Distributed Systems
Bachelor In Informatics Engineering
Universidad Carlos III de Madrid
Midterm 2011/2012

Exercise 1. Answer the following questions briefly, justifying your answer.

a) State the main features of distributed systems.

· Heterogeneity is the variety and difference in the components that make up a distributed system. For example, networks, computer architectures, the ordering of bytes, etc.
· Naming is the feature that allows the identification of objects, computers want to share resources.
· Communication and synchronization has to do with how to make the communication between processes and how they are to be synchronized (blocking or non-blocking communication primitives).
· Scalability or growth capacity, is the ability to preserve system capacity when significantly increases the number of users or resources.
· Transparency is hiding the user of the components that make up the distributed system. Transparency can be given in different.
· Reliability is the probability that a system develops a particular function under prescribed conditions and a period of time. Reliability has several aspects: consistency, security and handling faults.
· Quality of service is the ability to meet the time requirements and process when transmitting multimedia data streams in real time.
· Structure of software represents the architecture of the distributed system. There are several possibilities: network operating system, distributed operating system and middleware.

b) Describe briefly the communication paradigms.

· Message passing, on the lowest level of abstraction processes communicate by message passing. The basic primitives are send (message, destination) and receive (message, origin).
· Client-server applications where two processes with distinct roles (client and server) will communicate to share resources that reside on the server. There is an active participant (client) that is initiating the communication and a passive (server) that expects service requests from customers.
· Remote Procedure Calls. This communication paradigm allows invocation semantics of local procedures distributed applications. When a process requests a remote procedure is invoked client stub that masks communication with the process server and sends the request to the arguments. In the process server, the server stub reads the request, remote procedure calls locally and responds to the client with the response.

· Peer-to-peer. In this communication paradigm, unlike the client-server paradigm, all processes have the same role. A typical example of such applications is multimedia file sharing.
· Distributed objects is based on remote procedure calls. This paradigm extends the concept of remote procedure remote object, taking into account the programming languages, object-oriented.
· Mobile agents. In this communication paradigm no message exchange but a binary file is exchanged.

· Network Services. In this communication paradigm gets a reference to the server to access a particular resource. This reference is provided by a directory service that is responsible for recording the various network services.
· Collaborative or group communication is a communication paradigm for creating collaborative work sessions involving a set of processes.

c) Reason briefly the following sentence: "The client-server computing is not only distributed access distributed computing".

In a distributed algorithm processes work together to achieve a common goal. Each of these processes for a certain task partially used to obtain the final result to be achieved. The cooperation of each and every one of these processes leads to the result that can be achieved. In the case of client-server applications, the server offers a particular service (eg a share) but usually there is a distributed algorithm running different processes for the server response.

e) Indicate what types of servers we count.

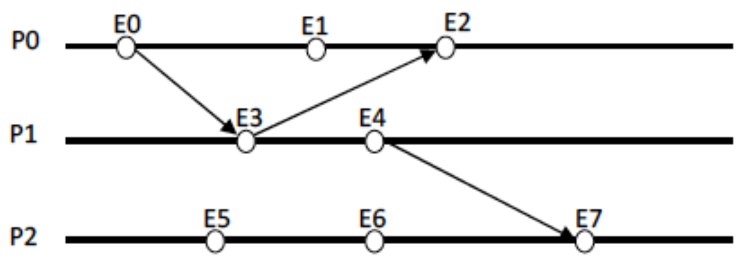
The servers can be classified from different points of view.
1. Depending on the number of requests that can attend simultaneously speak sequential server when a request can only be seen at the same point in time, and concurrent servers when multiple requests could be processed at the same instant of time.
2. Depending on whether or not server stores information about the service request from a client, we talk about stateless servers when the server does not store any kind of information requests and therefore each is treated independently, and stateful servers, those which store customer information. The status information could be divided into two types: global (shared by all clients) and information request (customer-specific).
3. Depending on whether or not has established a connection before sending data to the server, server-oriented those for connection to be established a connection between the client and server before you can send or receive data, and servers connection-oriented not those not requiring the connection phase.

f) What is a name service and what it is used? Indicate which provides generic services nameserver.

A name service maps hostnames to IP addresses on the Internet. Users generally operate while hostnames IP applications handle. The name server provides several services:
· Register name: it allows servers register the service address.

· Solve the name or look for, giving the address of a service.
· Remove or delete a name service.

Exercise 2. Consider the processes P1, P2 and P3 running in a distributed system. These processes generate events marked in the figure below.



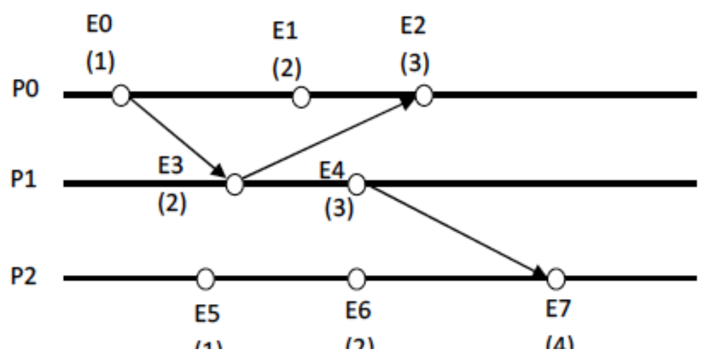
a) Extract potential causal relationships between events shown in the figure.

