Purpose-Built Languages in System Design

Mike Shapiro
mws@sun.com

Little Languages in UNIX: Early 80's

- **Debuggers**
  - adb
  - dbx
  - stabs
- **Programming Tools**
  - lex
  - make
  - m4
  - ratfor
  - yacc
- **Shells and Utilities**
  - sh
  - csh
  - awk
  - bc, dc
  - ed, ex, vi
  - sed
- **Games**
  - rogue
  - trek
- **Text Formatting**
  - eqn
  - pic
  - nroff, troff
- **Libraries**
  - regex
- **Configuration**
  - sendmail.cf

Solaris 10: About 20 Years Later ...

- DTrace
- Fault Management
- Service Management
- Zones
- ZFS
- FireEngine TCP/IP

Solaris 10: About 20 Years Later ...

- DTrace
  - "D" - Language for describing dynamic tracing queries
- Fault Management
  - "Eversholt" - Language for describing fault propagations
- Service Management
  - XML DTD and svccfg(1M) configuration language
- Zones
  - XML DTD and zonecfg(1M) configuration language
- ZFS
- FireEngine TCP/IP

Part 1: Evolution Trumps Intelligent Design

ODT-8 on the PDP circa 1967

- *nnnn*B - set a breakpoint at nnnn
- *nnnn*G - go to location nnnn
- C - continue from a breakpoint
- / - edit the value of a location

> **400** / - display location 400
> **400** / 6046 - observe its value is 6046
> **400** / 6046 **2345** - rewrite it to 2345

ODT-8 Octal Debugging Technique Manual
DEC-D8-COCO-D December 1967

UNIX V3 db(1) circa 1971

- *expression* / - print a word in octal
- *expression* \ - print a byte in octal
- *expression* ‘ - print a byte as ASCII
- *expression* ? - disassemble an instruction
- *expression* = - print expression in octal
- *expression* : - find the closest symbol
- $ - print the registers

UNIX Version 3 Manual Pages, DB(I), 11/3/1971

AT&T UNIX SVR3 adb(1) circa 1980

- *expression* / *f* - retrieve value from memory
- *expression* ? *f* - retrieve value from object file
- *expression* = *f* - compute value of expression
- Format characters:
  - o or O for octal 2 or 4 bytes
  - d or D for decimal 2 or 4 bytes
  - x or X for hexadecimal 2 or 4 bytes
- `$r` - register dump `:b` - breakpoint
- `$a` - Algol backtrace `:c` - continue
- `$c` - C backtrace `$<` - read macro

Kernel Debugging on SunOS, 1984

```
address$<proc
./"link"16t"rlink"16t"nxt"16t"prev"nXXXX
+/"as"16t"segu"16t"stack"16t"uarea"nXXXX
+/"upri"
+/"upri"8t"pri"8t"cpu"8t"stat"8t"time"8t"nice"nbbbbbb
+/"slp"8t"cursig"16t"sig"bbX
+/"mask"16t"ignore"16t"catch"nXXX
+/"flag"16t"uid"8t"suid"8t"pgrp"nXddd
+/"pid"8t"ppid"8t"xstat"8t"ticks"nddxd
+/"cred"16t"ru"16t"tsize"nXXX
+/"dsize"16t"ssize"16t"rssize"nXXX
+/"maxrss"16t"swrss"16t"wchan"nXXX
+/16+"%cpu"16t"pptr"16t"tptr"nXXX
+/"real itimer"n4D
+/"idhash"16t"swlocks"ndd
+/"aio forw"16t"aio back"8t"aio count"8t"threadcnt"nXXXX
```

