OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

OFVWG: ABI discussion

Netlink based serialization

Liran Liss, Matan Barak

Notes

- The code in this presentation is untested and is just given in order to present a concept.

Buffer format

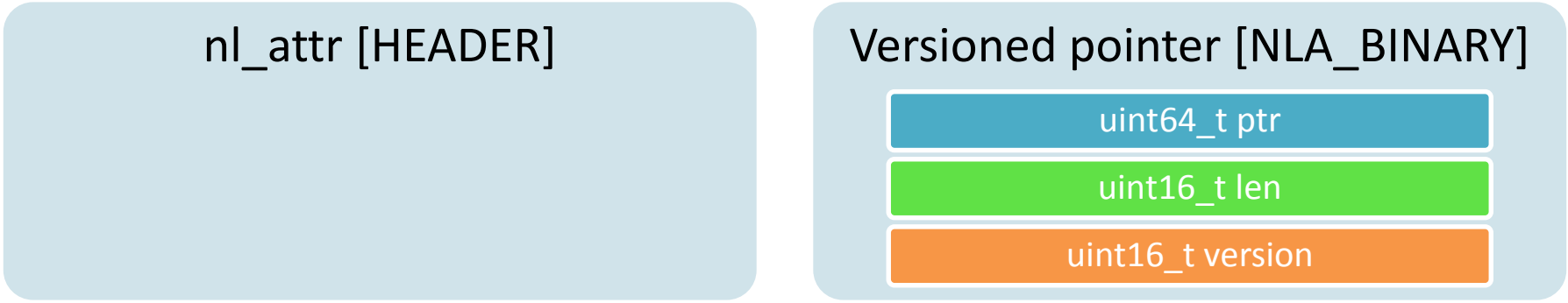
```
#define IB_USER_VERBS_IOCTL_COMMAND \
        _IOWR(IB_IOCTL_MAGIC, 1, struct ib_uverbs_ioctl_hdr)      /* TODO: need to check if 1 is free */

struct ib_uverbs_ioctl_hdr {
        __u8  ver;              /* header version */
        __u8  flags;            /* flags – for example: vendor specific*/
        __u16 object_type;      /* QP, CQ, device, port, .... */
        __u32 length;           /* packet length including header */
        __u16 reserved;         /* future extensibility */
        __u16 action;           /* 0 – 7: common actions, 8-65535: object specific */
        __u32 user_handler;     /* object type, not valid in create */
};

enum ib_uverbs_common_actions {
        IB_UVERBS_COMMON_OBJECT_CREATE,
        IB_UVERBS_COMMON_OBJECT_DESTROY,
        IB_UVERBS_COMMON_OBJECT_QUERY,
        IB_UVERBS_COMMON_OBJECT_MODIFY,
        IB_UVERBS_COMMON_OBJECT_MAX = 8
};
```

Versioned ptr Versioned ptr

Buffer format – Discussion part

- Why only one IOCTL?
  - strace could parse the buffer itself and stringify the parameters from there, so what's the value of having more than once IOCTL?
- Dividing the core's part and vendor's part
  - Maintaining ABI between libibverbs and vendor's user-space library
  - Allow to change the core part only without touching the vendor's part
  - Vendor's that will be willing to switch to netlink, could indicate this in an indirect core header
  - TLVs are written in order. Verbs usually write the vendor part before the core part.
- Response
  - Response attributes are written directly to the core output buffer, wrapped in one NLA_NESTED
  - If there is vendor response, it's written to the vendor response pointer. A "version pointer netlink attribute" (next slide) in the core part points to this BLOB.

Versioned pointer netlink attribute



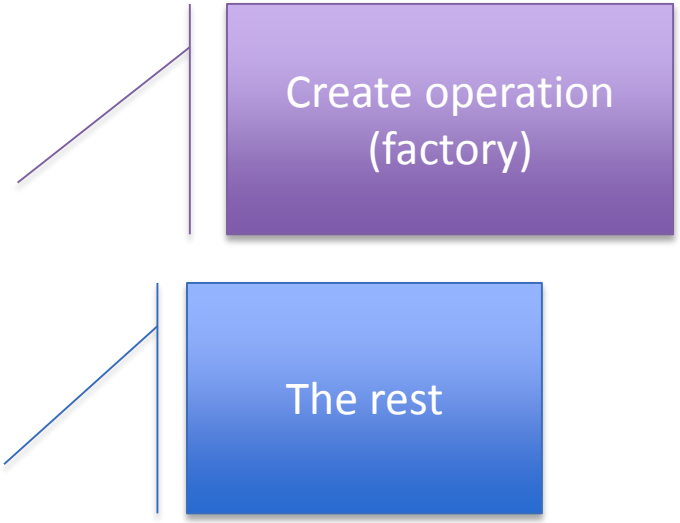
- Part of the core part response / core command part
- Points to the vendor's response part
- The buffer is wrapped in ib_udata (as of today)
- The vendor could set the version field when it switches the format

