (a7))

Now for a "real-life" example: The following scriptfile shows you how you can monitor the Open()/Close() pairs of the dos.library :

---

base=dos,dos.library ; open DOS

* watch Open() and Close()

watch=dos,-36,D1L,Close($%lx)

watch=dos,-30,D1L/D2L/RLD0,Open("%s", $%lx) = $%lx

Now, the base-statement should be quite clear. What does the first watch-statement mean ?

"watch=" - the keyword tells Snoopy that this line is a watch

description

"dos," - tells Snoopy that this watch applies to the dos.library

"-36," - tells Snoopy that you want to watch the function with the

offset _LVOClose, which is -36

"D1L," - tells Snoopy that you want to see the Register D1 ("D1")

and you want to interpret it as a Longword ("L")

"Close($%lx)" - tells Snoopy that you want to see a line "Close($<hexadecimal representation of register D1>)"

Understood ? Now take a look at the second statement and try to understand it. Its quite simple, really..... If you have problems making a script, try BUILDWATCH from the support directory. More advanced "concepts" (ha!) can be seen in the example scriptfiles provided together with this distribution.

1.7 THE HIDE KEYWORD

You can hide certain tasks from Snoopy. This is a very usefull feature because it reduces the amount of output given by Snoopy, thereby allowing you to look only for calls by specific tasks. An example: If you want to monitor critical calls to DOS functions like Open() and Close(), why bother about calls made from the Workbench ? They are harmless and probably ok - and besides you should think that the commodore programmers make some reasonable code in their own operating system (at least if you're not working with obscure beta release versions ("developers only")). By hiding the Workbench from Snoopy you will get a lot less calls to Open() and Close() (which are called, for example, on each icon when you enter a directory on your drive!). Besides, you probably still see all the important calls, don't ya ?! The syntax for this keyword is :

HIDE=<taskname>

where <taskname> is the name of the task you want to hide. The <taskname> is actually an abbreviation (not case sensitive!) of the real task name; so for example

hide=Work

would hide both the tasks "Workbench" and "WORKING CLASS HERO". If the task is a CLI, give the name of the command loaded in this CLI. You can get all the current task names by using the TASKLIST tool from the "support/" directory of this distribution.

1.8 THE SHOW KEYWORD

Sometimes it is more interesting to see calls of only one task (or: a selected list of tasks) instead of hiding all tasks that can be ignored. For instance, if you are debugging task "A" and want to see calls relevant only for this task, you would have to hide all other tasks [and there are many on them on a multitasking machine like the amiga]. The SHOW keyword does just this; it allows you to give an explicit list of tasks you want to snoopy into. The syntax for this keyword is

SHOW=<taskname>

where <taskname> is the name of the task you want to show. The <taskname> is actually an abbreviation (not case sensitive!) of the real task name; so for example

Show=Work

would show both the tasks "Workbench" and "WORKING CLASS HERO". If the task is a CLI, give the name of the command loaded in this CLI. You can get all the current task names by using the TASKLIST tool from the "support/" directory of this distribution. Note that SHOW disables any HIDE commands : it doesn't make much sense.

1.9 THE PRI KEYWORD

One very important thing about Snoopy is that you can change its running priority. This is VERY usefull [and therefor has been added in version 1.1] because some (=several(=many)) programs use local = volatile memory such as the stack for setting up their arguments. The problem is that if you place a string on the stack, the memory gets (probably) scratched by the time Snoopy is able to show it. If however you set Snoopy priority higher than the task calling the output is made BEFORE the memory is made invalid => lucky charm ! An example : tracking down calles by "internal" programs such as "alias" : if you use PRI=0 [default] you'll get scratched args for _Open(); if you use PRI=3 [recommended] you'll get the real thing! Great! Disko, Disko! The Syntax for this command is

PRI=<value>

where <value> is the numeric value (signed decimal) of Snoopys priority. You should use prioritys in the range -3 to +3. I cannot guarantee for any Snoopy function if you use prioritys outside this range : if your pri is too high, you might crash the event loop (especially if your pri is higher than the pri of the output handle (i.e. "CON:")), if it is too low you might never see anything. I recommend using "PRI=3" for most purposes.

Notes: a) only one PI statement per scriptfile is taken into account (actually: the last PRI statement in a scriptfile)

b) if you use the PRI command line argument it takes "priority" over any PRI statement in a scriptfile

c) again : keep in range!!!

