whdload

COLLABORATORS			
	TITLE : whdload		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 2, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

whdload

1.1 WHDLoad.guide

WHDLoad

(the ultimate HD-Installer for OS-Killer)

send comments, bug reports, new slaves to:

Bert.Jahn@fh-zwickau.de

booah

Greeting and thanks are flying to:

Harry, Mr Larmer, Jeff, John Girvin, Dark Angel, Olbi

and all other guys who have part on the development of whdload !

1.2 What for this piece is ?!

WHDLoad is designed to easy install programs (demos,games...) which kill the operating system on harddisk.

To install a program, a so called "slave" must be written. The slave is the connection between the program and WHDLoad, and coordinates the reading and writing of files. On every disk access WHDLoad will switch from the installed program to the Operating System, and after the access back.

Because the simple interface and the example sources it will be very easy for anybody to install nearly any program on harddisk.

1.3 Features

Features of WHDLoad

startable via CLI or Workbench

memory protection using MMU (if available)

absolutely clear return to the OS (depends on the Slave)

possibility to use "disk images" and/or real files

variable MemSize for the program from $1000 up to $200000

intelligent allocation of the memory, partial absolute and partial independent

caching of files on start-up and dynamically on runtime (LRU)

caching of disk images via preload on start-up

automatic decrunching of files

emulation of MOVE SR,ea from User-Mode on 68010+

emulation of unsupported integer-instructions on 68060

debugging support (memory dump, register dump, file access logging)

Enforcer/Cyberguard and VMM proof

WHDLoad is tested on the following machines :

A1200 Blz 030/50/50 2 MBChip 4 MBFast AGA Kick 39.106

A2000 GVP 030/25/25 1 MBChip 1+4 MBFast ECS Kick 40.063(soft)

A4000/030/25 2 MBChip 16 MBFast CV64 Kick 40.068(soft)

A4000 CS 060/50 2 MBChip 28 MBFast CV64 Kick 39.106

UltraSPARC-2 Solaris 2.5 uae 0.5.3

1.4 Required is ?!

Requirements

to run installed programs :

CPU 68000

(68010+ is recommend because some installs require the VBR for the quit feature)

Kickstart 2.0 (version 37+)

a minimum of 1.0 MB RAM (depends on the program)

to write new slaves / install programs :

recommend is a 68030 equipped with a MMU to use all features of WHDLoad

know how on machinecode, assembler, amiga hardware ...

an Assembler (should be able to optimize)

some other tools will be useful (hexedit,disasm,...)

some more MB's RAM

access to various machines ('000-'060,ECS,AGA,GfxCard) for testing reasons

a NMI key to interrupt a running program or to enter a monitor/software-freezer may be also a fine tool

1.5 Installation

Installation

The easiest way to install WHDLoad and the supplied tools is to use the provided Installer script.

Important is, that the tools and WHDLoad are accessible without a path specification. Therefore the recommend place is "C:". Anyhow WHDLoad can also installed in the same directory where the program is located (notice that in the Project Icons of the Examples only "WHDLoad" is used, without any path).

The tools "DIC" and "Patcher" should be always installed because they are required by most installs.

1.6 Options and global Configuration

Options and global Configuration

There are two ways to configure WHDLoad. First you can set global options in a configuration file. And second set local options via the command line if started from CLI/Shell or via ToolTypes if started from the Workbench (Project Icon). Some options can be used global and local. In such a case a local Option overides the global one.

The global configuration file "S:whdload.prefs" is a usual ASCII file and contains one option per line. Empty lines and comments starting with ";" are ignored. An example config-file is contained in the developer package.

