RTAI Programming

Sowjanya
C-DAC, Hyderabad

Topics

- Synchronization
  - RT Semaphores
- Inter-process Communication
  - RT Fifos
  - RT Mailboxes
  - RT Messages
  - Remote Procedure Calls (RPCs)

RT Semaphores

- There are 3 types of Semaphores in RTAI
  - Binary Semaphores
    - Provides mutual exclusion or signalling
  - Resource Semaphores
    - Provides priority inheritance and recursion, to avoid problems in mutual exclusion
  - Counting Semaphores
    - To guard multiple instances of the resource

RT Semaphores

- Initializing the RT Semaphore
  - **void rt_typed_sem_init (SEM *sem, int value, int type)**
  - Type: CNT_SEM, BIN_SEM, RES_SEM
  - Queuing Policy: FIFO_Q, PRIO_Q
  - Eg: rt_typed_sem_init(&S1, 0, BIN_SEM | FIFO_Q)
- Alternatively, it can be initialized as
  - **rt_sem_init (SEM *sem, int value)** for counting semaphores

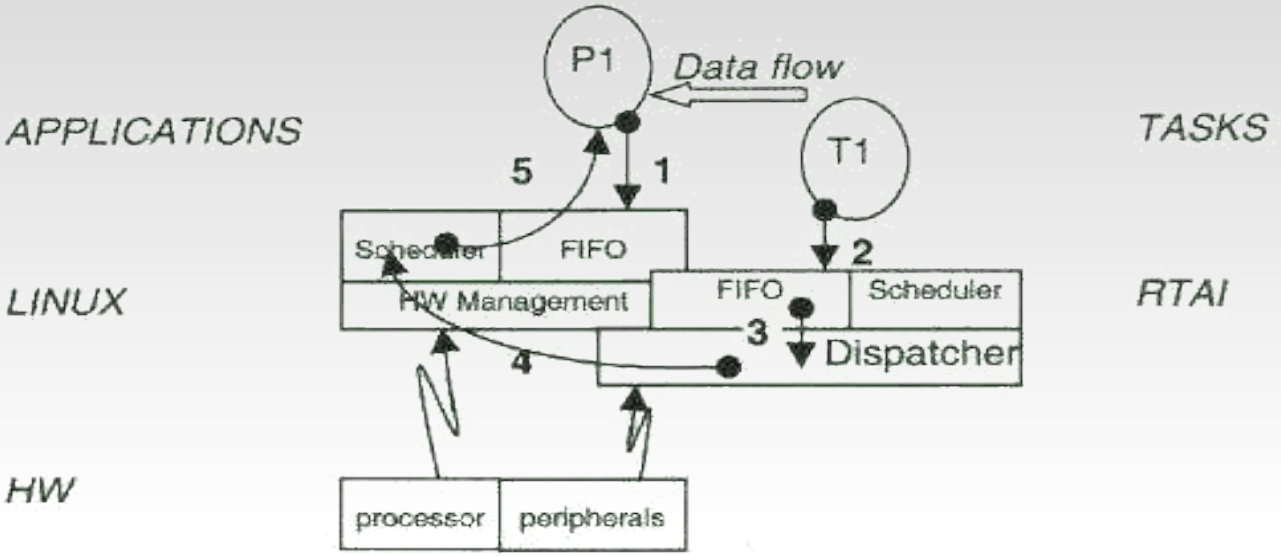
RT Semphores

- Gain access to the Semaphore
  - int **rt_sem_wait** (SEM *sem) / **rt_sem_wait_if** (SEM *sem)
  - int **rt_sem_wait_until** (SEM *sem, RTIME time)
  - int **rt_sem_wait_timed** (SEM *sem, RTIME delay)
- Release the Semaphore
  - int **rt_sem_signal** (SEM *sem)
  - int **rt_sem_broadcast** (SEM *sem)
- Delete a Semaphore
  - int **rt_sem_delete**(SEM *sem)

RT FIFOs

- RT FIFOs allow Linux processes and RTAI tasks to exchange byte-oriented data steams.
  - One Way Channel - Duplexing requires 2 FIFOS
  - Non Blocking from RTAI perspective, either blocking or non blocking from the Linux perspective
  - To perform synchronous reads (and to avoid polling) RTAI makes it possible to attach a user real time handler to a fifo.

