RTnet

Application Programming Interface

University of Hannover
ISE – Real Time Systems Group

Outline

- BSD Sockets in RTnet
  - Introduction
  - Available Protocols
  - Differences and Extensions
- RTmac / TDMA Interface
- Low-Level Interfaces
  - RTDM Introduction
  - RTnet's RTDM Devices
  - Direct Device Access
- Configuration Interface

BSD Sockets

- Generic interface for exchanging information between processes
- Socket: source or sink of transmitted information
- Creation:

```
fd = socket(proto_family, sock_type, proto);
```

`fd` – File descriptor (integer), used in succeeding calls

`proto_family` – e.g. **`PF_INET`** for IP protocols

`sock_type` – message (**`SOCK_DGRAM`**) or stream (**`SOCK_STREAM`**) oriented, etc.

`proto` – actual protocol (e.g. **`IPPROTO_UDP`**)

➔ Further information: man socket(2), udp(7), packet(7)

BSD Sockets (2)

- Reception:

```
result = recv(fd, buf, len, flags);
result = recvfrom(fd, buf, len, flags,
                  from, fromlen);
result = recvmsg(fd, msg, flags);
```

`result`	– received bytes, negative on error
`flags`	– `MSG_DONTWAIT` (non-blocking) `MSG_PEEK` (keep message in queue)
`from/fromlen`	– source address buffer/size (`struct sockaddr`[`_in`, `_ll`, ...])
`msg`	– scatter/gather buffer (`struct iovec`), address, control data

➔ Further information: man recv(2), readv(2) (iovec)

BSD Sockets (3)

- Transmission:
  `send(), sendto(), sendmsg()`

- Fixed addresses:

```
result = bind(fd, my_addr, addrlen);
result = connect(fd, serv_addr, addrlen);
```

`my_addr` – fixed local address (e.g. IP/port) over which data may arrive or can be sent

`serv_addr` – address which is used when no other destination is specified (connection-less) or to which a connection shall be established (connection-oriented)

➔ Further information: man send(2), bind(2), connect(2)

BSD Sockets (4)

- Set socket/protocol parameters:

  ```
  result = setsockopt(fd, level, optname,
                            optvalue, optlen);
  result = ioctl(fd, request, arg);
  ```

  Parameters will be explained later.

- Get socket/protocol information:

  ```
  getsockopt(), getsockname(), getpeername(),
  ioctl()
  ```

- Socket clean-up:

  ```
  result = close(fd);
  ```

➔ Further information: man ...

Supported Protocols in RTnet

- UDP/IP:

  (`PF_INET, SOCK_DGRAM, 0`) or
  (`PF_INET, SOCK_DGRAM, IPPROTO_UDP`)

- Packet Sockets:

  (`PF_PACKET, SOCK_DGRAM, <PROTO>`)

  `<PROTO>` – link layer protocol identifier
  (i.e. Ethernet protocol ID)

Note: ICMP/IP only accessible as "ping" command via
Linux misc-device (used by rtping)

Differences and Limitations

- Real-time socket functions carry "**`_rt`**" suffix (e.g. **`send_rt`**)
- Return value also contains the error code (no errno support)
- User's **`iovec`** structures are modified by **`recvmsg()`** and **`sendmsg()`** [bug]
- Only one listener can register per IP port, no **`ETH_P_ALL`** for packet sockets allowed (RTcap uses different interface)

Differences and Limitations (2)

- Socket creation and clean-up may run both in real-time and non-real-time context, but don't mix it up!
- `close_rt()` can fail if socket is busy!
  => polling loop with delay required (see examples)
- Don't kill a task which is running some socket function, close the socket first! [RTAI-specific]

IOCTLs and Socket Options

Standard:

- Get list of network devices
  IOCTL: **`SIOCGIFCONF`**

- Get devices flags
  IOCTL: **`SIOCGIFFLAGS`**

➔ Further information: man netdevice(7)

- Set Type of Service (TOS) field in IP headers
  sockopt, level: **`SOL_IP`**, optname: **`IP_TOS`**

Parameters (2)

Extensions:

- Define transmission priority per socket
  IOCTL: **`RTNET_RTIOC_PRIORITY`**
  arg: **`(int *)prio, SOCK_MAX_PRIO < SOCK_MIN_PRIO`**
- Define timeout of blocking socket calls per socket
  IOCTL: **`RTNET_RTIOC_TIMEOUT`**
  arg: **`(__s64 *)nanosecs,`** 0 = infinite (default)
- Set callback handler (kernel mode only)
  IOCTL: **`RTNET_RTIOC_CALLBACK`**
  arg: **`(struct rtnet_callback *)handler_and_arg`**

  Note: Handler prototype has changed in 0.7.0, file descriptor can now be obtained via **`context->fd`**, see examples.