Overview

Name of Option Type Local Global default value

Cache switcher x -

CoreDump switcher x -

CoreDumpPath string - x PROGDIR:

DebugKey numerical x x -

DCache switcher x -

FileLog switcher x -

FreezeKey numerical x x $5d

MMU switcher x x

NoAutoVec switcher x x

NoCache switcher x -

NoDCache switcher x -

NoTrapHandler switcher x -

NoMMU switcher x x

NoVBRMove switcher x -

NTSC switcher x -

PAL switcher x -

Preload switcher x -

QuitKey numerical x x -

Slave string x - whdload.slave

WriteDelay numerical x x 150

Description of each Option

Cache/S

this option enables the instruction cache for the installed program

the option has no effect if NoCache/S is also set

CoreDump/S

if selected on every regular exit from an installed program a memory and register dump is created

this may be useful if you like to rip a music-module a similar stuff

CoreDumpPath

the destination directory for all dump files created by whdload

default: "PROGDIR:"

DCache/S

this option enables the instruction and the data cache for the installed program (it implies Cache/S)

the data cache will not be enabled if the option NoDCache/S is also active, the instruction cache will not be enabled if NoCache/S is set

DebugKey/K/N

set the rawkey code to quit the program via debug (write coredump files and quit)

this works only if the VBR is moved by whdload (NoVBRMove is not set) and/or the slave support it

FileLog/S

this option is only for debugging purposes, if set all disk accesses will logged to a file (.whdl_filelog)

this option disables the option PRELOAD

FreezerKey/K/N

if you are using one of the supported software freezers HrtMon 2.1 or Thrillkill 1.04 you can use this option to setup a rawkey code on which WHDLoad will enter the freezer

to work the vbr must be moved by whdload (therefore NOVBRMOVE must not be set) and the freezer must be active

default: $5d (gray '*' on the numerical keyblock)

MMU/S

this must be used on 68030 machines to get the mmu related features working (memory protection and improved cache management)

on 68040/060 this option has no effect because the MMU will be used on default

if the option NoMMU/S is also set this option has no effect

NoAutoVec/S

if selected whdload will not quit if an unexpected autovector or NMI occures (vectors #25-31 / $64-$7c)

this should be used on machines/hardware which will create at random such interrupts to prevent whdload from exiting

NoCache/S

if selected the caches for the installed program are disabled, also the slave cannot enable them (via resload_SetCACR)

this option overrides Cache/S and DCache/S (it disable them)

NoDCache/S

disables the data cache for the installed program

this option override DCache/S

NoTrapHandler/S

enables original OS vector table

with this WHDLoad will not catch any exceptions

Warning: if the OS tries to create an exec.Alert the machine will crash

NoMMU/S

this can be used to prevent that whdload uses the mmu, this may be necessary if there are "Access Fault" exceptions in conjunction with some older installs

the option overides MMU/S

NoVBRMove/S

disables the moving of the vector table on 68010+ machines

use this in conjunction with old slaves to prevent WHDLoad from exiting

NTSC/S

if selected WHDLoad switches to NTSC-videomode (60Hz) via custom.beamcon0

PAL/S

if selected WHDLoad switches to PAL-videomode (50Hz) via custom.beamcon0

Preload/S

if selected WHDLoad tries to cache as far as possible files/disk images on startup

this should be always enabled

QuitKey/K/N

set the rawkey code to quit the program

this works only if the VBR is moved by whdload (NoVBRMove is not set) and/or the slave support it

Slave

name of the slave which should be used

default: "WHDLoad.slave"

WriteDelay/K/N

time in 1/50 secs that WHDLoad will wait after writing anything to disk (savegames, highscores)

this makes sense for filesystems who does'nt write data immedately to disk

you can set it to 0, but then you should never exit by a reset because saved data may perhaps not been written physically to disk

default: 150 = 3 seconds

1.7 rawkey codes

keyboards RAWKEY codes

All keys used in WHDLoad must be specified as raw key codes. Therefore here a short list which includes some of the most used keys.

Key Hex Dec

ESC $45 69

F1..F10 $50..$59 80..89

DEL $46 70

HELP $5f 95

BS $41 65

NumL $5a 90

ScrL $5b 91

PrtSc $5d 93

1.8 WHDLoad and the MMU

WHDLoad and the Memory Managment Unit (MMU)

A MMU is contained in the following processors of the 68000er family: MC68030, MC68040, MC68060. For the MC68020 there is an external MMU called MC68851 available, but this is currently not supported by WHDload. There are also so called EC versions of these processors which have a broken MMU. For example all standard A4000/030 have only a MC68EC030 cpu. On third party accelerator boards this is different, look in the appropriate documentation to learn about. As far as known all 68040/68060 ever build in an Amiga are full cpu's containing a working MMU (because burstmode and Zorro III requires MMU mapping of io space). WHDLoad cannot determine, by software if the current cpu is a full or a EC version. Therefore this distinction must be done by the user.

