Lab Session 4: Review and Distributed Systems
21653-Sistemes Operatius

1 Introduction

First you are going to program some exercises that you will have to deliver in paper (it can be this very same paper printed). In the second hour we are going to test ROS a distributed computing architecture used in robotics.

2 Synchronization review: Bank deposit

Look and test the provided code of the bank account deposit. The code starts a bank with N=4 Accounts, 100 threads and does 10000 deposits of 1 euro per thread to random accounts. Thus the resulting bank total amount should be: 100*10000.

```
#define N 4
#define NThreads 100
#define NDeposits 10000
Account bank[N];
```

Now look at the code in the deposit function:

```
void deposit(Account* acc, double amount) {
 /* AFEGIR codi de sincronitzacio d'entrada */

 // Deposit del thread
 usleep(10);
 acc->balance += amount;

 /* AFEGIR codi de synchronitzacio de sortida */
}
```

To deliver:

- Compile the code bankDeposit.c and report what happens.

- Write a solution using the lock of the account and report time

- Write a solution using monitors: the condition variable and report time and compare it with previous solution and justify. Remember when we do the actual deposit we must be free of lock.

3 Threads review: Barber example

Compile and test the code barber.c a solution to the barbers problem with monitors.

To deliver:

- Make every customer write its id. Explain how you solve it, and write the parts of the code that you added or modified.

- Make the barber know which customer it is serving by using a shared variable. Explain how you solve it, and write the parts of the code that you added.

4 ROS

We are going to follow a tutorial to understand the basic mechanisms avialable in ROS. We are going to follow the tutorial `http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics`. Open roscore in one terminal:

```
roscore
```

Open the turtlesim node in another terminal:

```
rosrun turtlesim turtlesim_node
```

And the tele-operation node in another one:

```
rosrun turtlesim turtle_teleop_key
```

Now let's look on how messages are published by opening another terminal;looking at the topic. Whenever a key is pressed in the teleop node appears:

```
rostopic echo /turtle1/cmd_vel
```

Now we are going to publish a message at a certain frequency so that we can control the turtle:

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

An explanation of the parameters of the previous call follows:

- "pub" command will publish messages to a given topic.
- "-1" option (dash-one) causes rostopic to only publish one message then exit
- "/turtle1/cmd_vel" This is the name of the topic to publish to
- "geometry_msgs/Twist" This is the message type to use when publishing the topic
- "–" This option (double-dash) tells the option parser that none of the following arguments is an option. This is required in cases where your arguments have a leading dash -, like negative numbers.
- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' a geometry_msgs/Twist msg has two vectors of three floating point elements each: linear and angular. In this case, '[2.0, 0.0, 0.0]' becomes the linear value with x=2.0, y=0.0, and z=0.0, and '[0.0, 0.0, 1.8]' is the angular value with x=0.0, y=0.0, and z=1.8. These arguments are actually in YAML syntax