Win32 API Emulation on UNIX for Software DSM

Sven M. Paas, Thomas Bemmerl, Karsten Scholtyssik

Lehrstuhl für Betriebssysteme, RWTH Aachen, Germany

`http://www.lfbs.rwth-aachen.de/`

`contact@lfbs.rwth-aachen.de`

Agenda:

- Background
- Our approach: emulating a reasonable Win32 API subset
- Implementation details of **nt2unix** (multithreading, memory mapped I/O, ...)
- A case study: SVMlib: *Shared Virtual Memory Library*
- Conclusions

Background

The Problem:

- Given a **console** application written in C / C++ for Win32;

  --> Visual C++ 5.0 + STL, Windows NT 4.0 / Windows 95
- Compile (and execute) the same code on a UNIX system

  --> gcc 2.8.1 + STL, Solaris 2.6 [SPARC/x86], Linux 2.0 [x86]

Available Solutions:

- *Wind/U* (Bristol Technology, Inc., http://www.bristol.com/)
- *MainWin XDE* (MainSoft Corp., http://www.mainsoft.com/)
- *Willows Twin API* (Canopy Group, http://www.willows.com/)

A reasonable Win32 Subset

- NT Multithreading

  Creating / Destroying / Suspending / Resuming preemptive threads
  Synchronization and Thread Local Storage (TLS) functions;

- Virtual Memory (VM) Management

  Allocating / Committing / Protecting VM on page level
  Memory Mapping I/O, File Mapping

- NT Structured Exception Handling (SEH)

  User Level Page Fault Handling by SEH

- Networking using WinSock

  Windows Sockets API for TCP/IP

Windows NT Multithreading

Creating a Thread under NT:

```
WINBASEAPI HANDLE WINAPI CreateThread(
       LPSECURITY_ATTRIBUTES lpThreadAttributes,
       DWORD dwStackSize,
       LPTHREAD_START_ROUTINE lpStartAddress,
       LPVOID lpParameter, DWORD dwCreationFlags,
       LPDWORD lpThreadId);
```

with

```
typedef DWORD (WINAPI *PTHREAD_START_ROUTINE)(
       LPVOID lpThreadParameter);
typedef PTHREAD_START_ROUTINE LPTHREAD_START_ROUTINE;
```

UNIX Multithreading

Creating a Thread using POSIX API:

```
int pthread_create(
     pthread_t *new_thread_ID,
     const pthread_attr_t *attr,
     void *(*start_func)(void *), void *arg);
```

... and using the Solaris Thread API:

```
int thr_create(void *stack_base, size_t stack_size,
     void *(*start_func)(void *), void *arg, long flags,
     thread_t *new_thread_ID);
```

--> we must ignore `LPSECURITY_ATTRIBUTES.`
(like Windows 95 / 98 does)

NT Thread Synchronization

Problems:

- Susending / Resuming Threads is **not** possible within the POSIX Thead API! (-> SuspendThread(), ResumeThread())
- This fact implies that some Win32 thread concepts are hard to implement **efficiently** within POSIX environments:

```
struct ThreadInfo {
   DWORD state, suspendCount, exitCode;
#ifdef __POSIX_THREADS__
   pthread_cond_t cond, pthread_mutex_t mutex;
#else
   volatile BOOL threadHasBeenResumed;
#endif
};
```

Virtual Memory (VM) Management

Emulating a Windows NT File Mapping Object:

```
struct FileMapping {
  LPVOID lpBaseAddress;
   // the virtual base address of the mapping
  DWORD dwNumberOfBytesToMap;
   // the mapping size in bytes
  HANDLE hFileMappingObject;
   // the file handle
  char FileName[MAX_PATH];
   // the file name
  DWORD refcnt;
   // the number of references to the mapping
 };
 static vector<FileMapping> FileMappings;
```

NT Structured Exception Handling

Two methods:

- by embracing code with a `__try{}` ... `__except(){}` block;
- by installing a user level exception handler by calling **`SetUnhandledExceptionFilter`**().

