Advanced IO capabilities

Or how some parts of r_io api work internally and how to write r_io plugins to simulate memory mapped hardware

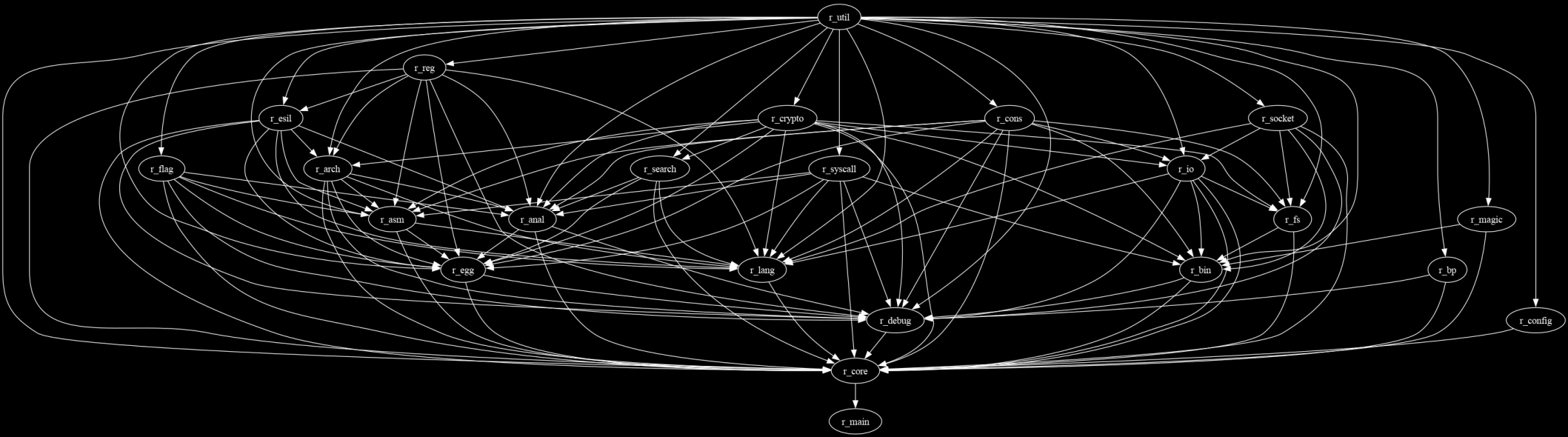
whoami

condret
- author of ~70% of the r_io api

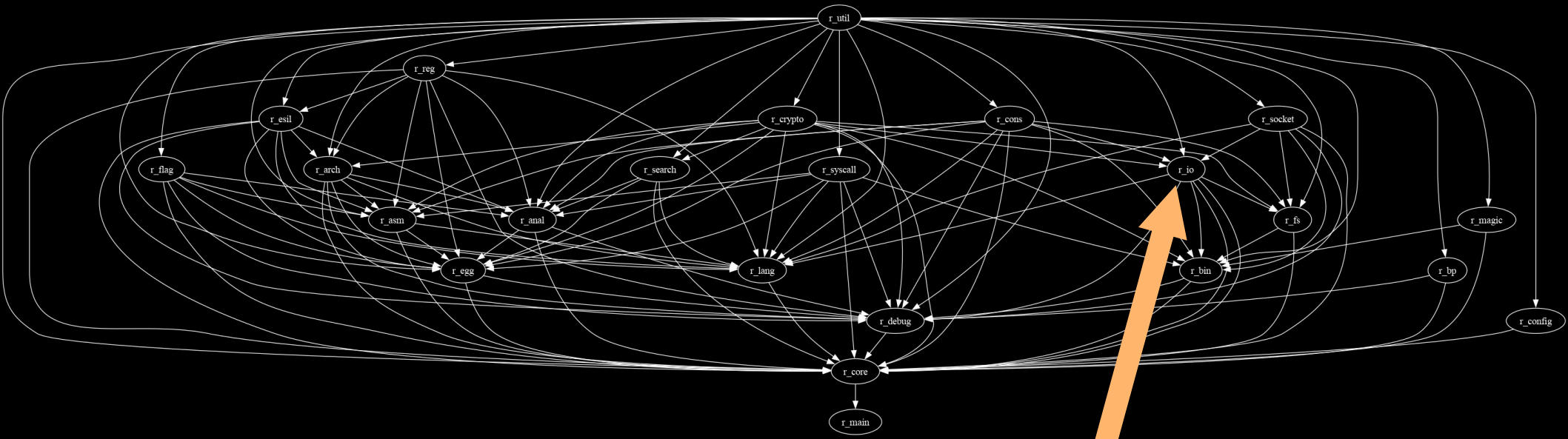
- @condret@fedi.absturztau.be

- @condret@shitposter.world

libr


r_util
r_reg
r_esil
r_crypto
r_cons
r_socket
r_flag
r_arch
r_search
r_syscall
r_io
r_asm
r_anal
r_fs
r_magic
r_egg
r_lang
r_bin
r_bp
r_debug
r_config
r_core
r_main

libr



we are going here

The r_io stack

simple scenario

```
e io.cache = false
e io.va = false
e io.pcache = false
px
```

simple scenario

```
+==========================+
| RIO                      | io/io.c              r_io_read_at
+==========================+
| P                        | io/io.c              r_io_pread_at
+==========================+
| fd                       | io/io_fd.c           r_io_fd_read_at
+==========================+
| Desc                     | io/io_desc.c         r_io_desc_read_at
+==========================+
| Plugins                  | io/io_plugin.c       r_io_plugin_read_at
+==========================+
| Plugin Implementation    |
+==========================+
```

fd, RIODesc and RIOPlugin

- fd api "wraps" RIODesc
  - intended for integration of existing things into r2

- RIODesc wraps specific plugin
  - provides void * for the plugin context (e.g. offset)

RIOPlugin

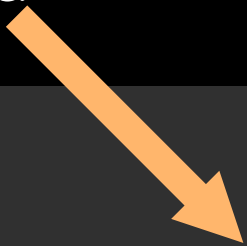
```
typedef struct r_io_plugin_t {
        const RPluginMeta meta;
        void *widget;
        const char *uris;