Parameters (3)

- Set blocking/non-blocking mode of socket
  IOCTL: **`RTNET_RTIOC_NONBLOCK`**
  arg: **`(int *)nonblock`**, **`≠0`** means non-blocking

  Note: there is no fcntl_rt() to switch the mode the standard way.

- Extend / shrink buffer pool of socket
  IOCTL: **`RTNET_RTIOC_EXTPOOL`** / **`RTNET_RTIOC_SHRPOOL`**
  arg: **`(int *)delta`**

  Note: To receive / transmit a message, all required buffers are taken from the pool of the destination / source socket.

  If the socket was created in real-time, these IOCTLs also require real-time context. If creation was performed in non-real-time, the IOCTLs must be called in non-real-time as well.

  See Documentation/README.pools for further details

RTmac/TDMA Interface

- Real-time misc device for every RTmac-managed NIC
  e.g. `rteth0` => `TDMA0`

  ```
  fd = open_rt("TDMA0", O_RDONLY);
  ```

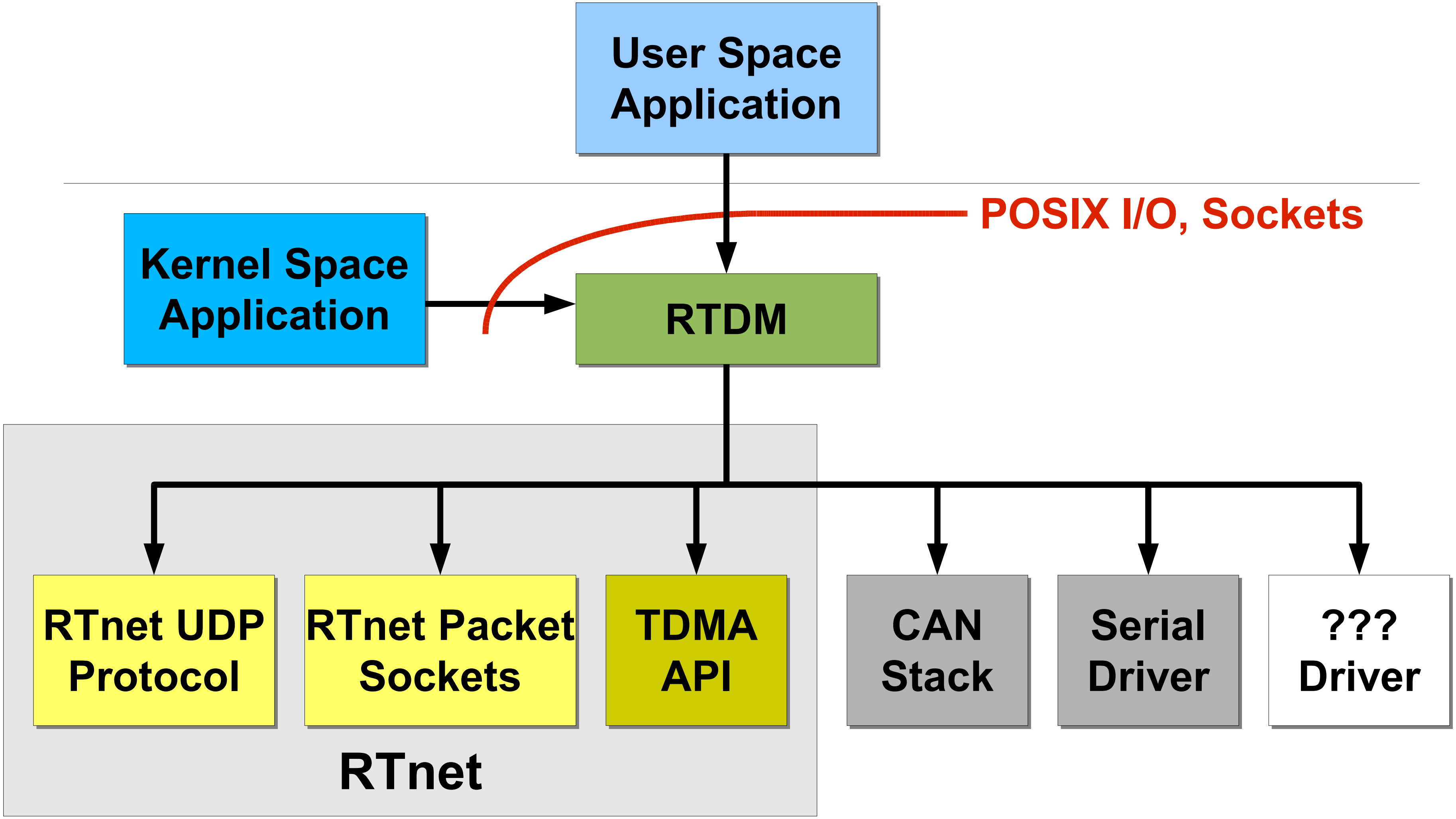
- Get global time offset
  IOCTL: `RTMAC_RTIOC_TIMEOFFSET`
  arg: `(__s64 *)delta_buffer`

