Introduction to *RTEMS*

RTEMS in one day

by Thomas Dörfler
embedded brains GmbH
Obere Lagerstr. 30
D-82178 Puchheim
Germany

Navigator

Overview

Host And Target Environment

RTEMS Structure

Classic API

System Configuration

“Hello World” Tour

Basic Features

- Object-oriented
- Multitasking
- Multi-Processor Support
- Portable
- Various APIs
- C, C++, Ada supported
- Realtime-oriented
- Versatile Synchronization and Communication Mechanisms

- Configurable
- User Extensible
- File System Support
- Networking Support

And by the way...

- Open Source ☺

RTEMS: What It Is

- Operating System for Realtime and/or Embedded Applications
- Supports same APIs on all major 32 bit architectures
- Allows efficient use of processing time and memory resource
- Reliable realtime behaviour
- Tailored for
  - low memory footprint (e.g. 256KByte RAM, 512 kByte ROM)
  - Low processing power (e.g. 25MHz M68k systems)

RTEMS: What it isn't

- No sophisticated MMU support
  - No virtual memory
  - No memory protection
- No (or limited) multi-user environment
- No access security between internal tasks
  - File system
  - Memory
  - OS objects

RTEMS APIs

Classic API

- Implements “RTEID” “Real-Time Executive Interface Definition” standard
- Also available in Closed-Source products like pSOS^{+}™

Posix API

- Implements subset of POSIX 1003.1 standard
- Also available e.g. under Linux vxWorks™

ITRON API

- “The Realtime Operating Nucleus”
- Only partially implemented

Other APIs

- Can be implemented based on RTEMS “SuperCore” architecture

Navigator

Overview

Host And Target Environment

RTEMS Structure

Classic API

System Configuration

“Hello World” Tour

Host Platforms

Primary supported Host Platforms:

- **Linux**
- **Win32/cygwin**
- **New: Win32/MinGW**

Usable Host Platforms:

- **Apple Mac OS-X**
- **SunOS/Solaris**
- **Other Unix Derivates with GNU Toolchain support**

Build Environment

Primary Tools used to build RTEMS and Applications:

- **GCC(4.x)**
- **Binutils(2.17)**
- **Gnu Make**

Secondary Tools for Source Maintainance:

- **automake/autoconf**
- **Tex tools (Documentation)**

Debugging

Gdb-based

- **gdb-stub can be linked to application**
- **Alternative interfaces:**
  - **Mpc8xx-lib, Abatron bdi-2000, ... others**
- **Gdb can be integrated into various IDEs:**
  - **Insight, DDD, Eclipse...**
- **OS object display through gdb macros**

Other vendor tools

- **Lauterbach TRACE32, ... others**

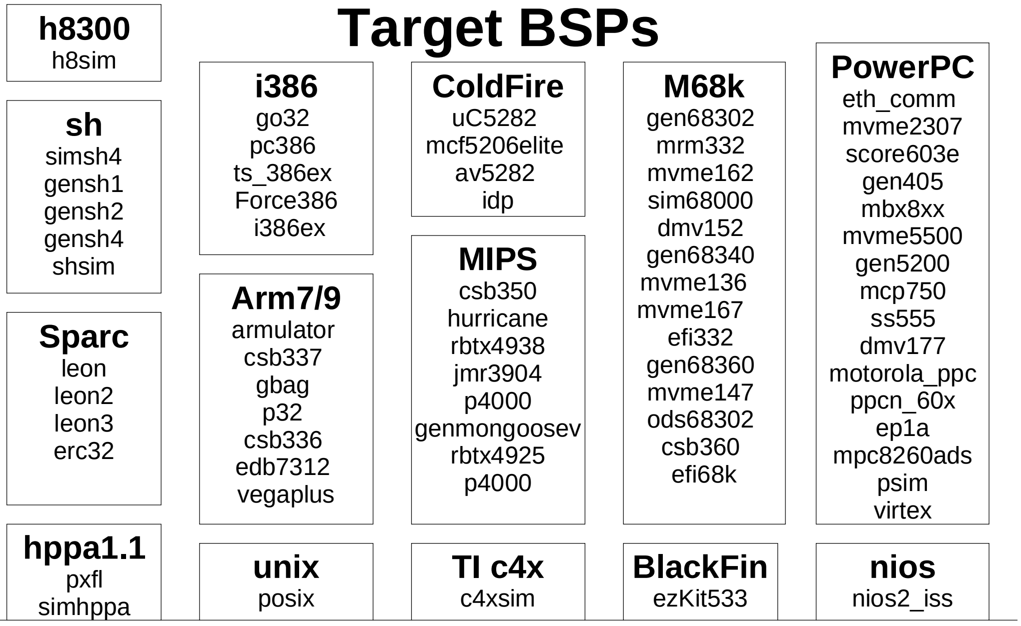
Target Architectures

- RTEMS can be ported to almost all 32 bit architectures
- Some special architectures also supported

Current Architectures:

- PowerPC
- M68K/ColdFire
- I386
- ARM
- BlackFin
- MIPS
- NIOS
- Super-H
- SPARC