        int (*listener)(RIODesc *io);
        bool (*init)(void);
        bool isdbg;
        char *(*system)(RIO *io, RIODesc *fd, const char *);
        RIODesc* (*open)(RIO *io, const char *, int perm, int mode);
        RList* (*open_many)(RIO *io, const char *, int perm, int mode);
        int (*read)(RIO *io, RIODesc *fd, ut8 *buf, int count);
        ut64 (*seek)(RIO *io, RIODesc *fd, ut64 offset, int whence);
        int (*write)(RIO *io, RIODesc *fd, const ut8 *buf, int count);
        bool (*close)(RIODesc *desc);
        bool (*is_blockdevice)(RIODesc *desc);
        bool (*is_chardevice)(RIODesc *desc);
        int (*getpid)(RIODesc *desc);
        int (*gettid)(RIODesc *desc);
        bool (*getbase)(RIODesc *desc, ut64 *base);
        bool (*resize)(RIO *io, RIODesc *fd, ut64 size);
        bool (*extend)(RIO *io, RIODesc *fd, ut64 size);
        bool (*accept)(RIO *io, RIODesc *desc, int fd);
        int (*create)(RIO *io, const char *file, int mode, int type);
        bool (*check)(RIO *io, const char *, bool many);
} RIOPlugin;
```

RIOPlugin (important parts)

```
typedef struct r_io_plugin_t {
        const RPluginMeta meta;
        const char *uris;
        RIODesc* (*open)(RIO *io, const char *, int perm, int mode);
        int (*read)(RIO *io, RIODesc *fd, ut8 *buf, int count);
        ut64 (*seek)(RIO *io, RIODesc *fd, ut64 offset, int whence);
        int (*write)(RIO *io, RIODesc *fd, const ut8 *buf, int count);
        bool (*close)(RIODesc *desc);
        bool (*check)(RIO *io, const char *, bool many);
} RIOPlugin;
```

