The language muddle and the system middle

Stephen Kell
s.r.kell@kent.ac.uk
Two myths
Programming languages are great...
Programming languages are great, but... programming systems > programming languages
Programming languages are great, but... 

programming systems \(\supset\) programming languages
Programming languages are great, but . . .

programming systems ⊇ programming languages

What do I mean, ‘programming systems’?
What defines Unix?

- programs and processes
- build a program (compile, assemble, link)
- run it
- the world is bytes
isEmpty

"Answer whether the receiver contains any elements."

t self size = 0
What defines Smalltalk?

- the program and process are the same: *the image*
- edit the program (no ‘compile, assemble, link’)
- … *as you run it*
- the world is objects
What is a programming system? Two things

First: ‘what it lets programmers do’

- includes language(s), but broader
- also: *the act* of programming
- create, execute, debug, modify, understand
What is a programming system? Two things

Second: ‘what it lets programs do’

- execute and interact
- includes standard libraries, but broader:
- also: the *between* (IPC; I/O; linkage; ‘foreign’)
- also: the meta-level (reflection, metaprogramming)
“An operating system is a collection of things that don’t fit into a language. There shouldn’t be one.”
“A programming language is more opinionated than an operating system; there can’t be just one.”

Everyone wants to invent their own language…

How about ‘one system, many languages’?

■ okay, but system has to accommodate diversity
What’s not a (whole) programming system?

The JVM, the CLR, . . .
- also includes some host environment (Unix-like)

The web
- also includes a server-side environment (Unix-like)

Theme: environments *build on* Unix
- and (in practice) seem to *include* it
The path of least resistance: make Unix better!

Making C better

- more \{debuggable, secure, ‘safe’\}

Making the *Unix process* a better environment

- including better at hosting high-level languages
Three little stories

A dynamically safe implementation of C
  ■ what does that mean?

Towards multiple languages *without* FFIs
  ■ instead, evolve Unix-like *memory*

Foundations for (many) garbage collectors
  ■ GC is the technical pretext for FFIs
A dynamically safe implementation of C

Our first story...

Once upon a time there was a language called C

It was very popular for a while

Then everyone started calling it names like ‘unsafe’

(... but still depended on it to do their dirty work)
Intro to undefined behaviour, due to John Regehr

```c
#include <limits.h>
#include <stdio.h>

int main (void)
{
    printf ("%d\n", (INT_MAX+1) < 0);
    return 0;
}
```
Intro to undefined behaviour, due to John Regehr

```c
#include <limits.h>
#include <stdio.h>

int main (void)
{
    printf ("%d\n", (INT_MAX+1) < 0);
    return 0;
}
```

```
$ cc test.c -o test
$ ./test
Formatting root partition, chomp chomp
```
Let’s try it!
Safety as a property of *evaluators*...
Safety as a property of evaluators…

“A [program] that works properly has the same behavior on both [unsafe and safe] evaluators. Unfortunately, for a program that misinterprets data we cannot predict how it behaves under the unsafe evaluator.”
Safety as a property of *evaluators*…

“A [program] that works properly has the same behavior on both [unsafe and safe] evaluators. Unfortunately, for a program that misinterprets data we **cannot predict** how it behaves under the unsafe evaluator.”

–Krishnamurthi & Felleisen

“Safety in programming languages”

Rice COMP TR99-352, 1999
“A [program] that works properly has the same behavior on both [unsafe and safe] evaluators. Unfortunately, for a program that misinterprets data we cannot predict how it behaves under the unsafe evaluator.”

–Krishnamurthi & Felleisen
“Safety in programming languages”
Rice COMP TR99-352, 1999

Safety is always about predictability

- whether static or dynamic
Safety as a property of *evaluators*…

“A [program] that works properly has the same behavior on both [unsafe and safe] evaluators. Unfortunately, for a program that misinterprets data we *cannot predict* how it behaves under the unsafe evaluator.”

–Krishnamurthi & Felleisen

“Safety in programming languages”
Rice COMP TR99-352, 1999

Safety is always about predictability

- whether static or dynamic

Can we have a “safe evaluator” for C?
The ‘infamously unsafe’ pointer arithmetic:

```c
int *p = /* ... */;
int *q = p + off;   // if this arithmetic is bad,
*q = /* ... */;    // anything could happen!
```
The ‘infamously unsafe’ pointer arithmetic:

```c
int *p = /* ... */;
int *q = p + off;  // if this arithmetic is bad,
                   *q = /* ... */;  // anything could happen!
```

Claim: this is ‘unsafe’, because... the C spec says:

If both the pointer operand and the result point to elements of the same array object, or one past the last element of the array object, the evaluation shall not produce an overflow; otherwise, the behavior is undefined.
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?