Target BSPs



h8300
h8sim

sh
simsh4
gensh1
gensh2
gensh4
shsim

Sparc
leon
leon2
leon3
erc32

hppa1.1
pxfl
simhppa

i386
go32
pc386
ts_386ex
Force386
i386ex

Arm7/9
armulator
csb337
gbag
p32
csb336
edb7312
vegaplus

unix
posix

ColdFire
uC5282
mcf5206elite
av5282
idp

MIPS
csb350
hurricane
rbtx4938
jmr3904
p4000
genmongoosev
rbtx4925
p4000

TI c4x
c4xsim

M68k
gen68302
mrm332
mvme162
sim68000
dmv152
gen68340
mvme136
mvme167
efi332
gen68360
mvme147
ods68302
csb360
efi68k

BlackFin
ezKit533

PowerPC
eth_comm
mvme2307
score603e
gen405
mbx8xx
mvme5500
gen5200
mcp750
ss555
dmv177
motorola_ppc
ppcn_60x
ep1a
mpc8260ads
psim
virtex

nios
nios2_iss

Navigator

Overview

Host And Target Environment

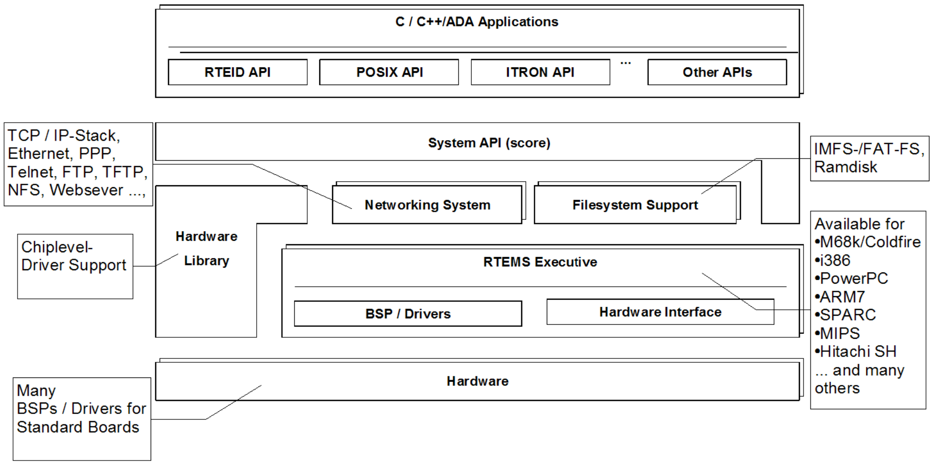
RTEMS Structure

Classic API

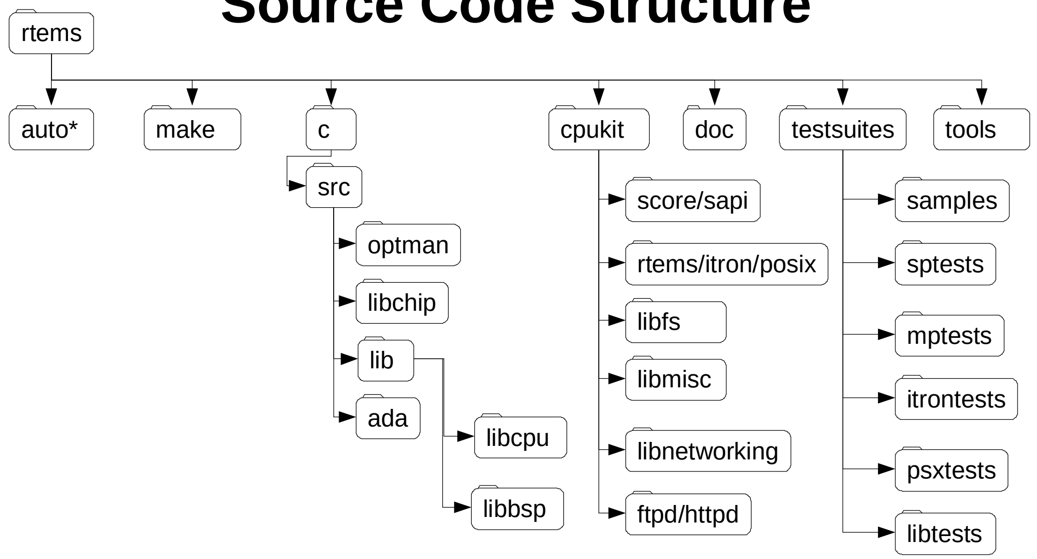
System Configuration

“Hello World” Tour

SW Structure



Source Code Structure



- rtems
  - auto*
  - make
  - c
    - src
      - optman
      - libchip
      - lib
        - libcpu
        - libbsp
      - ada
  - cpukit
    - score/sapi
    - rtems/itron/posix
    - libfs
    - libmisc
    - libnetworking
    - ftpd/httpd
  - doc
  - testsuites
    - samples
    - sptests
    - mptests
    - itrontests
    - psxtests
    - libtests
  - tools

RTEMS File Systems

Internal File Systems

- IMFS “In Memory FS”
  - Root-FS
  - Kept in RAM (Heap)
- tar fs
  - Linked from IMFS into tar image
- miniIMFS
  - Reduced version of IMFS

Non-volatile File Systems

- DOSFS: FAT file system
  - ☺ compatible to “industrial standard”
  - Used for hard discs and CF

Networked File Systems

- FTPFS
  - File based access to FTP server
- TFTPFS
- NFS

Networking

FreeBSD Stack ported to RTEMS

- robust
- flexible
- standard socket calls
- multiple Interfaces
- support for Ethernet and PPP

Network Servers

- HTTP server (GoAhead)
- FTP server
- SNMP server (contrib)
- Telnet server (“shell”)

