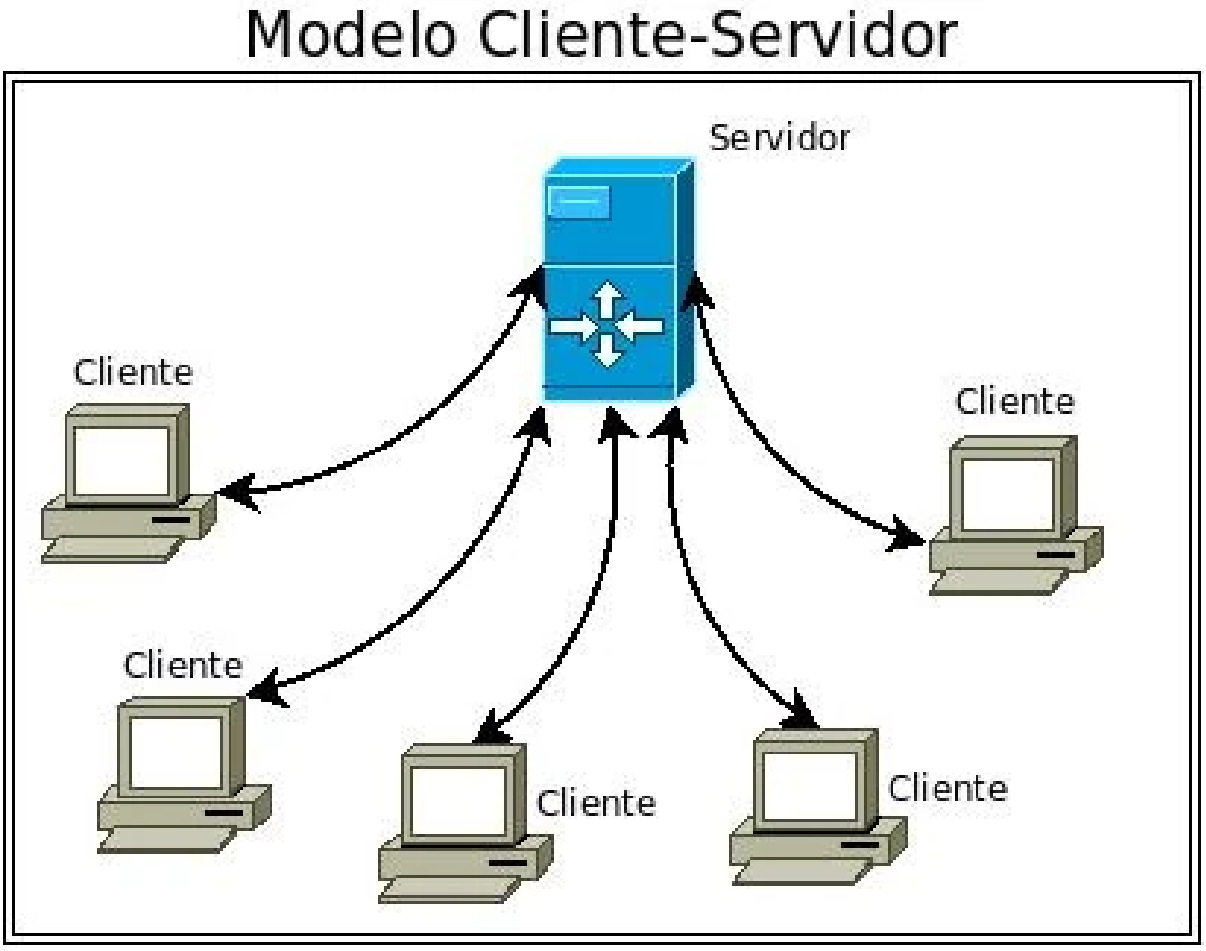
PRÁCTICA 2

COMUNICACIONES EN RED Y CONCURRENCIA



Iván Álvarez Gárcia y Andrea Gómez Bobes Grupo 1

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 en la máquina remota 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 del comando solicitado por el cliente.

