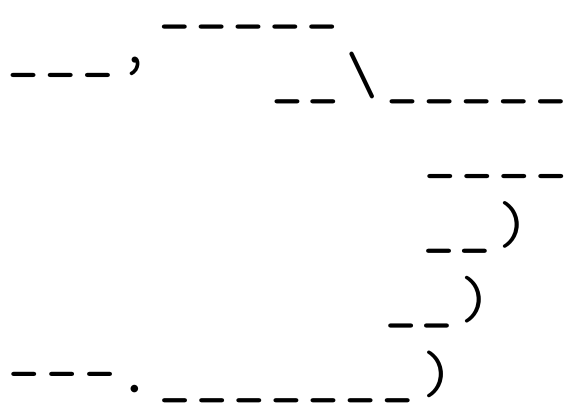


GNU poke
The 2020 Update

The extensible editor
for structured binary data

Jose E. Marchesi

GNU Tools Track @ LPC 2020

Disclaimer

This is still **fun in progress**

Contents

Introduction to GNU poke

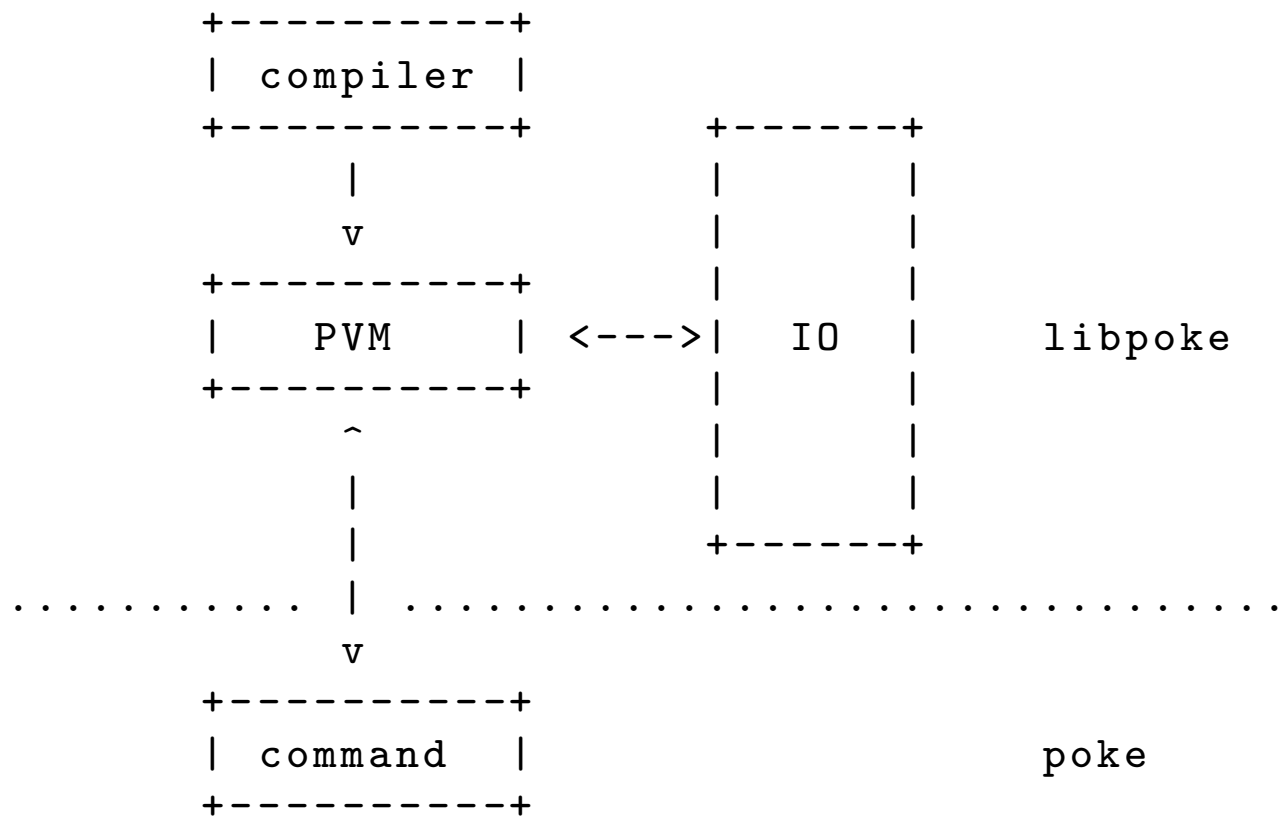
Motivation

- Need to edit object files, among others.
- Scripts break easily, and are a PITA to maintain.
- Format-specific tools are... too specific.
- Decided to hack a general-purpose binary editor in 2017.
- Published first version in September 2019.
- About to release the first version.

Developing the idea

- Took a while.
- From C structs “plus something” to a domain-specific programming language.
- Some existing work:
  - **Datascript** by Godmar Back: nice but unsatisfactory.
  - 010 Editor: proprietary and simplistic approach.
- After many design failures and blind alleys... finally got it right... or so I hope! :D

Architecture



poke, Poke and pickles

- We call **poke** the program. When the context may induce confusion (since “poke“ is a pretty common word) then we use “GNU poke“.
- **Poke** (with upper case P) is the name of the domain-specific language implemented by poke.
- A **pickle** is a Poke source file containing definitions that conceptually apply to some definite domain.

The Poke language - Values

- Integers:

```
10, 0xff, 8UB, 0b1100, 0o777