Kernel Debugging on SunOS, 1984

```
#include <sys/param.h>
#include <sys/time.h>
#include <sys/proc.h>

proc
./"link"16t"rlink"16t"nxt"16t"prev"n{p_link,X}{p_rlink,X}{p_nxt,X}{p_prev,X}
+/"as"16t"segu"16t"stack"16t"uarea"n{p_as,X}{p_segu,X}{p_stack,X}{p_uarea,X}
+/"upri"8t"pri"8t"cpu"8t"stat"8t"time"8t"nice"n{p_usrpri,b}{p_pri,b}{p_cpu,b}
{p_stat,b}{p_time,b}{p_nice,b}
+/"slp"8t"cursig"16t"sig"n{p_slptime,b}{p_cursig,b}{p_sig,X}
+/"mask"16t"ignore"16t"catch"n{p_sigmask,X}{p_sigignore,X}{p_sigcatch,X}
+/"flag"16t"uid"8t"suid"8t"pgrp"n{p_flag,X}{p_uid,d}{p_suid,d}{p_pgrp,d}
+/"pid"8t"ppid"8t"xstat"8t"ticks"n{p_pid,d}{p_ppid,d}{p_xstat,x}{p_cpticks,d}
+/"cred"16t"ru"16t"tsize"n{p_cred,X}{p_ru,X}{p_tsize,X}
+/"dsize"16t"ssize"16t"rssize"n{p_dsize,X}{p_ssize,X}{p_rssize,X}
+/"maxrss"16t"swrss"16t"wchan"n{p_maxrss,X}{p_swrss,X}{p_wchan,X}
+/"%cpu"16t"pptr"16t"tptr"n{p_pctcpu,X}{p_pptr,X}{p_tptr,X}
+/"real itimer"n{p_realtimer,4D}
+/"idhash"16t"swlocks"n{p_idhash,d}{p_swlocks,d}
+/"aio forw"16t"aio back"8t"aio
count"8t"threadcnt"n{p_aio_forw,X}{p_aio_back,X}{p_aio_count,X}
{p_threadcnt,X}
```

Solaris mdb(1) circa 1997-99

- Complete rewrite with full adb(1) compatibility
- Extended command set - `::`*command*
- Loadable modules provide dcmds and walkers
  - dcmds – custom debugging utilities written to C API
  - walkers – perform iteration of complex data structures
- Arithmetic “pipelines” for forming queries:

```
> ::walk proc p | ::print proc_t p_cred->cr_uid |
  ::grep .==19 | ::eval <p::ps

S     PID PPID   PGID    SID  UID      ADDR NAME
R 100755     1 100755 100755   25 781a68e0 sendmail
```

Looking Back

- A language can have no design, no parser, no name, and last 30+ years if it connects to its users
- Key idea is that when debugging a dump you:
  - Start with an address (global symbol, register, stack)
  - Perform a query to compute a new address
  - Take this new address, perform another query ...
- Implementation often guides syntax: with modern tools and CPUs, this isn't necessarily a good thing

Part 2: Simplicity and the Power of What You Know

DTrace Program Structure

probe-description

/ predicate-expression /

{

    probe statements

    ...

}

DTrace Program Structure

```
kmem_alloc:entry
/ arg0 > 64 /
{
    stack( );
}
```

Early Design Decisions

- Expressions, syntax, and types would follow C as closely as possible: D is C's debugging companion
- It should be very easy to write programs on the command-line and then move into an editor if needed
- Language would be statically typed like C, but infer types and declarations automatically if possible
- Syntax should reflect concepts and notation already understood by users, not the implementation
- Type system would support multiple namespaces, including the implicit inclusion of all kernel types

Static Types with Inferred Declaration

```
probe-description
/ x++ /
{
   trace(x);
}
```

```
probe-description
/ x++ && y == 0 /
{
   trace(x + y++);
}
```

```
probe-description
/ x == 0 /
{
   trace(x++);
}
```

Early DTrace Program

```
bcopy:entry
/ arg2 > 1000 &&
  curthread->t_procp->p_cred->cr_uid == 31992 /
{
     stack(20);
}
```

```
dtrace: 2 probes enabled.
CPU     ID                    FUNCTION:NAME
  1   8576                      bcopy:entry

genunix`getproc+0x4f8
genunix`cfork+0x64
```

D Translators and Inlines

```
translator lwpsinfo_t < kthread_t *T > {
  pr_syscall = T->t_sysnum;
  pr_pri = T->t_pri;
  pr_clname = `sclass[T->t_cid].cl_name;
  ...
}

inline lwpsinfo_t *curlwpsinfo =
  xlate <lwpsinfo_t *> ( curthread );

trace(curlwpsinfo->pr_pri);
```