RIOPlugin (important parts)

ignored



```
typedef struct r_io_plugin_t {
        const RPluginMeta meta;
        const char *uris;
        RIODesc* (*open)(RIO *io, const char *, int perm, int mode);
        int (*read)(RIO *io, RIODesc *fd, ut8 *buf, int count);
        ut64 (*seek)(RIO *io, RIODesc *fd, ut64 offset, int whence);
        int (*write)(RIO *io, RIODesc *fd, const ut8 *buf, int count);
        bool (*close)(RIODesc *desc);
        bool (*check)(RIO *io, const char *, bool many);
} RIOPlugin;
```

SEEK

Use R_IO_SEEK... instead of SEEK...

R_IO_SEEK_END should always seek to eof
- resizing works via .resize or .extend callback

Gameboy MCB2 RAM

A000-A1FF - 512x4bits RAM, built-in into the MBC2 chip (Read/Write)

The MBC2 doesn't support external RAM, instead it includes 512x4 bits of built-in RAM (in the MBC2 chip itself). It still requires an external battery to save data during power-off though. As the data consists of 4bit values, only the lower 4 bits of the "bytes" in this memory area are used.

Build a minimal version of this without saving capabilities

Source: https://gbdev.gg8.se

Skelleton, makefile and testing

https://github.com/condret/r2con24_io/tree/master/0

open r2 --
load your plugin with 'l <path_to_so>'
open desc with 'on mbc2ram://'

RIOMap

- provides mapping from pa to va

- references RIODesc via fd

- has it's own rwx perms
  - desc->rwx >= map->rwx (intended but not enforced)
  - checkout omf command

RIOMap

```
typedef struct r_io_map_t {
        int fd;
        int perm;
        ut32 id;
        ut64 ts;
        RInterval itv;          // vaddr range: used as closed interval!
        ut64 delta;             // paddr = vaddr - itv.addr + delta
        RRBTree *overlay;
        char *name;
        ut32 tie_flags;
} RIOMap;
```

RIOMap

you can create maps with om command

or with #!c

Gameboy MBC2

MBC2 (max 256KByte ROM and 512x4 bits RAM)

0000-3FFF - ROM Bank 00 (Read Only)

Same as for MBC1.

4000-7FFF - ROM Bank 01-0F (Read Only)

Same as for MBC1, but only a total of 16 ROM banks is supported.

A000-A1FF - 512x4bits RAM, built-in into the MBC2 chip (Read/Write)

The MBC2 doesn't support external RAM, instead it includes 512x4 bits of built-in RAM (in the MBC2 chip itself). It still requires an external battery to save data during power-off though. As the data consists of 4bit values, only the lower 4 bits of the "bytes" in this memory area are used.

0000-1FFF - RAM Enable (Write Only)

The least significant bit of the upper address byte must be zero to enable/disable cart RAM. For example the following addresses can be used to enable/disable cart RAM: 0000-00FF, 0200-02FF, 0400-04FF, ..., 1E00-1EFF. The suggested address range to use for MBC2 ram enable/disable is 0000-00FF.

Gameboy MBC2

0000-1FFF - RAM Enable (Write Only)

The least significant bit of the upper address byte must be zero to enable/disable cart RAM. For example the following addresses can be used to enable/disable cart RAM: 0000-00FF, 0200-02FF, 0400-04FF, ..., 1E00-1EFF. The suggested address range to use for MBC2 ram enable/disable is 0000-00FF.

ram only enables if 0x0A is written
other values will disable the ram

Gameboy MBC2 strategy

Gameboy MBC2 strategy

create another plugin mbc2:// (in the same file)

- with it's own instance of RIO
- pretend to have size of 0xC000
- use existing plugin with r_io_desc_open_plugin
- map ram to 0xA000
- forward reads to r_io_read_at
- check writes for ram enable/disable
  - modify map or desc permissions

eof