```

- Strings:

```
"foo\nbar"
""
```

- Arrays:

```
[1,2,3]
[[1,2],[3,4]]
[[1,2,3],[4]]
```

- Structs:

```
struct { name = "Donald␣Knuth", age = 100 }
struct {}
```

The Poke language - Offset values

- Named units:

```
8#b
23#B
2#Kb
```

- Numeric units:

```
8#8
2#3
```

- Even better:

```
deftype Packet = struct { int i; long j; }
23#Packet
```

- Operations:

```
OFF +- OFF -> OFF
OFF *  INT -> OFF
OFF /  OFF -> INT
OFF %  OFF -> OFF
```

The Poke language - Array Types

- Unbounded:

```
int[]
int[][]
```

- Bounded by number of elements:

```
int[2]
int[foo+bar]
```

- Bounded by size:

```
int[8#B]
```

The Poke language - Struct Types

```
deftype Elf64_Shdr =
  struct
  {
    offset<Elf_Word,B> sh_name;
    Elf_Word sh_type;
    Elf64_SectionFlags sh_flags;
    Elf64_Addr sh_addr;
    Elf64_Off sh_offset;
    offset<Elf64_Xword,B> sh_size;
    Elf_Word sh_link;
    Elf_Word sh_info;
    Elf64_Xword sh_addralign;
    offset<Elf64_Xword,b> sh_entsize;
  };
```

The Poke language - Variables

Block oriented. Lexically scoped.

```
defvar a = 10
defvar b = [1,2,3]
defvar c = { foo = 10, bar = 20L }
```

The Poke language - Mapping

A central concept in poke:

- Poke variables are in memory.
- The IO space is the data being edited (file, memory, ...)
- Both can be manipulated **in the same way**.
- ... or that's the idea.

