UNIVERSIDAD CARLOS III DE MADRID
BACHERLOR IN INFORMATICS ENGIEERING. DISTRIBUTED SYSTEMS
Ordinary Call. May 22th, 2015.

To conduct this exam will be available **2 hours**. Books, notes or calculators are not allowed.

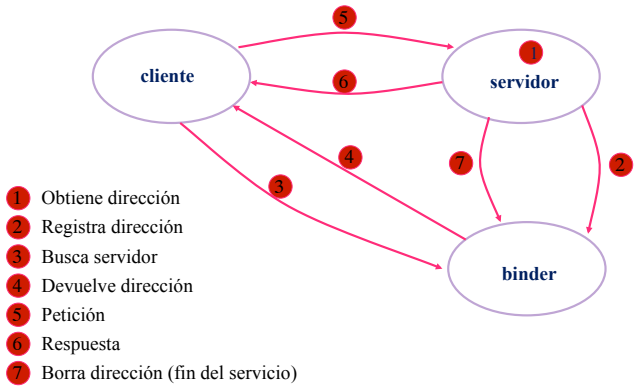
Name: _____ **Group:** 89

Exercise 1 (3 points). Response the following questions, justifying your answers:

a) Describe which information is included in the WSDL.

- **Types:**
- **Messages:**
- **PortTypes**
- **Bindings:**.
- **Services:**

b) Graphically represent the registration and access process to a remote service using SUN' RPC.



c) Describe the features of the stub code in remote procedure calls.

Stub code is responsible for hiding the complexity in the process of communication between a client and a server process RPC process. Stub codes perform tasks such as flattening and representation of data and low level management using sockets.

d) Given the following distributed system consisting of N nodes, N = 6, using dynamic voting to maintain consistency of the replicas. Nodes have the following values NV and SC:

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
NV	5	5	5	6	6	6
SC	6	6	6	3	3	3

Could you update the replica on the partition {4,5,6}? Fill the following table properly and justify the answer

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
NV	5	5	5	7	7	7
SC	6	6	6	3	3	3

No se puede actualizar la replica de la partición {4,5,6} porque no pertenece a una partición mayoritaria.

M = max{NV} = 6

I = {4,5,6}

N = max{SCi} = 3

Si |I| > N/2 -> Si 3 > 3/2 se puede actualizar la partición. Por tanto sí se puede actualizar.

e) Which is the portmapper service and what is its functionality?

The portmapper process allows RPC SUN to register and access the listening port of a service running on the same machin. The portmapper service registers both the identifier of the program, version, listening port and protocol.

f) Explain the bully algorithm and define the messages that are needed to perform the algorithm. Propose two scenarios where this algorithm may be useful.

The bully algorithm is a coordination mechanism of election. The aim is to choose a coordinator within a set of nodes. When the current coordinator is not responding messages, a new election process starts. The algorthim relies on three kind of messages: ELECTION COORDINATOR, OK.

This algorithm can be used in the following manner in distributed systems. First, it can be used to define the process of coordinatior among a set of web servers. Second, to define the coordinator on a centralized distributed synchronization mechanism.

Exercise 2 (7 points). We want to develop an application for an online subscription video system, namely NetVideo. The system consists of two basic functions: subscription service and streaming video service. The video subscription system allows registered users to access multimedia content. For this purpose, we count with the following remote procedure calls:

- `int login (char *username, char *password, int *token)`: This procedure allows users to access the remote client authentication system. For this, two arguments will be passed: the user name and password. If the user does not exists 0 is returned, otherwise an authentication token is returned. The token is an integer number in the range of 10,000 to 60,000.
- `int getAccess (int token, char* content, long *size, int *port)`: This remote procedure is used to determine whether the requested content is accessible to the user. If it is possible, the call returns the value 1. The last parameter is the returned port given that an on-demand service is created for that specific user.

On the other hand, once granted access to contents, customers can access the visualization of the requested multimedia content. Customers have only direct access to the port indicated by the `getAccess` call. Customers must access information depending on the available bandwidth and server saturation. For this, the following service is defined:

- `int downloadVideo (int token, char* buffer, int offset, int currBlockSize, int *nextBlockSize)`: This call permits to read `currBlockSize` bytes from the server and stored in a memory buffer. The `offset` parameter defines the offset of the acceded block. In addition, the server will inform costumers of the available length for the next request by using the `nextBlockSize` parameter. Consider an initial block size of 1 Kbytes.