Network Clients

- BOOTP/DHCP client
- TFTP filesystem
- FTP filesystem

Navigator

Overview

Host And Target Environment

RTEMS Structure

Classic API

System Configuration

“Hello World” Tour

Classic API

- Conforms to RTEID realtime API standard (like pSOS$^+$ etc.)
- First API available on RTEMS
- Design based on various objects

Object types available:

- Tasks
- Semaphores
- Message Queues
- Regions
- Partitions

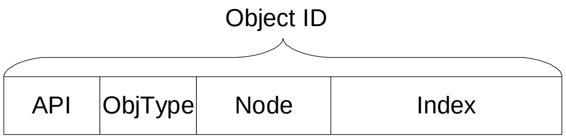
Classic API: Objects

Classic API object types:

- Tasks
- Semaphores
- Message queues
- Rate monotonic periods
- Partitions
- Regions

Identification

- Each object has unique (32 bit) identifier
- ID contains:
  - Object type
  - Node number
  - Index



API	ObjType	Node	Index

Classic API: Object Methods

`rtems_xxx_create(rtems_name name,rtems_id *id,...)`

- Creates object of type xxx
- name: up to four ASCII characters as arbitrary name
- `*id`: ptr to location, where object id will be stored

`rtems_xxx_delete(rtems_id *id)`

- Deletes object of type xxx
- Wakes up any tasks waiting for this object

`rtems_xxx_ident(rtems_name name,rtems_id *id)`

- Determines object id of (first) object with given name

Classic API: Tasks

Definition

- Task is a processing entity that processes and data and interacts with (OS) objects
- Multiple tasks share the processor hardware
- Scheduler decides, which task is allowed to use the processor

Attributes

  - Fixed at creation time
- FPU context (yes/no)
- Stack size

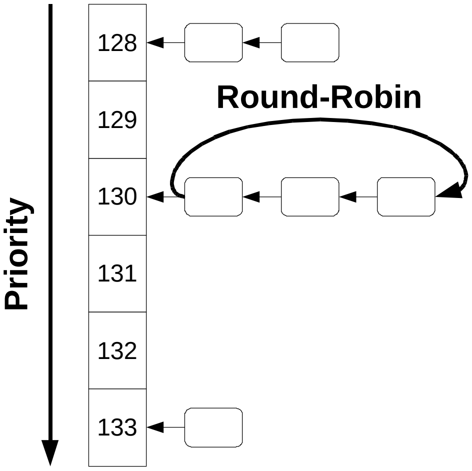
Mode options

  - Changeable during runtime
- (scheduling) priority
- Preemption
- Timeslicing
- Async. Signal Handling

RTEMS Scheduling

RTEMS supports

- **256 task priorities**
- **Unlimited tasks with same priority**
- **“Ready” queue for each priority level**
- **Bitmap-based fast lookup to determine highest-priority level with a “ready” task**



RTEMS Scheduling Options

Time Slicing

- **Task will yield the processor when it has executed for one timeslice**
- **Other tasks of same priority have a chance to run**
- **Implements a Round-Robin scheduling scheme between tasks of same priority**

Preemption

- **When enabled: Scheduler can preempt task at any time to run a different (higher priority) task**
- **When disabled: Task continues to execute, even if a higher priority task becomes ready**
- **Non-preemptive task executes, until it calls a blocking function**

Classic API: Inter Task Communication

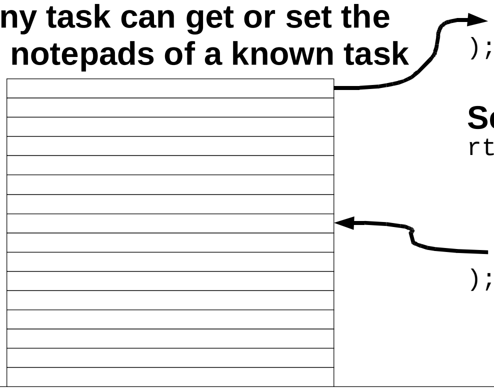
Mechanisms available to implement Inter Task Communication

- **Notepads**
- **Events**
- **Signals**
- **Semaphores**
- **Message queues**
- **Shared memory/variables**

Classic API: Task Notepads

Each task has 16 32-bit-“Notepads” as local IPC variables

Any task can get or set the notepads of a known task



Get a Notepad:

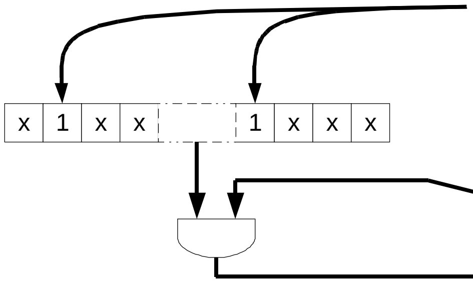
```
rtems_status_code
  rtems_task_get_note(
  rtems_id          id,
  rtems_unsigned32  notepad,
  rtems_unsigned32 *note
);
```

Set a Notepad:

```
rtems_status_code
  rtems_task_set_note(
  rtems_id          id,
  rtems_unsigned32  notepad,
  rtems_unsigned32  value
);
```

Classic API: Task Events

Each task has a 32 bit event word



Set an event:

```
rtems_status_code
  rtems_event_send(
  rtems_id         tid,
  rtems_event_set  event_in
);
```

Get own events:

```
rtems_status_code
  rtems_event_receive (
  rtems_event_set  event_mask,
  rtems_option     option_set,
  rtems_interval   ticks,
  rtems_event_set *event_out
);
```

Note:

- **There is no “queue” of events!**

Options:

- **wait/nowait**
- wait for **any/all** events

Classic API: Signals and ASRs

What is a signal?