The Poke language - Mapping

TYPE @ OFFSET `-> MAPPED_VALUE`

- Simple types

```
(poke) defvar a = 10
(poke) defvar b = int @ 0#B
```

- Arrays

```
(poke) defvar a = [1,2,3]
(poke) defvar b = int[3] @ 0#B
```

- Structs

```
(poke) defvar a = Packet { i = 10, j = 20 }
(poke) defvar b = Packet @ 0#B
```

The Poke language - Functions

```
defun ctf_section = (Elf64_Ehdr ehdr) Elf64_Shdr:
{
 for (s in Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff)
   if (elf_string (ehdr, s.sh_name) == ".ctf")
     return s;

 raise E_generic;
}
```

What is New

Styled Output in Terminal

- Based on Bruno Haible's **libtextstyle**.
- User can customize styling using CSS (`poke-default.css`).
- Command-line:

```
--color=yes
--color=no
--color=html
```

- Language-level support in **printf**:

```
printf "%<CLASSNAME:...>";
```

Terminal Hyperlinks

- Also based on **libtextstyle**.
- Hyperserver.

```
hserver listening in port 53577.
```

- **Execute** links:

```
app://ankh-morpork:46603/1592/e/.file #0
```

- **Insert** links:

```
app://ankh-morpork:46603/1592/i/0x00000000#b
```

- Considering adding a customizable toolbar:

```
<dothis><dothat>
(poke) _
```

Union Types

```
deftype Id3v2_Frame =
  struct
  {
    char id[4] : id[0] != 0;
    uint32 size;
    ...
    union
     {
       /* Frame contains text related data.  */
       union
       {
         struct
         {
           char id_asciiz_str = 0;
           char[size - 1] frame_data;
         } : size > 1;

         char[size] frame_data;
       } : id[0] == 'T';

       /* Frame contains other data.  */
       char[size] frame_data;
     };
  };
```

Initial value for a struct field

```
deftype Elf64_Ehdr =
  struct
  {
    struct
    {
      byte[4] ei_mag = [0x7fUB, 'E', 'L', 'F'];
      byte ei_class;
      byte ei_data;
      byte ei_version;
      byte ei_osabi;
      byte ei_abiversion;
      byte[6] ei_pad;
      offset<byte,B> ei_nident;
    } e_ident;

    Elf_Half e_type;
    Elf_Half e_machine;
    Elf_Word e_version;

    [...]
  };
```

Denotes a constraint **and** a default initial value.

Optional struct fields

```
deftype Elf64_File =
  struct
  {
    Elf64_Ehdr ehdr;

    Elf64_Shdr[ehdr.e_shnum] shdr @ ehdr.e_shoff
      if ehdr.e_shnum > 0;

    Elf64_Phdr[ehdr.e_phnum] phdr @ ehdr.e_phoff
      if ehdr.e_phnum > 0;

    [...]
  };
```

Fields may be absent.

Struct constructors

There are two ways to create a struct value in Poke:

- **Mapping** a struct at some IO space.
- **Constructing** a struct from a template with initial values.

```
deftype Exception =
  struct
  {
    int<32> code;
    string msg;
    int<32> exit_status;
  };

defvar E_div_by_zero
  = Exception {code = EC_div_by_zero,
               msg = "division␣by␣zero",
               exit_status = 1};
```

Definitions inside Struct Types

Variables, functions and other types can be defined inside struct types, and are available at value construction time.

Definitions inside Struct Types

```
deftype Packet =
  struct
  {
    byte magic = 0xab;
    byte size;

    defvar real_size = (size == 0xff ? 0 : size);

    byte[real_size] payload;
    byte[real_size] control;

    defun corrected_crc = int:
    {
      try return calculate_crc (payload, control);
      catch if E_div_by_zero { return 0; }
    }

    int crc = corrected_crc;
  };
```

Definitions inside Struct Types

```
deftype BTF_Type =
  struct
  {
    offset<uint<32>,B> name;
    struct [...] info;

    deftype BTF_Func_Proto =
      struct
      {
        BTF_Param[info.vlen] params;
      };

    union
    {
      BTF_Int integer                          : info.kind == BTF_KIND_I
      BTF_Array array                          : info.kind == BTF_KIND_A
      BTF_Enum[info.vlen] enum                 : info.kind == BTF_KIND_E
      BTF_Func_Proto func_proto                : info.kind == BTF_KIND_F
      BTF_Variable variable                    : info.kind == BTF_KIND_V
      BTF_Member[info.vlen] members            : info.kind == BTF_KIND_U
      BTF_Var_SecInfo[info.vlen] datasec       : info.kind == BTF_KIND_D
    } data;

  };
```