- Wait on RTmac cycle
  IOCTL: `RTMAC_RTIOC_WAITONCYCLE`
  arg: `(int *)cycle_type`

  `RTMAC_WAIT_ON_DEFAULT` – Discipline default
  `RTMAC_WAIT_ON_XMIT` – Actual packet transmission time

  `TDMA_WAIT_ON_SOF` – Start of TDMA frame (TDMA default)

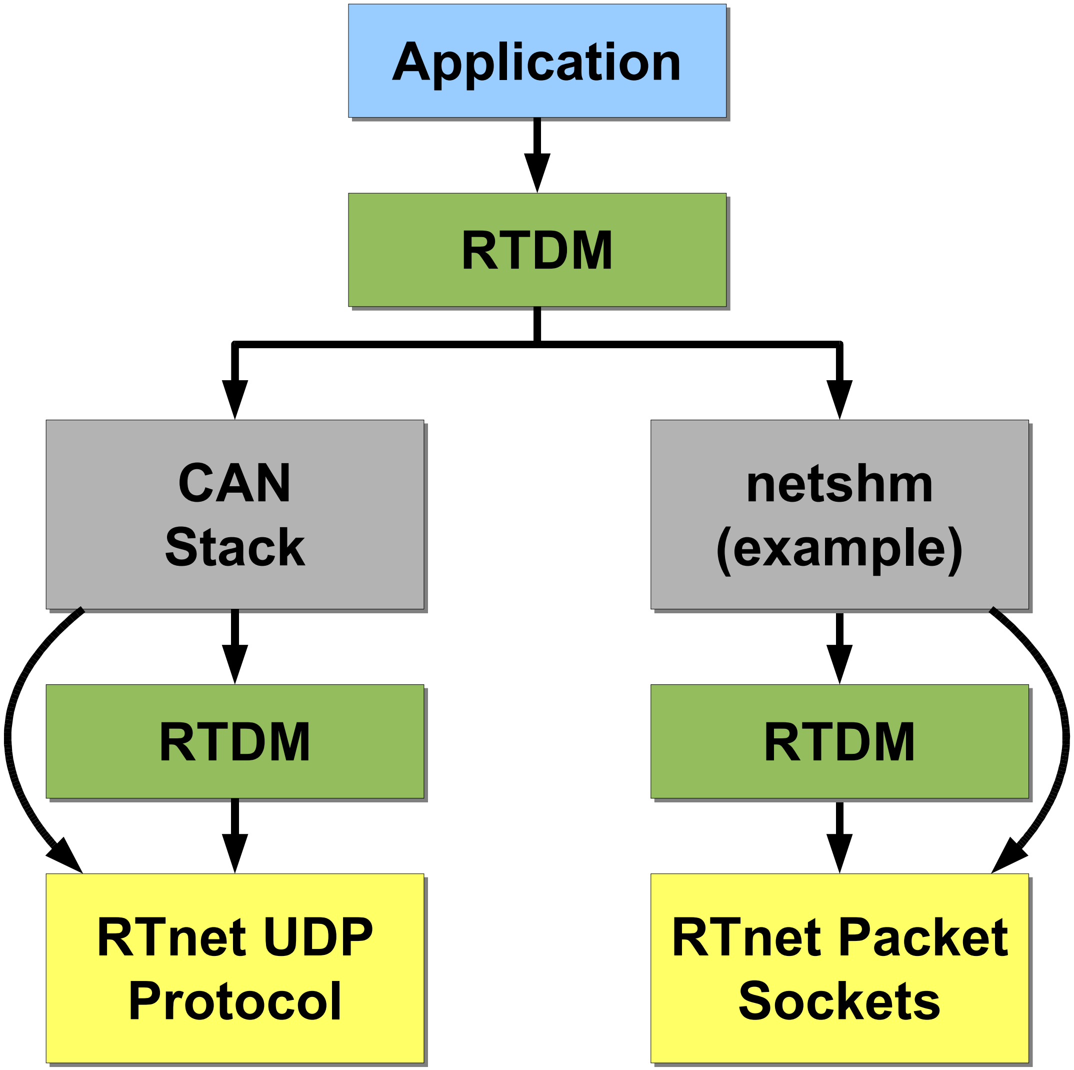
Real-Time Driver Model



Real-Time Driver Model (2)

- Provides POSIX I/O API for *named devices* (character and misc devices)
  `open_rt / close_rt`
  `read_rt / write_rt`
  `ioctl_rt`
- Provides Socket API for *protocol devices*
  `socket_rt / close_rt`
  `recvmsg_rt / sendmsg_rt`
  `ioctl_rt`
- Any other functions are mapped on **`recv`**/**`sendmsg_rt`** or on IOCTLs
- *Profiles* define what functions and IOCTLs a driver has to provide for a specific device class

Driver Stacks



Direct Device Access

- Unique context data structure per opened instance
- Get context structure from file descriptor (kernel mode)
  IOCTL: **`RTIOC_GETCONTEXT`**
  arg: **`(struct rtdm_getcontext_args *)vers_and_ptr`**

  Note: Context structure remains valid until lower device has been successfully closed. Stacked drivers need to take care of potential race conditions.
- Driver function can be called directly,
  avoids file descriptor lookup

```
result =  ctx->ops->read_rt(ctx, call_flags, ...);
result =  ctx->ops->read_nrt(ctx, call_flags, ...);
```

`_rt` / **`_nrt`**: call in real-time / non-real-time context

Configuration Interface

- Misc device (e.g. /dev/rtnet), minor = 240
- Core IOCTLs
