Mozart under MicroSoft Windows

- How to build a Mozart system from scratch under Windows
- Implementation differences between the UNIX and Windows version

Building a Mozart Distribution for Windows

- Configure, compile and install
  - Using Cygwin
  - Using Mingw32
  - Using MS Visual C++
- Build a self-extracting EXE file

Compiling Mozart: using Cygwin

- URL: `http://sourceware.cygnus.com/cygwin`
- Library (DLL) plus C headers to ease porting UNIX applications to windows
- Includes EGCS and many UNIX tools (bash, grep, ls, mv, . . .)
- Drawbacks
  - All applications build with this package need `cygwin.dll` at runtime
  - Some functions don't work perfectly (like `select()`)
  - Mozart only used very few functions from the library
- Latest version can create binaries without reference to `cygwin.dll`:

```
gcc -mno-cygwin foo.c ...
```

Compiling Mozart: Using Mingw32

- URL:

  `www.xraylith.wisc.edu/~khan/software/gnu-win32`

- EGCS

  plus native Windows headers

  plus library wrappers for common Windows DLLs (kernel32.dll, wsock32.dll, ...)

- Builds native Windows applications: no special runtime DLL needed

Compiling Mozart: Using MS Visual C++

- Can compile and link emulator.
- Test suit passes.
- Difference to gcc: objects of classes with virtual functions seam to have their pointer to the method table in the *first* word (GCC puts it ”‘somewhere’” else).
- Drawbacks:
  - No threaded code
  - Does not inline all functions in the emulator

Building Mozart

- Have to be able to run configure: need a bourne shell:
  - Use Cygwin:
    - Does not yet work completely
    - Very slow
  - Use a cross compiler (recommended)

Building Mozart Using a Cross Compiler

- Build a cross compiler for Mingw32:

  `www.xraylith.wisc.edu/~khan/software/gnu-win32/mingw-cross-howto.txt`

- Installed into `/opt/mingw32-cross`

- Runs under Linux

- Problem: a complete bootstrap needs to compile the library

  $\longrightarrow$ need a Mozart system on the host system

  - First do a bootstrap on Linux

  - Then compile a Windows version

Building Mozart

```
setenv PATH /opt/mingw32-cross/bin:$PATH
cvs get -d ...  mozart

mkdir build-linux; cd build-linux
../mozart/configure
make bootstrap; make install; cd ..

mkdir build-win; cd build-win
../mozart/configure --target=i386-mingw32
make; make install
```

Running Mozart under Windows

- Create a self-extracting EXE (see below) and install it.

or run from the installation directory:

- Assumptions
  - your UNIX home dir is mounted under `Y:` on the windows machine
  - the installation dir is `Y:\system`
  - the source dir is `Y:\mozart`
- Just start:

  `Y:\system\winbin\oz`

  (use `winbin` since Linux shell scripts will go into `bin`)

Creating a self-extracting EXE

- Problem: Files are loaded using `http://www.mozart-oz.org`
- Installation dir contains a cache for this, which is just a link:

  `cache/http/www.mozart-oz.org/home`

  points to: `../../..`
- There are no links on Windows! (at least no real ones :-)
- Solution: create the cache directory and move files into it:

  ```
  make win32-i486
  ```

  Will create directory `oz3` containing the exact file system structure as it will be installed on Windows.

Creating a self-extracting EXE (cont.)

- Use Freeman installer (`http://netnet.net/~freeman`)
- It is freeware now
- Takes as input:
  - Files to be installed (expected under `Y:/mozart/oz3`)
  - Installation description (expected under `Y:/mozart/platform/wininstall/oz3.inf`)
- Output: self extracting exe `C:/temp/prj/online/run/setup.exe`
- Rename this to `mozart-1999MMDD.exe` (be careful: name must not be too long, otherwise you get strange errors!)
- TODO: check other installers (InstallShield)
- Demo!