E0 ->E1
E1 -> E2
E0 -> E2 (transitivity)
E0 -> E4 (transitivity)
E3 -> E4
E5 -> E6
E6 _ E7
E5 -> E7 (transitivity)
E0 -> E3
E3 -> E2
E4 -> E7
E3 -> E7
E0 ->E7

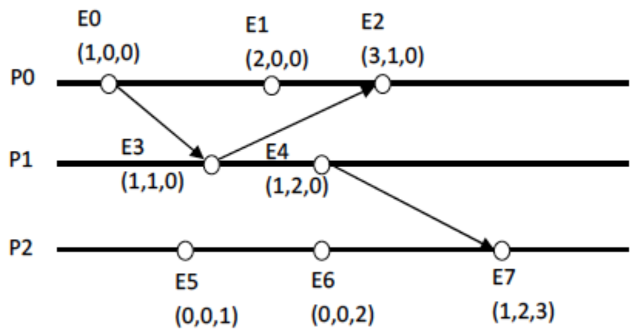
b) What events can not extract an order relation and why?

E1 || E3
E1 || E4
E2 || E4
E3 || E5
E3 || E6
E4 || E5
E4 || E6

c) Using Lamport logical clocks indicate the time stamps for the events of the previous processes.



d) Using vector clocks, set the time stamps for the events of the previous processes.



Exercise 3. We want to develop a distributed application that allows to manage the marketing of movie tickets through a service called CINEPOLIS. This service lets you manage several cinemas entries. Customers can use the service to buy and cancel tickets. They can also use this service to know the next time and date to which to project a certain movie in a particular film. He asks:

a) Design the previous client-server application using sockets, indicating and specifying all aspects necessary for your design.

In a client-server application need to consider at least the following design aspects:
1. To design a client-server application we need to consider common issues of distributed applications:

- Names: need to know how to identify the machine where the application runs. To do this, we use static IP addresses and assume that the client knows the IP (or name) of the server machine.

- Scalability: to provide a scalable service (to remain effective while the number of requests increases) we design a concurrent server, ie that that can simultaneously address multiple service requests simultaneously. The concurrent server could be implemented with conventional processes or processes light, the latter being the most efficient option. Since we consider a server concurrent need to protect those shared variables that could lead to race conditions and thus lead to incorrect results in the application (aspects concurrency and synchronization).

- Heterogeneity: as client and server machines may have different hardware architectures for sending data to the network, we must be translated into standard network (network byte order or big endian). Similarly, in receiving network data should passed to the host format.

2. The transport protocol to use will depend on the reliability requirements of the application. Transport protocols are implemented over TCP/IP and UDP. Both applications (client and server) must agree on the transport protocol use. In particular, for this application, possibly send the payment information of the movie tickets. This information is critical and it is important that messages reach their destination with some reliability (lossless, sorted, without duplicates). Thus, we select TCP/IP as the transport protocol to use in the application. No need implement QoS aspects.

3. The service protocol
The service protocol defines the exchange of messages between the client application and furthermore the server message format.

For each service (buy and cancel entries) we define two messages: the first of them sends the client to the server, and contains the request for the purchase (or cancellation) of entries, the second of which is sent by the server to the client with the response.

There are three services to be provided by the server:
· Buy Tickets
· Cancel
· Next pass

There are therefore three types of requests that can be ordered. For each request will establish a connection, it will send the arguments of the request and will receive the result. Then closes the connection.

To identify each request, you can use a byte that is sent from client to server to establish the connection. You can use 0 to buy inputs, 1 input and 2 to cancel to know the next pass of a movie.

For tickets, you must then send the following information:
· Film ID: 32-bit integer in big endian format.
· Name of the film: character string up to a certain maximum. Either sends an integer in big endian format indicating the number of characters in the film and then the name of the movie.
· Date: You can use different formats, eg day / month / year, day one byte, another month and year an integer in big endian format.
· Number of entries: one byte can be used for this field.
· Card payment: string with the card number to use stop buy.
· Holder of the SD: Character string holder (of a certain size).
· Expiry date: month / year

The response message to this request would be:
· Code Information: integer in big endian format.

· Number of room: one byte
· Lisa of seats: a list with the number of reserved seats (each an integer
big endian format).

It can be considered that if the sales code is negative, then the purchase could not be perform. The request for cancellation of tickets sales will contain the code that will be sent after send the request code (a 1). The answer in this case may be a byte indicating failure or success. For the next pass is necessary to send the name of the movie, the identifier cinema. It will return a time and date for the next pass. For the time can use a formatted string format: HH: MM: SS. To dates, we can send a byte for the day, one byte for the month (1 January 12 December), for the year big endian. Sending the number format month, allows the client may be adapted to different languages.

