TC2006 Programming Languages

Assignment 6: Distributed Programming in Erlang

(* Teams of 2 or 3 members *)

This assignment will give you the opportunity to practice and become familiar with the Erlang programming language. Individual works will **NOT be accepted.** Any situation in this regard must be negotiated as soon as possible with the professor.

Internally, through Erlang comments, you must include into the system's source file your student ids and names, the description of the interface functions for the members and the store, the creation and communication protocols of the processes, and the auxiliary functions. You will be heavily **penalized** for NOT properly documenting your code.

SYSTEM OPERATION

You must program a prototype of a **distributed purchasing system** that involves the following 3 types of entities:

- **Partners** – Buyers who signed up by paying for store membership.
- **Products** – items, registered in the store, that can be purchased by the partners (**slave processes**).
- **Store** – Server that manages partner subscriptions and product sales (**master process**).

Buyers must subscribe to the store as **partners** by paying a membership. To subscribe, they must use the **subscribe_partner(Partner)** command, where Partner must be an atom (name) that uniquely identifies the partner so that he can buy products. If a partner with that name already exists, you will need to try again with another name. A subscription can be removed using the **delete_partner(Partner)** command.

Only partners can order products from the store using the **create_order(Partner,ProductList)** command, where the list of products must be a list of tuples identifying ***{Product,QuantityOrdered}***. The store will generate an order number and respond with the order list adjusted according to stock. The customer can request a list of products in stock using the command **stock_list(ProductList)** .

Products in stock must be registered in the store using the **register_product(Product,Quantity)** command**, where** Product is an atom that uniquely identifies the product, while Quantity must be an integer greater than or equal to 0. Product registration will create a process that will control the amount of product in stock. You can remove a product, by

removing its process, with the command **remove_product(Product).** The quantity of a product can be increased or decreased using the **modify_stock(Product,Quantity)** command, where the increase or decrease is determined by the sign of the quantity. It should not be reduced if the quantity exceeds the stock. All these tasks are performed through the Store process.

The **store** should be managed as a process that concentrates the information of subscribed partners, products, and orders. Of the partners you should only keep a list with names. Of the products you should only save the list of names and PIDs of those that have a live process. And of the orders you must keep the counter to control the order # and a list with the historical record of the total of each product sold (name and quantity).

The store process is created with the **open_store()** command and stopped with the **close_store()** command. The store process should be registered with the **store** name so that partners and product registrants do not need to know its PID. This process must implement the communication protocols with the partners for the orders and with the products for their creation, modification of stocks and termination. In addition, it must be possible to display the list of partners with the command **list_partners()** and the list of total of each product sold with the command **sold_products()**.

It is worth mentioning that if the store process terminates, all product processes should automatically terminate as well.

As it is a prototype of a distributed system, you must include displaying code that clearly and in full detail indicates what is happening into each distributed node of the system. Particularly, when sending and receiving messages. For example, when a shopper makes a subscription request, the shopper should display, in its node of request, something like "*Partner <name> requests subscription*", before pausing to wait for the store's response, and after a successful subscription display "*Suscription accepted*". Also, when the store process receives a subscription request from a buyer, it must display in its own node "*Subscription request from <name>*", before attending said request.

Finally, **you MUST** include within the program itself, by means of Erlang comments, information that describes the steps to test your system and a script to show the execution of all the features that you successfully programmed into your distributed system. This must include all the details to create the nodes and processes, as well as use all your interface functions, so that all the programmed facilities are illustrated. It will be **penalized strongly** NOT to document the detailed use of your system.