UNIVERSIDAD CARLOS III DE MADRID
BACHERLOR IN INFORMATICS ENGIEERING. DISTRIBUTED SYSTEMS
Extraordinary Call. June 16th, 2014.

To conduct this exam will be available **3 hours**. Books, notes or calculators are not allowed.

Student: ____ **Group:** __________

Exercise 1 (3 points). Each correctly answer has 0.2 points, every question answered incorrectly subtracts 0.066 points, and empty answers do not count.

1. Which of the following calls allows a server to know the IP address of a client?
   - A) bind()
   - B) connect()
   - C) accept()
   - D) getpeername()

2. The function `read()` applied to a datagram socket, which of the following statements is true?
   - A) It cannot be applied to datagram sockets.
   - B) It is a service for use with files, so you cannot use with sockets.
   - C) It always returns the number of bytes specified in the argument.
   - D) It could be used in datagram sockets.

3. Regarding the sockets, which of the following statements is true?
   - A) A UDP socket cannot send any message size.
   - B) A TCP socket cannot send any message size.
   - C) A socket created in Java is not allowed to send data to a socket created in an application written in C language.
   - D) A datagram socket cannot be used in concurrent servers.

4. What is false about NFS v3?
   - A) The write policies are write-delayed or write-through.
   - B) You can negotiate the file's block size.
   - C) We count with one I/O server.
   - D) Ensure consistency between the information stored in the client cache and server.

5. In a voting system comprises N = 8 nodes, where R = 4 and W = 5. The cost of the readings is 1C and cost of writes is 3C. If the probability of reading the system is 0.25, which is its cost?
   - **A) 12.25**
   - B) 10.5
   - C) 4.5
   - D) The quorum R=4 and W=5 is invalid.

6. In POSIX message queues:
   - A) The data flow of data is unidirectional.
   - B) They are synchronization mechanism.
   - C) They have a direct naming mechanism.
   - D) They provide a synchronous sending mechanism.

7. Consider a system with five processes (0 to 4). If node 4 is fallen and the node starts the Bully algorithm is the number 2, what is the number of messages exchanged to decide the next coordinator.
   - A) 3
   - B) 5
   - **C) 7**
   - D) 9

8. In a reliable multicast:

A) There is no guarantee that the message is delivered to all nodes.
B) It ensures that messages are delivered in accordance with the causal relationships.
C) The protocol ensures that all group members will receive messages from different nodes in the same order.
D) All running nodes receive the message.

9. The `gethostbyname` function:
A) Gets the hostname and port of the server.
B) Allows resolving the hostname, returning the address in domain-point format.
C) Allows resolving the hostname, returning the address in decimal-point format.
D) Allows resolving the hostname, returning the address in big-endian format.

10. A stateful server:
A) Consume fewer system resources.
B) Use shorter request messages.
C) Increase the fault tolerance.
D) All messages are idempotent.

11. Indicate which of the following statements is NOT correct. Using distributed cache allows applications to:
A) Reduce network traffic, in particular, the number of requests from clients to servers.
B) Improve scalability.
C) Improve co-use of data across multiple clients.
D) Reduce the number of disk accesses, keeping in the cache the latest client request.

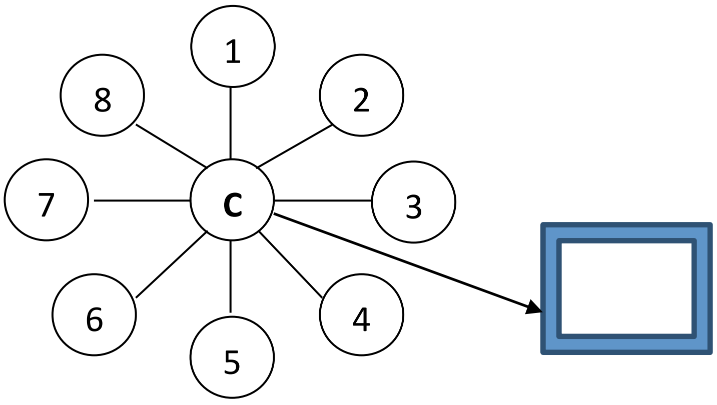
12. Which of the following statement is correct about the differences between grid and cloud computing paradigms?
A) Only grid computing offers heterogeneous resources.
B) Cloud computing is suitable for undetermined computation peaks.
C) Remote services could be deployed only in grid computing.
D) Cloud computing lies on volunteer computing.