Struct Methods

```
deftype Elf64_File =
  struct
  {
    [...]

    method get_string = (offset<Elf_Word,B> offset) string:
      {
        defvar strtab = ehdr.e_shstrndx;
        return string @ (shdr[strtab].sh_offset + offset);
      }

    method get_section_by_name = (string name) Elf64_Shdr:
      {
        for (s in shdr where get_string (s.sh_name) == name)
          return s;

        raise E_generic;
      }
  };
```

Pretty-printers

```
deftype BPF_Reg =
  struct
  {
   uint<4> code;

   method _print = void:
   {
    print "#<";
    if (code < BPF_R9)
      printf "%<insn-register:%sr%i32d%>", "%", code;
    else
      printf "%<insn-register:fp%>";
    print ">";
   }
  };
```

#<...> convention

Pretty-printers

```
(poke) .set pretty-print yes
(poke) (BPF_Insn[shdr.sh_size] @ shdr.sh_offset)[0]
BPF_Insn {
  opcode=BPF_Insn_Opcode {
    alujmp=#<mov>
  },
  regs=BPF_Insn_Regs {
    src=#<%r0>,
    dst=#<%r0>
  },
  offset=0x0H#64,
  imm=struct {
    imm32=0xa
  }
}
```

Pretty-printers

```
(poke) .set pretty-print no
(poke) (BPF_Insn[shdr.sh_size] @ shdr.sh_offset)[0]
BPF_Insn {
  opcode=BPF_Insn_Opcode {
    alujmp=struct {
      code=0xbUN,
      src=(uint<1>) 0x0,
      class=(uint<3>) 0x7
    }
  },
  regs=BPF_Insn_Regs {
    src=BPF_Reg {
      code=0x0UN
    },
    dst=BPF_Reg {
      code=0x0UN
    }
  },
  offset=0x0H#64,
  imm=struct {
    imm32=0xa
  }
}
```

Memory IO spaces

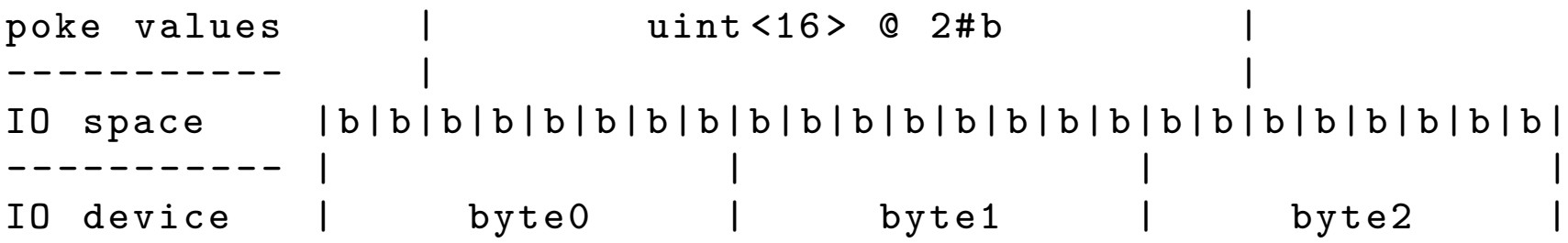
```
(poke) .mem foo
The current IOS is now ‘*foo*’.
(poke) dump
76543210  0011 2233 4455 6677 8899 aabb ccdd eeff  0123456789ABCDE
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
(poke) iosize / #B
0x1000UL
```

Auto-growing memory buffers

NBD IO spaces

- Provides accessing data served by an arbitrary NBD (Network Block Device) server.
- Based on libnbd (`http://libguestfs.org/libnbd.3.html`).
- `nbd+unix:///?socket=path/to/file`
- Contributed by Eric Blake.