- **RTEMS defines 32 signals (bits) per task**
- **Tasks/ISRs can send signals to arbitrary tasks**
- **Signals are rejected, when receiving task has not ASR**
- **Signals of a task are OR'd together to a 32 bit mask**

What is an ASR?

- **“Asynchronous Service Routine”**
- **Executes in the task's**
  - **context**
  - **processing time**
- **called, when**
  - **signal was sent to the task**
    and
  - **task gets scheduled again**

Classic API: ASR Installation

Task can establish an ASR to handle received signals

```
rtems_status_code
  rtems_signal_catch(
  rtems_asr_entry  asr_handler,
  rtems_mode mode
);
```

Task can enable/disable ASR execution using its own task mode setting
(RTEMS_ASR/RTEMS_NO_ASR)

```
rtems_status_code
  rtems_task_mode(
  rtems_mode  mode_set,
  rtems_mode  mask,
  rtems_mode *previous_mode_set
);
```

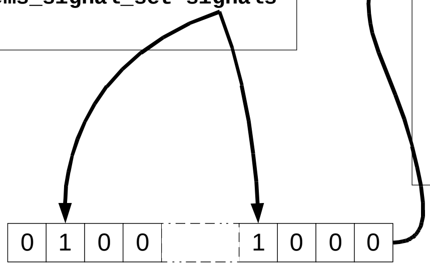
Classic API: Signals and ASRs

Any task can send signals to a certain task:

```
rtems_status_code
  rtems_signal_send(
  rtems_id task_id,
  rtems_signal_set signals
);
```

Example of an ASR

```
rtems_asr_entry my_asr
 (rtems_signal_set signals)
{
  if (signals & MY_KILL_SIG) {
    /* ... cleanup resources ..*/
    rtems_task_delete(RTEMS_SELF);
  }
  if (signals & MY_DUMP_SIG) {
    my_dump_task_state(...);
  }
  ...
}
```



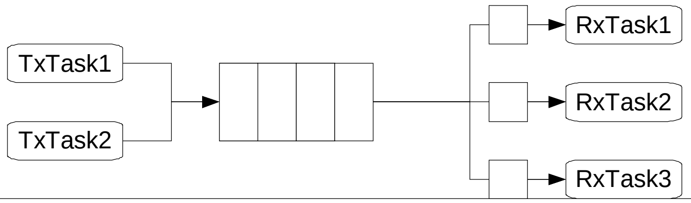
Classic API: Message Queues

Definition:

- A message queue is like a FIFO for messages
- Queue can store a limited number of messages

Features and Options:

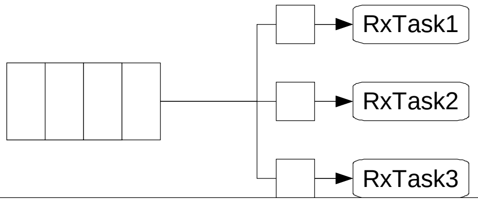
- Max. message size
- Max. message count
- “urgent” and “broadcast” messages
- wait policy
  - FIFO or PRI



Classic API: Message Queues

```
rtems_message_queue_receive
  (id,msgbuf,*size,RTEMS(_NO)_WAIT,timeout)
```

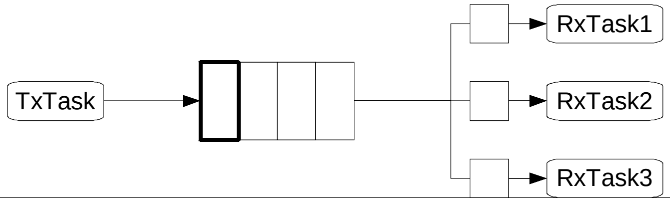
- Get message from front of queue



Classic API: Message Queues

`rtems_message_queue_send(id,msgbuf,size)`

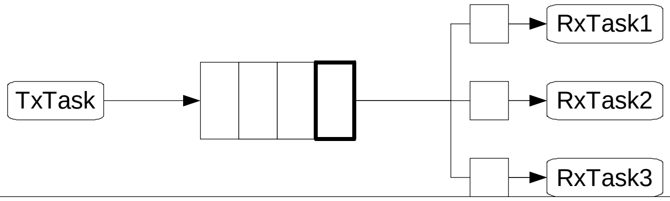
- **Put message to rear of a queue**



Classic API: Message Queues

`rtems_message_queue_urgent(id,msgbuf,size)`

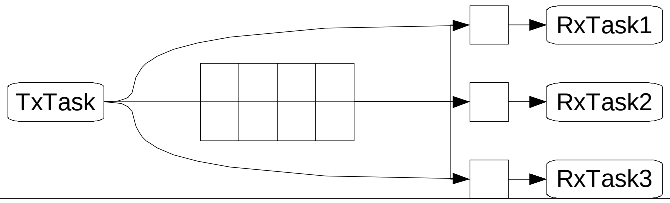
- **Put message to front of a queue**



Classic API: Message Queues

```
rtems_message_queue_broadcast
     (id,msgbuf,size)
```