Features of a MMU and using in WHDLoad

The main purpose of the MMU is to translate logical addresses to physical ones. This is required for virtual memory and separated address spaces (for example in a protected multi process system). A other feature is to specify special properties like Supervisor Only, Write Protected and Caching mode for every physical address space (on a Page base).

WHDLoad does not use logical to physical address translation. But it uses the MMU for memory protection, cache managment and some special features concerning the custom and cia registers.

Memory protection in WHDLoad

WHDLoad scans on startup the memory list and will build a translation tree which includes all accessible memory. At this point all memory is marked as invalid (not accessible). Now it marks the following address spaces as valid and accessible : $0...BaseMem (using the information from the slave), $dff000...$dff200 (custom registers), $bfd000...$bff000 (cia registers) and the memory used by the slave and whdload. If the vector table is moved, or a freezer is found in memory also these areas will be marked as valid.

All other address space is marked as invalid, and therefore every access to such a area if it's a Read or a Write will create an Access Fault Exception which will normaly end in a appropriate error requester created by WHDLoad.

Cache managment using the MMU in WHDLoad

The caches are normally controled via the CacheControlRegister (CACR). With the MMU this may be adjusted finer because the cacheability can be set for each memory location separately. On the 68030 a memory page can be cacheable or not cacheable. On a 68040/68060 it can be cachable WriteTrough, cachable CopyBack, noncachable and noncachable serialized. The area of WHDLoad and the Slave are always marked as cacheable (WHDLoad=CopyBack, Slave=WriteTrough). If the MMU is used by WHDLoad on default the BaseMem area is marked as noncachable, and the Data and Instruction cache are enabled in the CACR. So the program located in the BaseMem area runs without caches but WHDLoad and the Slave uses the caches to improve performance.

The Slave can enable (or disable if previous enabled) the cache via the function resload_SetCACR. These function will depending on the cache set also modify the MMU tables for the BaseMem area.

User control of the MMU handling in WHDLoad

There are 3 different modes how WHDLoad does affect an existing MMU.

1. ignore MMU

In this mode WHDLoad does not change any MMU related register. This may be useful if you have running a program banging the MMU which is essential for your system. Or for example if you want to use a software freezer which uses the MMU. Warning: if you are running Enforcer or a similar tool your machine will hang because a lot of Enforcer hits during swapping the operating system.

2. disable MMU

In this mode the MMU is switched off, no MMU related features are available.

3. use MMU

In this mode WHDLoad takes full control over the MMU and realizes memory protection and cache managment as explained above.

On the 68030 the default mode is "disable MMU". On the 68040/68060 default is "use MMU".

There are two options to control this behavior. "MMU" enables the MMU feature and must be used on 68030 systems to enable the MMU functionality. "NOMMU" disables the MMU usage. Both option can be used in the global configuration file and also local as Tooltype/CLI-Argument.

"An Enforcer hit is an Enforcer hit, period" (Michael Sinz)

1.9 How to install

have it in mind

general rules

the easiest case

One disk trackloader without savegames

possible problems

non standard disk format

more than one disk

highscores saving

savegames

OS accesses

common compatibility problems

limited address space

different stackframes on different processors

movem on 68000 and 68020

tips & tricks

choice between diskimages/files

writing installs

1.10 Simple One Disk TrackLoader

How to install a simple one disk trackloader

I. Prework

Make a drawer which will hold all files.

Create disk image using DIC in this drawer.