RT FIFOs



1) blocking read.
2) rtf_put call.
3) SRQ initialization.
4) wake-up action in queue.
5) wake-up action de-queued at next Linux scheduling

RT FIFOs

- Creating fifos
  - Real-time side,
    - **int rtf_create (unsigned int fifo, int size)**
    - ***fifo* –** Fifo number ranging from 0 to RTF_NO (currently 63), used in further fifo operations.
  - Linux side
    - Fifo numbers are associated with character devices /dev/rtf0 ... /dev/rtf63 (Created on RTAI install)
    - **fd = open("/dev/rtf0", O_RDONLY)**

RT FIFOs

- int **rtf_get**(unsigned int fifo, void * buf, int count)
- int **rtf_put**(unsigned int fifo, void * buf, int count)
- int **rtf_destroy**(unsigned int fifo)
  - Closes fifo (after the last close, fifo is really destroyed)
- int **rtf_sem_init**(unsigned int fifo, int value)
- int **rtf_sem_post**(unsigned int fifo)
- int **rtf_sem_trywait**(unsigned int fifo)
- int **rtf_sem_destroy**(unsigned int fifo)

RT FIFOs

- FIFO Handlers
  - Created using the function
    rtf_create_handler(int fifo_num, my_handler)
  - The handler function is declared as
    void my_handler(int fifo_num)
  - To receive an optional argument in the handler, signifying read/write operation from the user space on FIFO. **rtf_create_handler (fifo_num, X_FIFO_HANDLER(my_handler))**

RT FIFOs

- Example Handler

```
rtf_create_handler(fifo_number, X_FIFO_HANDLER(my_handler))

int my_handler(unsigned int fifo, int rw) {
        if (rw == 'r') {
                // handle a read call and return appropriate value
        } else {
                // handle a write call and return appropriate value
        }
}
```

RT Mailboxes

- Mailbox is a buffer managed by the RT Kernel
  - Allows variable number of messages, of variable length, to be queued.
  - Multiple tasks can send to and receive from the same mailbox.
  - A receiving task reads the messages in the order of arrival.
  - These are one way communication channels

RT Mailboxes

- There are 4 variants in mailboxes
  - Unconditional Mailboxes (Default)
  - Best Effort Mailboxes (Extension _wp)
  - Conditional or Availability Mailboxes (_if)
  - Timed Mailboxes (_up or _timed)

RT Mailboxes

- Initialize a Mailbox
  - **int rt_mbx_init (MBX *mbx, int size)**
- Send Data to a Mailbox
  - int **rt_mbx_send**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_send_wp**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_send_if**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_send_until**(MBX *mbx, void *msg, int msg_size, RTIME time)
  - int **rt_mbx_send_timed**(MBX *mbx, void *msg, int msg_size, RTIME delay)

RT Mailboxes

- Receive Data from a Mailbox
  - int **rt_mbx_receive**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_receive_wp**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_receive_if**(MBX *mbx, void *msg, int msg_size)
  - int **rt_mbx_receive_until**(MBX *mbx, void *msg, int msg_size, RTIME time)
  - int **rt_mbx_receive_timed**(MBX *mbx, void *msg, int msg_size, RTIME delay)
- Delete a Mailbox
  - **int rt_mbx_delete(MBX *mbx)**

RT Messages

- Message passing mechanism, allows sending a four byte message from a sender to a receiver
  - Blocking send and receive calls.
  - **RT_TASK * rt_send (RT_TASK *task, unsigned int msg)**
  - **RT_TASK * rt_receive (RT_TASK *task, unsigned int *msg)**
    - If ***task*** is equal to 0, the caller accepts messages from any task

RT Messages

- Non blocking calls
  - **rt_send_if**
  - **rt_receive_if**
- Timeout calls with Absoulte / Relative time delay
  - **rt_send_until / rt_send_timed**
  - **rt_receive_until / rt_receive_timed**

RPCs

- Remote Procedure Calls (RPCs) provide inter-task messaging facility
  - 32-bit values can be passed
  - Tasks are coupled awaiting a reply from the receiver.
- Make a remote procedure call (send message)
  - **RT_TASK *rt_rpc (RT_TASK *task, unsigned int msg, unsigned int *reply)**
  - ***reply* -** Points to a buffer provided by the caller for the returned result message(4 bytes int) to be placed

RPCs

- To receive message
  - **RT_TASK* rt_receive (RT_TASK* task, unsigned int *msg)**
- To return the result back to the task that made the related RPC
  - **RT_TASK *rt_return (RT_TASK *task, unsigned int result)**
  - ***task* -** Task pointer returned by the receive function
  - ***result* -** Gets placed in the ***reply*** buffer

Thank You