TP 1 - PARTE A

1) Explicar qué es un socket y los diferentes tipos de sockets.

Un socket actúa como un punto de conexión que facilita la comunicación entre dispositivos o programas distintos en una red utilizando descriptores de archivo estándar, o sea un entero asociado con un archivo o recurso abierto. Su función principal radica en la transmisión y recepción de datos entre estos puntos de conexión.

Al hacer una llamada al sistema socket(), se obtiene un descriptor de socket que puede ser utilizado para realizar otras llamadas al sistema como send() o read().

Existen cuatro tipos principales de sockets. En primer lugar, se encuentra el Socket de Datagrama, también conocido como UDP socket. Este tipo de socket ofrece comunicación no orientada a la conexión, lo que significa que no garantiza la entrega de los datos ni su orden. Los paquetes pueden llegar desordenados o incluso perderse durante la transmisión.

En segundo lugar, está el Socket de Paquete, utilizado para capturar y enviar de un dispositivo en la capa de enlace, lo que implica que el encabezado de la placa de red puede ser accedido.

El tercer tipo es el Socket de Flujo, conocido como TCP socket, que a diferencia del Socket de Datagrama, establece una conexión orientada a la conexión y fiable entre dos hosts en la red. Este tipo de socket se utiliza para la transmisión de datos en flujo continuo y garantiza que los paquetes lleguen en el mismo orden en que fueron enviados.

Por último, encontramos el Raw Socket, que ofrece acceso de bajo nivel a la capa de red y permite enviar y recibir paquetes en su forma cruda, sin procesar. Este tratará con el datagrama sin procesar sin encabezados de nivel de enlace.

2) Cuáles son las estructuras necesarias para operar con sockets en el modelo C-S y cómo se hace para ingresar los datos requeridos. Explicar con un ejemplo.

Las estructuras varían según estemos del lado del servidor o del cliente. Para el servidor, se requieren: la dirección del servidor, un socket de escucha, un socket para cada cliente, así como estructuras de datos para gestionar las conexiones entrantes y salientes, además de procesar y responder a las solicitudes de los clientes.

Por otro lado, el cliente necesita la dirección del servidor, un socket para la comunicación con el servidor y estructuras para enviar y recibir datos del servidor.

Una de las estructuras a las que mas recurrimos es:

```
struct sockaddr_in{
        short int sin_family;   -> Familia de direcciones=AF_INET
        unsigned short int sin_port;  -> Número de puerto
        struct in_addr sin_addr;  ->Dirección de Internet
        unsigned char sin_zero[8];  ->Conserva tamaño igual a struct sockaddr
        }
```

Esta nos permite guardar y referirnos a información sobre la dirección de un socket. Luego esta estructura de información es utilizada para poder identificar un socket en el otro extremo de la comunicación, para finalmente establecer una conexión con el y sobre ella realizar otros procesos.

3) Explicar modo bloqueante y no bloqueante en sockets y cuáles son los "sockets calls" a los cuales se pueden aplicar estos modos. Explicar las funciones necesarias.

El modo bloqueante y no bloqueante en sockets se refiere al comportamiento de las llamadas a funciones de socket en términos de bloqueo del proceso.

En el modo bloqueante, las llamadas a las funciones de socket hacen que el proceso que las realiza se bloquee hasta que la operación correspondiente se complete. Por ejemplo, si un proceso llama a la función recv() para recibir datos de un socket en modo bloqueante y no hay datos disponibles para recibir, el proceso se bloqueará y esperará hasta que se reciban datos o se produzca un error.

En contraste, en el modo no bloqueante, las llamadas a las funciones de socket no bloquean el proceso, incluso si la operación solicitada no se puede completar de inmediato. En lugar de bloquearse, las funciones de socket en modo no bloqueante devuelven un valor especial que indica que la operación no se pudo completar en ese momento. Esto permite que el proceso continúe ejecutándose y realice otras tareas mientras espera que ocurra algún evento relacionado con el socket, como la disponibilidad de datos para recibir o la capacidad de enviar datos.

