Problemas de Redes de Computadores.
Ingeniería Técnica en Informática de Gestión
Conjunto de problemas 1

Preguntas cortas

Pregunta 1.1: Si configuro mi servidor Web para que no acepte conexiones desde la dirección IP 130.206.1.1, ¿Qué nivel de protocolo es el que está decidiendo si aceptar o no aceptar a un cliente?

a) El nivel de red, porque utiliza la dirección IP

b) El nivel de transporte, porque lo que se rechaza es la conexión

c) El nivel de aplicación, porque es la configuración del servidor Web

d) El nivel de enlace, porque utiliza la dirección ethernet

Problema 1.2: ¿Cómo se detecta, desde un programa que utiliza el API de sockets, que un puerto ya está ocupado por otra aplicación y no puede ser utilizado de nuevo por nuestro programa?

Problema 1.3: Cuando la función `connect()` devuelve -1 ¿Cómo puedo saber si es porque no hay nadie escuchando en ese puerto o si el servidor no acepta una conexión desde la dirección IP desde la que he lanzado la conexión?

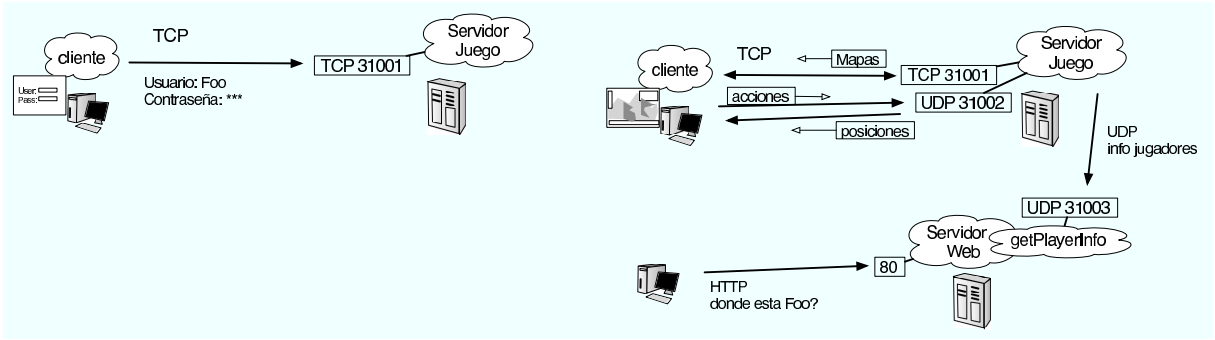
Problema 1.4: ¿Cuáles de estas funciones pueden bloquear el programa por mas de 5 segundos?

`socket()` `connect()` `recvfrom()` `sendto()` `bind()` `listen()` `accept()`

Problema 1.5: ¿Qué nivel de la pila de protocolos se encarga en Internet de separar los datos recibidos para diferentes aplicaciones?

Problemas largos de examen: Sobre un servidor de juegos

Las comunicaciones de un juego multijugador online se hacen usando TCP y UDP como se ve en la siguiente figura. El cliente del juego desde el ordenador del usuario establece una conexion TCP al servidor del juego correspondiente en el puerto 31001. A través de esta conexión se intercambian mensajes para negociar la autentificación y que el cliente demuestre que tiene un usuario que ha pagado la cuota mensual. Una vez autetificado la conexion TCP se mantiene abierta para enviar al cliente información de mapas y informaciones que necesitan transporte fiable. Pero las informaciones de tiempo real sobre el estado del juego se mandan mediante UDP. El servidor tiene un socket UDP en el puerto 31002 al que los clientes autentificados le envian mensajes con sus acciones y movimientos y desde este socket el servidor les envía informaciones de tiempo real (los movimientos de los demas).Cuando se cierra la conexión TCP el cliente deja de jugar. Aparte el servidor de juego envía información sobre los jugadores al servidor web como se ve más adelante



Pregunta 1.6: El equipo de desarrollo que trabaja en la siguiente versión del juego sugiere utilizar el puerto 61000 tanto para la autentificacion sobre TCP como para el funcionamiento el tiempo real sobre UDP. ¿Ocasionaría algún problema ese funcionamiento?