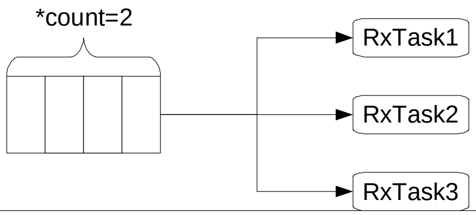
- **Send message to all waiting tasks**



Classic API: Message Queues

```
rtems_message_get_number_pending(id,*count)
```

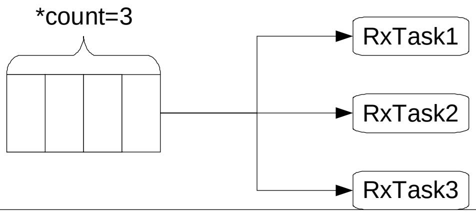
- **Get current message count of a queue**



Classic API: Message Queues

```
rtems_message_queue_flush(id,*count)
```

- **Clear all messages in queue**



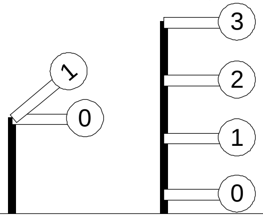
Classic API: Semaphores

Definition:

- **OS object used to ensure, that task access to shared resources is properly limited.**
- **Can have two (binary) or multiple (counting) states**
- **State “0” means that access is blocked**

Features:

- **Binary(0/1) or Counting (0-n)**
- **FIFO or priority scheduling**
- **Priority inheritance**
- **Priority ceiling**



Classic API: Semaphore Creation

```
rtems_semaphore_create (name, count, attrib,
  ceiling, *id)
```

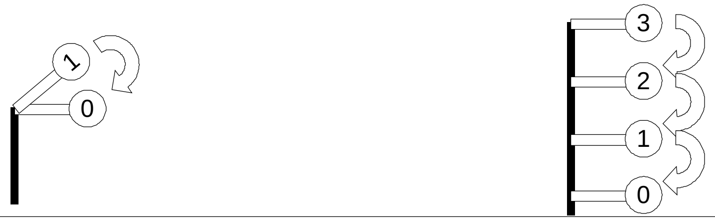
- Create a semaphore
- Attributes:
  - Task wait by: RTEMS_FIFO or RTEMS_PRIORITY
  - Semaphore type: COUNTING or BINARY or SIMPLE_BINARY
  - Priority inheritance: YES or NO
  - Priority ceiling: YES or NO
  - Scope: LOCAL or GLOBAL

Classic API: Semaphore Obtain

```
rtems_semaphore_obtain (name, options, timeout)
```

aquire lock to shared resource

- Optionally wait, until semaphore is available

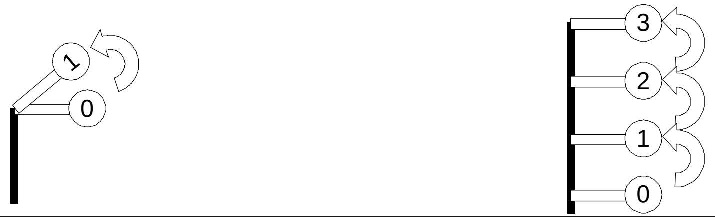


Classic API: Semaphore Release

`rtems_semaphore_release(name)`

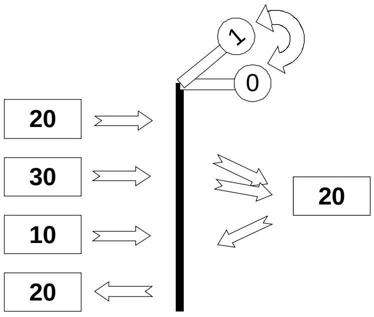
release lock of shared resource

- **The next waiting task may immediately gain the semaphore**



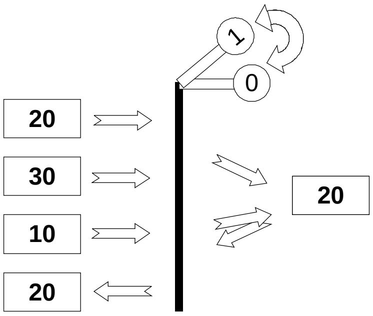
Classic API: Semaphore FIFO Scheduling

- **When Semaphore becomes available, first waiting task gets it**



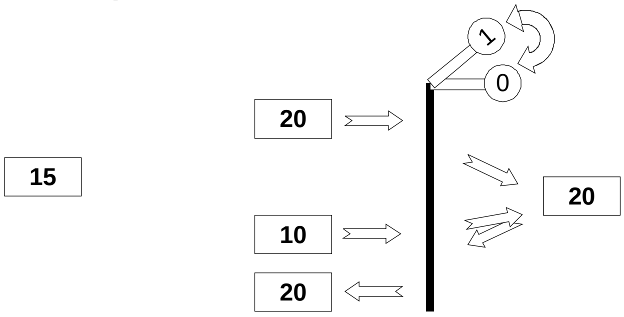
Classic API: Semaphore Priority Scheduling

- **When Semaphore becomes available, highest priority waiting task gets it**



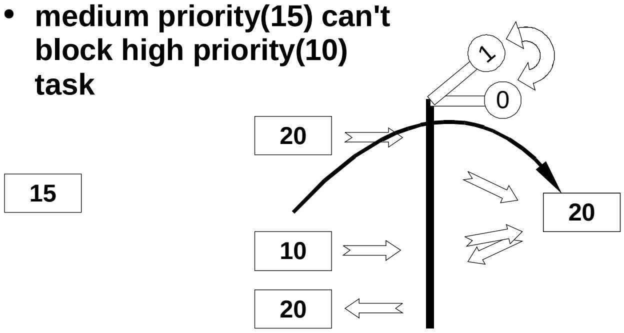
Classic API: Priority Inversion

- **Task with medium priority(15) can block high priority(10) task, because low priority(20) task holds semaphore**