Las funciones necesarias para trabajar en modo no bloqueante y bloqueante, aparte de las funciones de socket call mencionadas, incluyen:

fcntl(): Utilizada para establecer o cambiar el modo de bloqueo de un descriptor de archivo, incluidos los sockets.

select(): Permite al proceso monitorear múltiples descriptores de archivo, incluidos los sockets, y determinar cuáles están listos para lectura, escritura o excepción.

poll(): Ofrece una interfaz similar a select() pero con mayor flexibilidad y eficiencia en algunas situaciones.

epoll(): Proporciona una interfaz eficiente para monitorear múltiples descriptores de archivo en sistemas Linux.

4) Describir el modelo C-S aplicado a un servidor con Concurrencia Real. Escribir un ejemplo en lenguaje C.

En el enfoque Cliente-Servidor con concurrencia real, el servidor puede gestionar múltiples clientes simultáneamente. Esto se logra mediante técnicas como la creación de subprocesos o procesos hijos para manejar las solicitudes individuales de cada cliente. Esta arquitectura permite al servidor atender a varios clientes al mismo tiempo sin esperar a que una solicitud previa se complete, mejorando así significativamente la eficiencia y la capacidad de respuesta del sistema en entornos con alta carga de trabajo.

Servidor:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#include <arpa/inet.h>
#include <time.h>
#include <stdlib.h>
#define MAX 80
#define PORT 8088
#define SA struct sockaddr
double tiempo_transcurrido(struct timespec* inicio, struct timespec* fin) {
        return (fin->tv_sec - inicio->tv_sec) + (fin->tv_nsec - inicio->tv_nsec) ;
}
// Función principal
int main(){
   fd_set master, read_fds;
   int fdmax,yes=1,active_connections=0;
   int i,j;
```

```
            FD_ZERO(&master);
            FD_ZERO(&read_fds);
            char buff[MAX];
            int n;
            struct timespec start,end;
            double tiempo_t;

            int sockfd, connfd, len;
            struct sockaddr_in servaddr, cli;

            // socket create and verification
            sockfd = socket(AF_INET, SOCK_STREAM, 0);
            if (sockfd == -1) {
                printf("Falla la creación del socket...\n");
                exit(0);
            }
            else
                printf("Socket creado...\n");
            //bzero(&servaddr, sizeof(servaddr));

                if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==
-1){
                perror("setsockopt");
                exit(0);
            }

            // asigno IP, PORT
            servaddr.sin_family = AF_INET;
            servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
            servaddr.sin_port = htons(PORT);

            memset(&(servaddr.sin_zero),'\0',8);

             // Bind del nuevo socket a la dirección IP y lo verifico(asocio ip
a puerto)
              if ((bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)))
== -1) {
                printf("Falla socket bind ...\n");
                exit(0);
            }
            else
                printf("Se hace el socket bind ..\n");

            // El servidor está en modo escucha (pasivo) y lo verifico
            if ((listen(sockfd, 5)) == -1) {
```

```
            printf("Falla el listen ...\n");
            exit(0);
        }else{
            printf("Servidor en modo escucha ...\n");
            FD_SET(sockfd,&master);
            fdmax=sockfd;
        }
        len = sizeof(cli);


         for (;;) {
            read_fds = master;
            if (select(fdmax + 1, &read_fds, NULL, NULL, NULL) == -1) {
                perror("select");
                continue;
            }

            for (i = 0; i <= fdmax; i++) {
                if (FD_ISSET(i, &read_fds)) {
                    if (i == sockfd) {
                        // New connection request
                        len = sizeof(cli);
                            connfd = accept(sockfd, (struct sockaddr*)&cli,
(unsigned int*)&len);
                        if (connfd == -1) {
                            perror("accept");
                        } else {
                            // Add new connection to master set
                            FD_SET(connfd, &master);
                            if (connfd > fdmax) {
                                fdmax = connfd;
                            }
                              printf("Nuevo cliente conectado desde %s en el
socket %d\n", inet_ntoa(cli.sin_addr), connfd);
                            active_connections++;
                        }
                    } else {
                        // Existing connection has data to read
                        bzero(buff, MAX);
                        clock_gettime(CLOCK_MONOTONIC,&start);
                        if ((n = recv(i, buff, sizeof(buff), 0)) <= 0) {
                            if (n == 0) {
                                printf("Cliente %d se desconectó\n", i);
                                active_connections--;
                            } else {
```

