LDOS

Leonard's Demo Operating System

Great stuff!!! Easy to use, and now no excuses for not making that new Trackmo ;)

by DanLemon

The perfect trackmo tool, very simple, very efficient, thanks Leonard!

by Soundy

Very easy to use indeed!

by ok3anos

Mel o Dee's
A Music Disk By Resistance
... Loading ... Please wait
LOADING
Amigatari
The ultimate music floppy
Original 2020 Atari Music Section
Atari Music Modern Era Section
OXYGENE brings

The easiest way to code a trackmo for your Amiga

INTRODUCTION

LDOS is a framework to easily build Amiga multi-part demos. You can chain several amiga executable effects. LDOS is managing memory allocation, floppy disk loading, data depacking and image disk creation. LDOS also includes an HDD loader to run your demo from harddisk. LDOS toolchain is running on Windows platform.

CREDITS

- LDOS is written by Arnaud Carré aka Leonard/Oxygene (@leonard_coder)
- ARJ depackers by Mr Ni! / TOS-crew
- Light Speed Player by Leonard/Oxygene (https://github.com/arnaud-carre/LSPlayer)

TABLE OF CONTENTS

FILES TYPES

Here's a summary of the different files types you're going to deal with:

- `.asm` ⇒ your assembly source code
- `.bin` ⇒ output of the assembler/linker. These files are generated by `m.cmd` and are used by LDOS to create an ADF file.
- `.cfg` ⇒ Compiler & linker configuration file
- `.cmd` ⇒ Windows Batch File, similar to the good old `.bat`
- `.lsmusic & .lsbank` ⇒ LSP music player native files formats (see https://github.com/arnaud-carre/LSPlayer)
- `.s` ⇒ same as .asm

GETTING THE FILES YOU NEED

You already have coded some effects, maybe you also have a music and you want to make a Trackmo? Easy!

1. Grab LDOS's latest version here: https://github.com/arnaud-carre/ldos
2. In folder `LDOS/demo/` you will find a suggested folder organization:

- demo/
    - build.cmd
    - parcade.mod
    - script.txt
    - greetings/
        - greetings.asm
        - m.cmd
        - run.cmd
        - script.txt
        - vc.cfg
    - sprites_loader/
        - sprite.asm
        - m.cmd
        - run.cmd
        - script.txt
        - vc.cfg

Here's an explanation of the above file organization, remember this is just a **suggestion**:

- **demo**: Global folder that contains the whole demo, and specific files to build the full demo:
    - **build.cmd**: used to build the full demo (described below)
    - **parcade.mod**: just the music for your demo (can be several)
    - **script.txt**: same as for single fx, but this one is bigger: it lists all the .bin of all your FX, and the music (see below for more explanations about script.txt)
- **greetings**, **sprite_loader**, …: Folders containing your various FX. Each is composed of:
    - source and data files
    - **m.cmd**: builds and links your code. The output is a "`.bin`" file.
    - **run.cmd**: creates a `.ADF` out of the `.bin` and launches WinUAE.
    - **script.txt**: the list of files to include in the ADF file for this FX (more details about script.txt below).
    - **vc.cfg**: you need it in every FX folder. It contains the arguments (parameters) for the assembler and the linker (vasm, vlink). You can use the one provided here: `LDOS\demo\greetings\vc.cfg` .

SCRIPT.TXT

This is a text file containing the names of all the files to include in your Trackmo (basically, the list of files to copy to the ADF disk). 1 file path per line, as easy as that.

- FX Development use case:
    - You need to build an ADF containing only FX1 to test it? Just type a single line in script.txt:

      ```
      FX1.bin
      ```

      See demo file `LDOS\demo\greetings\script.txt` for example.

      Wonder what a `.bin` is? It's generated by build.cmd, described later.
    - Maybe you want to test FX1, but at the end of FX1 you coded a transition to FX2. Wanna test it? Just add FX2.bin in your script. Now your script file in

the FX1 folder contains 2 lines:

```
FX1.bin

FX2.bin
```

- Building the whole trackmo use case:

  Now you want to generate an ADF that contains your whole demo. Here's the script.txt you need:

  ```
  music.lsmusic

  music.lsbank

  ../FX1/FX1.bin

  ../FX2/FX2.bin
  ```

  See demo file `LDOS\demo\script.txt` for example.

  The 2 first lines are the files generated by LSP when processing your .MOD music file. This processing is handled in build.cmd (described below). Putting the music first guarantees the music is loaded before anything else and we are sure it can start with the first FX. Then comes the list of FX binaries to add to the trackmo.