a) No

b) Si, porque 61000 no es un puerto válido, solo hay puertos hasta el 32767

c) Si, porque para hacerlo el servidor debería tener dos sockets escuchando el puerto 61000, uno de tipo `SOCK_STREAM` y otro de tipo `SOCK_DGRAM` y por tanto en uno de los dos fallara el `bind()`

d) Sólo que no se podrá tener dos clientes del juego en un mismo ordenador porque el primero que lance el programa y se conecte al servidor ocupara el puerto 61000 y no podra conectarse otro desde el mismo ordenador. Aunque esto normalmente no es un problema en un juego

Se pretende ofrecer en la pagina web de la empresa la informacion en tiempo mas o menos real de donde estan los jugadores en el mundo del juego. Para ello los servidores tienen que informar periódicamente al servidor web de las posiciones de los jugadores para que se puedan integrar en la pagina web. Como queremos cargar lo menos posible a los servidores del juego se enviaran mensajes UDP a un programa en el servidor web encargado de recopilar la información. Se utilizara el puerto 31003 para este programa

Se pretende construir el programa que recibirá la información de localización de los jugadores desde los servidores de juego para posteriormente integrarla en la web. La información se enviará mediante mensajes UDP conteniendo una cadena de la forma

```
<identificadorservidor>  <nombredeljugador>  <idmapa>  (<x>,<y>) \n
```

Es decir

```
2 Foo 5 (16,23)
```

significa que el servidor 2 informa de que el jugador Foo esta en el mapa 5 en las coordenadas x=16 y=23.

Se enviara una única cadena de ese formato en cada mensaje UDP. Nuestro programa no debe de contestar nada a los servidores ni hacer ninguna petición, simplemente recogerá la información que estos le envíen periódicamente y la integrará en la pagina web para que muestre esta información.

Pregunta 1.7: La anterior descripción de los mensajes y acciones de nuestros programa es la descripción de un protocolo. Pero... ¿de que nivel?

a) Enlace, porque enlaza los servidores del juego con el servidor Web

b) Red, porque la información atraviesa routers hasta llegar a su destino

c) Transporte, porque utiliza sockets UDP

d) Aplicación, porque es parte de nuestra aplicación así que esta por encima del nivel de transporte

El programa que recibe los datos tiene la siguiente función `main()` que llama a otras dos funciones

```
int main(int argc, char *argv) {
    int s;
    printf("getPlayerInfo running...\n");
    s = prepara_socket_udp(32003);
    while (1) {
        get_info_from_player_using_socket(s);
    }
}
```

La funcion `prepara_socket_udp()` nos devuelve un socket ya preparado para escuchar en el puerto indicado. Se ha escrito este código.

```
int prepara_socket_udp(int puerto) {
    struct sockaddr_in midireccion;
    int sock;
    int err;

    sock = socket(PF_INET, SOCK_DGRAM,0);

    midireccion.sin_family = AF_INET;
    midireccion.sin_addr.s_addr = htonl(INADDR_ANY);
    midireccion.sin_port = htons(puerto);

    err = bind(sock, (struct sockaddr *)&midireccion, sizeof(midireccion));
    if ( err == -1 ) {
        exit(-1);
    }
    listen(sock,5);
    return sock;
}
```

Pregunta 1.8: ¿Qué errores se han cometido al escribir la función `prepara_socket_udp()`?

a) Ninguno

b) A la función `socket` hay que pasarle `SOCK_STREAM` para que no pierda los mensajes

c) No hay que convertir `INADDR_ANY` con `htonl()` porque de lo contrario solo recibirá mensajes de 127.0.0.1

d) Otro error (indicar cual) _____

e) Hay que hacer `return s;` en lugar de `return sock;` porque si no la función `main` no recibirá el socket

La función `get_info_from_player_using_socket()` recibe un paquete e interpreta la información contenida en él pasándosela a la función `enter_player_info()` que la añade a la web. El código de la función es el que sigue

```
void get_info_from_player_using_socket(int sock) {
    unsigned char buf[70000];
    struct sockaddr_in dir;
    unsigned int len=sizeof(dir);
    unsigned int leidos;
    int serverID;    char playerName[40];
    int mapID;    int x,y;

    leidos = recvfrom(sock, buf, 70000, 0, (struct sockaddr *)&dir, &len);

         /* queremos insertar condiciones aqui para la pregunta 12 */
```

```
    buf[leidos]=0;
    sscanf(buf,"%d %s %d (%d,%d)\n",&serverID,playerName,&mapID,&x,&y);
    enter_player_info(serverID,playerName,mapID,x,y);
}
```