Create a #?.info file with "WHDLoad" as >Default Tool< and a Tooltype "SLAVE=#?" contain the name of the slave. (or simple copy the icon from an Example Install, and disable all tooltype except >SLAVE=<)

II. The slave

To write the slave we need some informations:

1. Where on disk is main executable file located ?

2. Where inside the main executable is the disk loader located ?

To get these informations we first analyze the bootblock. Most times the main exe will loaded from here via exec.DoIo(). Sometimes a special trackloader stands in the bootblock. We now write a slave which will simulate the bootblock and loads the main exe from the disk image, then he returns via TDREASON_DEBUG and we make a coredump. After that we have to find the loader in the main exe. An fast way is to search for the pattern $AAAAAAAA (used by MFM decoding) with a hex-editor. Then cut the area (+/- $1000 bytes) found, disassemble it, and search the start of the routine. Understand the parameterlist. Now we create some code for the slave which will patch the loader routine, that all calls to the loader will redirect to the slave, which will call WHDLoad (the parameters must be adjusted). So, the program should now run from HD (this is the ideal case).

III. The end

Now create an nice Icon and all is done.

DCLXVI. I know this sounds easier as it is

Sorry, I can't and I don't want describe all possible problems here which can occur. See also the other chapters one page before, and look in the examples.

1.11 non standard trackloader

non standard trackloader

Some programs use a very own disk format. With the result, that DIC is unable to create the disk images. So you must use the orginal loader routine. Most times this routine is located in a preloader (blocks 2-11) which will loaded from the bootblock (sometimes it is in the bootblock, further simple). So write a slave which will create the disk image using the orginal loader (patch the loader to read the required disk area and save them with resload_SaveFile).

Some times the loaders are unable to read a whole disk or you will have to less memory, in this case let the install slave create smaller files (10 cyls for example). And later join them using "C:Join".

1.12 multiple disks

more than one disk

If the program uses more than one disk the slave must redirect the disk accesses to the appropriate image file. Sometimes this is not easy. Some programs support more than one drive, so you can use the drive number for selecting the disk. Most programs use an id on every disk to distinguish them. In this case use a variable which holds the disk number, and on every access to the disk id (determinate this by analyzing the parameters for the disk loader) increase the var (if last disk reached decrease, of course). So hopefully the loader will read the id again and again up the right disk is inserted ! Perhaps there is an request from the program that the user should insert the right disk, disable it.

1.13 highscore saving

saving highscores

Not much to say here. Use resload_SaveFile to write the appropriate memory area to the disk. If you like encrypt it a bit that not every lamer can patch it. Remember, that always a separate file must be used. So the initial read of the highscores, most times loaded by the program from a disk image, must be redirected to the highscore file. (Example Gods.slave)

1.14 savegames

writing savegames

Savegame handling is near the same as with highscores. But you must disable the savegames from caching (remember all files which are loaded by WHDLoad are cached automatically if enough memory is available). To do this use a pattern which matches all savegames (e.g. name all savegames to "save.[0-9]" and use the pattern "save.#?").

(Starting whdload 0.143 it's no longer required to use ws_DontCache because resload_SaveGame will now writing to the cache)

1.15 os accesses

accesses to the operating system

At the time the slave or the program is executed absolutely no os exist nor is accessible nor makes any sense to access them ! Therefore all accesses which the program will do, must be disabled. If there are less of them, and without function (like exec.Disable() or exec.SuperState()) simple $4e71 them. If the accesses have an important function (like exec.DoIo()), redirect them to the slave and emulate them. If there are a lot of them, create an simple exec.library in an unused memory area (set $4 to this). (Example Oscar.slave emulates exec.AllocMem())

To detect such accesses the initial execbase is set to $f0000001 with the intention that all routines which like to use the execbase will force an "Address Error" exception.

1.16 limited address space

Limited address space on 68000/68010/68ec020

On this processors the address space is limited to 16MB ($000000...$ffffff) because these have only 24 address lines. As a result all accesses to higher spaces are performed to 16MB and the most signifikant 8 bits are ignored. Some programs use these bits to store any data, or simple forgets to clear these. On a processor with full 4 GB address space like 68020/680ec30/68030/68040/68060 this will not work, because the full 32bit address will be accessed.

To solve this you have to patch these accesses and redirect them to the appropriate address.

Sometimes the reason for such accesses may be an uninitialized pointer. In this case it may help to clear $400 - ws_BaseMemSize.