Classic API: Priority Inheritance

- Low priority(20) task holding semaphore inherits high priority(10) of blocked task
- medium priority(15) can't block high priority(10) task



Classic API: Regions

Definition:

- A region is a memory area, that provides service to allocate (and return) <u>variable length</u> memory segments
- Allows flexible memory usage

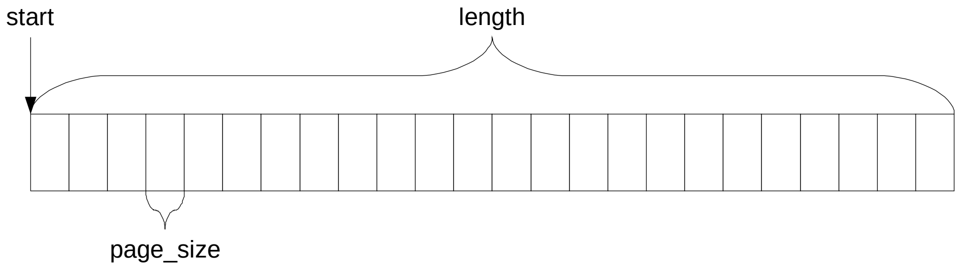
Features:

- Region is created in contiguous memory area
- Can be extended from different areas
- Allocates segments in multiples of its page size
- Tasks waiting for segment can optionally be blocked

Classic API: Regions

```
rtems_region_create (name, *start, length,
     page_size, FIFO|PRIO, *id)
```

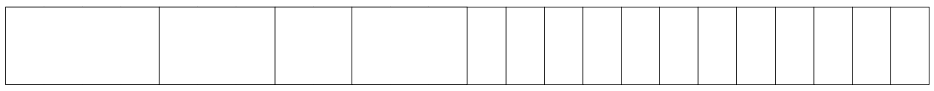
- **Create a region**



Classic API: Region Get Segment

```
rtems_region_get_segment (id, size,
 RTEMS_(NO_)WAIT, timeout, void **segment)
```

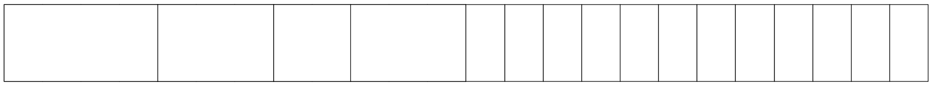
- **Get segment from region**



Classic API: Region Return Segment

```
rtems_region_return_segment
    (id, void *segment)
```

- **Return segment to region**



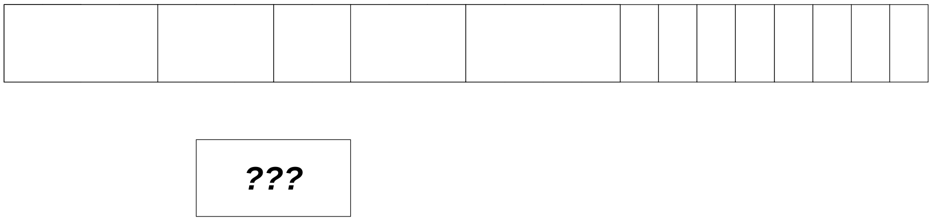
Classic API: Region Fragmentation

Example:

1: Get four different segments

2: Return two segments

3: Get same two segments again, with different order



Classic API: Partitions

Definition:

- **A partition is a memory area, that provides service to allocate (and return) fixed length memory buffers**
- **Allows simple memory management**
- **Avoids fragmentation**

Features:

- **Partition is created in contiguous memory area**
- **Allocates buffers with fixed size**
- **Tasks waiting for buffer cannot block**

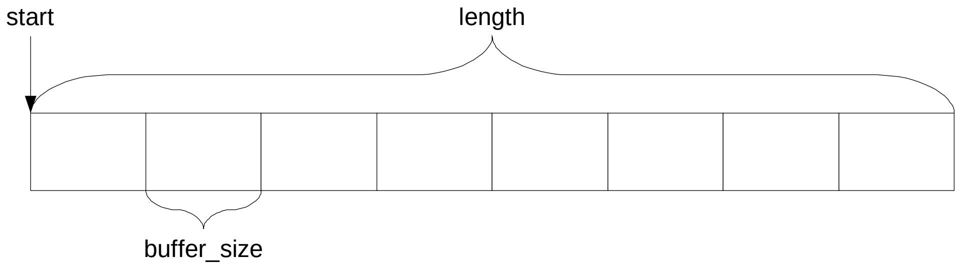
Question:

- **How can we implement a blocking mechanism for tasks needing a buffer?**

Classic API: Partitions

```
rtems_partition_create (name, *start,
 length, buffer_size, LOCAL|GLOBAL, *id)
```

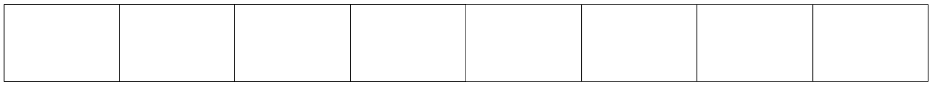
- **Create a partition**



Classic API: Partition Get Buffer

```
rtems_partition_get_buffer (id, void
 **buffer)
```

