TestHandler

COLLABORATORS

	TITLE : TestHandler		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TestHandler

1.1 main

The Contents Page

1.2 intro

Introduction

You're either one of two people. The person writing that simple device handler or the person writing a full blown file system. Which ever you are, you're going to face a problem possibly worse than the actual coding of the handler itself, testing it in a real world situation. Forgetting that for a moment, testing the code at all. As you know, similar to a library or a device their is not an easy no standard way of testing a device handler in a debugger and single stepping through it. That is, until now. I recently faced this same problem, and so started searching for programs that would help. I found none. So then I started searching for solutions to the problem, experimented with my ideas and came up with this program. Which really works! I have written a program that will allow you to load your code into your usual debugger and handle the other task, dos packets. You can happily debug away on your handler and communicate with it normally as you would with dos as TestHandler will automatically take care of this set up for you. Now out can get a grip, get on with the job, and not worry about putting fiddly on-line messages in your handler and coding blind.

1.3 quick

Quickstudy

TestHandler is a dos command to mount an already loaded process as a device to enable easy debugging and testing. It will allow you to single step through your code as you deal with dos packets in real time.

1.4 install

Installation

TestHandler is a dos command, so you can copy it where you like. However, by double-clicking on the Install icon will start up a script that will install it for you. There are two main files to install, the program file and the guide file. You will be asked where you would like to install the files.

1.5 instruct

Instructions

TestHandler has quite a few options in its command line. It's not too complicated to set up and I will explain the basics of what to do. Running the command by itself will give you a brief on the template arguments.

How it works

Basically you load the executable into your debugger. Then you run TestHandler with the process name of the handler and the requested device name, with an optional start-up string among other options. It will take care of the rest, attempt to mount the device and send it a packet. Then wait. You begin debugging your code, first dealing with the packet as normal. If all goes well in your handlers' initialisation TestHandler will receive your reply and if no problems are encountered, TestHandler will notify you and quit as it's job is now done. You will have a new device to play with which you can now use it as a normal dos device in a shell, Workbench or whatever you choose. From now on, dos will take care of the rest, as though nature intended it. Transparently, of course.

The Template

The template for TestHandler is:

TASK=PROCESS/A, DEVICENAME/A, NAME=HANDLERNAME/K, DP=DEBUGPORT/S, NC=NOCLI/S, SS=STACKSIZE/K/N, PRI/K/N, GV=GLOBVEC/K/N, STARTUP/F

TASK=PROCESS/A

This is the name of the process (or task) you want the packets sent to, the one you have loaded into your debugger.

DEVICENAME/A

This is the name of the dos device added to the doslist you want to address the device as. The colon at the end is optional.

NAME=HANDLERNAME/K

This is the name of the handler to be specified in the dosentry. Default is the process name you specified.

DP=DEBUGPORT/S

This is a special option that will refer to a port name as opposed to a task name when addressing your task during debugging. More information can be found in the examples below.

NC=NOCLI/S

This option will clear the CLI entry of the task structure in case you need to fool it into thinking it started from Workbench, for message retrieval purposes.

SS=STACKSIZE/K/N

This will set the stacksize of the handler process in the dosentry. Default is the process' stacksize.

PRI/K/N

This will set the priority of the handler process in the dosentry.

Default is the process' priority

GV=GLOBVEC/K/N

This will set up the global vector type in the dosentry. Default is -1.

STARTUP/F

The last available option, this will set the start-up string in the dosentry and also of course, in the dos start-up packet.

Examples

First example

You have loaded a handler into the MonAm debugger. Let's say the handler's filename is "L:Test-Handler" and we wish to mount it as "TEST:". Here is what we do on the command line:

TestHandler L:Test-Handler TEST:

Next we go back to MonAm and single step through the code, retrieving the start-up packet and replying to it. TestHandler will mount the device and you can continue debugging. Simple!

Second example

You are working on a new file-system, written in Amiga E. You have loaded it into the E source level debugger. The filename is "New-File-System" and we wish to mount it as "NEW:". However, the code is in it's own directory and we wish to refer to it as "L:New-File-System" as well as direct TestHandler's output to the debugger's screen. Here is what we do.