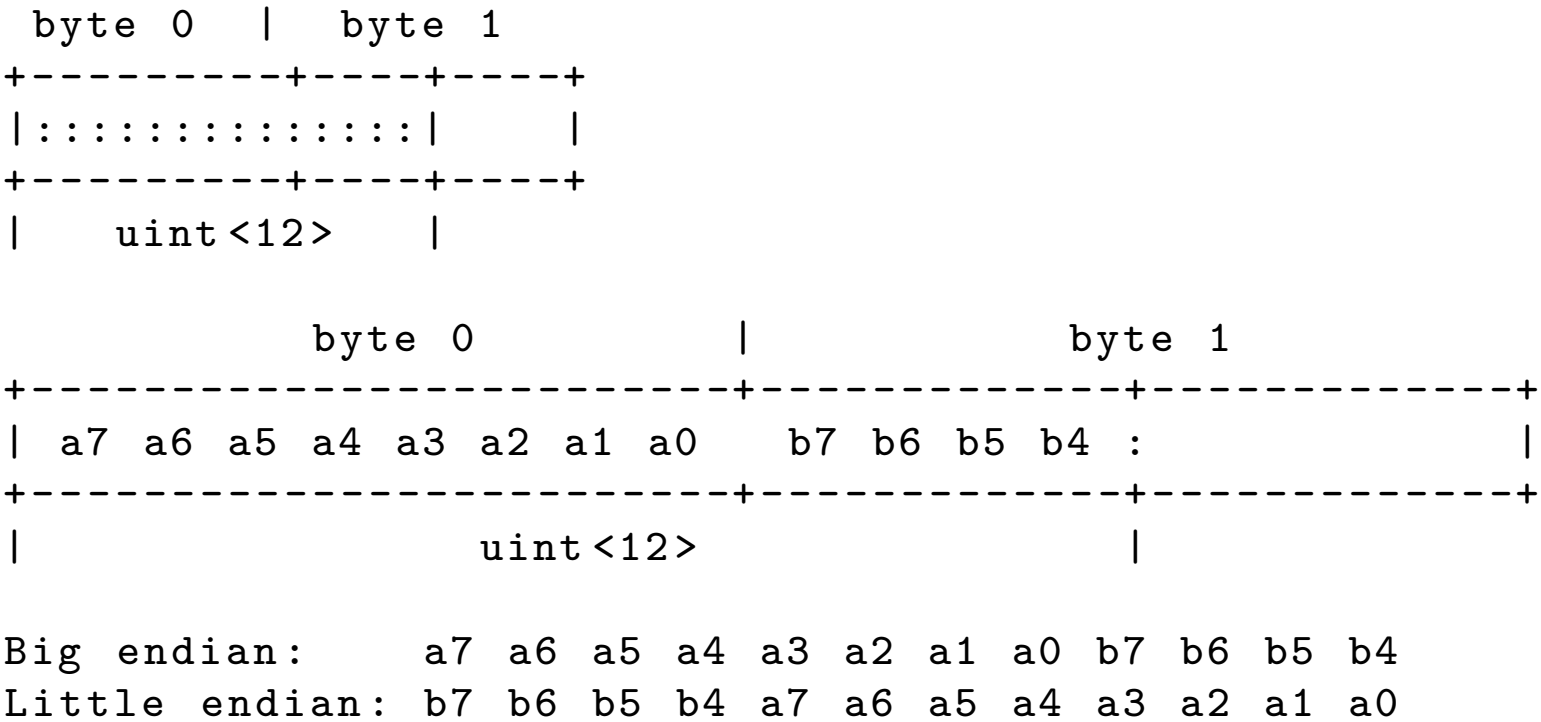
The IOS Abstraction



WYPIWYG (What You Poke is What You Get)