Translation of NT Exception Codes to UNIX signals:

Windows NT **EXCEPTION_*** Code	UNIX Signal
ACCESS_VIOLATION	SIGSEGV
FLT_INVALID_OPERATION	SIGFPE
ILLEGAL_INSTRUCTION	SIGILL
IN_PAGE_ERROR	SIGBUS
SINGLE_STEP	SIGTRAP

Catching Page Faults

1st problem: **where** was the fault ?

```
switch (sig) {
case SIGSEGV:
// A segmentation violation.
ExceptionInfo.ExceptionRecord->
 ExceptionCode = EXCEPTION_ACCESS_VIOLATION;
ExceptionInfo.ExceptionRecord->
 ExceptionInformation[0] =
#if defined(__SPARC)
   (*(unsigned *)((ucontext_t*)uap)
    ->uc_mcontext.gregs[REG_PC] & (1<<21));
#elif defined(__X86)
   (((ucontext_t*)uap)->
    uc_mcontext.gregs[ERR] & 2);
#elif defined(__LINUXX86)
   stack[14] & 2;
#endif
```

Catching Page Faults (cont‘d)

<u>2nd problem</u>: **what** was the reason for the fault?

```
  if (ExceptionInfo.ExceptionRecord->
   ExceptionInformation[0])
   ExceptionInfo.ExceptionRecord->
    ExceptionInformation[0] = 1;
    // 1 == write access; 0 == read access
  ExceptionInfo.ExceptionRecord->
   ExceptionInformation[1] =
#ifdef __LINUXX86
    stack[22];
#else
    (DWORD)sip->si_addr;
#endif
  break;

  // other signals processed here ...
}
```

TCP/IP Networking using WinSock

Ideas: - Restrict WinSock 2.0 to BSD Socket API

- Translate data types, definitions, and error codes

For example:

```
typedef int           SOCKET;
#define INVALID_SOCKET(SOCKET)(-1)
#define SOCKET_ERROR  (-1)
```

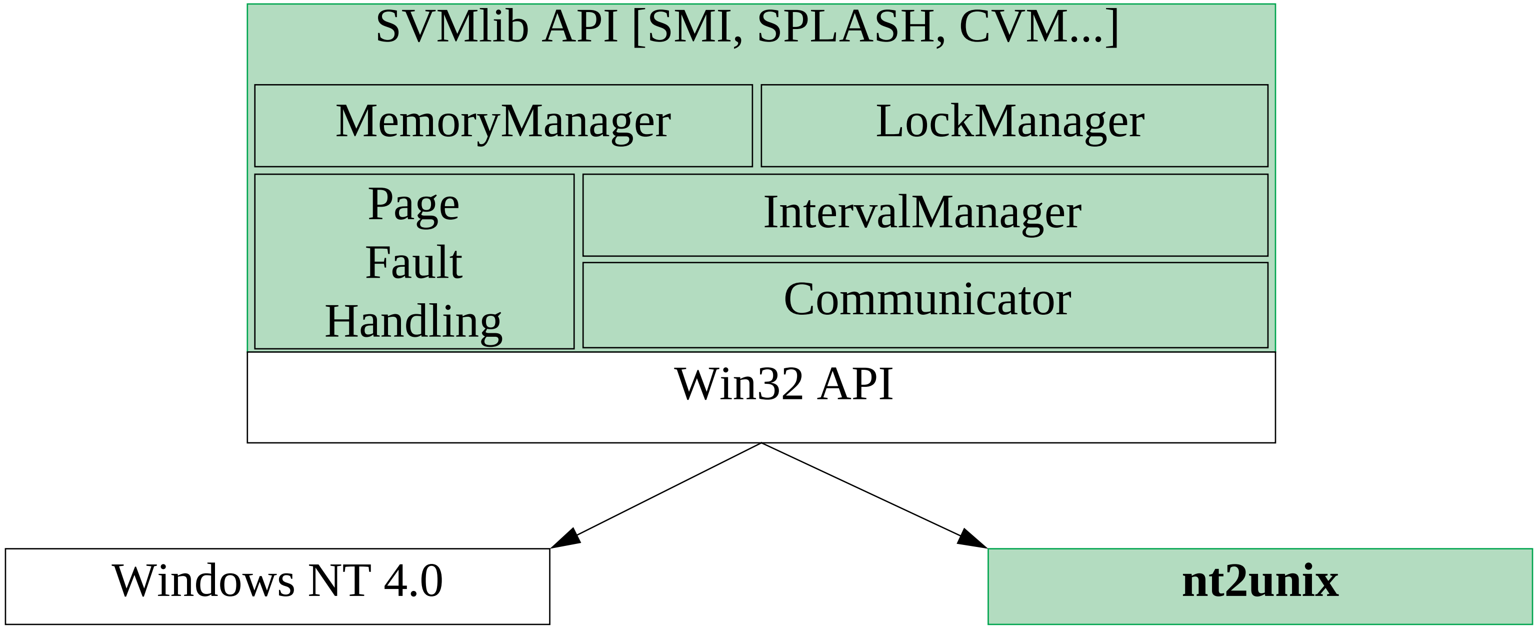
Pitfalls: - some types are hard to map (e.g. `fd_set`)