```
                                perror("recv");
                            }
                             // Close the connection and remove from master
set
                            close(i);
                            FD_CLR(i, &master);
                        } else {
                            // Print received message from client
                            printf("Del cliente %d: %s\n", i, buff);

                            // Check if the message is "SALIR" to close the
connection
                            if (strncmp("SALIR", buff, 5) == 0) {
                                printf("Cliente %d solicita salir\n", i);
                                active_connections--;
                                close(i);
                                FD_CLR(i, &master);
                                if(active_connections==0){
                                            printf("Se cerraron todas las
conexiones.\nSalgo del servidor...\n");
                                    exit(0);
                                }
                            } else {
                                int padre=getpid(),hijo=getppid();
                                int num = atoi(buff);
                                int factorial = 1;
                                for (int k = 1; k <= num; ++k) {
                                    factorial *= k;
                                }
                                  // Get input message from server and send
to client
                                printf("Servidor: %d\n",factorial);
                                bzero(buff, MAX);
                                n = 0;
                                            write(i,  (void*)&factorial,
sizeof(factorial));
                                clock_gettime(CLOCK_MONOTONIC,&end);
                                tiempo_t=tiempo_transcurrido(&start,&end);
                                write(i,&tiempo_t,sizeof(tiempo_t));
                                write(i,&padre,sizeof(padre));
                                write(i,&hijo,sizeof(hijo));
                            }
                        }
                    }
                }
```

```
            }
        }

        // Close the socket when finished
        close(sockfd);
    }
```

Cliente:

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8088
#define SA struct sockaddr
void func(int sockfd)
{
    int fact,hijo ,padre;
    double tiempo_t;
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Ingrese texto : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        if ((strncmp(buff, "SALIR", 5)) == 0) {
            printf("Saliendo...\n");
            break;
        }
        bzero(buff, sizeof(buff));
        read(sockfd, &fact, sizeof(fact));
        printf("Servidor : %d\n", fact);
        read(sockfd,(void*)&tiempo_t, sizeof(tiempo_t));
         printf("Tiempo de respuesta del servidor: %f microsegundos\n",
tiempo_t);

        read(sockfd,(void*)&padre,sizeof(padre));
        read(sockfd,(void*)&hijo,sizeof(hijo));
        printf("PID Padre:%d     PID Hijo:%d\n", padre,hijo);
```

```
            if ((strncmp(buff, "SALIR", 5)) == 0) {
                printf("Saliendo...\n");
                exit(0);
            }
        }
    }
    int main()
    {
        int sockfd, connfd;
        struct sockaddr_in servaddr, cli;
        // socket: creo socket y lo verifico
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
            printf("Falla la creación del socket...\n");
            exit(0);
        }
        else
            printf("Socket creado ..\n");
        bzero(&servaddr, sizeof(servaddr));
        // asigno IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
        servaddr.sin_port = htons(PORT);
        // Conecto los sockets entre cliente y servidor
        if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
            != 0) {
            printf("Falla de conexión con servidor...\n");
            exit(0);
        }
        else
            printf("Conectado al servidor..\n");
        // Función para el chat
        func(sockfd);
        //Cierro el socket
        close(sockfd);
    }
```

Este código establece un servidor capaz de recibir conexiones de clientes y mantener conversaciones con cada uno de ellos. Esta arquitectura permite al servidor atender a múltiples clientes al mismo tiempo mientras continúa aceptando conexiones adicionales.

5) ¿Cómo se cierra una conexión C-S ?. Métodos que eviten la pérdida de información.

Se cierra siguiendo un procedimiento específico dependiendo del protocolo de comunicación utilizado y la implementación particular. Algunos métodos que evitan la pérdida de información son:

A- Notificación de cierre previa: El cliente o servidor envía un mensaje de cierre antes de cerrar la conexión para notificar que esta será finalizada. Esto permite que realicen cualquier acción necesaria, como por ejemplo realizar una limpieza de recursos y enviar los últimos datos.

B- Protocolo de cierre explícito: En este método, como en el protocolo TCP, cuando se desea cerrar la conexión se envía un paquete FIN al otro lado, que responde con un ACK y luego envía su propio paquete ACK. Después de recibir esta confirmación, el primer lado envía una última para confirmar el cierre.