1.10 RESTRICTIONS

Yes, there are several restrictions. They should be very reasonable, and I don't think you will find them annoying or whatever. They are

* You need Kickstart V37 or higher to run Snoopy (actually, V36

might do, too, but I didn't check on that one and you never know)

* The base alias can be only up to 32 characters long

1.11 FUTURE PLANS

Well, this is already the fourth release of Snoopy, and if you would know how lazy I am you probably would consider this a miracle in itself. However, "due to popular demand" I had to add some features. Some things I could do if I ever get myself back to work on this are :

* enable "ini.library" support

1.12 AUTHOR

Send your bug reports, ideas, love letters, fresh'n'phunky vinyl to

Gerson Kurz

Karl Köglsperger Str.7 / A303

80939 München

West Germany

This tool is dedicated to the **ULTRAWORLD** (Fuck the Sperrstunde!)

Greetings to : the ULTRAWORLD (munichs BEST rave - 100% pure mind-fucking techno) plus all munich DJs, the food-4-thought-suppliers RECORD STORE, DELIRIUM!, OPTIMAL, all munich techno ravers, my friends in the industry, RAVE SYSTEM K plus GABBERMAN of DISKO LOVERS INTERNATIONAL, Stefan Nocon (Rave On) and all other ravers. Hey! I want to get in contact with some GABBERFANS because ROTTERDAM RULES! [fuck "trance"], so if you have some tapes : GO BERSERK!!!!!!!!!!!!!!!!!!

See you all on the next Mayday in Berlin (or on any munich rave). ROTTERDAM ROTTERDAM ROTTERDAM!!!!

1.13 DISTRIBUTION

** Snoopy is I-WANT-TO-BE-FREE-WARE **

dedicated to the global house & techno scene

written by Gerson Kurz

Snoopy may be used by and FREELY distributed with any application be it commercial or public domain. There are no pagan users fees, Trump-esque licenses, or other forms of rabid capitalist trickery associated with using this tool and its support files. You do not even have to acknowledge the secret of your superb and efficient whatever-it-is result you gain by using Snoopy. The only limitation is that you may not alter the actual executable of Snoopy, nor sell the product and its support files as a distinct product (i.e. represent it as such). I don't give any guarantee for the fitness of Snoopy for any purpose : use at own risk.

1.14 SNOOPY SUPPORT TOOLS

The directory "support/" contains several tools you will find usefull in your work with Snoopy (given that you see any sense in doing so ;-). These tools are provided together with their sourcecodes under the terms of the Snoopy distribution. They are :

* BUILDWATCH - build watch commands for an FD file

* TASKLIST - show the names of all existing tasks

1.15 BUILDWATCH

In case you have trouble making script files for your libraries, try this tool. It takes as input the base name for a FD file, that fits to the following name description :

FD:<base>_lib.fd

Example:

buildwatch dos

Builds all commands for the dos.library from the file FD:dos_lib.fd

The results are given to stdout - so you must redirect this if you want to get a script file. You can then hand-edit the result to take only the functions you wish to include. Also note that a script generated by BUILDWATCH is not sensible to the actual data types - BUILDWATCH assumes all data to be LONG registers; so it doesn't recognize STRPTRs for example. Probably what you would do is :

- Generate a scriptfile with BUILDWATCH

- make your own copy of this scriptfile and handedit it to

contain

a) only those few functions you are interested in

(its not very reasonable to Snoop into all of Execs stuff at

the same time, for instance...)

b) proper output formats (e.g. replace %lx with %s for strings

and so on)

1.16 TASKLIST

This is a very small tool that gives you a list of the names of all tasks currently active in your system. If you don't have any other monitor program at hand, you can use "tasklist" to see if you can put some hide statements in your Snoopy monitor file. "tasklist" has no arguments or tricky features; just try it and look what happens ;-)

1.17 HISTORY

This page describes the changes that came about the different release versions of Snoopy

1.0 First released version.

1.1 - added PRI keyword

1.2 - BASE keyword now can handle a specific library version

- SHOW keyword added

- rewrote this document to fit what one might call

"the latest developments in the wide field of the English Language" ;-)

- tested with ENFORCER and found no hits (lucky me!)

1.3 - internal bugfixes only

1.4 Major update

- enables register indirect "due to popular demand"

- workbench interface

- detaches itself automagically

- added snoopy.doc