4. Options on the server:
The server will be given connection-oriented transport protocol that we will use TCP. This implies that there must be a connection establishment between the client and server prior to the exchange of data.

In addition, the server is stateful as we maintain the global information
service requests from customers. For example, the server must check that the inputs for a film have been exhausted, and return the appropriate error if you try to buy more tickets for that movie. Also we keep buying codes to perform a subsequent cancellation.

Once the service protocol and design aspects to consider, the functions
perform in the client application and the servant are: functions on the client: connect to server using sockets, send the request and wait for the answer.

Functions on the server: waiting connection requests customer service, calculating the result and return the response.

b) According to the previous design, specify which calls to the socket library used on the client and on the server and in what order.

Here are the calls to the sockets library:

On the client:
1. int socket (int domain, int type, int protocol)
where domain is AF_INET, type

2. int connect (int socket, struct sockaddr * dir, int long)

where socket is the socket returned by socket, dir is the remote socket address long is the length of the address. This primitive provides the connection to the server.

3. int write (int sd, char * buffer, long int);

where sd is the socket returned by socket, buffer is a pointer to the data to send and long is the data size.

4. int read (int sd, char * buffer, long int);
where sd is the socket returned by socket, buffer is a pointer to the data to get long and the size of the data to receive.

5. int close (int sd)

On the server:

1. int socket (int domain, int type, int protocol)
where domain is AF_INET, SOCK_STREAM type, protocol is 0 (choose the OS).

2. int bind (int sd, struct sockaddr * dir, int long)
where sd is the socket returned by socket, dir is the direction of the socket and shank is the size of the address. Enables socket to receive incoming connections.

3. int listen (int sd, int baklog)
where sd is the socket returned by socket, and backlog may be queued before the server makes an (SOCK_STREAM, protocol is 0 (choose the OS.) is the number of requests that are accepted.

4. int accept (int sd, struct sockaddr * dir, int * length)
where sd is the socket returned by socket, dir is the address of the customer who placed connect and long is the size of the address.

6. int read (int sd, char * buffer, long int);
where sd is the socket returned by socket, buffer is a pointer to the data to be received and is long the size of data to receive.

7. int write (int sd, char * buffer, long int);
where sd is the socket returned by socket, buffer is a pointer to the data to send and is long the size of the data.

8. int close (int sd)
where sd is the socket returned by socket.

c) Whereas a used RPC interface definition language similar to C language syntax, define the interface for the service described. Indicate which parameters are input and which are output.

```
#define MAX_ACCOUNT 20
#define MAX_DATE 10
#define MAX_TIME 8
#define MAX_FILM 1000
#define MAX_SEATS 10
struct req_buy{
     int id_cine;
     char film[MAX_FILM];
     char date[MAX_DATE];
     int num_tickets;
     char bank_account[MAX_ACCOUNT];
};

struct res_buy{
    int buy_id;
    int id_room;
    int list_seats[MAX_SEATS];
};
struct req_cancel{
   int buy_id;
};

struct next {
      char date[MAX_DATE];
      char time[MAX_TIME];
};

struct req_buy buy_tickets(struct req_buy request);
int cancel_tickets(struct req_cancel request);
struct next get_next(char *film, int id_cinema);
```

d) Specify the same interface using Sun's RPC.

```
const MAX_DATE=10;
const MAX_ACCOUNT=20;
const MAX_FILM=100;
const MAX_SEATS=10;

struct req_buy{
   int id_cinema;
   string film[MAX_FILM];
   string date[MAX_DATE];
   int num_tickets;
   string bank_account [MAX_ACCOUNT];
};

struct req_cancel{
   int buy_id;
};

struct res_buy{
   int id_client;
   int id_room;
   int list_seats [MAX_SEATS];
};

struct res_next {
   char date[MAX_DATE];
   char time[MAX_DATE];
};

program CINEPOLIS_PRGM {
   version CINEPOLIS_VER {
      res_buy buy_tickets(struct req_buy) = 1;
      int cancel_tickets(struct req_cancelar) = 2;
      struct res_next get_nextfilm (char film<>, int id_cinema) = 3;
   }= 2;
} = 100000;
```

e) When using the Sun RPC, indicate what steps you need to perform the RPC client to invoke a remote procedure.

To invoke a remote procedure RPC, the client must:
1. - Get a handle on the client by **clnt_create.** This handler allows identify the remote service, version and transport protocol used in communication.
2. - Invoke the remote procedure as if it were a local procedure.
3. - Destroy the handler once the customer does not want to continue invoking procedures, by calling **clnt_destroy**.

f) Indicate what tasks they have to perform the client stub and server.

The client stub must be provided for each remote procedure:

1) Remote Service Location
2) Pack a message with arguments (marshalling)
3) Send the message
4) Get server response
5) Unpack the answer (unmarshalling)
6) Return Customer

The server stub must:

1) Register the remote service
2) Receive remote procedure request
3) Unpack the request (unmarshalling)
4) Invoke the procedure locally
5) Get the response procedure
6) Package response (marshalling)
7) Send the answer