Pregunta 1.9: Queremos añadir una condición despues del recvfrom() que solo permita que aceptemos informacion de los servidores del juego y no de cualquier otro origen. Para ello se han definido las variables globales siguientes con las que tenemos definidos los servidores aceptables

```
#define N_SERVERS 3
int servidores[N_SERVERS];
```

Y posteriormente se han inicializado con

```
servidores[0]=inet_addr("87.6.5.25");
servidores[1]=inet_addr("87.6.5.26");
servidores[2]=inet_addr("87.6.5.27");
```

Escriba el codigo para comprobar que solo aceptamos paquetes procedentes de servidores de la lista definida en la variable `servidores`

Pregunta 1.10: En la siguiente versión del juego se pretende enviar la información de los jugadores al puerto 32004. Dado que queremos tener funcionando a la vez servidores con las dos versiones del juego. Se pretende modificar el programa para que escuche la información de los jugadores de todas las versiones. Por lo que se realiza la siguiente modificación del código de la función `main()` para que pueda atender a varios sockets

```
int main(int argc, char *argv) {
    int s1,s2;
    printf("getPlayerInfo running...\n");

    s1 = prepara_socket_udp(32003);
    s2 = prepara_socket_udp(32004);
    while (1) {
        get_info_from_player_using_socket(s1);
        get_info_from_player_using_socket(s2);
    }

}
```

A la vista de las funciones anteriores. ¿Qué problema tiene este código?

a) Ningun problema, debería funcionar bien porque el programa no se bloquea en la funcion get_info_from_player_using_socl

b) No funcionará, un programa no puede abrir más de un socket asi que el segundo prepara_socket_udp() dará error en el bind

c) Si no llegan mensajes a uno de los sockets los que lleguen al otro se quedaran sin ser atendidos porque el programa se quedara atascado en una de las `get_info_from_player_using_socket()`

d) Si llegan mensajes muy cercanos en el tiempo a los dos sockets una de las funcionaes get_info_from_player_using_socket() interrumpirá a la otra lo que puede causar que se pierdan mensajes

d) Si llegan mensajes muy cercanos en el tiempo a los dos sockets una de las funcionaes get_info_from_player_using_socket() recibira varias lineas en una sola llamada a `recvfrom()` pero como solo hace un `sscanf()` suponiendo que hay una linea se perdera la información de uno de los mensajes

Problemas largos de examen: Sobre un sistema peer-to-peer

Problema 1.11: Como parte de un sistema peer-to-peer, se pretende construir un programa *tracker* que lleve la cuenta de los diferentes peers que están disponibles en el sistema. Para ello el *tracker* escuchará en el puerto 15051. Los peers interesados en unirse al sistema deberán informar al *tracker* mediente el siguiente protocolo:

- establecer una conexión TCP con el tracker en el puerto 15051
- enviar un mensaje de texto sobre la conexión. El formato del mensaje será una primera linea de texto indicando la dirección IP y el puerto en el que escucha el peer, a continuación enviara los nombres de los ficheros que desea compartir con otros peers enviando un nombre de fichero en cada linea de texto
- cerrar la conexión para inicar el fin del mensaje

```
PEER direccionIP puerto
fichero1
fichero2
fichero3
<FIN DE LA CONEXIÓN>
```

El *tracker* no debe enviar ningun dato como respuesta al peer simplemente apuntara los datos para actualizar la información de todos los peers.

a) Complete la función para inicializar el socket en el programa tracker y asociarlo al puerto 15051

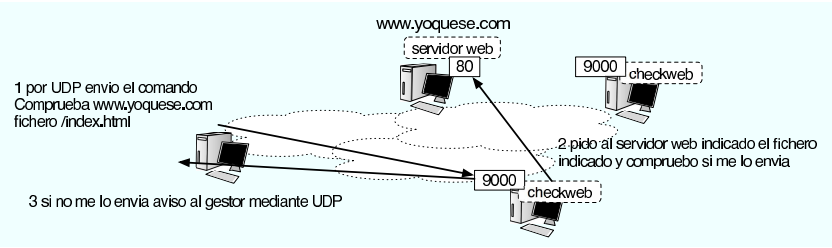
```
int init_tracker_socket () {
    int sock;
    struct sockaddr_in servidor;

    sock=socket(PF_INET,SOCK_STREAM,0);
    /* rellenar la estructura servidor */
    servidor....

    if ( bind( sock, (struct sockaddr *)&servidor , sizeof(servidor) ) == -1 ) {
        return -1;
    }
    return sock;    }
```