- WinSock‘s `select`() is **not** BSD `select`()!

A Case Study: SVMlib

SVMlib: *Shared Virtual Memory Library*

- all software, user-level, page based
- about 15.000 lines of (Visual) C++ code, natively for Win32



SVMlib API [SMI, SPLASH, CVM...]

MemoryManager

LockManager

Page Fault Handling

IntervalManager

Communicator

Win32 API

Windows NT 4.0

nt2unix

SVMlib Performance (1)

Page Fault Detection Time:

	Super-SPARC, 50 MHz	Pentium, 133 MHz	Pentium Pro, 200 MHz
Windows NT 4.0 Server / WS	-	28 µs	19 µs
Solaris 2.5.1 (native)	105 µs	70 µs	40 µs
Solaris 2.5.1 & nt2unix	135 µs	92 µs	48 µs

--> UNIX Signal handling is **expensive**.

SVMlib Performance (2)

Page Fault Handling Times:

Nodes	R / W / Avrg Fault Time [ms] CVM on Solaris (Sun SS20)	R / W / Avrg Fault Time [ms] SVMlib on nt2unix (Sun SS20)	R / W / Avrg Fault Time [ms] SVMlib on Windows NT (Intel Pentium 133)
2	11.3 / 0.8 / 4.4	4.5 / 1.3 / 2.2	3.4 / 1.1 / 1.8
3	12.0 / 0.8 / 5.8	4.6 / 1.8 / 2.7	3.4 / 1.4 / 2.3
4	16.7 / 0.9 / 7.1	4.9 / 1.8 / 3.1	4.0 / 1.5 / 2.4

Test Application: FFT

Conclusions

- Win32 API Emulation under UNIX **is** possible.
- If the Emulation is „application driven“,
  it can be implemented within **finite** time (3 MM for SVMlib);
- **nt2unix** is a reasonable first step to develop portable
  low level applications.

Next Steps:

- More complete implementation of Win32 base services;
- More applications (NT Services <-> UNIX Daemons)

Further Information

nt2unix Project Homepage:

`http://www.lfbs.rwth-aachen.de/~sven/nt2unix/`

SVMlib Project Homepage:

`http://www.lfbs.rwth-aachen.de/~sven/SVMlib/`

E-Mail:

`contact@lfbs.rwth-aachen.de`

Phone/Fax:

`+49-241-7634`, `+49-241-8888-339`