13. In the SUN RPC:
A) In the server implementation, it is required to define the type of network protocol used.
B) The client stub does not change when you change the server implementation.
C) The binder is responsible for sending data from server to client and vice versa.
D) The messages are encoded in XML through the XDR interface.

14. Given A and B be two events that occur in a distributed system. If the event A is marked with Lamport' logical value 3 and the event B the value 4. Which of the following statements is true?
A) The event A always precedes B.
B) The event A precedes B only if they occur in different processes.
C) The events A and B may be concurrent.
D) The events A and B can never be concurrent.

15. Calculate the time to transmit a 500 KBytes file from a client to a server, assuming that packets sent have a maximum length of 2 KBytes, the latency is 1 ms, and a bandwidth of 1000 MBytes/s.
A) 200.5
B) 250.5
C) 500.5
D) 125.5

Exercise 2 (2 points). Consider a system consisting of N processes where the first process, coordinator process (process C in the figure) serves as a writer on the screen and the other processes (numbered 1 to 9) write requests asking the coordinator process. The system architecture for N = 9 is:



The coordinator process must block until a process requests for writing a character. The other processes must request for writing to the coordinator in order (specified by the characters 'a' to 'z') until they have completed a total of M = 100 requests. The write request is made character by character. When the requesting process ends, the application should print on the screen the number of characters requested for writing for each worker. Implement the functions `writer` (for the coordinator process) and `worker` (for other processes) in the C programming language, in order to obtain the following sequence on screen:

```
coordinator....
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuv
coordinator end=100
END...Thread=6 Characters written = 12
END...Thread=1 Characters written = 12
END...Thread=0 Characters written = 12
END...Thread=7 Characters written = 12
END...Thread=5 Characters written = 13
END...Thread=4 Characters written = 13
END...Thread=3 Characters written = 13
END...Thread=2 Characters written = 13
```

Solución:

Este problema admite varias soluciones. Sólo se pide el código de la función escritor y trabajadores. A continuación se proporciona el código completo.

```
#define         N       8
#define         M       100

/* VARIABLES GLOBALES */
int caracter='a';
int cont_threads=0;
int copiado=0;
int fin=0;
pthread_mutex_t mutex;
pthread_mutex_t mutex2;
pthread_cond_t cond_copiado;
pthread_cond_t cond_escribe;
pthread_cond_t cond_caracter;
pthread_cond_t cond_fin;

/* PROTOTIPOS DE FUNCIONES */
int escritor();
int trabajador(int *id);

/* IMPLEMENTACION DE FUNCIONES */
int escritor(){
        int i=0;

        pthread_mutex_lock(&mutex);
        pthread_cond_broadcast(&cond_escribe);
        pthread_mutex_unlock(&mutex);

        while(i<M){
                pthread_mutex_lock(&mutex2);
                while(!copiado)
                        pthread_cond_wait(&cond_escribe, &mutex2);
                copiado=0;
                printf("%c",caracter);
                pthread_cond_signal(&cond_caracter);
                pthread_mutex_unlock(&mutex2);
                i++;
        }
        printf("\ncoordinador termina=%d \n",i);
        pthread_mutex_lock(&mutex);
        fin=1;
        while(cont_threads<N){
                pthread_cond_signal(&cond_caracter);
                pthread_cond_wait(&cond_fin,&mutex);
        }
        pthread_mutex_unlock(&mutex);

}
int trabajador(int *id){
        int identifier;
        int cont_parcial=0;

        pthread_mutex_lock(&mutex);
        identifier=*id;
        copiado=1;
        pthread_cond_signal(&cond_copiado);
        pthread_mutex_unlock(&mutex);

        // Barrier
        pthread_mutex_lock(&mutex);
        pthread_cond_wait(&cond_escribe,&mutex);
        pthread_mutex_unlock(&mutex);

       caracter='a';

        while(1){
                pthread_mutex_lock(&mutex2);
                copiado=1;
                pthread_cond_signal(&cond_escribe);
                pthread_cond_wait(&cond_caracter, &mutex2);
                if (fin==1) {
                        cont_threads++;
                         printf("FIN...Thread=%d  Numero  de  caracteres  escritos=%d\n",  identifier,
       cont_parcial);
                        pthread_cond_signal(&cond_fin);
                        pthread_mutex_unlock(&mutex2);
                        pthread_exit(0);
                }
                caracter++;
                if (caracter=='z'+1){
                        caracter='a';
                }
                pthread_mutex_unlock(&mutex2);
                cont_parcial++;
        }
        pthread_exit(0);
}

int main(int argc, char* argv[]){
        pthread_t arrayTh[N];
        int i;
        pthread_attr_t attr;

        system("clear");
        pthread_mutex_init(&mutex, NULL);
        pthread_mutex_init(&mutex2, NULL);
        pthread_cond_init(&cond_copiado, NULL);
        pthread_cond_init(&cond_escribe, NULL);
        pthread_cond_init(&cond_caracter, NULL);
        pthread_cond_init(&cond_fin, NULL);
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);


        for(i=0;i<N;i++){
                pthread_create(&arrayTh[i],NULL,(void*)trabajador, &i);
                pthread_mutex_lock(&mutex);

                       while(!copiado)
                pthread_cond_wait(&cond_copiado, &mutex);
            copiado=0;
            pthread_mutex_unlock(&mutex);
        }

        printf("coordinador....\n");
        escritor();

        pthread_mutex_destroy(&mutex);
        pthread_mutex_destroy(&mutex2);
        pthread_cond_destroy(&cond_copiado);
        pthread_cond_destroy(&cond_escribe);
        pthread_cond_destroy(&cond_caracter);
        pthread_cond_destroy(&cond_fin);
        pthread_attr_destroy(&attr);
        exit(0);
}
```

Exercise 3 (1 point). Consider a XDR file (`remote.x`) that defines an interface of a server using the SUN RPC. Indicate which files are generated by the rpcgen compiler, which is the objective of each file, which files are used by the client and server, and whether it is necessary for the developer implement any additional file.

Solución:

El proceso de compilación con `rpcgen` genera, al menos, los siguientes archivos:

- `funciones_clnt.c`: que incluye el código del *stub* del cliente.
- `funciones_xdr.c`: incluye funciones para empaquetado y desempaquetado XDR.
- `funciones.h`: fichero de cabecera con los tipos y declaración de los prototipos de procedimientos en C.
- `funciones_svc.c:` incluye el código del *stub* del servidor.

Para poder implementar el cliente y el servidor es necesario que el programador implemente el código del cliente, por ejemplo `cliente.c` y el código del servidor, por ejemplo `servidor.c`

El ejecutable del cliente se obtiene a partir de los archivos: `funciones_clnt.c`, `funciones_xdr.c`, `funciones.h` y `cliente.c`
El ejecutable del servidor se obtiene a partir de los archivos: `funciones_svc.c`, `funciones_xdr.c`, `funciones.h` y `servidor.c`

Exercise 4 (3 points). It is desired to design and implement a messaging service for mobile devices called 3GChat. The messaging service allows communication between different devices via a wireless connection. The system is a centralized service, in which all messages pass through 3GChat headquarters. The application installed on mobile devices will use the following primitives:

- `send_message`: this operation sends a message to a 3GChat user, identified with an integer. The message has a maximum of 256 characters.
- `send_photo`: this operation sends a photo of variable size to a 3GChat user.
- `receive_message`: this operation receives a message of a 3GChat user, identified with an integer. The message has a maximum of 256 characters.
- `receive_photo`: this operation obtains a photograph of variable size. The user id of the users is also received.

