Sistemas Operativos
2016/2017

Práctica 2

Comunicaciones en red
Y
Concurrencia

“COMUNICACIONES EN RED”

Implementar una aplicación cliente-servidor mediante la utilización de sockets cuya funcionalidad será la ejecución remota de comandos sin parámetros. El ejercicio constará de dos partes (procesos): el cliente que enviará el comando a ejecutar al servidor y el servidor que recibirá la petición desde el cliente, ejecutará el comando y devolverá los resultados al cliente.

El cliente se ejecutará en la máquina local y su función es establecer la comunicación con el servidor, enviar al servidor el comando, recibir los resultados y mostrarlos por pantalla. Para ello, el usuario escribirá en el prompt el comando:

ClienteRemoto IP_Servidor Comando

El servidor se lanzará con la orden **ServidorRemoto** y debe estar en todo momento escuchando por el puerto 9999, preparado para aceptar conexiones. Tras una petición de conexión por parte de un cliente, debe crear un hijo que será el encargado de realizar todas las operaciones necesarias relacionadas con la ejecución de comando solicitado por el cliente.

La puntuación del ejercicio será la siguiente:

- Implementación correcta del cliente y su estructura. **(1,5 puntos)**
- Implementación correcta del servidor y su estructura. **(1,5 puntos)**
- Ejecución remota de comandos. **(3 puntos)**

LLAMADAS AL SISTEMA RELACIONADAS CON LAS COMUNICACIONES EN RED

Estructuras necesarias

<sys/socket.h>, **struct sockaddr**.
<netinet/in.h>, **struct in_addr; struct sockaddr_in** (incluye un campo in_addr).
<sys/types.h>

Llamadas al sistema principales

socket
Permite establecer el canal de comunicación, está definida en <sys/socket.h> y necesita <sys/types.h>.

Ejemplo:
conector = socket(AF_INET,SOCK_STREAM,0) //(Familia protocolos, tipo transporte, protocol).

bind
Se emplea para comunicar a la red la dirección del canal. Utiliza como parámetro la estructura sockaddr_in definida en el archivo de cabecera <netinet/in.h>

Bind(conector, addr, addrlen) //(descriptor del conector, dirección servidor, longitud dir)

listen
Disponibilidad para recibir peticiones de servicio.

listen(conector, tamaño_cola) // (descriptor del conector, cola peticiones pendientes)

accept
Bloquea al servidor en espera de peticiones de conexión por parte de los clientes.

accept(conector, addr, addrlen) //(descriptor del conector, dir. servidor, longitud dir)

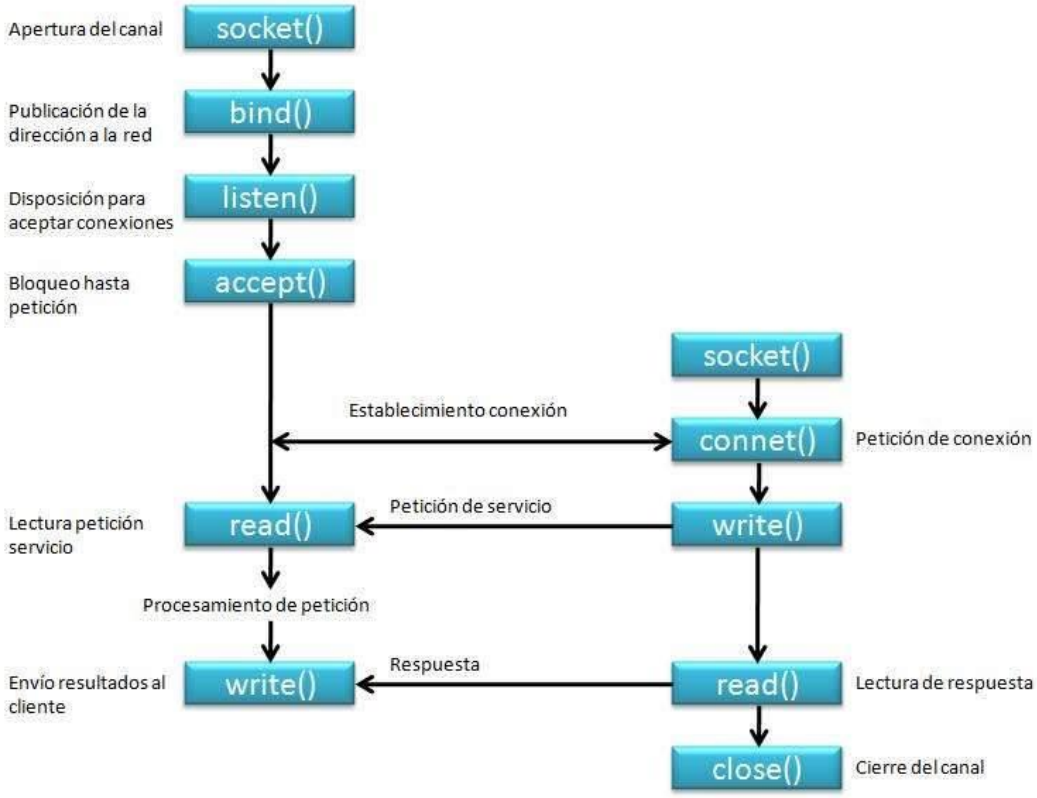
connect
Es invocada por el proceso cliente para establecer una conexión, el servidor debe estar esperando.

connect(conector_cliente, addr, addrlen) // (descriptor conector, dirección servidor, longitud)

Funciones de ayuda
inet_addr(cadena) //traduce una cadena de caracteres en dirección IP.
inet_ntoa(in_addr) //conversión inversa.

htonl, htons, ntohl, ntohs. Convierten datos unsigned long y unsigned short del formato de host al formato de la red y viceversa.

Estructura de toda aplicación cliente-servidor



“CONCURRENCIA”

Sincronización de procesos. Semáforos. Diseñar un programa de concurrencia mediante el entorno JBACI.

http://code.google.com/p/jbaci/

Enunciado

En una nave industrial existe una máquina de inyectar que deja cada pieza producida (una cada vez) en una cinta transportadora de tamaño limitado. Existe un robot que recoge las piezas de la cinta (una cada vez) y las deja en las cajas de embalaje. Finalmente, tenemos un operario que, de vez en cuando, recoge 3 piezas para realizar el control de calidad, si no hay tres piezas en la cinta lo intentará más tarde, pero no se bloquea. Resuelve el escenario anterior mediante semáforos.

Notas

Para la corrección y calificación del ejercicio además de la corrección del programa se valorarán los siguientes aspectos:

- Uso correcto de los semáforos atendiendo a los dos principales aspectos de exclusión mutua y sincronización
- Uso del interfaz gráfico para la claridad del problema

La puntuación del ejercicio será de **4 puntos.**

FECHA DE ENTREGA: **Semana del 14 al 18 de noviembre**