C- Temporizador de inactividad: Si un cliente o servidor deja de recibir o enviar datos durante un cierto tiempo, se puede cerrar la conexión automáticamente para evitar la inactividad, todo esto mediante un temporizador.

D- Manejo de errores: En este caso, si se produce un error durante la comunicación, como una conexión interrumpida, se puede cerrar la misma de manera inmediata para evitar la pérdida de datos adicionales. Es importante, con este método, manejar correctamente los errores y notificar al otro extremo sobre la terminación de la conexión.

6) Probar el código del chat entre cliente y servidor. Cambiar tipo de socket y volver a probar.

Código del cliente:

```
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/socket.h>
#include <unistd.h>

#define MAX 80
```

```
#define PORT 8080
#define SA struct sockaddr

void func(int sockfd, struct sockaddr_in servaddr)
{
    char buff[MAX];
    int n, len;
    len = sizeof(servaddr);
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Ingrese texto : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        sendto(sockfd, buff, sizeof(buff), 0, (SA*)&servaddr, len);
        bzero(buff, sizeof(buff));
        recvfrom(sockfd, buff, sizeof(buff), 0, (SA*)&servaddr,
&len);
        printf("Servidor : %s", buff);
        if (strncmp("SALIR", buff, 4) == 0) {
            printf("Cliente cierra conexión...\n");
            break;
        }
    }
}

int main()
{
    int sockfd;
    struct sockaddr_in servaddr;

    // socket: creo socket y lo verifico
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado ..\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
```

```
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // Función para el chat
    func(sockfd, servaddr);

    //Cierro el socket
    close(sockfd);
    return 0;
}
```

Código del servidor:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h> // inet_ntoa()
#include <netdb.h>

#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Función para el chat entre el cliente y el servidor
void func(int sockfd, struct sockaddr_in cliaddr)
{
    char buff[MAX];
    int n, len;
    len = sizeof(cliaddr);
    // Loop para el chat
    for (;;) {
        bzero(buff, MAX);

        // Leo el mesaje del cliente y lo copio en un buffer
        recvfrom(sockfd, buff, sizeof(buff), 0, (SA*)&cliaddr,
&len);
        // Muestro el buffer con los datos del cliente
        printf("Del cliente: %s\t: ", buff);
```

```
        bzero(buff, MAX);
        n = 0;
        // Copio el mensaje del servidor en el buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // y envío el buffer al cliente
        sendto(sockfd, buff, sizeof(buff), 0, (SA*)&cliaddr, len);

        // si el mensaje dice "SALIR" salgo y cierro conexión
        if (strncmp("SALIR", buff, 4) == 0) {
            printf("Salgo del servidor...\n");
            break;
        }
    }
}

// Función principal
int main()
{
    int sockfd, len;
    struct sockaddr_in servaddr, cliaddr;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        printf("Falla la creación del socket...\n");
        exit(0);
    }
    else
        printf("Socket creado...\n");
    bzero(&servaddr, sizeof(servaddr));

    // asigno IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Bind del nuevo socket a la dirección IP y lo verifico
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("Falla socket bind ...\n");
        exit(0);
    }
```

```
    else
        printf("Se hace el socket bind ..\n");

    printf("Servidor en modo escucha ...\n");

    // Funcion para el chat entre el cliente y el servidor
    func(sockfd, cliaddr);

    // Cierro el socket al terminar el chat
    close(sockfd);
    return 0;
}
```

Para cambiar el tipo de socket de SOCKET_STREAM a SOCKET_DGRAM se modificó el tipo socket y luego se quitó tanto el connect() como el listen(). Al utilizar UDP, la dirección del cliente es utilizada en cada llamado a la funciones sendto() y recvfrom().
Output del cliente:

Socket creado ..
Ingrese texto : hola
Servidor : chau
Ingrese texto : SALIR
Servidor : SALIR
Cliente cierra conexion...

Output del servidor:

Socket creado...
Se hace el socket bind ..
Servidor en modo escucha ...
Del cliente: hola
: chau
Del cliente: SALIR
: SALIR
Salgo del servidor...