Subscription and streaming services will run in the same memory space.

Considering that we want to develop the application using sockets, we are asked:

a) Represent the previous three services using the SUN's XDR.
b) Make a detailed application design. Take into account the characteristics of each of the services offered in order to maximize performance in the video streaming service.
c) Implement the following services using the C socket API:
    a. Implement the `login` service on the client side.
    b. Implement `downloadVideo` service on the client side.
    c. To improve network performance, it is proposed to make a parallel implementation using the service implemented `downloadVideo` in b), using 4 threads. Each thread will access a portion of the video. Suggest a design and implement this solution.

a)

```
struct login_res{
        int código;
        int token
    };

    struct acceso_res{
        int código;
        int tamaño;
        short puerto;
```

```
};


struct bloque_res{

        int código;

        char buffer<>;

        int nextBlockTam;

};

program NetVideo {

        version NetVideo_V1{

           login_res login (char usuario<>, char contraseña<>) = 1;

           acceso_res solicitar_acceso (int token, char contenido<>) = 2;

            bloque_res descargaBloqueVideo (int token, int offset,

                                            int actBloqueTam)= 3;

        } = 1;

} = 10005;
```

b) The system consists of two services: one subscription and other video streaming. The subscription system and will be processed by TCP, and personal information will be transmitted, and users can be connected over long distances. Moreover, the download service run by UDP as performance is critical and the order of the packets is not important.

The system consists of 2 clients, who will invoke the services described in the statement, and two servers, one for login and access and another for launch the on-demand streaming subscription service.

The servers are sesión-based given that token must be stored at each of the service users.

The subscription server is multithreaded and able to support multiple concurrent clients. The streaming server will initially monothread as only a client will access the service.

Both services is located on the same computer, sharing the IP. The subscription service will use port 8080. The streaming service will automaticly reserve a ramdon port number.

Messages exchanged for services are:

```
#define MAX_STRING 50

Mensahe login_pet
       char username[MAX_STRING]
       char password[MAX_STRING]

Size = 100 bytes

Message access_pet
       int token
       char content[MAX_STRING]

Size = 54 bytes

Mensaje download_pet
       int token
       int offset
        int actBloqueTam
```

```
Size login_res
       int result

Size = 4 bytes

Message access_res
       int port
       long size
       int result
Size = 12 bytes

Mensaje download_res
       int result
       int nextBlock
```

Size = 8 bytes + content

Size = 12 bytes

c) 1. Login

```
#define MSG_P_SIZE 100
#define MSG_R_SIZE 8
int login (char *username, char *password, int *token) {

            int sockd;

struct sockaddr_in serv_name;
struct hostent *hp;
int err;
char  msg_p[MSG_P_SIZE];
char  msg_r[MSG_R_SIZE];

int sresponse, int stoken, response;
            /* socket socket creation */

sockd = socket(AF_INET, SOCK_STREAM, 0);
if (sockd == -1){
perror("Error in socket creation");
exit(1);
}

/* server address*/
bzero((char *)&serv_name, sizeof(serv_name));
hp = gethostbyname ("netvideo.com");

memcpy (&(serv_name.sin_addr), hp->h_addr, hp->h_length);
serv_name.sin_family = AF_INET;

serv_name.sin_port = htons(8080);
/* connection process */

status = connect(sockd, (struct sockaddr*)&serv_name, sizeof(serv_name));
if (err == -1)
{
perror("Error");
return(-1);
}

bzero (msg_p, MSG_P_SIZE);

memcpy(msg,username,MAX_STRING);
memcpy(msg + 50,password,MAX_STRING);


if ( write(sockd, msg, MSG_P_SIZE) < 0) {
perror("Error");
close(sockd);
return (-1);
}

if ( read(sockd, msg_r, MSG_R_SIZE) < 0) {
perror("Error");
close(sockd);
return (-1);
}

close(sockd);

       scanf(msg_r,"%d%d",&response,&stoken);

       sresponse = ntohs(response);
        stoken = ntohs(stoken);

if (sreponse == 0) {
```

```
        return 0;
} else {
        *token = stoken;
        return sresponse;
}
}
```