1.17 different stackframes

different stackframes on each processor

The stackframes created by the processor on interrupts and exceptions are different for the members of the 68k family. On the 68000 a stackframe is 6 byte, except on Bus and Address Error. The stackframe contains first the saved SR at (a7) and the saved PC at (2,a7). On all other processors (68010+) the minimal stackframe is 8 byte and contains additional the vector number as word at (6,a7). This Four-Word stackframe format $0 is created for "Trap #xx" and Interrupts on 68010-68060. The stackframes on other exceptions are different on each processor.

The RTE instruction works different on the 68000 against 68010+. On a 68000 it simply writes the SR and PC back and continue program execution at the interrupted address. On the 68010+ it will additional frees the stackframe depending on the stackframe format.

Some programs are pushing an address (PC) and a SR and then execute a RTE instruction. This works on a 68000 only, on 68010+ this will have undefinable results.

If a program do so, you have to emulate this stuff. Sometimes it may be enough to replace the rte with a rtr.

1.18 movem on 68000 and 68020

movem.x rl,-(an) on 68000/010 and 68020/030/040

There is a difference if the register used in predecrement mode is also contained in the register list. For the 68020, 68030 and 68040 the value written to memory is the initial register value decemented by the size of the operation. The 68000 and 68010 write the initial register value (not decremented).

Because such a construction is not very useful no problems to existing software are known.

1.19 diskimages or files

what's better diskimages or files

Sometimes you will have the choice to use disk images or real files. Both has advantages. The use of disk images is most time the easier and faster way to create the slave. But real files are better cacheable (if there is less memory or the memory is fragmented diskimages cannot be cached). The needed space on harddisk will also smaller with real files than with disk images. Only if there are a lot of files (more than 30) you should use disk images.

1.20 writing installs

writing installs

* If you want to distribute the slave/install, check it at least on 68000/OCS and 68060/GfxCard equipped machines.

* Use an Installer script if you provide an install. (best you modify one on the provided scripts, I think they are good.)

1.21 general rules

general rules

Don't modify cpu registers present in higher processors like VBR or CACR. VBR is always 0 from the slave programmers view, even if it's moved because Auto-Vectors (and Traps if Flag is set) are emulated. The bits in the CACR are different for each processor. There is only one valid way to modify the caches, use resload_SetCACR and the bitdef's from exec/execbase.i.

Of course this means also that all stuff which may present in the program to install banging these registers must be disabled or skipped.

Never modify disk images. So, if anybody wishes to start the program from a floppy he must only write the images back to a disk.

Never use orginal stuff from the program directly in the slave (copyright problem).

Disable all stuff which tries to access the os. Otherwise the system will be destroyed.

Enable the Instruction Cache only if you are sure it runs on all processors. (Data Cache cannot be enabled in the current version)

Use less memory as possible for ws_BaseMemSize. Some people have resident tags on the end of the Chip memory, so it helps to use only $1f0000 instead $200000 and WHDLoad can use absolute allocated memory.

1.22 Technics

Executing flow

here is a typical executing sequence (I hope it helps a bit to understand how WHDLoad works) :

USER starts

WHDLoad will be loaded and started

loads and checks the slave

allocates required memory

preloads diskimages/files (if enabled)

switches os off

calls the slave

slave loads the main program (via calling WHDLoad)

patches the program that the program will load his files via the slave

patches the program to enable an exit from the program

calls the main program

program will do his stuff

USER exits the program (for example by pressing ESC)

slave will return to WHDLoad

WHDLoad switches OS on

frees all resources

quits

1.23 System Monitors / Freezer

System Monitors / Freezer

There are a lot of pure software freezers out there. The usage of such tools may be a great help on developing and debugging slaves for WHDLoad.

Directly supported by WHDLoad is HrtMon 2.1 and ThrillKill 1.04. Other kinds may be used via the option NoTrapHandler/S (this has some other disadvantages of course). If you want direct support for an other one, you can contact me and if it can be done I will include support for it.