b) Complete la función `espera_siguiente_peer( )` que se encarga de esperar la conexión de un peer y almacenar los datos llamando a las funciones indicadas

```
/* Tenemos disponibles dos funciones a las que hemos de llamar con la informacion del nuevo peer */

int nuevo_peer( char *direccionIP , int puerto);
void peer_tiene( int peerid, char *nombrefichero);



int espera_siguiente_peer ( int sock ) {
    struct sockaddr_in dir;
    int s2;
    /* datos del peer */
    char direccionipdelpeer[20];
    int puertodelpeer;
    char nombrefichero[200];
    /* para almacenar la informacion nos dara un identificador del peer */
    int peerid;

    s2 = accept( (struct sockaddr *)&dir , sizeof(dir) );

    /* leer informacion del peer */
    ......
    ......

    printf( "recibida informacion del peer en la dirección:
                          \%s y puerto \%d\n ",direccionipdelpeer, puertodelpeer );
    peerid=nuevo_peer( direccionipdelpeer , puertodelpeer );

    /* leer informacion de los ficheros */
    /* y llamar a peer_tiene( peerid, nombrefichero) con cada uno*/
    ......
    ......

    }

}
```

Problema 1.12: ¿Cámo modificaría la función anterior para que compruebe si el peer esta dando la información correcta de su dirección o si esta enviando una dirección falsa?

Problemas largos de examen: Sobre un monitorizador de servidores web



Pregunta 1.13: Se pretende construir un sistema para monitorizar la disponibilidad de servidores web de forma automática. El sistema se basara en un programa pequeño que llamaremos `checkweb` que se colocará en diferentes puntos desde los que queramos asegurarnos que se ven los servidores web y realizara pruebas de peticiones web bajo la dirección de un control central. El proceso será como se ve en la figura. El programa `checkweb` escuchara comandos en el puerto UDP 9000 y estos comandos le indicarán que haga una petición web a un servidor pidiendo un fichero concreto, si el servidor contesta no debe hacer nada y si no consigue conectar enviará un mensaje avisando al cliente UDP de control que le pidió realizar la comprobación.

Esta es la función `main` del programa, se omiten las cabeceras e includes

```
struct info_peticion {
    int direccion_ip;
    char nombre_fichero[500];
};

int main(int argc, char *argv[]) {
    int sock;     int fichero_ok;
    struct info_peticion peticion;
    struct sockaddr_in fdir;


    sock = inicia_escucha( 9000 );

    while ( 1 ) {
        if ( espera_peticion(sock, &peticion, &fdir) != 0 ) {
            fichero_ok = peticion_web(&peticion);
            if ( ! fichero_ok ) {
                envia_error(sock, &peticion, &fdir);
         }}}}
```

Señale los errores presentes en el código de la funcion `inicia_escucha` a continuación

```
int inicia_escucha( int puerto ) {
    int sock;
    struct sockaddr_in dir;

    sock = socket(PF_INET,SOCK_STREAM,0);
    dir.sin_family=AF_INET;
    dir.sin_port=htons(puerto);
    dir.sin_addr.s_addr=htonl(INADDR_ANY);
    if ( bind( sock,(struct sockaddr*)&dir,sizeof(dir)) == -1 )
        sal_con_error("No puedo coger el puerto");

    return sock;
}
```

a) El socket `sock` al ser un socket UDP debe construirse con SOCK_DGRAM en lugar de SOCK_STREAM

b) dir.sin_family debe ser PF_INET al igual que el valor pasado al socket

c) El error se producirá cuando bind devuelve 0 en lugar de -1

d) El puerto hay que convertirlo con ntohs() en lugar de con htons() porque queremos convertir del formato de red al de host

e) No hay errores

Pregunta 1.14: La función espera_peticion escucha mensajes por el socket y rellena la estructura de petición con la dirección y el nombre de fichero que llegan en el mensaje

```
int espera_peticion(int sock, struct info_peticion *peticion, struct sockaddr_in *fdir) {
    unsigned int leidos,lenfdir;
    char buf[5000];
    char host[500];
    unsigned int direccion_ip;

    lenfdir=sizeof(*fdir);
    leidos=recvfrom(sock,buf,5000,0,(struct sockaddr *)fdir,&lenfdir);
    if (leidos > 0) {
        printf("recibido: [%s]\n",buf);
        sscanf(buf,"%s %s",host,peticion->nombre_fichero);
        direccion_ip=inet_addr(host);
        peticion->direccion_ip=direccion_ip;
        return 1;
    } else {
        return 0;
    }
}
```