- same set of \{features, valid programs\}
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?

- same set of \{features, valid programs\}
- same set of behaviours in error-free executions
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?

- same set of \{features, valid programs\}
- same set of behaviours in error-free executions
- smaller set of behaviours on error
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?

- same set of \{features, valid programs\}
- same set of behaviours in error-free executions
- smaller set of behaviours on error

Not a subset or dialect. It’s a *refinement*!
Imagine a ‘safe’ implementation that translates to:

```c
int *p = /* ... */;
int *q = SAME_OBJECT(p, p+off) ? p + off : TRAP_VALUE(p + off);
*q = /* ... */; // traps if our arithmetic was bad
```

Is this some kind of new dialect of C? Perhaps a subset of C?

- same set of \{features, valid programs\}
- same set of behaviours in error-free executions
- smaller set of behaviours on error

Not a subset or dialect. It’s a *refinement*!

- undefined behaviour = *freedom to refine*
- we can ‘decide to be safe’, i.e. to fail cleanly
Let’s try it!
Our second story: memory as a shared abstraction

C’s trickiest undefined behaviours are about memory

- defined only if <some property of memory right now>

Our \texttt{SAME\_OBJECT()} instrumentation can only work by

- ... checking against a \textit{dynamic} model of memory

Something related: C doesn’t have an FFI

- it has memory! that’s it!
- memory is ‘shared’, in a general sense
static void walk_instrs(unsigned char *pos, unsigned char *end, 
    void (*cb)(unsigned char *, unsigned, void *), void *arg)
{
    unsigned char *cur = pos;
    while (cur < end)
    {
        unsigned len = instr_len(cur, end);
        cb(cur, len, arg);
        cur += (len ? len : 1);
    }
}
Programs abstract over memory

- memory holding stuff defined in C
- *alien* memory

The same expressions (lvalues) abstract over both!

Alien memory is “first-class”!

- supports the usual operations (read, write, execute)
- → little need for FFI
“If we succeed in making an Intergalactic Network, then our main problem will be learning to communicate with Aliens.”

J.C.R. Licklider
Custom memory allocators: the ultimate alien technology

In C, you can even get your memory from outside C

- `int *ptr = (int*) myalloc(sizeof (int));`
- `myalloc()` could be written in any language!
- could use memory obtained by any mechanism
- we still write ordinary C code against those objects
Some ordinary C code

```c
struct Point
{
    int x_; 
    int y_; 
};
```
Use it from JavaScript? The FFI horror

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();
}
The lesson of C

To get beyond FFIs, build a memory-like abstraction…

- … that *many* language implementations can share
- think *system*, not language!
- my answer is ‘typed allocations’
Evolving Unix: a reflective model of memory

```
$ cat /proc/self/maps
55f0bf200000  -55f0bf208000  r-xp  00000000  103:02  20578329  /bin/cat
55f0bf407000  -55f0bf408000  r--p  00007000  103:02  20578329  /bin/cat
55f0bf408000  -55f0bf409000  rw-p  00008000  103:02  20578329  /bin/cat
55f0bf68b000  -55f0bf6ac000  rw-p  00000000  00:00  0 [heap]
7f5f32bde000  -7f5f335ad000  r--p  00000000  103:02  30546546  /usr/lib/locale/locale-archive
7f5f335ad000  -7f5f33794000  r-xp  00000000  103:02  23728700  /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33794000  -7f5f33994000  ---p  001e7000  103:02  23728700  /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33994000  -7f5f33998000  r--p  001e7000  103:02  23728700  /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33998000  -7f5f3399a000  rw-p  001eb000  103:02  23728700  /lib/x86_64-linux-gnu/libc-2.27.so
7f5f3399a000  -7f5f3399e000  rw-p  00000000  00:00  0
7f5f3399e000  -7f5f339c5000  r-xp  00000000  103:02  23728672  /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33b9e000  -7f5f33bc5000  rw-p  00000000  00:00  0
7f5f33bc5000  -7f5f33bc6000  r--p  00027000  103:02  23728672  /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33bc6000  -7f5f33bc7000  rw-p  00028000  103:02  23728672  /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33bc7000  -7f5f33bc8000  rw-p  00000000  00:00  0 [stack]
7ffde13f1000  -7ffde1413000  rw-p  00000000  00:00  0 [vvar]
7ffde143d000  -7ffde1440000  r--p  00000000  00:00  0 [vdso]
7ffde1440000  -7ffde1440000  r-xp  00000000  00:00  0 [vsyscall]
```