Single IOCTL command

```
static long ib_uverbs_ioctl (struct file *filp, unsigned int cmd, unsigned long arg)
{
        struct ib_uverbs_ioctl_hdr __user *user_hdr = (struct ib_uverbs_ioctl_hdr __user *)arg;
        struct ib_uverbs_ioctl_hdr hdr;

        if (cmd != IB_USER_VERBS_IOCTL_COMMAND)
                return -ENOIOCTLCMD;

        if (copy_from_user(&hdr.ver, user_hdr, sizeof(hdr.ver)))
                return -EINVAL;

        if (hdr.ver)
                return -EOPNOTSUPP;

        if (copy_from_user(&hdr, user_hdr, sizeof(hdr)))
                return -EINVAL;

        /* currently there are no flags supported */
        if (hdr.flags)
                return -EOPNOTSUPP;

        if (hdr.reserved)
                return -EOPNOTSUPP;

        if (hdr.length > IB_UVERBS_MAX_CMD_SZ || hdr.length <= sizeof(hdr))
                return -EINVAL;

        return ib_uverbs_cmd_verbs(filp, &hdr, (__user void *)arg + sizeof(hdr));
}
```

Objects and actions - declarations

```
struct mandatory_fields {
        unsigned long *bitmap;
        unsigned int   max;
};

struct validate_op {
        const struct nla_policy         *policy;                /* validating sizes of parts of command */
        struct mandatory_fields         mandatory_fields;       /* validate that all mandatory parts exist */
        struct response_validator_op    response;               /* validate that the response is of valid size */
};

struct object_action {
        struct {
                struct validate_op validator;
                long (*fn)(struct file *filp, struct ib_uverbs_ioctl_hdr *hdr,
                           struct nlattr **tb);
                unsigned int max_attrs;
        } create;
        /* modify, query should come here too */
        /* other intersting per object type functions */
        struct {
                struct validate_op validator;
                long (*fn)(struct file *filp, struct ib_uverbs_ioctl_hdr *hdr,
                           struct nlattr **tb, uint32_t id);
                unsigned int max_attrs;
        } ops[];
};
```



When we reach the function pointer, all command parts are of valid size, all mandatory parts exist. If a response exists its large enough to contain the minimum mandatory response. In actions other than create, idr is fetched and validated.

Objects and actions - Example

```
/* Standard response descriptor */
struct ib_uverbs_ioctl_resp {
        __u64 response;
        __u16 core_out;
        __u16 reserved;
};

enum ibnl_create_device {
        IBNL_CREATE_DEVICE_RESPONSE_DESC,
        IBNL_CREATE_DEVICE_CORE,
        IBNL_CREATE_DEVICE_PROVIDER,
        IBNL_CREATE_DEVICE_MAX
};

static const struct nla_policy ibnl_create_device_policy[] = {
      [IBNL_CREATE_DEVICE_RESPONSE_DESC] = {.type = NLA_BINARY, .len = sizeof(struct ib_uverbs_ioctl_resp)},
      [IBNL_CREATE_DEVICE_PROVIDER]      = {.type = NLA_BINARY, .len = sizeof(struct ib_uverbs_ioctl_ver_ptr)},
};


static const struct object_action object_actions[] = {
        [IB_UVERBS_OBJECT_TYPE_DEVICE] = {
                .create = {
                        .fn = ib_uverbs_context_create,
                        .validator = {.policy = ibnl_create_device_policy,
                                      .mandatory_fields = IB_UVERBS_MANDATORY_FIELDS(IBNL_CREATE_DEVICE_RESPONSE_DESC),
                                      .response = {
                                              .resp_min_sz = sizeof(struct ib_uverbs_get_context_resp),
                                              .response_id = IBNL_CREATE_DEVICE_RESPONSE_DESC,
                                      }
                                },
                        .max_attrs = IBNL_CREATE_DEVICE_MAX,
                }
        }
};
```