2. descargarBloqueVideo

Vamos a asumir que el puerto que devuelve la llamada solicitar_acceso se encuentra almacenada en la variable global:

```
        short puerto;

        #define MSG_P_SIZE 12
        #define MSG_R_SIZE 8

        int descargaBloqueVideo (int token, int offset, char* buffer,
                        int actBloqueTam, int *nextBlockTam) {


                struct sockaddr_in server_addr, client_addr;
                struct hostent *hp;
                ssize_t recv;

                char  msg_p[MSG_P_SIZE];
                char  msg_r[MSG_R_SIZE];

                int result;
                int nextBlock;


                s =  socket(AF_INET, SOCK_DGRAM, 0);
                hp = gethostbyname ("netvideo.com");
                bzero((char *)&server_addr, sizeof(server_addr));

                memcpy (&(server_addr.sin_addr), hp->h_addr, hp->h_length);
                server_addr.sin_family = AF_INET;
                server_addr.sin_port = htons(puerto);

                bzero((char *)&client_addr, sizeof(client_addr));
                client_addr.sin_family = AF_INET;
                client_addr.sin_addr.s_addr = INADDR_ANY;
                client_addr.sin_port = htons(0);

                bind (s, (struct sockaddr *)&client_addr, sizeof(client_addr));

                bzero (msg_p, MSG_P_SIZE);
                sprintf(msg_p,"%d%d%d", htons(token), htons (offset), htons(actBloqueTam));

                sendto(s, msg_p, MSG_P_SIZE, 0,
                (struct sockaddr *) &server_addr, sizeof(server_addr));


                recvfrom(s, msg_r, sizeof(int), 0, NULL, NULL);

                scanf(msg_r,"%d%d",&result,&nextBlock);

                result = ntohs(result);
                *nextBlockTam = ntohs(nextBlock);

                trecv = 0;
                while (trecv < actBloqueTam) {
                        recv = recvfrom(s, buffer + trecv, pend, 0, NULL, NULL);
                        trecv += recv;
                }
```

```
            close(s);

            return result;
      }
```

3. The solution will be to divide the content into four contiguous blocks in order to parallelize downloading content. For this, you can use a distribution of this style, so that each thread download the next block to reproduce:

Content

T0	T1	T2	T3	T0	….

It is considered that the block size does not vary during the transmission.

```
pthread_mutex_t mutex__no_copia;
int nocopy = TRUE;  /* TRUE con valor a 1 */
pthread_cond_t cond_mensaje;

#define SIZE_BLOCK 1024
#define NUM_THREADS 4

struct request {
      int id;
      char *buffer;
      int size;
      int token;
}

void tratar_bloques(struct request *p) {
      int j;
      int sig;
      struct request tp;
      /* el thread copia el mensaje a un mensaje local */
      pthread_mutex_lock(&mutex_mensaje);
      memcpy(&tp, p, sizeof(struct request));
      /* ya se puede despertar al servidor*/
      nocopy = FALSE;            /* FALSE con valor 0 */
      pthread_cond_signal(&cond_mensaje);
      pthread_mutex_unlock(&mutex_mensaje);

      for (j = 0; j < tp.size / (4 * SIZE_BLOCK * NUM_THREADS); j++) {
          int offset;

          offset =  (j * NUM_THREADS * SIZE_BLOCK) + (tp.id * SIZE_BLOCK);
    descargaBloqueVideo (tp.token, offset , tp.buffer + offset, SIZE_BLOCK, &sig);
}

pthread_exit(0);
}



int paralleldownload(int token, char * buffer, int size)
{
      int i =0;
      pthread thid;
struct mq_attr q_attr;
      pthread_attr_t t_attr;

pthread_mutex_init(&mutex_mensaje, NULL);
      pthread_cond_init(&cond_mensaje, NULL);
      pthread_attr_init(&attr);

      for (i = 0; i < NUM_HILOS; i++) {
            struct request p;
```

```
            p.id = i;
            p.size = size;
            p.buffer = buffer;
            p.token = token;
            pthread_create(&thid, &attr, tratar_bloques, p);
while (nocopy)
                  pthread_cond_wait(&cond_mensaje, &mutex_mensaje);
            nocopy = TRUE;
            pthread_mutex_unlock(&mutex_mensaje);

}

}
```