On the startup of WHDLoad it will check if one of the supported monitors are active. If one is found WHDLoad will do some special stuff. If the MMU is used by WHDLoad it will declare the memory used by the monitor as valid and WriteTrough cacheable. During the game/demo runs it will forward all NMI exceptions to the nmi vector saved from the monitors vectortable. Additional, if the vbr is moved (by whdload, this means NoVBRMove is NOT set and the cpu is at least a 68010) it will compare

the "FreezerKey" with the actual value in _ciaa+ciasdr at each interrupt. If the values are matching WHDLoad will do the required keyboard stuff, transform the stackframe to a NMI stackframe and enter the monitor via his nmi handler. The keycode (rawkey) for this operation can be controlled via the global or local option "FreezerKey". Default is the gray '*' on the numerical keyblock.

HrtMon

The detection in memory should be relative sure. I think it will also work with future versions.

Be careful if the MMU is used by whdload: dont access area outside BaseMem from HrtMon. It will crash because HrtMon does not handle the resulting Access Fault Exception.

(tested with 2.1demo)

ThrillKill

There is no useable signature in the freezer, so some code compares are used. Therefore the detection will not work with other versions. (Contact me if you have problems)

(tested with 1.04demo)

1.24 Future plans

Future plans

emulating of some OS functions (exec.AllocMem,exec.DoIO,...)

support for nonvolatile.library (savegames)

1.25 The Disk-Image-Creator

DIC * The Disk-Image-Creator

A small tool to create disk-image-files on your harddisk, from NDOS floppy disks.

Installation:

copy it to C: , or a similar place

Usage:

enter directory where the images should stored

type "DIC" on the Shell and press RETURN

now insert the disks and follow the instructions from DIC

after the last disk is read by DIC break it by pressing CTRL-C

Options:

DEVICE,FD=FIRSTDISK/K/N,LD=LASTDISK/K/N

DEVICE

the device from which DIC will read the disks

default "DF0:"

example "RAD:"

FD=FIRSTDISK/K/N

number of first disk

default "FD=1"

example "FD=0"

LD=LASTDISK/K/N

number of last disk

default "LD=999999"

example "LD=3"

Return code

The return code is set to 0 if all disks from "FD" up to "LD" are read successfully. Otherwise 10 is returned.

Background:

DIC can read any diskgeometry, also HD's.

If a track cannot be read, the area in the diskimage is filled with the pattern "TDIC" and DIC goes to the next track, anyway an error message will you inform about this.

1.26 Find Access

Find Access * Find Access to a memory location

This tool will load the spcified file and scans it for accesses to a given address. It will find all absolut and relative 8bit, 16bit and 32bit references. I have written this to analyse the coredump files. It is similar to the Freezer command FA but will not disassemble the access but show it as pure hex dump.

Installation:

copy it to C: , or a similar place

Usage:

I use the following alias (added to my S:shell-startup) to search in the last memory dump written by whdload:

alias fa FindAccess C:.whdl_memory

so I can open a Shell and type for example:

fa $1500

and will get the output:

FindAccess 1.0 (04-Aug-96 01:25:31) by Bert Jahn

loading file Workbench:CE/.whdl_memory

scanning accesses to $15000 (86016) file: $0-$80000

Relative Word at $ 14240 -> 06C00000 07C00000 0DC0 00000CC0 000004C0

Options:

FILE/A,ADDRESS/A,ORG

FILE/A

the name of the file to scan

ADDRESS/A

the address on which all accesses will be shown

you can use simple expressions like these :

"$5000" = "20480" = " -$Ff +-33+ 19512+$4e8"

(prefix "$" for hexadecimal numbers, supported operators are "+" and "-")

ORG

the logical start address of the file

if not specified $0 is used

you can use same simple expressions as with ADRESS/A

1.27 Relocating Executables

Reloc * Relocating Executables

A small tool to relocate a standard AmigaDOS Executable to an absolute address and save this as a simple data file. This is required because WHDLoad is not able to handle AmigaDOS executables.

