PICO RUBY

Episode III

REVENGE OF THE STDIN

RubyWorld Conference 2022

hasumikin>&Monstarlab

Nov. 10, 2022

Episode II

ATTACK OF THE RAKE

find on the internet

This slot has Q&A time

PicoRuby@SUZURI

PicoRuby: An alternative mruby

```
puts "Hello World"
```

- mruby = mruby VM + mruby compiler
  => **136 KB**
- MicroRuby = mruby VM + PicoRuby compiler
  => **88 KB**
- PicoRuby = mruby/c VM + PicoRuby compiler
  => **11 KB**

All measurement on 64-bit Ubuntu

PicoRuby: An alternative mruby

github.com/picoruby/picoruby

PRK Firmware: Keyboard firmware

e.g.) picoruby/prk_crkbd

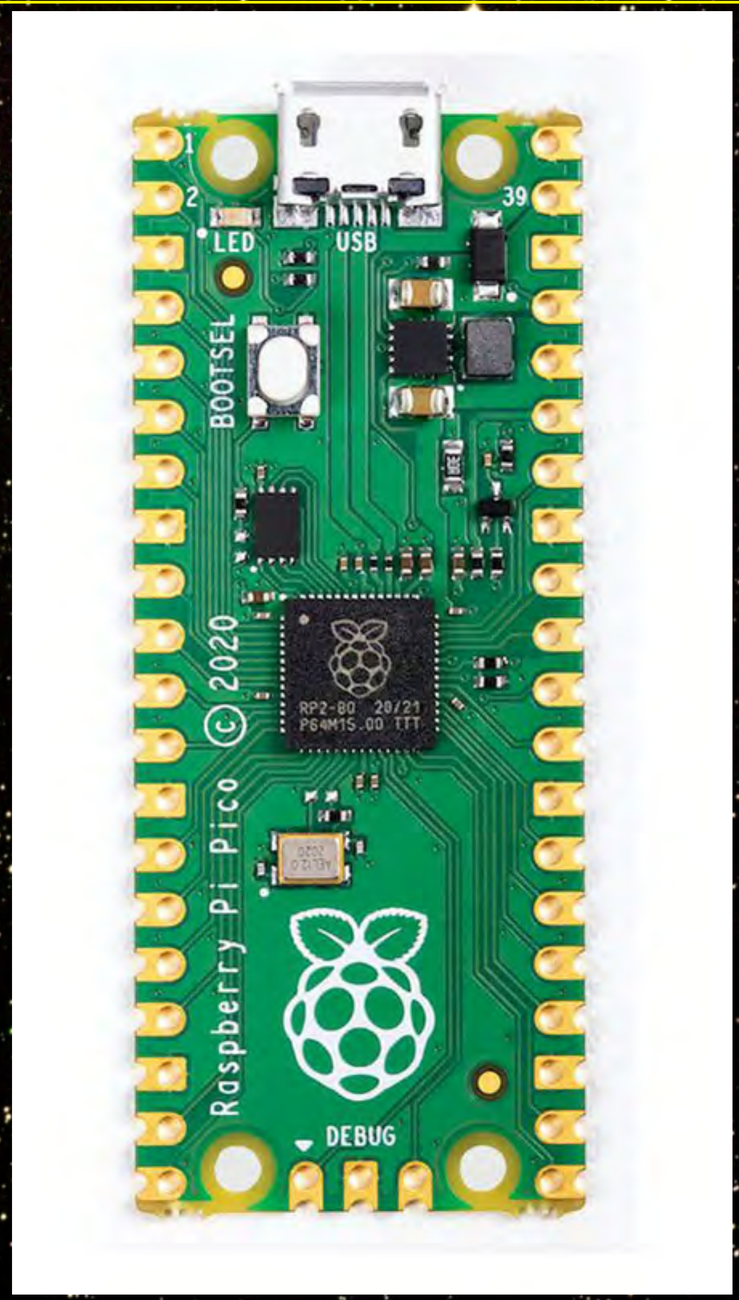
Displayed in Large exhibition hall (1F)

PRK Firmware: Keyboard firmware

github.com/picoruby/prk_firmware

Raspberry Pi Pico

- RP2040
  - Arm Cortex-M0+ (dual)
  - 264 KB RAM
- 2 MB ROM
  - (RP2040 supports upto 16 MB)
- $4~ (￥600~)
- "Pi Pico W" supports WiFi
  - Got 技適 (new!)