On the other hand, mobile devices should be able to receive notifications from 3GChat headquarters, announcing that they have a content waiting. For this, it has the following call:

- notify: This operation notifies devices that a message or photo is pending for download. This call uses the user identifier and the type of content that is ready for download.

State:

a) Assuming that we will develop the service using sockets, design the application protocol, specifying: transport protocol used, server type, message types, message exchange sequence, and message format..
b) Implement a possible code using sockets, a program that sends and receives a photo (code client and server). Furthermore, it is important to implement and invoke `notify` function but is not necessary to implement the receipt of the notification. The implementation will be done according to the protocol designed in paragraph a).
c) Specify using SUN RPC, the XDR interfaces necessary for developing.

Solución:

a) Para el desarrollo del servicio se va a utilizar el protocolo TCP. Tanto los dispositivos móviles como la oficina central de 3GChat tendrán el rol de cliente y servidor. Por un lado el teléfono móvil es cliente porque inicia la comunicación con los servidores de 3GChat, y por otro lado, también es servidor porque espera las notificaciones del

   Los servidores se van a diseñar de forma que sean concurrente y realizando una conexión por petición. Según el enunciado se distinguen 4 tipos de peticiones entre cliente a servidor, que se corresponderán con mensajes de:

   1. enviar_mensaje: El cliente envía el identificador del destinatario y un mensaje de longitud fija de 256 caracteres. Como respuesta se obtiene un mensaje que indicará éxito o error.
   2. enviar_foto: el cliente envía el identificador del destinatario, el tamaño de en bytes de la fotografía y los datos de la fotografía. Como respuesta, el servidor envía un mensaje con un código de error.
   3. recibir_mensaje: el cliente envía su identificador usuario. El servidor devuelve un mensaje y el código que indica el resultado de la operación.
   4. recibir_foto: el cliente envía el identificador de su usuario. El servidor responde con un mensaje que incluye el resultado de la operación, la longitud de la fotografía y los datos de la fotografía.

Como formato de los mensajes se puede utilizar el siguiente:

- Identificador de usuario: un integer .
- mensaje: Cadena de caracteres con longitud máxima = 256 bytes.
- Longitud fotografía: long.
- Tipo de contenido: byte.
- Código de error: un byte. El valor 0 indica éxito, el -1 error.

Por último, en el caso de utilizar sockets, es necesario, especificar la dirección IP y el puerto donde ejecuta el servidor.

