Liberating the lurking Smalltalk in Unix

(a work continually in progress)

Stephen Kell

stephen.kell@cl.cam.ac.uk

Computer Laboratory
University of Cambridge
$ cc -o hello hello.c
$ ./hello
Hello, world!
$ nano hello.c
$ cc -o hello hello.c
$ ./hello
Hello, world!
isEmpty

"Answer whether the receiver contains any elements."

self size = 0
<table>
<thead>
<tr>
<th>Unit</th>
<th>&quot;Unix&quot;</th>
<th>&quot;Smalltalk&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>Unit</td>
<td>&quot;Unix&quot;</td>
<td>&quot;Smalltalk&quot;</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>People say</td>
<td>binary, process, &quot;static&quot;</td>
<td>VM image, &quot;dynamic&quot;</td>
</tr>
<tr>
<td></td>
<td>“Unix”</td>
<td>“Smalltalk”</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>unit</td>
<td>binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>people say</td>
<td>“static”</td>
<td>“dynamic”</td>
</tr>
<tr>
<td>program from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td></td>
<td>“Unix”</td>
<td>“Smalltalk”</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>unit</td>
<td>binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>people say</td>
<td>“static”</td>
<td>“dynamic”</td>
</tr>
<tr>
<td>program from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td>change from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td></td>
<td>“Unix”</td>
<td>“Smalltalk”</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>unit</td>
<td>binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>people say</td>
<td>“static”</td>
<td>“dynamic”</td>
</tr>
<tr>
<td>program from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td>change from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td>deal in</td>
<td>opaque bytes</td>
<td><em>described</em> objects</td>
</tr>
<tr>
<td></td>
<td>“Unix”</td>
<td>“Smalltalk”</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>unit</td>
<td>binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>people say</td>
<td>“static”</td>
<td>“dynamic”</td>
</tr>
<tr>
<td>program from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td>change from</td>
<td>outside</td>
<td>inside</td>
</tr>
<tr>
<td>deal in</td>
<td>opaque bytes</td>
<td>described objects</td>
</tr>
<tr>
<td>outside is</td>
<td>opaque bytes</td>
<td>invisible magic</td>
</tr>
</tbody>
</table>
It’s demo time
“Any debugger [must] access the address space of the debugged program, ...
[but currently] there is no dynamic binding [of debuggers to programs]. ... We have tried to overcome [this] by providing an interface that is as uniform as possible.”

T.J. Killian

*Processes as files*

Usenix summer conference, 1984
Shared Libraries in SunOS

Abstract

The design and implementation of a shared library facility for Sun's implementation of the UNIX† operating system (SunOS) is described. Shared libraries extend the resource utilization benefits obtained from sharing code between processes running the same program to processes running different programs by sharing the libraries common to them.

In this design, shared libraries are viewed as the result of the application of several more basic system mechanisms, specifically:

- Kernel-supplied facilities for file-mapping and "copy-on-write" sharing
- A revised link editor supporting dynamic binding
- Compiler and assembler changes to generate position-independent code

The use of these mechanisms is transparent to applications code and build procedures, and also to library source code written in higher-level languages. Details of the use and operation of the mechanisms are provided, together with the policies by which they are applied to create a system with shared libraries. Early experiences and future plans are summarized.

1. Introduction

The UNIX operating system has long achieved efficiencies in memory utilization through sharing a single physical copy of the text (code) of a given program among all processes that execute it. However, a program text usually contains copies of routines from one or more libraries, and occasionally a program consists mostly of library routines. Considering that virtually every program makes use of routines such as printf(3), then at any given time there are as many copies of these routines competing for system resources as there are different active programs.

In an environment containing single-user systems, such as workstations, the likelihood of achieving acceptable resource utilization is usually determined by the limits set on the number of programs permitted to execute. In the limit of a single program in a system (the canonical example of such a system category), we observe the waste in storage resources, containing yet more copies of common library routines. Thus, there is a decreasing motivation to exploit the benefits of sharing to processes executing different programs, by sharing the libraries common to them.

This paper describes the design and implementation of a shared library facility for Sun's implementation of the UNIX† operating system, SunOS. We discuss our goals for such a facility, our approach to its design and implementation, and our plans for its use. We also discuss our early experiences, and our plans...

Robert Gingell, et al

Shared libraries in SunOS

Usenix summer conference, 1987
“Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of . . .”