Command processing

```
static long ib_uverbs_cmd_verbs(struct file *filp, struct ib_uverbs_ioctl_hdr *hdr, void __user *buf)
{
        enum ib_uverbs_object_type obj_type = hdr->object_type;
        void *cmd_buf = NULL;
        struct nlattr **tb = NULL;
        int err = 0;

        if (obj_type >= IB_UVERBS_OBJECT_TYPE_MAX)
                return -EOPNOTSUPP;

        cmd_buf = kmalloc(hdr->length, GFP_KERNEL);
        if (!cmd_buf)
                return -ENOMEM;

        if (copy_from_user(cmd_buf, buf, hdr->length - sizeof(*hdr))) {    /* We copy the whole command to kernel space */
                err = -EINVAL;
                goto err;
        }

         switch (hdr->action) {
         case IB_UVERBS_COMMON_OBJECT_CREATE: {
                if (!object_actions[obj_type].create.fn) {
                        err = -EOPNOTSUPP;
                        goto err;
                }

                if (hdr->action || hdr->user_handler) {
                        err = -EINVAL;
                        goto err;
                }

                tb = kcalloc(object_actions[obj_type].create.max_attrs + 1,
                             sizeof(*tb), GFP_KERNEL);
                if (!tb) {
                        err = -ENOMEM;
                        goto err;
                }

                err = nla_strict_parse(tb, object_actions[obj_type].create.max_attrs, (const struct nlattr *)cmd_buf, hdr->length - sizeof(*hdr),
                                       &object_actions[obj_type].create.validator);
                if (err)
                        goto err;

                err = object_actions[obj_type].create.fn(filp, hdr, tb);
                break;
```

Generic parsing of netlink buffer

```
static int nla_strict_parse(struct nlattr **tb, int maxtype, const struct nlattr *head, int len, const struct validate_op *validate)
{
        const struct nlattr *nla;
        int rem, err = 0;
        unsigned long *fields;
        const struct nla_policy *policy = validate->policy;

        fields =  kcalloc(maxtype / BITS_PER_LONG + !!(maxtype % BITS_PER_LONG), sizeof(unsigned long), GFP_KERNEL);
        if (!fields)
                return -ENOMEM;

        memset(tb, 0, sizeof(struct nlattr *) * (maxtype + 1));

        nla_for_each_attr(nla, head, len, rem) {
                u16 type = nla_type(nla);

                if (type < 0 || type >= maxtype || validate_nla(head, len, maxtype, policy)) {
                        err = -EOPNOTSUPP;
                        goto errout;
                }
                set_bit(type, fields);
                tb[type] = (struct nlattr *)nla;
        }

        if (unlikely(rem > 0)) {
                pr_warn("ib_uverbs: malformed command, %d bytes remainging in command\n",
                                rem);
                err = -EINVAL;
                goto errout;
        }

        bitmap_and(fields, fields, validate->mandatory_fields.bitmap,
                    validate->mandatory_fields.max);
        if (!bitmap_equal(fields, validate->mandatory_fields.bitmap,
                          validate->mandatory_fields.max)) {
                err = -EINVAL;
                goto errout;
        }

         /* TODO: validate response if exists */
errout:
        kfree(fields);
        return err;
}
```

ib_udata object

- Core command part
  - No need for ib_udata any more, as we pass an array of netlink attributes.
- Core response part
  - All response parts are written to one NLA_NESTED attribute wrapping them. No need for ib_udata too.
- Legacy vendor command part
  - As of today, ib_udata just wraps the vendor's command buffer.
  - The only change is an API that gives the vendor an ability to query the command version.
- Legacy vendor response part
  - As of today, ib_udata just wraps the vendor's command buffer.
  - The only change is an API that gives the vendor an ability to write the command version in the versioned pointer.
- Netlink style vendor command part
  - Vendor could get the input buffer and size and just treat them as NLA_NESTED netlink attribute.
- Netlink style vendor response part
  - Vendor could write straight to the output buffer (any format it wants)
  - Need to update the versioned pointer length at the end.

Libibverbs – some questions

1. Serialize commands – maybe using libmnl/libnl-tiny?
2. Avoid changing vendor's user space libraries. Target should is either ABI compliance or at least re-compile only.
3. Add support to strace to deeply decode libibverbs actions
4. Maintain current libibverbs← → vendor's user-space driver API and ABI
5. Maintain current libibverbs uAPI and uABI
6. Statically linked application? Are distributions going to support a solution for this?

Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP