**Escuela Politécnica Nacional**

**Facultad de Ingeniería de Sistemas**

**Computación Distribuida**

**Taller Final**

**Nombre: Andres Velasco**

**Los siguientes ejercicios son enunciados para realizar el diseño de los siguientes sistemas distribuidos. En todos los casos se debe realizar el diseño del sistema: a nivel físico, lógico, de arquitectura y el algoritmo para la aplicación:**

1. **Se plantea un sistema distribuido cliente/servidor para almacenamiento y verificación de contraseñas. Una aplicación cliente envía un par <usuario, contraseña> hacia el servidor para su almacenamiento. En el servidor se almacena el usuario y el hash de la contraseña. Posteriormente, la aplicación para comprobar que la contraseña indicada es la almacenada. Se plantea un servicio concurrente. Se deberá determinar estrategias para medir el nivel de disponibilidad, dado como la relación entre solicitudes atendidas y solicitudes recibidas.**

**Requisitos:**

* Almacenamiento y verificación de contraseñas.
* Servicio concurrente.

**Diseño Arquitectónico.**

**Entidades Comunicantes.**

Las entidades que se comunican son procesos ya que la información que se va a transmitir a través de la red no es muy grande, de tal manera que es posible manejar esta información a través de procesos.

**Paradigmas de Comunicación**

Este sistema utilizará la comunicación entre procesos, ya que se utilizarán sockets para la comunicación de estos. Así mismo, se utilizará una comunicación del tipo TCP ya que el cliente debe esperar una confirmación de que su contraseña a sido almacenada correctamente o que se ha verificado su contraseña. De esta forma la comunicación termina cuando el cliente recibe el mensaje de confirmación por parte del servidor.

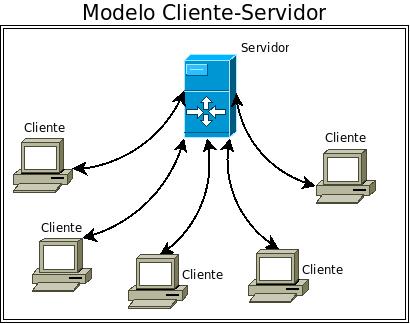
**Roles y Responsabilidades**

La arquitectura del sistema será la de un cliente-servidor.

El servidor del sistema debe ser concurrente y la estrategia para medir el nivel de disponibilidad está dada por la siguiente relación:



**Placement**



**Patrones Arquitectónicos**

El patrón arquitectónico que seguir para esta aplicación es la de un Cliente-Servidor. El cliente es un proceso que necesita acceder a los datos o hacer uso de los recursos de otro computador o servidor. El servidor es quien maneja los datos, los procesa y pone a disposición de los clientes sus recursos. La interacción entre el cliente y el servidor empieza cuando el cliente hace un llamado al servicio y este recibe un mensaje como respuesta al uso de ese servicio.

**Algoritmo**

**Cliente**

client\_socket = socket();

client\_socket.start();

recibir(){

      msg = client\_socket.recive();

      listaMensajes.append(msg);

}

enviar(){

      msg = mensaje.get();

      mensaje.set(‘’);

      cliente\_socket.send(msg);

      if msg == ‘{Salir}’{

            cliente\_socket.close();

}

}

enviar()

recibir()

**Servidor**

aceptarConeccion(){

      accept();

      thread(manejar\_cliente, cliente).start();

}

almacenarClave(){

      informacion = server.recive();

      user = informacion[0];

      pass = informacion[1]

      almacenar(user,generarHash(pass));

      return verificarContraseña(user,contraseña)

}

verificarContraseña(user, contraseña){

      if(consultar(user, generarHash(contraseña))){

            return “Contraseña valida”

}else{

      return “Contraseña no valida”

}

}

server = socket();

server.bind();

server.listen();

thread(aceptarConeccion());

thread.start();

almacenarClave();

server.close();

1. **Se requiere diseñar una aplicación cliente/servidor para control de acceso (asistencia, horarios de ingreso-salida, tiempo de permanencia) del personal de una empresa- existen sensores biométricos en los accesos de la empresa (5 puertas), que registran la asistencia en una base de datos, además de consultar en ella sobre los datos biométricos.**

**Requisitos:**

* Almacenamiento de personal
* Registro de hora de llegada de un personal

**Diseño arquitectónico**

**Entidades comunicantes**

Las entidades que se comunican son objetos, debido a que se tiene un conjunto de elementos que interactúan entre sí, para así dar una solución al problema.

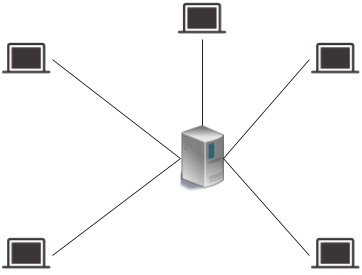
**Paradigmas de comunicación**

La aplicación usará una comunicación entre procesos, debido a que la conexión se realiza mediante sockets. De igual manera, la única información que se envía es la de la hora y fecha del ingreso o salida. Por lo que, es una comunicación sencilla, apoyando el criterio de uso y comunicación entre procesos.

**Roles y responsabilidades**

Para los roles y responsabilidades se hace uso de una arquitectura Cliente/Servidor, debido a que el servidor estará siempre corriendo, esperando que le llegue una solicitud del cliente.

**Placement**

****

**Patrones arquitectónicos**

Para los patrones arquitectónicos se hará uso de clientes ligeros, debido a que no existe una conexión persistente y la información que se envía al servidor es la hora de acceso.

**Algoritmo**

**Cliente**

s = socket()

s.start()

enviar(){

    msg.append(datos.getName())

    msg.append(datos.getTime())

    s.send(msg)

    s.close()

}

**Servidor**

inicializarServidor():

    s = socket()

    s.bind()

    s.listen()

    s.accept()

    s.close()

class conexion():

    def conectar():

        c = connectionMySQLdb()

    def desconectar():

        c = close()

def consulta(query):

    c = conexion()

    c = conectar()

    c.execute(consulta)

c.desconectar()

    return c.fetchall()

**3.     Se plantea una plataforma para almacenamiento compartido para un centro de datos. Se tienen múltiples nodos en un sistema distribuido. Los nodos son computadores top-of-the-shelf, que tienen discos duros de distintos tamaños. Al ser almacenamiento compartido, todos los nodos deben tener acceso de lectura/escritura al agregado total del almacenamiento.**

**Requisitos:**

* Almacenamiento compartido para un centro de datos.
* Acceso de lectura y escritura para todos los nodos.

**Diseño Arquitectónico.**

**Entidades comunicantes**

Las entidades que se comunican son objetos, debido a que se tiene un conjunto de elementos que interactúan entre sí, para así dar una solución al problema.

**Paradigmas de comunicación**

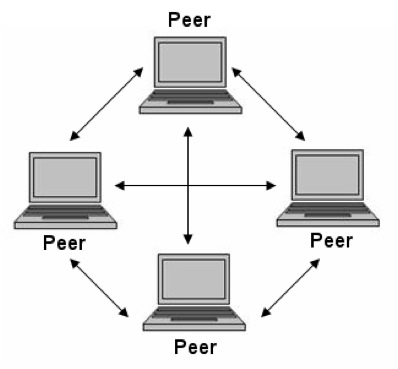
La aplicación usará comunicación entre procesos, ya que es una solución de middleware, que no necesita de interacción con los usuarios finales.

**Roles y Responsabilidades**

Se utilizará una arquitectura del tipo peer-to-peer, en la cual cada nodo mantendrá una tabla similar a las de tipo fat, en las cuales se guardará la dirección en la que se encuentran los archivos. Para esto, se utilizará las direcciones ip de cada computador con un alias, además de manejar un mismo puerto para todos.

Para el manejo correcto de la concurrencia, junto con cada archivo se tendrá dos campos, el primero el estado del archivo, para conocer si está siendo modificado por alguien o no. y el segundo se guardará quien está con el control del documento, esto para que se pueda dar acceso solo a los documentos que no se encuentran siendo editados por otra persona.

**Placement**



**Patrones Arquitectónicos**

Se manejan servidores ligeros, ya que no es necesario mantener una integración con el usuario por lo tanto no se utilizan capas.

**Algoritmo**

socket = bind()

if socket.accept():

    f = socket.read()

    if addr.isRemote():

        s = socket(addr.getPC())

        s.connect()

        s.write(‘Bring me’, addr.getFile())

        file = s.read()

        socket.write(file)

    else:

        file = open(addr, getFile())

        socket.write(file)

socket.close()

**4.  Se requiere diseñar una aplicación Peer-to-Peer para streaming de video bajo demanda. Se tienen videos diseminados a través de la red, localizados en los discos duros de los nodos pertenecientes y registrados en el sistema. Cuando un nodo accede a la red visualiza una lista de videos para ver, lo selecciona y descarga. La descarga del video se hace por partes, y cada parte perteneciente a un nodo diferente registrado en la red.**

**Requisitos:**

* Ver vídeos bajo demanda
* Alta disponibilidad

**Diseño Arquitectónico.**

**Entidades comunicantes**

Las entidades que se comunican son objetos, debido a que se tiene un conjunto de elementos que interactúan entre sí, para así dar una solución al problema.

**Paradigmas de comunicación**

La aplicación usará comunicación entre procesos, debido a que la comunicación es mediante sockets. Lo que hace que la comunicación no sea compleja. Además, se utilizará un protocolo de comunicación UDP, debido a que no nos interesan que se transmitan todos los datos, sino que la información que se transmita sea legible por el receptor, dada la gran cantidad de datos que se debe trasmitir.

**Roles y Responsabilidades**

Dentro de este aspecto, se hará uso de una arquitectura híbrida entre cliente servidor y peer to peer. Cada nodo primero debe conectarse a un servidor, el cual tendrá todos los registros acerca de los contenidos multimedia que tiene cada nodo de la red, de esa forma el servidor le responderá al cliente con una lista de los nodos que tienen el contenido que se solicita. Posteriormente el nodo podrá establecer una comunicación broadcast, con los nodos determinados por el servidor. Además, cada nodo al tener obtener un nuevo contenido multimedia o perder alguno, deberá informar al servidor con el fin de que este actualice sus registros.

**Placement**

****

**Patrones Arquitectónicos**

Este sistema distribuido, estará dividido en capas. Es decir, habrá una capa encargada del direccionamiento, otra encargada de gestionar el contenido multimedia y otra de presentar el contenido.

* **Capa de direccionamiento.** Esta capa se encarga de entablar la comunicación entre los nodos a un bajo nivel, es decir a través de las direcciones IPv4, que estén registradas en el servidor.
* **Capa de gestión de contenido.** Esta capa, determina la distribución del trabajo, que cada nodo deberá hacer para compartir el contenido.
* **Capa de presentación.**Esta capa ordenará las partes del video que reciba de cada nodo, con el fin de hacer que el contenido sea legible por el usuario.

**Algoritmo.**

**Cliente-Servidor.**

**Cliente**

    s=socket()

    s.connect(ip,port)

    cambios\_contenido=escanear\_disco()

    solicitud=valor\_entrada

if cambios\_contenido:

s.send(nuevo)

s.send(solicitud)

lista\_nodos=s.recv()

s.close()

**Servidor**

s=socket.bind()

if s.accept():

     msg = s.recv()

      if msg == nuevo:

        actualizar\_registros()

      else:

lista\_nodos=consultar\_registro(mensaje)

       s.send(lista\_nodos)

s.close()

**Peer to Peer**

s=socket(UDP)

i=1

global video=[array]

thread⇒ servidor() #Hilo que hace de servidor

for nodo in lista\_nodos:

       conn=s.connect(nodo)

       if conn⇒ success:

s.send(i,contenido) #Comunicamos que parte del contenido queremos recibir

    thread ===> repositorio\_partes(conn,i)

 mostrar\_video(video)

 i++

 servidor(){

        s=socket.bind()

        while True:

            s.listen(10)

            if s.accept():

                # Inicio thread

                    contenido,i = s.recv()

                    contenido=fetch\_contenido(contenido)

                        contenido\_parcial=contenido[ i ] #Divide el contenido

                        s.send(contenido\_parcial)

                #Fin thread

}

    repositorio\_partes(conn,i){

        parte\_video=conn.recv()

        video[ i ] = parte\_video

}