Weird Integers

- Incomplete Bytes



Weird Integers

- Quantum Bytenics

```
     byte
+-----+-----+
|:::::|     |
+-----+-----+
uint<5>

                byte
+-----------------+-----------+
| b7  b6  b5  b4  b3 |         |
+-----------------+-----------+
|     uint<5>       |

Value: b7  b6  b5  b4  b3
```

Unaligned Integers

```
poke values      |          uint<16> @ 2#b          |
-----------      |                                  |
Virtual bytes    | virt. byte1    |  virt. byte2    |
-----------      |                |                 |
IO space     |0|1|1|1|1|1|1|1|0|1|0|0|0|1|0|1|0|1|0|0|1|1|0|0|
-----------  |                 |                 |                 |
IO device    |      0x7f       |       0x45      |       0x4c      |
```

```
(poke) .set obase 2
(poke) .set endian big
(poke) uint<16> @ 2#b
0b1111110100010101UH
(poke) .set endian little
(poke) uint<16> @ 2#b
0b0001010111111101UH
```

Integral Structs

- Many data structures can be translated 1-1 from C to Poke
- C

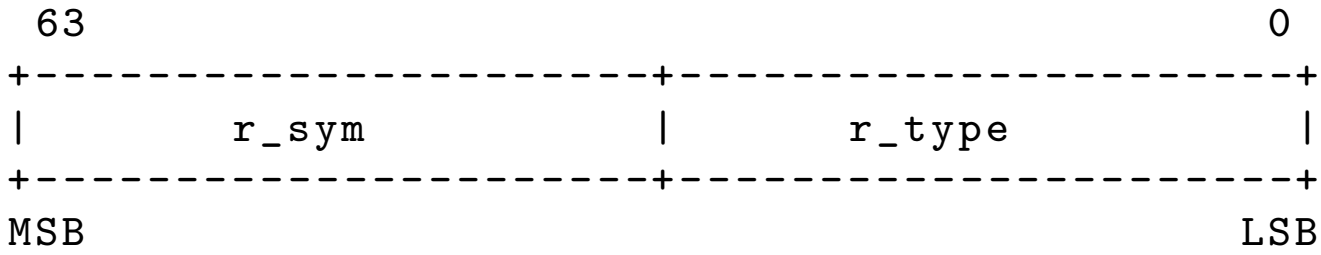
```
typedef struct
{
  Elf64_Addr   r_offset;
  Elf64_Xword  r_info;
  Elf64_Sxword r_addend;
} Elf64_Rela;
```

- Poke

```
deftype Elf64_Rela =
struct
{
  Elf64_Addr r_offset;
  Elf64_Xword r_info;
  Elf64_Sxword r_addend;
};
```

Integral Structs

- But sometimes integral values are to be interpreted as composite data:



- In C

```
#define ELF64_R_SYM(i)          ((i) >> 32)
#define ELF64_R_TYPE(i)         ((i) & 0xffffffff)
#define ELF64_R_INFO(sym,type) ((((Elf64_Xword) (sym)) << 32)
```

- We could mimic that in Poke:

```
defun Elf64_R_Sym = (Elf64_Xword i) uint<32>:
   { return i .>> 32; }
defun Elf64_R_Type = (Elf64_Xword i) uint<32>:
   { return i & 0xffff_ffff; }
defun Elf64_R_Info = (uint<32> sym, uint<32> type) Elf64_Xword
   { return sym as Elf64_Xword <<. 32 + type; }
```

Integral Structs

- But using an **integral struct** is way much better:

```
deftype Elf64_RelInfo =
  struct uint<64>
  {
    uint<32> r_sym;
    uint<32> r_type;
  };
```

- Integral structs can also be used as integers:
  - Using explicit casts.

    ```
    Elf64_RelInfo {} as uint<64>
    ```

  - Converted automatically wherever an integer is expected.