- Note about musics: LSP requires 2 files: a `.lsbank` and a `.lsmusic` file for each music. You always have to put both files one after another in your script.txt. Also note, it's better to put the `.lsbank` first, then the `.lsmusic`

BUILD.CMD

This is a text file containing the instructions to compile your full demo and make an ADF:

See demo file `LDOS\demo\build.cmd` for example:

1. calls `m.cmd` for each of the FX of the demo to generate a .bin for each of them
2. converts the music to the LSP format
3. generates the ADF from all the .bin and music files according to `script.txt`

4. invokes WinUAE

API DOCUMENTATION

All the available functions are listed in "`../../ldos/kernel.inc`", and guess what, your code must `include "../../ldos/kernel.inc"`

Then, all functions are invoked the same way:
```
move.l  (LDOS_BASE).w,a6
jsr FUNCTION(a6)
```

For example:

```
move.l  (LDOS_BASE).w,a6
jsr LDOS_PRELOAD_NEXT_FX(a6)
```

Note that `../../ldos/kernel.inc` already provides a basic documentation for the API, below is some extra info for the functions that need more details:

- **LDOS_PRELOAD_NEXT_FX** : load & depack the next file in `script.txt`. Usually used by a FX to pre-load the next FX. Once the current FX ends (using RTS) the next one is executed immediately.
    - Note #1: If the next file in `script.txt` is not a .bin, but a music or a data file, then this file will be loaded.
    - Note #2: If you don't pre-load, next FX will be loaded & depacked when the previous one ends
- **LDOS_MUSIC_START**: Will start playing the latest loaded music.2 ways to load a music:
    - Either the music files are the 2 first files in script.txt and the music will be ready to play when your first FX starts (see `LDOS\demo\script.txt` for an example).
    - Either the music is NOT at the beginning of script.txt, and it needs to be preloaded using **LDOS_PRELOAD_NEXT_FX**. This method is generally used when you have several musics in a single demo.
- **LDOS_MUSIC_GET_TICK**: Get LSP music frame tick. Use this counter if you want to sync gfx with music (output: `d0.l` = music frame tick). The return value is actually the CIA counter ticks, and it totally depends on the music’s tempo. There is no trivial way to convert the tick counter into patter/pos/row in the original

.MOD file.

- **LDOS_MUSIC_GET_SEQ_POS**: Returns the current MOD music sequence position (output: `d0.w` = music sequence position). Basically, this number increases every time a new pattern starts. Note this is not the pattern number, but the index in the pattern sequence.
- **LDOS_MUSIC_STOP**: Stops the current music and frees the associated RAM.
- **LDOS_PERSISTENT_CHIP_ALLOC**: Allows FX to allocate memory that will not be erased when your FX ends (see *"How do I allocate RAM in my FX?"* in the Q&A section below*).*

2 DISKS DEMOS

Here's how to handle multiple disks demos.

- First, you need 2 script.txt files in your demo/ folder: one for each disk, listing which .bin and which musics go to which disk.
- Then, here's how you check for disk swap:

```
move.l     (LDOS_BASE).w,a6
move.w     #5*50,d0 ; wait 5 seconds in case of HDD version
jsr        LDOS_IS_DISK2_INSERTED(a6)
cmp.b      #$ff,d0
bne.b      .retry
; success
move.l     (LDOS_BASE).w,a6
jsr        LDOS_PRELOAD_NEXT_FX(a6)
```

DEBUGGING

LDOS crash screens will give you an explanation of why it crashed. But you can use this feature to implement your own assert/error system. To see examples, look for “`trap #0`” instructions in LDOS source files. For example:

```
InvalidParam:

      move.w      d0,-(a7)         ; push param to display on stack

      movea.l     a7,a1

      lea         .txt(pc),a0      ; error message

      trap        #0               ; assert

.txt:

      dc.b        'D0.W = %w Invalid param!',0

      even
```

Another example:

```
mallocError:

      lea         .txt(pc),a0            ; error string

      pea         fastMemTable(pc)       ; 2nd displayed argument

      pea         chipMemTable(pc)       ; 1st displayed argument

      movea.l     a7,a1

      trap        #0                     ; assert

.txt:

      dc.b        'MALLOC ERROR!',10     ; “,10” goes to the next line

      dc.b        'Chip Table: $%l',10   ; %l ⇒ chipMemTable(pc)

      dc.b        'Fake Table: $%l',10   ; %l ⇒ fastMemTable(pc)

      dc.b        0                      ; classic 0 terminated string

      even                               ; better safe than sorry ;)
```

Q&A

- ***How do I allocate RAM in my FX?***
  - The best way is to use BSS sections. DATA sections are fine too, but obviously make your .bin files bigger than a BSS. LDOS will allocate your sections for you when your FX starts and free them when your FX ends.
  - You may want to allocate buffers in a FX that will be used by other FX. In that case, see: `LDOS_PERSISTENT_CHIP_ALLOC` / `LDOS_PERSISTENT_CHIP_GET` / `LDOS_PERSISTENT_CHIP_TRASH`