Evolving Unix: memory mappings are ‘allocations’

```
$ cat /proc/self/maps

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Type</th>
<th>Offset</th>
<th>Flags</th>
<th>File</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>55f0bf200000</td>
<td>16K</td>
<td>r-xp</td>
<td>00000000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
</tr>
<tr>
<td>55f0bf407000</td>
<td>16K</td>
<td>r-p</td>
<td>00007000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
</tr>
<tr>
<td>55f0bf408000</td>
<td>16K</td>
<td>rw-p</td>
<td>00008000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
</tr>
<tr>
<td>55f0bf68b000</td>
<td>16K</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[heap]</td>
</tr>
<tr>
<td>7f5f32bde000</td>
<td>16K</td>
<td>r-p</td>
<td>00000000</td>
<td>103:02</td>
<td>30546546</td>
<td>/usr/lib/locale/locale-archive</td>
</tr>
<tr>
<td>7f5f335ad000</td>
<td>16K</td>
<td>r-xp</td>
<td>00000000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f5f33794000</td>
<td>16K</td>
<td>r-p</td>
<td>001e7000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f5f33994000</td>
<td>16K</td>
<td>r-p</td>
<td>001e7000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f5f33998000</td>
<td>16K</td>
<td>rw-p</td>
<td>001eb000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f5f3399a000</td>
<td>16K</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7f5f3399c000</td>
<td>16K</td>
<td>r-xp</td>
<td>00000000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f5f33b9e000</td>
<td>16K</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7f5f33bc5000</td>
<td>16K</td>
<td>r-p</td>
<td>00270000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f5f33bc6000</td>
<td>16K</td>
<td>rw-p</td>
<td>00280000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f5f33bc7000</td>
<td>16K</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7ffde13f1000</td>
<td>16K</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[stack]</td>
</tr>
<tr>
<td>7ffde143d000</td>
<td>16K</td>
<td>r-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vvar]</td>
</tr>
<tr>
<td>7ffde1440000</td>
<td>16K</td>
<td>r-xp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vdso]</td>
</tr>
<tr>
<td>ffffffff6000000-ffffffffff601000</td>
<td>r-xp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vsyscall]</td>
<td></td>
</tr>
</tbody>
</table>
```
Evolving Unix: memory mappings are *top-level* allocations
Evolving Unix: memory mappings are *top-level* allocations

Leaves in this tree can have *type information* attached
struct ellipse {
    double maj, min;
    struct { double x, y; } ctr;
};