DEMO

Serial port terminal emulator

- Traditional CUI/TUI tools have fallen to the dark side (apparently CR/LF issue)
  - cu🤦
  - screen😖
  - minicom🤦
- Linux -> GTKTerm😍
- Windows -> Tera Term? (not sure)
- macOS -> PuTTY? 🤷

Summary: It's still a PoC, though...

- An integrated Micon programming environment
  - Ruby compiler inside
  - Shell ready
  - Text editor equipped
  - Your own commands available
- Accelerates developing an IoT system with Ruby

IMPLEMENTATION

Command: builtin and executable

- Builtin commands are implemented in shell
  - cd, echo, export, pwd, type, etc.
- Executable commands are located in storage
  - cat, ls, rm, vim, etc.

Command: builtin

```ruby
# picoruby/mrbgems/picoruby-shell/mrblib/command.rb
def exec(params)
  args = params[1, params.length - 1]
  case params[0]
  when "echo"
    _echo(*args)
  when "type"
    _type(*args)
  when "pwd"
    puts Dir.pwd
  when "cd"
    puts if Dir.chdir(args[0] || ENV['HOME']) != 0
  when "free"
    Object.memory_statistics.each do |k, v|
      print "#{k.to_s.ljust(5)}: #{v.to_s.rjust(8)}", @feed
    end
```

Command: executable

```
picoruby/mrbgems/picoruby-shell/
├── mrbgem.rake
├── mrblib
│   ├── command.rb
│   └── shell.rb
├── shell_executables   👈
│   ├── cat.rb
│   ├── hello.rb
│   ├── ls.rb
│   ├── rm.rb
│   └── vim.rb
└── src
    └── shell.c

# Search path
ENV["PATH"] => ["/bin"]
```

Command: executable

- You can write your own executable in PicoRuby

```
$ vim hello_ruby.rb
  ```
  puts "Hello RubyWorld"
  ```
$ install hello_ruby.rb /bin/hello_ruby
$ hello_ruby
Hello RubyWorld
```

Filesystem

- Computer has filesystem(s)
  - MS-DOS: Microsoft Disk Operating System
  - NTFS, ext{2,3,4}, etc.
- Micon generally doesn't have a filesystem

FAT filesystem

- File Allocation Table (FAT)
- Developed by Microsoft along with MS-DOS
- 8-bit FAT, FAT12, FAT16, FAT32, exFAT
- Under construction for Flash ROM 😅
  - Today's demo is working on RAM disk

VFS (Virtual File System)

- VFS-like system written in PicoRuby
- Different devices and even different filesystems can mount under the root

```
VFS.mount(FAT.new(:flash),  "/")
VFS.mount(FAT.new(:ram),    "/ram")
VFS.mount(FAT.new(:sdcard), "/sdcard")

 /
  ├── bin/
  ├── home/
  ├── ram/
  │    └── tmp/
  ├── sdcard/  # SPI not impemented yet
  │    └── data.sqlite
```

mruby's build system

- mrbgems is a library management system for mruby
  - You can freely host your own gem
  - Matz says it's an advanced one (than MicroPython?)
  - Bundled mgems are automatically loaded

mruby's build system

```
mruby
 ├── Rakefile
 ├── bin/
 ├── build/
 ├── build_config/
 ├── include/
 ├── lib/
 │    └── mruby/    # MRuby module
 ├── mrbgems/
 ├── mrblib/
 ├── src/
 ├── tasks/
 │    ├──
 │    ├── mrbgems.rake
 │    ├──
```

PicoRuby's build system

```
picoruby
 ├── Rakefile                  👈 modified
 ├── bin/
 ├── build/
 ├── build_config/
 ├── include/
 ├── lib/
 │    ├── mruby/              # `module MRuby`
 │    └── picoruby/
 │         └── build.rb 👈 MRuby overridden
 ├── mrbgems/
 ├── mrblib/
 ├── src/
 ├── tasks/
 │    ├──
 │    ├── mrbgems.rake
 │    ├──
 │    ├── picogems.rake 👈 MRuby.each_target overridden
 │    ├──
```