Philip Greenspun
<table>
<thead>
<tr>
<th>Unix</th>
<th>Smalltalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>binary, process</td>
</tr>
<tr>
<td>people say</td>
<td>“static”</td>
</tr>
<tr>
<td>program from</td>
<td>either?</td>
</tr>
<tr>
<td>change from</td>
<td>either?</td>
</tr>
<tr>
<td>deal in</td>
<td>opaque bytes</td>
</tr>
<tr>
<td>outside is</td>
<td>opaque bytes</td>
</tr>
</tbody>
</table>
“If the v-code were translated to n-code . . ., the interpreter’s overhead could be eliminated and some optimizations become possible.”

L. Peter Deutsch & Allan Schiffman
*Efficient implementation of the Smalltalk-80 system*
POPL ’83
Software engineering wisdom

Target an interface, not an implementation.

Interfaces should hide change-prone details.

Don’t repeat yourself.
Local<Value> GetPointX(Local<String> property,
               const AccessorInfo &info) {
    Local<Object> self = info.Holder();
    Local<External> wrap = Local<External>::Cast(self->GetInternalField

    void* ptr = wrap->Value();
    int value = static_cast<Point*>(ptr)->x_;  
    return Integer::New(value);
}

void SetPointX(Local<String> property, Local<Value> value,
               const AccessorInfo& info) {
    Local<Object> self = info.Holder();
    Local<External> wrap = Local<External>::Cast(self->GetInternalField

    void* ptr = wrap->Value();
    static_cast<Point*>(ptr)->x_ = value->Int32Value();
}
On Fri, Jun 21, 2013 at 9:19 AM, Dan Carney <dca...@chromium.org> wrote:

The transition from Local to Handle won't happen for a while. It's more of a cleanup step after everything else is done, and there's no urgency since there shouldn't be any performance impact.

The callback signature changes alone break almost every single line of v8-using code I've written (tens of thousands of them), and I am still...
```javascript
var ffi = require("node-ffi");

var libm = new ffi.Library("libm", { "ceil": [ "double", [ "double" ] ] });
libm.ceil(1.5);  // 2

// You can also access just functions in the current process
var current = new ffi.Library(null, { "atoi": [ "int32", [ "string" ] ] });
current.atoi("1234");  // 1234
```
\texttt{libm.ceil(1.5)} \quad // \quad 2
\texttt{libc.atoi("1234");} \quad // \quad 1234

...?
double (*p_ceil)(double)
    = dlsym(RTLD_DEFAULT, "ceil");

Dl_info i;
dladdr(p_ceil, &i);
printf("%s\n", i.dli_sname); // "ceil"

// ... but only for "static" objects!
In Smalltalk’s “integrated environment”… there is little distinction between the compiler, interpreter, browser and debugger, [all of which] cooperate through shared data structures.... Pi is an isolated tool in a [Unix] “toolkit environment” [and] interacts with graphics, external data and other processes through explicit interfaces.

T.A. Cargill

Pi: a case study in object-oriented programming

OOPSLA ’86
Metadata demo
<table>
<thead>
<tr>
<th>Unix</th>
<th>Smalltalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit binary, process</td>
<td>VM image</td>
</tr>
<tr>
<td>people say “static”</td>
<td>“dynamic”</td>
</tr>
<tr>
<td>program from either?</td>
<td>inside</td>
</tr>
<tr>
<td>change from either?</td>
<td>inside</td>
</tr>
<tr>
<td>deal in described bytes</td>
<td>described objects</td>
</tr>
<tr>
<td>outside is opaque bytes</td>
<td>invisible magic</td>
</tr>
</tbody>
</table>
Enter liballocs

$ allocscc -o myprog ...
$ LD_PRELOAD=liballocs.so ./myprog
/* what's on the end of a pointer? */

struct uniqtype *
__liballocs_get_alloc_type(void *obj);

struct liballocs_err *__liballocs_get_alloc_info(
    const void *obj,
    memory_kind *out_memory_kind, /* ... */);
cf.

```
struct ellipse {
    double maj;
    double min;
}
struct point {
    double x, y;
    } ctr;
```
Quick demo – node + liballocs
Join me

http://github.com/stephenrkell/
feedback? wishlists?
Image credits

Squeak screenshot: Wolfgang Kreutzer

Phil Greenspun (minus Alex the Dog): Elsa Dorfman

Hot air balloons: AngMoKio

Flying cathedral: Friedrich Böhringer

Balloon over Brisbane: Cyron Ray Macey

Planets, rocket: NASA

copy of Escher’s “Hand in reflecting sphere”: Caia.or.kr

Vader reads the Bible: sirustalcelion
Conclusions

Unix is more dynamic than people think

Languages should be views, not worlds

Let’s build that infrastructure.

Thanks for your attention!