■ also: functions, enums, ‘dynamic things’…
One place type info can come from

```bash
$ cc -g -o hello hello.c && readelf -wi hello | column
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_compile_unit</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_language</code></td>
<td>1 (ANSI C)</td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>hello.c</td>
</tr>
<tr>
<td><code>AT_low_pc</code></td>
<td>0x4004f4</td>
</tr>
<tr>
<td><code>AT_high_pc</code></td>
<td>0x400514</td>
</tr>
<tr>
<td><code>TAG_pointer_type</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_byte_size</code></td>
<td>8</td>
</tr>
<tr>
<td><code>AT_type</code></td>
<td>&lt;0x2af&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_subprogram</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>main</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_base_type</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_byte_size</code></td>
<td>4</td>
</tr>
<tr>
<td><code>AT_encoding</code></td>
<td>5 (signed)</td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_formal_parameter</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>argc</td>
</tr>
<tr>
<td><code>AT_type</code></td>
<td>&lt;0xc5&gt;</td>
</tr>
<tr>
<td><code>AT_location</code></td>
<td>fbreg - 20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_base_type</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_byte_size</code></td>
<td>1</td>
</tr>
<tr>
<td><code>AT_encoding</code></td>
<td>6 (char)</td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>char</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TAG_formal_parameter</code></td>
<td></td>
</tr>
<tr>
<td><code>AT_name</code></td>
<td>argv</td>
</tr>
<tr>
<td><code>AT_type</code></td>
<td>&lt;0x7ae&gt;</td>
</tr>
<tr>
<td><code>AT_location</code></td>
<td>fbreg - 32</td>
</tr>
</tbody>
</table>

p.41
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  # ← a popular JavaScript implementation
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  
> process.lm.printf ("Hello, world!\n")

Hello, world!

14
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  # <--- ... with liballocs extensions
> process.lm.printf("Hello, world!\n")
Hello, world!
14
> require('−lXt ');
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  # <- - ... with liballocs extensions
> process.lm.printf("Hello, world!
")
Hello, world!

14
> require(’-lXt’)
> var toplvl = process.lm.XtInitialize ( 
    process.argv[0], "simple", null, 0, 
    [process.argv.length], process.argv);
> var cmd = process.lm.XtCreateManagedWidget(  
    "exit", commandWidgetClass, toplvl, null, 0);
> process.lm.XtAddCallback(  
    cmd, XtNcallback, process.lm.exit, null );
> process.lm.XtRealizeWidget(toplvl);
> process.lm.XtMainLoop();

Not “JS ↔ C”! One object space, many per-language views
That’s substructure; what about \textit{inter-reference}?

Our richer memory should ‘know where the pointers are’

- to enable automatic garbage collection
- to enable ‘liveness’ [even for C code!]
- to truly eradicate FFIs
Good news: Unix knows where the pointers are

Let’s see it…
Bad news: only until program starts

Relocs tell us about all pointers at start of day

... but not once program starts:

- writing new values
- using stack / registers
- dynamically allocating memory

What to do?
Good news: we (now) have run-time type information!

(gdb) print obj
$1 = (const void *) 0x6b4880 # unknown type!
Good news: we (now) have run-time type information!

(gdb) print obj
$1 = (const void *) 0x6b4880 # unknown type!
(gdb) print __liballocs_get_alloc_type (obj)
$2 = (struct uniqtype *) 0x2b3aac997630
  (__uniqtype__Pptr_int$32)
Good news: we (now) have run-time type information!

```
(gdb) print obj
$1 = (void *) 0x6b4880
(gdb) print __liballocs_get_alloc_type (obj)
$2 = (struct uniqtype *) 0x2b3aac997630
  <__uniqtype___PTR_int$32>
(gdb) print *(int **) $2
$3 = 42
```
/* uniqtype for stack frame main_0x4004f6_0x400516 */

struct uniqtype __uniqtype_main_0x4004f6_0x400516 = {
    32 /* size */,
    { composite: { COMPOSITE, /* nmemb */ 2 } },
    /* members */ {
        { &__uniqtype____PTR___PTR_signed_char$8,
          /* offset */ 0 } /* argv (size 8) */ ,
        { &__uniqtype__int$32,
          /* offset */ 12 } /* argc (size 4) (HOLE of 4B) */
    }
};

Is that hole harmless?
objtool: Add tool to perform compile-time stack metadata validation

This adds a host tool named objtool which has a "check" subcommand which analyzes .o files to ensure the validity of stack metadata. It enforces a set of rules on asm code and C inline assembly code so that stack traces can be reliable.

For each function, it recursively follows all possible code paths and validates the correct frame pointer state at each instruction.

It also follows code paths involving kernel special sections, like .altinstructions, __jump_table, and __ex_table, which can add alternative execution paths to a given instruction (or set of instructions). Similarly, it knows how to follow switch statements, for which gcc sometimes uses jump tables.
Many works in progress

- pointer metadata using relocs + objtool-like
- temporal checking for C
- ‘liveness’ / dynamic update
- more/better FFI-less retrofitting: node but also . . .
- faster, more robust, . . .
Image credits

Great pyramid of Giza: Olaf Tausch GFDL

Hot air balloons: AngMoKio CC-BY-SA

Ken and Den: Peter Hamer CC-BY-SA

Dan Ingalls: a friend of Dan Ingalls

Squeak screenshot: Wolfgang Kreutzer

Eye of Sauron: Selrond CC-BY-SA 3.0

Linker diagram: Alastair Reid
Conclusions

Systems are (literally) greater than languages

The Unix system *can* be dragged forwards

… and it might be our only hope!

Join me: https://github.com/stephenrkell

Thanks for your attention!