Installation:

copy it to C: , or a similar place

Options:

INPUTFILE/A,OUTPUTFILE,ADR/K,QUIET/S

INPUTFILE/A

the executable file which should relocated

must specified

OUTPUTFILE

the name of the file to write

if not specified the source file will be overwritten

ADR/K

the start address for the relocated file

if not specified $400 is used

you can use simple expressions like these :

"$5000" = "20480" = " -$Ff +-33+ 19512+$4e8"

(prefix "$" for hexadecimal numbers, supported operators are "+" and "-")

QUIET/S

disables all output, except error messages

enable this for using in conjunction with the Installer

Return code

The return code is set to 0 if all went OK, otherwise to 10.

Background:

the HUNK's from the executable are copied in the same order to the absolute file

BSS-HUNK's are blown up to their real sizes

supported are only pre 2.0 HUNK's

1.28 Installing Bootblocks

InstallBB * Installing Bootblocks

InstallBB read or writes Bootblocks from or to a disk. The bootblocks are handled as executables. On writing it will preserve the disk filesystem type by first reading the bootblock and merging the id (DOS\0,DOS\1,...) to the new. The bootblock checksum is automative calculated.

Installation:

copy it to C: , or a similar place

Options:

PROGRAM/A,UNIT/K/N,WRITE/S

PROGRAM/A

the executable file which will read or written

must specified

UNIT/K/N

the trackdisk unit number (number of floppy drive)

default is 0 (DF0:)

WRITE/S

switcher, if not specified the bootblock will read from disk to a file

if specified the given program will written to disk

1.29 The Patcher

Patcher * Patching Programs

The Patcher is a universal tool to modify files or disks. Some Installs especially for games with a very own disk format uses the Patcher to create files or disk images from the original disks. To learn more about the Patcher you should read his documentation, located in the patcher archive provided with this package.

1.30 Assembler Output

BASM 1.131 (12-Dez-95)

«Barfly Assembler (REG.)»

Copyright © Ralph Schmidt 1990-95

Assembling : Applications:Developer/Projects/WHDLoad/WHDLoad.asm

PASS 1

Optimize Pass 1...1330 Symbol Changes | 4 Length Failures...

Optimize Pass 2...1314 Symbol Changes | 3 Length Failures...

Optimize Pass 3...1289 Symbol Changes | 3 Length Failures...

Optimize Pass 4...1208 Symbol Changes | 3 Length Failures...

Optimize Pass 5...1182 Symbol Changes | 2 Length Failures...

Optimize Pass 6...1061 Symbol Changes | 2 Length Failures...

Optimize Pass 7...873 Symbol Changes | 2 Length Failures...

Optimize Pass 8...641 Symbol Changes | 2 Length Failures...

Optimize Pass 9...617 Symbol Changes | 0 Length Failures...

Optimize Pass 10...475 Symbol Changes | 0 Length Failures...

Optimize Pass 11...422 Symbol Changes | 0 Length Failures...

Optimize Pass 12...384 Symbol Changes | 0 Length Failures...

Optimize Pass 13...262 Symbol Changes | 0 Length Failures...

Optimize Pass 14...155 Symbol Changes | 0 Length Failures...

Optimize Pass 15...0 Symbol Changes | 0 Length Failures...

27174 Lines assembled in 60.809 Seconds(26812 Lines/Min)
Applications:Developer/Projects/WHDLoad/WHDLoad.asm contains 3203 Lines.
0 Symbol(s) declared global.
0 Symbol(s) referenced external.
129 Macros(s) defined.
2743 Macros(s) called.
0 Includes added to the Cache
1344 Includes found in the Cache
1 Hunk(s) created.
Optimizing saved 3482 Bytes Code. 15 Optimize Passes needed.
Hunk: 0 Name=``
Start:0 End:$8BE8 35816 Bytes Reloc:1 Mode:CODE Memory:PUBLIC
Output : c:WHDLoad
Errors: 0 Warnings: 0
Assembling Ok! No Errors found.
Code type is position relocatable.Output file size 35872 bytes.
File saved in 0.099 Seconds(361.766 KB/s).

1.31 Copyright

1.32 Disclaimer

1.33 author

The Author

jah@fh-zwickau.de

http://www.fh-zwickau.de/~jah/whdload.html

Bert Jahn

Franz-Liszt-Straße 16

Rudolstadt

07407

Germany