TestHandler >CON:////TestHandler/AUTO/SCREENEDBG EDBG NEW NAME=L:New-File-System DEBUGPORT NOCLI

That cryptic redirection string will open up a window on EDBG's screen. The NAME argument will name it as we want. As EDBG runs programs in it's own task, we must specify EDBG itself as the process. Here, using the DEBUGPORT option, we specify a port to find the process from. This is needed for EDBG. We also specify NOCLI so our program is forced to wait for a start-up message. Complicated!

Third example

You are working on a handler by the popular name of Foo, are writing it in assembler and have loaded it into your debugger. We wish to specify a lot of stuff ourselves. Here is out command line.

TestHandler L:Foo-Handler FOO: NAME="The Foobar" SS=4096 PRI=5 "Foo You!"

Okay. Device "FOO:" will be mounted with the name "The Foobar", stacksize set at 4096 and the string "Foo You!" send as the start-up string. Foo!

Hope this helps you to understand TestHandler.

1.6 distribute

Distribution

This program is what I consider to be EveryWare. Since I offer it free I give you the duty of distributing it everywhere. Which you may or may not do, it's your choice. Give it to your friends. Distribute it on the net. Just make sure all the files are together and intact.

1.7 program

About TestHandler

This program was written using Amiga E, and is the result of painstaking research into dos handlers. This is the first version of the program released to the public and installed on my harddrive as well.

1.8 author

About me

Born in Australia, I first discovered programming in my early teens by reading Usborne computer books and got the bug from there. I'd written my first BASIC program before I even acquired an actual computer! However, soon after I became the new proud owner of a Commodore C16. And I got to test out my program, it worked! More to my delight was finding out about the machine code monitor built in contrast with the lack of instructions with it. That didn't stop my inquisitive mind and pretty soon I was disassembling code and printing out hex dumps on the MPS1250 dot matrix, learning 8-bit machine code and assembly language in the process. Much to the dismay of my Mother, who along with my friends thought I was weird. "Why can't you act normal like the other kids and just play games?", was one encouraging comment. I continued to code in assembler as well as BASIC writing all sorts of stuff, mostly games cheat hacking. Until I started experimenting externally as well and ended up short fusing my machine. A sad day of my life, my life was empty for time until I had to accept new technologies and my parents offer to replace it with a brand new Commodore-Amiga A500.

Soon after I learnt all over again, and not dissimilar to the first time, began programming in AmigaBASIC and yearning to learn 68k assembler. I did so, this time from Abacus books and convinced my Dad to fork out $200 or so dollars for a copy of AssemPro, and since I considered myself an advanced user, $750 for a harddisk.

I spent years doing this, and gained a vested interest in music, sampling and trackers. Apart from games which I also liked, if I wasn't in AssemPro I'd be ProTracking around! I'd soon outgrow the hardware and eventually upgraded to an A1200. The dream machine. Especially since it became my dream machine. I explored other languages, including the popularised C which I couldn't get into. Assembler was hard enough, but this language just confused me! Believe my, I know, I became an expert on it when I started converting C to assembler just to learn how it worked! I stuck with assembler until one came along which caught my eye. Amiga E.

Although at first it just spewed out errors, I gave it a second chance and never looked back. From commands to programs with full GUI's. It was a big change from assembler. Allowing me to do more, especially with the up to date include files and soon would become my language of choice. Being able to include my own assembly was the icing on the cake, a perfect combination for me. I continued to do what I do best to this very day, more programming and less documenting. My worse programming habit!

I do tend to write essays in the finish, but I hope you enjoyed the story! If you want to contact me about anything just email me at:

damo_rules@yahoo.com

Where, thanks to me, Amiga has infiltrated the server. Long live Amiga, and keep on programming!

1.9 contact

Contacting me

My name is Damien Stewart and if you wish to contact me you can send email to:

damo_rules@yahoo.com