PicoRuby's build system

```
picoruby
 ├── build/repos/host/
 │                ├── mruby-bin-picorbc/    # picorbc
 │                └── mruby-pico-compiler/
 ├──
 ├── mrbgems/
 │    ├── default.gembox
 │    ├── picoruby-bin-picoirb/              # picoirb
 │    ├── picoruby-bin-picoruby/             # picoruby
 │    ├── picoruby-bin-prsh/                 # prsh [pəːrʃ]
 │    ├── picoruby-filesystem-fat/           # FAT filesystem
 │    ├── picoruby-io/                       # io gem
 │    ├── picoruby-mrubyc/                   # mruby/c VM
 │    ├── picoruby-sandbox/
 │    ├── picoruby-shell/
 │    ├── picoruby-terminal/
 │    ├── picoruby-vfs/
 │    └── picoruby-vim/
```

PicoRuby's build system

```ruby
# picoruby/mrbgems/default.gembox
MRuby::GemBox.new do |conf|
  conf.gem github: 'picoruby/mruby-pico-compiler'
  conf.gem github: 'picoruby/mruby-bin-picorbc'
  conf.gem core: 'picoruby-bin-picoruby'
  conf.gem core: 'picoruby-bin-picoirb'
  conf.gem core: 'picoruby-mrubyc'
end
```

`mruby-pico-compiler` and `mruby-bin-picorbc`
can be bundled in mruby (MicroRuby)

A typical picogem

```
picoruby/
└── mrbgems/picoruby-io/
    ├── mrbgem.rake
    ├── mrblib
    │   └── io.rb
    └── src/
        ├── hal/
        │   ├── hal.h
        │   └── posix
        │       └── hal.c
        └── io.c
```

`require`

- mruby doesn't have `require` out of the box
  - Neither does mruby/c
- PicoRuby has introduced `require`
  - Bundled Picogem code already consumed ROM, but
  - It isn't loaded onto RAM automatically
  - Consumes RAM only after `require`

Making a PicoRuby app elegantly

- CRuby code compatible with PicoRuby
- PicoRuby for PC (MRBC_USE_HAL_POSIX)
  - Write minimum HAL for POSIX
  - Extend PicoRuby's builtin class if necessary
- Keep the app code independent from Picoruby
  - Do not copy and paste files any longer

Keep the app code independent

```
example-IoT-app
 ├── CMakeLists.txt       # Just link src/*.c and libmruby.a
 ├── Rakefile
 ├── build
 │    ├── ...
 ├── include
 │    ├── ...
 ├── lib
 │    └── picoruby        # libmruby.a created by `rake`
 ├── mrblib
 │    ├── main_task.rb
 │    └── temperature_sensor_task.rb
 ├── pico_sdk_import.cmake
 └── src
      ├── hal.c           # Micon dependent hal
      ├── temperature_sensor.c
      ├── main.c
      ├── ...
```

Compatible CRuby with PicoRuby

```ruby
# picoruby-vim/mrblib/vim.rb
case RUBY_ENGINE
when "ruby", "jruby"
  require_relative "../../picoruby-terminal/mrblib/terminal"
when "mruby/c"
  require "terminal"
else
  raise "Unknown RUBY_ENGINE: #{RUBY_ENGINE}"
end

class Vim
  (...)
end
```

Compatible CRuby with PicoRuby

Run with CRuby on your PC, then

```
$ ruby -r./mrbgems/picoruby-vim/mrblib/vim.rb \
    -e 'Vim.new("myfile.txt").start'
```

You can easily implement and debug a "Pure" PicoRuby app with CRuby

DEMO

BACK TO IMPLEMENTATION

Shell and text editor

- Without Curses and Readline
- Pure Ruby just like Reline (irb of CRuby3.0+)

Shell and text editor

- Without Curses and Readline
- Pure Ruby just like Reline (irb of CRuby3.0+)

STDIN confuses you 😈

Standard INPUT

- Cooked mode **(by default on Unix-like OS)**
  - Blocked until submit (recall Kernel#gets)
    - "a" "b" "c" "[BSPACE]" "d" "[ENTER]" => "abd"
- Raw mode **(obviously by default on bare metals)**
  - Immediately receives any input
    - "a" "b" "c" "[BSPACE]" "d" => "abc[BSPACE]d"