Examining File Descriptors

```
inline fileinfo_t fds[int fd] =
  xlate <fileinfo_t> (
  fd >= 0 && fd < curthread->t_procp->
    p_user.u_finfo.fi_nfiles ?
  curthread->t_procp->p_user.u_finfo.
    fi_list[fd].uf_file : NULL );


syscall::write:entry
/ execname == "ksh" &&
  (fds[arg0].fi_oflags & O_APPEND) /
{
  printf("ksh %d appending to %s\n", pid,
    fds[arg0].fi_pathname);
}
```

Computing Stability

```
$ dtrace -e -v -s a.d
Stability attributes for script a.d:
  Minimum Statement Attributes
    Identifier Names: Stable
    Data Semantics: Stable
    Dependency Class: Common
  ...
```

Lessons

- Leverage existing knowledge, syntax, and paradigms as much as you possibly can
- Plan for namespaces, stability, versioning, and extension very early in language design
- Keep it simple: use syntactic sugar sparingly and only when there is a compelling need to do so
- Syntax should express intent, not implementation
  - a[b, c] = d;
  - @a[b, c] = sum(d);
  - printf(“%s%d”, “cs”, 169);

Part 3: Mutants

An Early Mutant: RATFOR

```
   if (a .gt. b) goto 10
      sw = 0
      write(6, 1) a, b
      goto 20
10 sw = 1
   write(6, 1) b, a
20 ...
```

```
if (a <= b) {
   sw = 0;
   write(6, 1) a, b
} else {
   sw = 1;
   write(6, 1) b, a)
}
```

Brian Kernighan, "RATFOR – A Preprocessor for a Rational Fortran," *Software—Practice and Experience*, October 1975.

Mystery Mutant #1

```
localsym(cframe)
L_INT cframe;
{
        INT symflg;
        WHILE nextsym() ANDF localok
                ANDF symbol.symc[0]!='~'
                ANDF (symflg=symbol.symf)!=037
        DO IF symflg>=2 ANDF symflg<=4
           THEN localval=symbol.symv;
                return(TRUE);
           ELIF symflg==1
           THEN localval=leng(shorten(cframe)+symbol.symv);
                return(TRUE);
           ELIF symflg==20 ANDF lastframe
           THEN localval=leng(lastframe+2*symbol.symv-10);
                return(TRUE);
           FI
        OD
        return(FALSE);
}
```

Algol-68 Revisited: cpp(1) in adb(1)

```
 1 #
 2 /*
 3  *      UNIX debugger
 4  */
 5
 6 #define TYPE    typedef
 7 #define STRUCT  struct
 8 #define UNION   union
 9 #define REG     register
10
11 #define BEGIN   {
12 #define END     }
13
14 #define IF      if(
15 #define THEN    ){
16 #define ELSE    } else {
17 #define ELIF    } else if (
18 #define FI      }
19
20 #define FOR     for(
21 #define WHILE   while(
22 #define DO      ){
23 #define OD      }
24 #define REP     do{
25 #define PER     }while(
26 #define DONE    );
27 #define LOOP    for(;;){
28 #define POOL    }
29
30 #define SKIP    ;
31 #define DIV     /
32 #define REM     %
33 #define NEQ     ^
34 #define ANDF    &&
35 #define ORF     ||
36
37 #define TRUE     (-1)
38 #define FALSE   0
39 #define LOBYTE  0377
40 #define HIBYTE  0177400
41 #define STRIP   0177
42 #define HEXMSK  017
43
44 #define SP      ' '
45 #define TB      '\t'
46 #define NL      '\n'
47 #define EOF     0
48
```

Mystery Mutant #2

```
headers
meta

code syscall? ( call# -- ok? )
  'user syscall-vec   %l0  nget
  bubble
32\ tos 2     tos   slln     \ multiply by 4
64\ tos 3     tos   slln     \ multiply by 8
  %l0 tos     %l0   ld       \ address of routine
  %l0 %g0     %g0   subcc
  0=  if
     %g0  1   tos   sub      \ (delay)
     %g0      tos   move
  then
```