    ```
    (poke) +Elf64_RelInfo {}
    0x0UL
    (poke) Elf64_RelInfo {} + 2
    0x2UL
    ```

Integral Structs

- An integral struct is **not** always the right abstraction to use when we see a C bit field!
- C:

```
struct regs
{
  __u8 dst_reg:4;
  __u8 src_reg:4;
};
```

- Poke:

```
deftype Regs =
  struct
  {
    defvar little_p = (get_endian == ENDIAN_LITTLE);

    uint<8> src @ !little_p * 4#b;
    uint<8> dst @ little_p * 4#b;
  };
```

Loading Pickles

- Not a proper modules system yet.
- **load_path**
- Dot-command

```
(poke) .load /path/to/file.pk
```

- Language construct

```
load elf;
load "bit-utils.pk";
```

Poke Scripts

- Shebang

```
#!/usr/local/bin/poke -L
!#
```

- Writing binary utilities.
- Command line arguments, **argv**:

```
#!/usr/bin/poke -L
!#

for (arg in argv)
  print "Argument: " + arg + "\n";
```

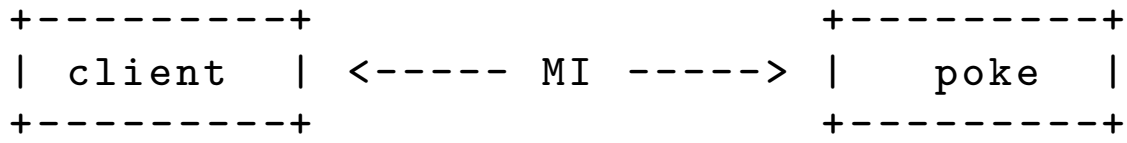
- Example: elfextractor.

libpoke

- Shared library providing compiler, PVM and IO space.
- Compilation services (file, buffer, expression).
- Disassembler.
- Management of IO spaces.
- Auto-completion.
- Loading of modules.
- Handling of PK values.

The Machine Interface (MI)

- Interaction with other pgorams (GUIs, testers, etc)



- Based on JSON, loosely inspired in the Debug Adapter Protocol https://microsoft.github.io/debug-adapter-protocol/specification.
- Being developed as part of GSOC 2020 by Konstantinos Chasialis.

The Machine Interface (MI)

- **Requests**
- **Responses**
- **Events**
- Example of dialogue:

```
MI: recv: {"seq":0,"type":2,"data":{"type":0,"args":{"mi_versi
MI: sent: {"seq":0,"type":0,"data":{"type":0}}
MI: recv: {"seq":2,"type":1,"data":{"type":0,"success_p":true,
```

- GUI prototype

New commands

- **copy** allows to copy regions of data within an IO space, or between different IO spaces.
- **save** writes a region from an IO space into a file.
- **extract** creates a temporary IO space with the contents of a mapped value.

First PokeConf at Mont Soleil, 2020

Current Status and Roadmap

Project Status - As of today

- # of commits: 3598
- Contributors (in order of appearance):

  Jose E. Marchesi, Egeyar Bagcioglu, John Darrington, Luca Saiu, Darshit Shah, Dan Cermak, Carlo Caione, Eric Blake, Tim Ruehsen, Aurelien Aptel, Bruno Haible, Konstantinos Chasialis

- Testsuite: 4247 tests in 10 testsuites

Future work

... after first release.

- Pattern matching
- Gradual typing.
- Support for sets (enums, bitmasks).
- Organize pickles better: module system, namespaces.
- Arguments to struct types
- Other language improvements.

Project Resources

- Homepage: **http://www.jemarch.net/poke.html**
- Savannah: **http://savannah.gnu.org/p/poke**
- Mailing list: **poke-devel@gnu.org**
- Bugs database: **http://sourceware.org/bugzilla**
- IRC channel: **#poke** in **irc.freenode.net**
- Applied pokology: **http://www.jemarch.net/pokology**

Hack with us!

See file **HACKING** in the source tree.