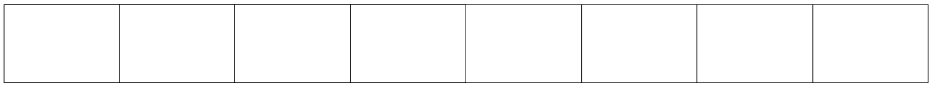
- Get one buffer from partition



Classic API: Partition Return Buffer

```
rtems_partition_return_buffer (id,
    void *buffer)
```

- Return one buffer to partition



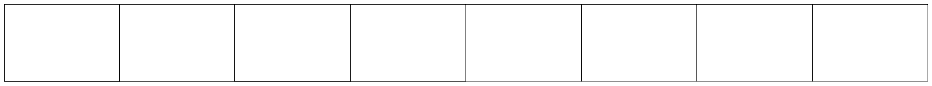
Classic API: Partition Fragmentation

Example:

1: Get four different buffers

2: Return two buffers

3: Get same two buffers again, with different order



Navigator

Overview

Host And Target Environment

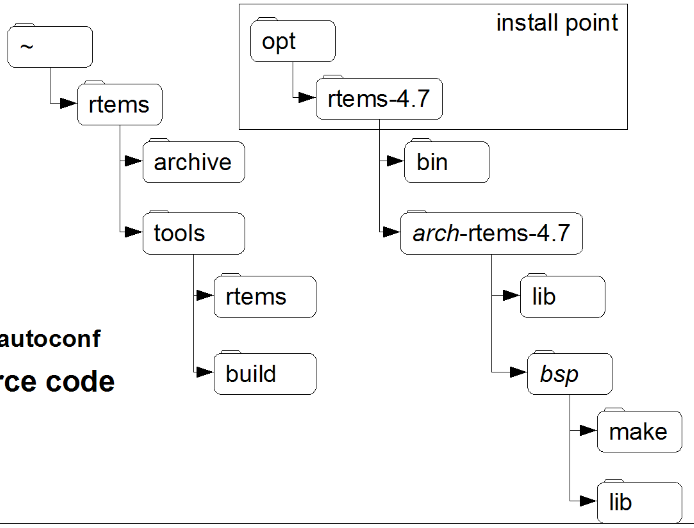
RTEMS Structure

Classic API

System Configuration

“Hello World” Tour

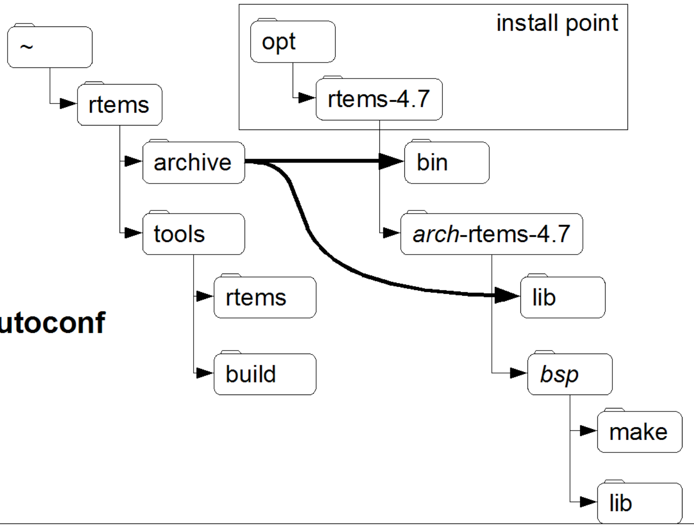
System Configuration
Step 1: Get Packages



Download:

- **Tools:**
  - **Binutils**
  - **Gcc**
  - **Gdb**
  - **Automake/autoconf**
- **RTEMS source code**

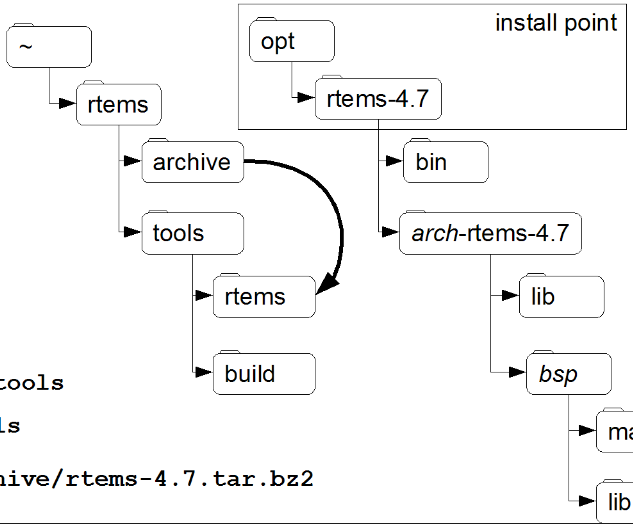
System Configuration
Step 2: Install Tools



Install:

- **Binutils**
- **Gcc**
- **Gdb**
- **Automake/autoconf**
- **Install via**
  - **RPM**
  - **Or “tar xjf”**