b) A continuación se indica un posible pseudocódigo para el cliente y servidor:

```
/* PARTE CLIENTE */


int foto(char* serv_nombre, int destino, char* foto , long foto_long)
{
  int sockd;
  struct sockaddr_in serv_name;
  struct hostent *hp;
  int err;
  byte peticion;
  /* 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 (argv[1]);
  memcpy (&(serv_name.sin_addr), hp->h_addr, hp->h_length);
  serv_name.sin_family = AF_INET;
  serv_name.sin_port = htons(1200);

  /* connection process */
  status = connect(sockd, (struct sockaddr*)&serv_name, sizeof(serv_name));
  if (err == -1)
  {
    perror("Error al connectar");
    exit(1);
  }

  peticion = 0;
  if ( write(sockd, &peticion, 1) < 0) {
     perror("Error en send");
     exit(1);
  }

  if ( write(sockd, &destino, sizeof(int)) < 0) {
     perror("Error en send");
     exit(1);
  }

  if ( write(sockd, &foto_long, sizeof(long)) < 0) {
     perror("Error en send");
     exit(1);
  }

  pbuf = foto;
  cont_r = foto_long;

  while (cont_r > 0) {
       cont_w = write(sockd, pbuf, cont_r);
       if (cont_w == -1) {
         perror("Socket write error");
         break;
      }
      cont_r =  cont_r - cont_w;
      pbuf = pbuf + cont_w;
  }

  close(sockd);

  exit(0);
}

/* PARTE SERVIDOR */

pthread_mutex_t m;
pthread_cond_t c;
int busy = FALSE;

#define MAX_LEN   1024

int main() {
  int sockd,sockd2;
  struct sockaddr_in dir, cliente;
  int err, len;

  /* crea el socket */
  sockd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockd == -1){
    perror("Error en socket");
    exit(1);
  }

  dir.sin_family = AF_INET;
  dir.sin_addr.s_addr = INADDR_ANY;
  dir.sin_port = htons(PUERTO);
  err = bind(sockd, (struct sockaddr*)&dir, sizeof(dir));
  if (err == -1){
    perror("Error en bind");
    exit(1);
  }

  err = listen(sockd, 5);
  if (err == -1){
    perror("error en listen");
    exit(1);
  }
  pthread_mutex_init(&mutex);
  pthread_cond_init(&mutex);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

  for (;;) {

     len = sizeof(peer_name);
     sockd2 = accept(sockd, (struct sockaddr*)&client, &len);
     if (sockd2 == -1) {
         perror("Error en accept");
         exit(1);
     }
     pthread_create(&thid, &attr, tratar_peticion, &sock2);

     /* esperar a que el hijo copie el descriptor */
     pthread_mutex_lock(&m);
     while(busy == TRUE)
        pthread_cond_wait(&m, &c);
     busy = TRUE;
     pthread_mutex_unlock(&m);
}

}

void tratar_peticion(int * s) {
       int s_local;
       unsigned char peticion;
       char name[MAX_LEN];
       int anio;
       char anio_string[4];
       char telefono[12], teléfono_new[12];
       int res_err;
       unsigned char res;

       pthread_mutex_lock(&m);
       s_local = *s;
       busy = FALSE;
       pthread_cond_signal(&c);
       pthread_mutex_unlock(&m);

       /* tratar la petición utilizando el descriptor s_local */
       read(s_local, &petición, 1);
       switch (petición) {
               case 0:
                               char *foto;
                               char *usuario;
                               long foto_long, toread,r_l;
                               int destino;
                               r_l = read(s_local, &destino, sizeof(int));
                               r_l = read(s_local, &foto_long, sizeof(long));
                               foto = malloc(foto_long);
                               toread = foto_long;
                               while(toread > 0) {
                                       r_l = read(s_local,buf,foto_long);
                                       buf = r_l + buf;
                                       toread = toread – r_l;
                               }

                               añadir_foto_base_datos(destino,foto,foto_long);
                               usuario = obtener_usuario(destino);

                               enviar_notificacion(usuario, 1); // 1 porque es una foto

                               break;


               default:
                               break;
       }
```

```
        close(s_local);

        pthread_exit(NULL);
}
```

c) Un posible fichero .x sería el siguiente:

```
struct mensaje_args{
    string mensaje<256>;
    int origen;
    int destino;
    int error;
};

struct foto_args {
    opaque foto<>;
    int origen;
    int destino;
    int error;
};


program 3GChatMovil {
    version 3GChatMovil Ver {
        int notificar(int id, byte tipo) = 1;
   } = 1;
} = 1000;

program 3GChatServidor {
    version 3GChatServidorVer {
        int enviar_mensaje(struct mensaje_args) = 1;
        int enviar_foto(struct foto_args) = 2;
        struct foto_args recibir_mensaje(int id) = 3;
        struct foto_args recibir_foto(int id) = 4;
   } = 1;
} = 1001;
```