Forth + SPARC = Crazy Delicious

```
headers

hex
00 op3 ld

: op3   ( opcode -- )
   createw, does> w@set-op3 ( rs1 [ rs2 | imm ] rd )
      rd src
;

: set-op   ( n class -- )   d# 30 <<   swap   d# 19   <<
      +   setbits   ;
: wcreate   ( n -- )   create w,   [compile] does>
      compile w@   ;   immediate
: set-op2   ( n -- )   2 set-op   ;
: w@set-op3   ( adr -- )   w@ 3 set-op   ;
: w@set-op2   ( adr -- )   w@ set-op2   ;
: createw,   ( n -- )   create w,   ;
```

Some Successful Mutations

- Classic UNIX:
  - lex and flex
  - yacc and bison
  - rpcgen
- Higher-level Software:
  - Visual Basic
  - RelaxNG

Designed Mutation: XML-RPC CrazyCall

```
<methodCall>
  <methodName>foo</methodName>
    <params>
      <param>
        <value><struct>
          <member>
            <name>bar</name>
            <value><int>123</int></value>
          </member>
          <member>
            <name>baz</name>
            <value><int>256</int></value>
          </member>
        </struct></value>
      </param>
  </params>
</methodCall>
```

Extension #1: MultiCall

```
x.foo({ bar: 123, baz: 456 });
```

```
x.multicall(
   { methodName: 'foo',
      params: [ { bar: 123, baz: 456 } ] },
   { methodName: 'foo',
      params: [ { bar: 789, baz: 987 } ] },
   { methodName: 'foo',
      params: [ { bar: 1, baz: 2 } ] },
   ...
)
```

Extension #2: CrazyCall

```
x.multicall(
   { methodName: 'foo.baz', methodParams: [
      { methodName: 'foo.bar', params: [ 1, 2, 3 ] }
   ] },
   ...
);


{ 'if': [ { equals: [ { methodName: 'foo.bar' }, 0 ]},[
   { literal: 'foo.bar() => zero' }
], [
   { literal: 'foo.bar() => non-zero' }
] ] }
```

Extension #3: CrazyCall Stack

```
x.multicall(
   { foreach: [ [ 2, 4, 6 ], [
      { push: [ ] },
      { div: [ { pop: [] }, 2 ] }
   ] ] }
);
```

- Result is a general-purpose system for using extant XML-RPC infrastructure to send arbitrary channel programs
- All of this functions in existing browser clients and with existing XML-RPC bindings (e.g. Perl, Python, Ruby)

Random Mutation: <unistd.h>

```
#if defined(_XPG4) || defined(__EXTENSIONS__)
extern size_t confstr(int, char *, size_t);
extern char *crypt(const char *, const char *);
#endif /* defined(_XPG4) || defined(__EXTENSIONS__) */
#if !defined(_POSIX_C_SOURCE) || defined(_XOPEN_SOURCE) || \
        defined(__EXTENSIONS__)
extern char *ctermid(char *);
#endif /* (!defined(_POSIX_C_SOURCE) ... */
#if !defined(__XOPEN_OR_POSIX) || defined(_REENTRANT) ||
defined(__EXTENSIONS__)
extern char *ctermid_r(char *);
#endif /* !defined(__XOPEN_OR_POSIX) || defined(_REENTRANT)
... */
/* Marked as LEGACY in SUSv2 and removed in SUSv3 */
#if !defined(_XPG6) || defined(__EXTENSIONS__)
extern char *cuserid(char *);
```

fcc: C Decls with Custom Attributes

```
#include <sys/types.h>

__fcc_attr xdr(string);

int8_t __fcc_attr xdr("xdr_int8_t");
float __fcc_attr xdr("xdr_float");

interface foo.bar.baz 1 {
  uint64_t gethrtime(void);
  ...
}
```

Small Languages in Systems

- The operating system is a set of resource managers:
  - abstractions with programming and administrative models
  - abstractions are organized into namespaces:
    - processes, threads, uids, sockets, file pathnames, zones, ...
- Purpose-built languages can serve to:
  - Create, destroy, and modify the base abstractions
  - Introspect into and glue together the rest of the system
  - Simplify the construction and extension of the system
  - Enumerate, organize, navigate, query namespaces
- As the system expands, we will need more of these in order to more competently manage growth and change