



NetworkCentric Core Avionics



RODOS tutorial ABC (Alice Bob Charly)

Version: 1.0
Date: 10.10.2009
Author: Sergio Montenegro

RODOS Tutorial ABC

This tutorial shows how to build bigger systems using very simple components (building blocks).

How to begin.

Read and understand first the RODOS-Tutorial and then the RODOS-Middleware tutorial.

use

```
% linux-executable <list of sources>  
to compile and  
%tst
```

to execute – e.g.

```
% linux-executable police.cpp  
% tst
```

For each example program please first read and understand the code, then compile and execute and see if it acts as expected. Then modify and continue trying.

Useful to know:

The middleware communication is based on a publisher/subscriber protocol. This is a multicast protocol. There is no connection from a sender to a receiver (connectionless).

Publishers make messages public under a given topic.

Subscribers to a given topic get all messages which are published under the given topic.

To establish a transfer path, both the publisher and subscriber have to share the same topic.

A Topic is a pair - data-type and an integer representing a topic identifier.

To match a communication pair, both data-type and topic identifier must match.

For a topic there may be zero, one or several subscribers. A message which is published under a topic will be distributed to 0 or more subscribers.

Each subscriber has a reference to the associated topic and a “putter” to store messages. Or in this more simple example Threads may be waiting for messages. If a message is published while a thread is waiting for it, the thread will get a copy of the message else not.

All subscribers are registered in a list. Each time a message is published the list of all subscribers will be searched and for each subscriber where the topic matches the associated putter will be called to store a copy of the published message.

Using a network interface and the corresponding gateways, the middleware may make the node borders transparent. Applications running on different computers may communicate as if they were on the same computer. (not in this example)

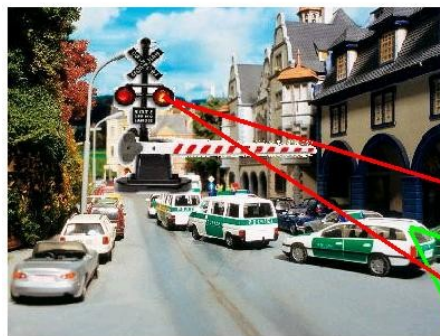
Examples

police.cc is a simple example of an array of threads using time synchronisation and semaphores to protect critical regions.

```
% linux-executable police.cpp
%tst
```

See the ID of each thread (this) to identify who is running.

Thread, Semaphore & Time Control (Synchronisation, Time-Events)



```
#include "rodos.h"

Semaphore onlyOne;


class Watcher : public Thread {
public:
    void run() {
        TIME_LOOP(3*SECONDS, 2*SECONDS) {
            onlyOne.enter();
            PRINTF(" only one, I am %lx", this);
            yield(); // Just to show that the semaphore protects
            PRINTF("time %l 0.6f\n", SECONDS_NOW());
            onlyOne.leave();
        }
    }
} team[20];
```

alice_bob_charly.cpp : Is a short demonstration of threads communicating by publishing messages and threads waiting for them. To see more elaborated publisher/Subscriber communication please refer to tutorial_middleware.

```
% linux-executable alice-bob-charly.cpp
%tst
```

alice, bob and charly share communication topics:

Inter-Task communication



```

/***** Common Data *****/
struct Greetings {
    TTime date;
    char msg[80];
};

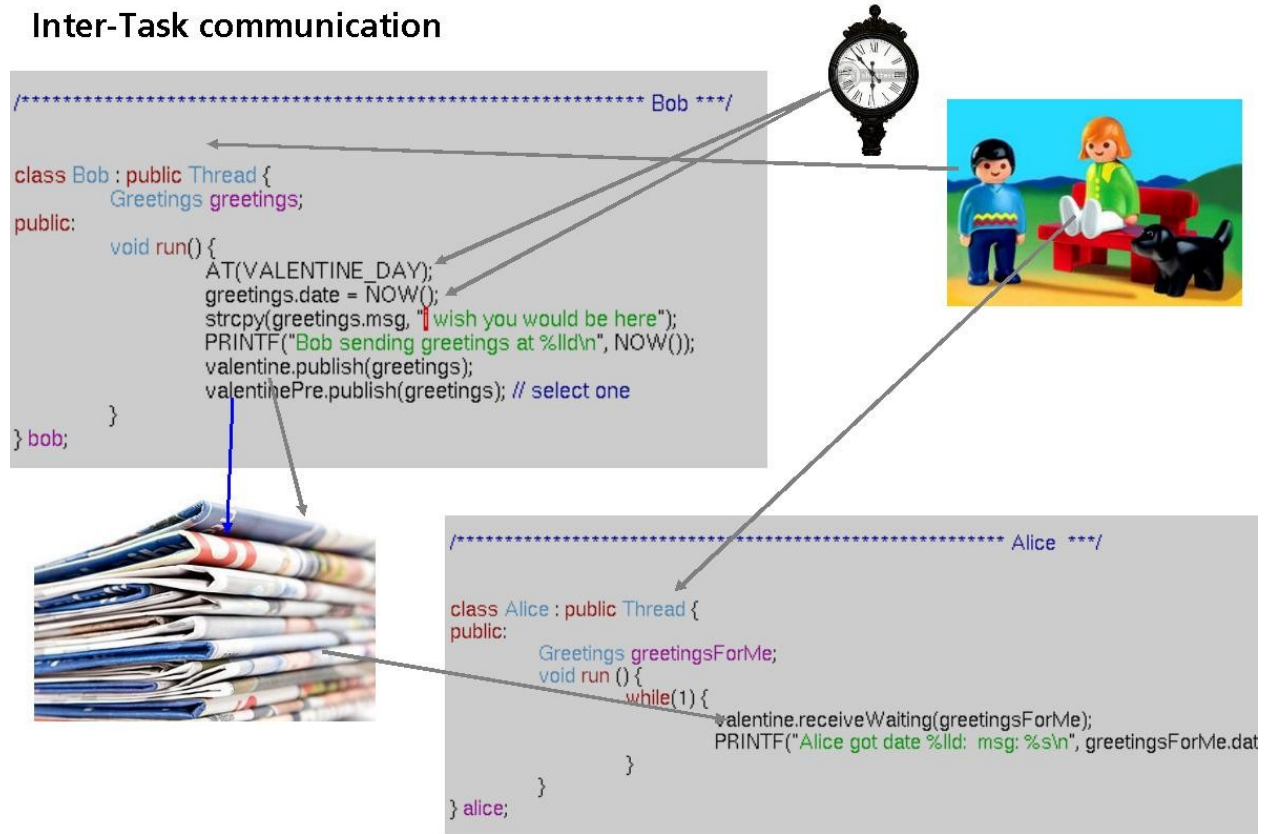
Topic <Greetings, 20> valentine;
Topic <Greetings, 21> valentinePre;

#define VALENTINE_DAY    2*SECONDS
    
```

A topic is public, not a directed letter

Bob publishes a message at time valentine day (in the example 2 seconds) and Alice waits for it

Inter-Task communication



In the next step comes Charly, who waits for the same messages makes a correction and sends the corrected version.

Inter-Task communication

```

/***** Bob *****/

class Bob : public Thread {
public:
    Greetings greetings;

    void run() {
        AT(VALENTINE_DAY);
        greetings.date = NOW();
        strcpy(greetings.msg, "I wish you would be here");
        PRINTF("Bob sending greetings at %ld\n", NOW());
        valentine.publish(greetings);
        valentinePre.publish(greetings); // select one
    }
} bob;

```



```

/***** Charly *****/

class Charly : public Thread {
public:
    SyncFifo<Greetings, 5> fifo;
    Greetings greetingsToCorrect;

    void run() {
        while(1) {
            valentinePre.receiveWaiting(greetingsToCorrect);
            PRINTF("Charly gets message %s\n", greetingsToCorrect.msg);
            greetingsToCorrect.msg[0] = 'I';
            waitUntil(NOW() + 1*SECONDS);
            valentine.publish(greetingsToCorrect);
        }
    }
} charly;

```