Building emulator.exe with MS VC++

- ❏ run configure as before under Linux
- ❏ do

  ```
  make first
  ```

  in

  ```
  platform/emulator
  ```

- ❏ On Windows do:

  ```
  nmake -f Makefile.msvc
  ```

Implementation

Overview

- Wrapper-Scripts
- Tcl/Tk
- Emulator

Wrapper-Scripts

- C++ implementation of shell scripts used on UNIX
- located under `mozart/platform/mswindows`
- `oz.cc`, `ozengine.cc`: determine and set `OZHOME` and `OZPATH`
- `ozenginew.cc`: windowed version of `ozengine.cc`
  (does not open/need console window)
- `oztool.cc`: same as UNIX shell script, but allows to select compiler
  (egcs, Watcom, MS VC++)
- `ozwrapper.cc`, `ozwrapperw.cc`: create servlets (like `ozc`):
  - UNIX: take shell script header and append applet
  - Windows: take `ozwrapper.exe` and append applet
    TODO: better mark start of applet (now: 3 times SYSLETHEADER)

Tcl/Tk: UNIX

file `platform/wish/ozMain.c`

```
Tk_CreateFileHandler(stdin, StdinProc, ...);
...
void StdinProc(...)
{
  char *cmd = read(stdin);
  Tcl_Eval(cmd);
}
```

Tcl/Tk: Windows

- file `platform/wish/winMain.c`
- UNIX version does not work: Tcl does not detect when data is avail on stdin
- Create separate reader thread

```
Lock L;
char *cmd = NULL;
void reader() {
    while(true) {
        char *aux = read(stdin);
        lock L {
            if (cmd==NULL)
                Tcl_AsyncMark(fun() Tcl_DoWhenIdle(DoEval) end);
            cmd = append(cmd,aux); }}
}
```

DoEval runs in main thread:

```
void DoEval() {
    lock L {
        Tcl_Eval(cmd);
        cmd = NULL; }
}
```

Emulator

- Heap management
- Foreign functions
- IO (sockets, pipes)
- Miscellaneous
  - Alarm timer: there is no SIGALRM under windows, so we use a separate OS thread
  - Loading of pickles via http/ftp: do not fork a subprocess, but use an OS thread
  - Creating subprocesses

Heap Management

- Heap is increased in blocks à 512 kB
- Address space must not exceed 1 GB (i.e. top most 2 bits must be zero)
- Problem under Windows:

```
void *ptr1 = malloc(512KB);
...
free(ptr1);
...
void *ptr2 = malloc(512KB);
```

does not imply that `ptr1==ptr2`! (at least under NT)

Processes under Windows do not need to use a contiguous address space.

- Use VirtualAlloc/VirtualFree under Windows (like mmap/munmap under UNIX):
  - First let Windows decide: `VirtualAlloc(0,...)`
  - If new block $>$ 1 GB: search in lower addresses ourself

```
for (char * base=0; base += ...; addr != NULL)
    addr = VirtualAlloc(base,512kB,...);
```

Foreign Functions

- Example: create foo.dll using `OZ_atom` to be loaded into emulator.exe.
- DLLs can only have external references to other DLLs??
- On Windows `mozart.h` declares `OZ_atom` to be a *pointer* to a function.
- `OZ_atom` is defined in `mozart.c` (automatically created from `mozart.h`), which is linked into foo.dll by oztool transparently
- `mozart.c` also defines

```
OZ_linkFF(void** funs)
{ ...
   OZ_atom = funs[23];
   ...
}
```

- When emulator links foo.dll, it calls `OZ_linkFF`

IO

- Every clock tick and when idle the emulator checks for IO ready using `select()`
- Sockets work as under UNIX (minor additions: WSAStartup, WSAGetLastError)
- In particular `select()` works for sockets *only*, but not for other file handles!!!
- Solution: for every input (output) channel c
  - Create a socket c_s
  - Start an OS thread that reads (writes) from c and writes (reads) to c_s
  - Use `select()` for c_s

Books

- Jeffrey Richter: Advanced Windows (3rd Ed)
  Microsoft Press; ISBN: 1572315482
- MS Visual C++ online documentation