  `IOC_RT_IFUP/_DOWN, IOC_RT_IFINFO`
- IP IOCTLs
  `IOC_RT_HOST_ROUTE_ADD/_SOLICIT/_DELETE,`
  `IOC_RT_NET_ROUTE_ADD/_DELETE,`
  `IOC_RT_PING`
- TDMA IOCTLs (RTmac itself doesn't provide any)
  `TDMA_IOC_CLIENT/_MASTER,`
  `TDMA_IOC_UP/_DOWN,`
  `TDMA_IOC_ADD/_REMOVE, ...`
- RTcfg IOCTLs
  An even longer list...

Examples

- frag_ip (RTAI-Kernel, UDP/IP)
  Exchange fragmented UDP packets.
- raw_packets (RTAI-Kernel, Packet Sockets)
  Exchange customised Ethernet packets.
- round_trip_time (RTAI-Kernel, UDP/IP)
  Measure round-trip delay at application level.
  Demonstrate UDP/IP interoperability with standard Linux application.
- rtnet_lxrt (LXRT, UDP/IP, RT-IOCTL)
  Exchange UDP packets between LXRT applications.
  Read list of interfaces and their parameters (IP and flags).
  Demonstrate UDP/IP interoperability with standard Linux applications.

RTmac/TDMA Examples

- event (RTAI-Kernel, UDP/IP)
  Compares distributed time stamps of an external event (serial or parallel port interrupt)
- rtt (RTAI-Kernel, UDP/IP)
  Round-trip delay measuring in RTmac-managed networks. Periodically or externally (parallel port) triggered.
- mrtt (RTAI-Kernel, UDP/IP)
  Measures round-trip delays between a single client and multiple servers.
- netshm (RTAI-Kernel, Packet Socket, RTDM)
  Simple distributed share-memory device driver (common read area, exclusive write sub-areas) with kernel demo application.

RTnet

www.rts.uni-hannover.de/rtnet

kiszka@rts.uni-hannover.de