Tal y como está hecha la función, ¿cuál es el formato de los mensajes que llegan de la red?

a) Los mensajes son una línea de texto con el formato 'direccionIP nombredefichero' la dirección IP es una cadena de la forma 'a.b.c.d' pero no es capaz de leer nombres de tipo 'www.yoquese.com'

b) Los mensajes son una línea de texto con el formato 'host nombredefichero' el host se lee con la funcion `inet_addr` por lo que es capaz de leer nombres de tipo 'www.yoquese.com' y convertirlos a direcciones IP

c) Los mensajes son una estructura binaria, los primeros 4 bytes son los bytes de la dirección IP en el orden de la red y luego vienen los bytes del nombre del fichero a pedir

d) Los mensajes son una estructura binaria, los primeros 4 bytes son los bytes de la dirección IP en el orden del host (por eso no se usa `ntoh`) y luego vienen los bytes del nombre del fichero a pedir

Pregunta 1.15: La función que realiza la petición web es esta. La función `respuesta_web_ok` analiza la primera linea de la respuesta para ver si tiene un codigo de OK o un error. Una vez obtenida la primera linea de la respuesta cerramos la conexión.

```
int peticion_web(struct info_peticion *peticion) {
    int sock;
    struct sockaddr_in dir;
    int len,leidos;
    int err;
    char buf[5000];
    FILE *cnx;

    len=sprintf(buf,"GET %s HTTP/1.0\r\n\r\n", peticion->nombre_fichero);

    sock = socket(PF_INET,SOCK_STREAM,0);
    dir.sin_family=AF_INET;
    dir.sin_port=htons(80);
    dir.sin_addr.s_addr=htonl(peticion->direccion_ip);

    err=connect(sock, (struct sockaddr*)&dir, sizeof(dir));
    if (err < 0) {
        return 0;
    }
    cnx=fdopen(sock,"r+");
    fprintf(sock,"%s",buf);

    if ( fgets(buf,5000,cnx)==NULL ) {
        return 0;
    }
```

```
    fclose(cnx);

    if ( respuesta_web_ok(buf) ) {
        printf("OK | %s\n",buf);
        return 1;
    }

    printf("Error | %s\n",buf);
    return 0;
}
```

¿Qué error o errores se han cometido al escribirla?

a) El `htons()` tiene que ser `ntohs()`

b) No se ha hecho bind antes de usar el nuevo socket

c) No hay que hacer `connect()` ya que es un socket UDP

d) Se utiliza el mismo socket (variable `sock`) que en la función `inicia_escucha`

e) No hay ningún error en ese trozo

f) A `fprintf` hay que pasarle la variable `cnx` en lugar de `sock`

Pregunta 1.16: Complete la función `envia_error` para que envíe el mensaje construido en `buf` al cliente que nos envío la petición

```
int envia_error(int sock, struct info_peticion *peticion, struct sockaddr_in *tdir) {
    char buf[5000];
    int escritos;
    escritos=sprintf(buf,"ERROR http://%x%s no disponible\n",
                                peticion->direccion_ip,peticion->nombre_fichero);

    /* Complete aqui el codigo */









}
```

Pregunta 1.17: Tal y como esta hecho el programa. ¿Que ocurre si llegan dos paquetes UDP muy seguidos con dos peticiones a comprobar?

a) La segunda petición no puede ser atendida porque se esta procesando la primera y el mensaje de la segunda petición se pierde con lo que no se comprueba. Sólo obtenemos los resultados del primero

b) La segunda petición interrumpe a la primera porque se llama a la función `peticion_web` otra vez, antes de que haya salido de la primera. Sólo obtenemos los resultados del segundo

c) La segunda petición espera a que termine de procesarse la primera, cuando sale de la función `peticion_web` lee el siguiente mensaje del socket. Obtenemos los resultados de los dos comandos con un tiempo de espera para el segundo, generalmente no mucho

e) Como los servidores UDP son básicamente concurrentes las dos peticiones se realizan a la vez (gracias a la multitarea). Obtenemos los resultados de los dos sin retardo apreciable para el segundo