Really confusing in making compatible code

Standard INPUT

```c
static struct termios save_settings;
static int save_flags;

/* Turn into raw mode on Linux (Windows has another solution) */
struct termios settings;
tcgetattr(fileno(stdin), &save_settings);
settings = save_settings;
settings.c_iflag = ~(BRKINT | ISTRIP | IXON);
settings.c_lflag = ~(ICANON | IEXTEN | ECHO | ECHOE | ECHOK | ECHONL);
settings.c_cc[VMIN]  = 1;
settings.c_cc[VTIME] = 0;
tcsetattr(fileno(stdin), TCSANOW, &settings);
save_flags = fcntl(fileno(stdin), F_GETFL, 0);
fcntl(fileno(stdin), F_SETFL, save_flags | O_NONBLOCK);

/* Turn back to cooked mode */
fcntl(fileno(stdin), F_SETFL, save_flags);
tcsetattr(fileno(stdin), TCSANOW, &save_settings);
```

Standard INPUT

```ruby
# picoruby/mrbgems/picoruby-terminal/mrblib/terminal.rb
case RUBY_ENGINE
when "ruby", "jruby"
  require "io/console"

  def IO.get_nonblock(max)
    STDIN.noecho{ |input| input.read_nonblock(max) }
  rescue IO::EAGAINWaitReadable => e
    nil
  end

when "mruby/c"
  require "io"

  def IO.get_nonblock(max)
    str = read_nonblock(max) # calls C implementation
    str&.length == 0 ? nil : str
  end

end
```

C object in RVALUE: mruby

```c
#define MRB_VTYPE_FOREACH(f) \
  /* mrb_vtype */  /* c type */     /* ruby class */ \
  f(MRB_TT_FALSE,     void,              "false") \
  ...
  f(MRB_TT_OBJECT,    struct RObject, "Object") \
  ...
  f(MRB_TT_ĐATA,      struct RĐata,   "Đata") \ // 👈
  ...

struct RĐata {
  MRB_OBJECT_HEAĐER;
  struct iv_tbl *iv;
  const mrb_data_type *type;
  void *data;  // 👈 You can pin any object here
};
```

C object in RVALUE: mruby/c

```c
typedef struct RInstance {
  MRBC_OBJECT_HEAĐER;
  struct RClass *cls;
  struct RKeyValueHandle ivar;
  uint8_t data[]; // 👈 The last member can extend
} mrbc_instance;
// File.new
mrbc_value obj = mrbc_instance_new(vm, v->cls, sizeof(FIL));
//                                             ^^^^^^^^^^^^
//                                                extend RVALUE
FIL *file_stream_ptr = (FIL *)obj.instance->data;
SET_RETURN(obj); // File.new => #<File:200098a0>
```

No need to `free(file_stream_ptr);`

Finally, the shell works 🎉

By the way,

How should I name this product🤔

Ruby on Raspberry Pi Pico

Ruby on Raspberry Pi Pico

R2-P2

R2-P2

“It is said that (account) names
should not contain
hyphens or underscores.”

[cited from `@kakutani']

R2-P2

R2P2

Be one the first stargazers of

github.com/picoruby/R2P2

(turned public repo today!)

Wrap-up

- R2P2 is an all-in-one programming Micon

Wrap-up

- R2P2 is an all-in-one programming Micon
- Runs on Raspi Pico which has 264 KB RAM and 2 MB ROM

Wrap-up

- R2P2 is an all-in-one programming Micon
- Runs on Raspi Pico which has 264 KB RAM and 2 MB ROM
- Picogem ecosystem easily builds your app

Wrap-up

- R2P2 is an all-in-one programming Micon
- Runs on Raspi Pico which has 264 KB RAM and 2 MB ROM
- Picogem ecosystem easily builds your app
- Filesystem ly makes you happy
- Filesystem will make you happy

Wrap-up

- R2P2 is an all-in-one programming Micon
- Runs on Raspi Pico which has 264 KB RAM and 2 MB ROM
- Picogem ecosystem easily builds your app
- Filesystem will make you happy
- STDIN is evil

Future work

- PicoRuby and Shell
  - More functionality of vim and the name
  - UTF-8 support
  - Multi-task controll on the fly
  - Redirect and pipeline
- Peripherals
  - FAT filesystem with SD card (SPI)
  - Real-time clock (RTC)
  - WiFi and TCP/IP on Raspberry Pi Pico W

Q AND A

GITHUB.COM/PICORUBY/R2P2

MAY THE

SOURCE

BE WITH YOU