System Configuration
Step 3: Install RTEMS Source


~

rtems

archive

tools

rtems

build

install point

opt

rtems-4.7

bin

arch-rtems-4.7

lib

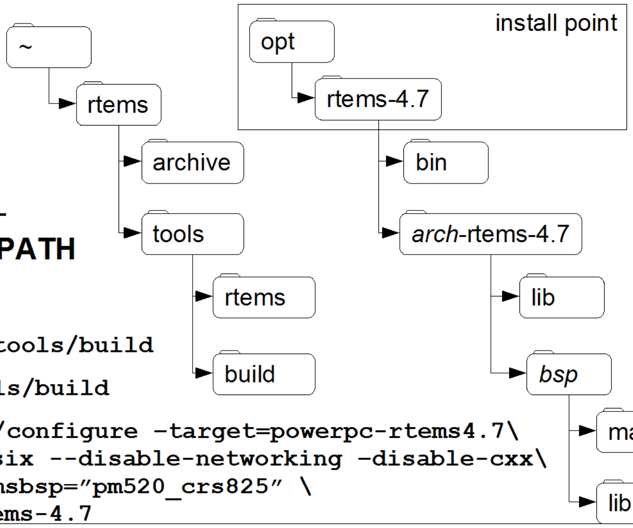
bsp

make

lib

- `mkdir ~/rtems/tools`
- `cd ~/rtems/tools`
- `tar xjf ../archive/rtems-4.7.tar.bz2`

System Configuration
Step 4: Configure RTEMS & BSP


~

rtems

archive

tools

rtems

build

install point

opt

rtems-4.7

bin

arch-rtems-4.7

lib

bsp

make

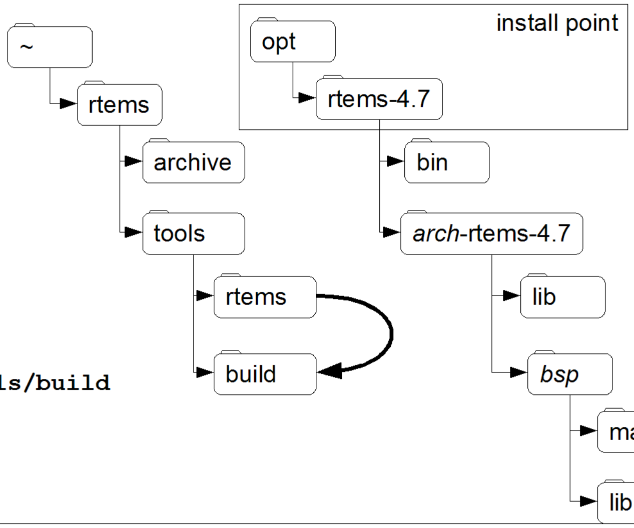
lib

Preparation:

- **Add** `<install-point>/bin` **to PATH**

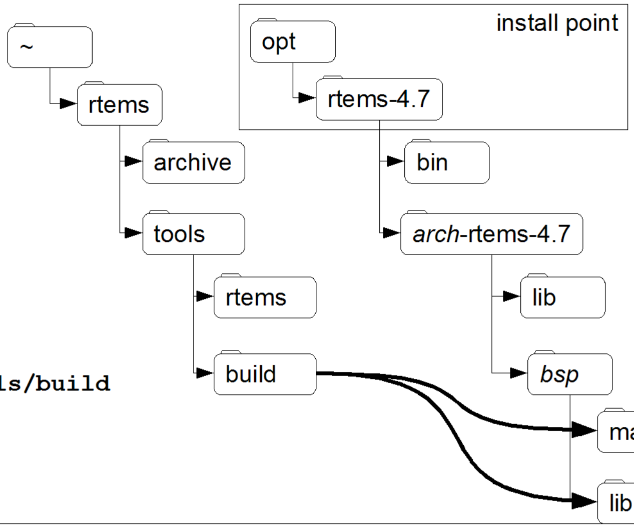
- `mkdir ~/rtems/tools/build`
- `cd ~/rtems/tools/build`
-
```
../rtems-4.7.1/configure –target=powerpc-rtems4.7\
  --disable-posix --disable-networking –disable-cxx\
  --enable-rtemsbsp=”pm520_crs825” \
--prefix=/opt/rtems-4.7
```

System Configuration
Step 5: Build RTEMS & BSP



- `cd ~/rtems/tools/build`
- `make all`

System Configuration
Step 6: Install RTEMS & BSP



- `cd ~/rtems/tools/build`
- `make install`

Navigator

Overview

Host And Target Environment

RTEMS Structure

Classic API

System Configuration

“Hello World” Tour

“Hello World” Tour

Live Demonstration:

- **How to create an RTEMS application**
- **How to test it in a gdb simulator**

(Place for your notes)

RTEMS Development

Development Model of RTEMS:

- New features and improvements are user/application driven
- User contributions or funding of development to support companies
- Source maintenance by OAR Corp, Huntsville

Support:

- User's Mailing List
- Commercial support:
  - OAR Corp, Alabama, USA
  - Cybertec, Australia
  - embedded brains GmbH, Germany

RTEMS Training

Regular Open Classes are held in:

- Huntsville, Alabama (organized by OAR Corp)
- Munich, Germany (organized by embedded brains GmbH)

Specialized and On-Site classes available on request

RTEMS Links

http://www.rtems.com

- Main website with
  - sources, tools, CVS
  - Documentation and Wiki
  - Mailing list archive

http://www.embedded-brains.de/

- Website of embedded brains GmbH
- Products and support services for RTEMS/embedded design

Coming soon:

http://www.rtems.de

- Mirror of RTEMS sources, tools, CVS, documentation

```
ERROR: undefined
OFFENDING COMMAND:

STACK:
```