El archivo servidor.c se encarga de la parte del servidor del ejercicio, en él primero se define el puerto y los structs necesarios para realizar el ejercicio. Luego se establece la conexión mediante el socket y crea un proceso hijo para ejecutar la orden, si el socket no se abre o se produce un error se notifica. Seguidamente, el servidor manda el resultado al cliente y este lo imprime.

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>

#define PUERTO 9999

int main(){

    FILE *file;
    char resultado[250];
    char buffer[200];
    struct sockaddr_in s_addr, c_addr;
    socklen_t size_a;
    pid_t pid;

    //abrimos el socket

    int socketfd = socket(AF_INET, SOCK_STREAM, 0);

    if(socketfd == -1){
        perror("No se pudo abrir el socket \n");
        exit(-1);
    }else{
        bzero((char *) &s_addr, sizeof(s_addr));    //inicializamos a 0
        s_addr.sin_family = AF_INET;
        s_addr.sin_port = htons(PUERTO);
        s_addr.sin_addr.s_addr = INADDR_ANY;

        //comprobamos que la dirección es correcta
        if(bind(socketfd, (struct sockaddr *) &s_addr, sizeof(s_addr)) < 0){
            perror("Error \n");
        }else{
            listen(socketfd, 2);    //cola de 2 peticiones

            while(1){

                printf("Aceptando peticion \n");
                pid = fork();   //creamos al proceso hijo

                if(pid == 0){
                    //CÓDIGO DEL PROCESO HIJO
                    size_a = sizeof(c_addr);

                    //aceptamos la petición del cliente
                    int socketfd2 = accept(socketfd, (struct sockaddr *) &c_addr, &size_a);
```

```
                if(socketfd2 < 0){

                    perror("Error aceptando conexion");

                }else{

                    bzero(buffer, sizeof(buffer)); //inicializamos a 0
                    read(socketfd2, buffer, sizeof(buffer)); //leemos del socket

                    file = popen(buffer,"r"); //guardamos el resultado del comando

                    //escribimos en el socket el resultado del comando
                    while(fgets(resultado,sizeof(resultado),file) != NULL){
                        write(socketfd2, resultado, sizeof(resultado));
                    }
                    //cerramos todo
                    pclose(file);
                    close(socketfd2);
                }
            }else{
                //CÓDIGO DEL PROCESO PADRE
                wait(NULL);
            }
        }
    }
    return 0;
}
```

El archivo cliente.c se encarga de la parte cliente. Este recibe por parámetros la dirección IP del servidor y el comando a ejecutar. Establece la conexión mediante socket con el servidor y envía el comando a ejecutar en la parte del servidor.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>

#define PUERTO 9999

int main(int argc, char *argv[]){
    char buffer[200];
    struct sockaddr_in s_addr;
    struct hostent *server;

        //abrimos el socket
        int socketfd = socket(AF_INET, SOCK_STREAM, 0);

        if(socketfd >= 0){
            //inicializamos a 0
            bzero((char*) &s_addr, sizeof(s_addr));

            s_addr.sin_port = htons(PUERTO);
            s_addr.sin_family = AF_INET;
            s_addr.sin_addr.s_addr = inet_addr(argv[1]);

            //petición de conexión
            if(connect(socketfd, (struct sockaddr *) &s_addr, sizeof(s_addr)) < 0){
                perror("Error al conectar al socket");
            }else{
                //escribimos en el socket
                write(socketfd,argv[2],sizeof(argv[2]));

                //leemos el resultado que nos pasa el servidor
                read(socketfd, buffer, sizeof(buffer));
                printf("%s", buffer); //imprimimos por pantalla el resultado
                //cerramos el socket
                close(socketfd);
            }

        }else{
            perror("Error al crear el socket");
        }

    return 0;
}
```

Aquí se muestra el resultado:

CONCURRENCIA

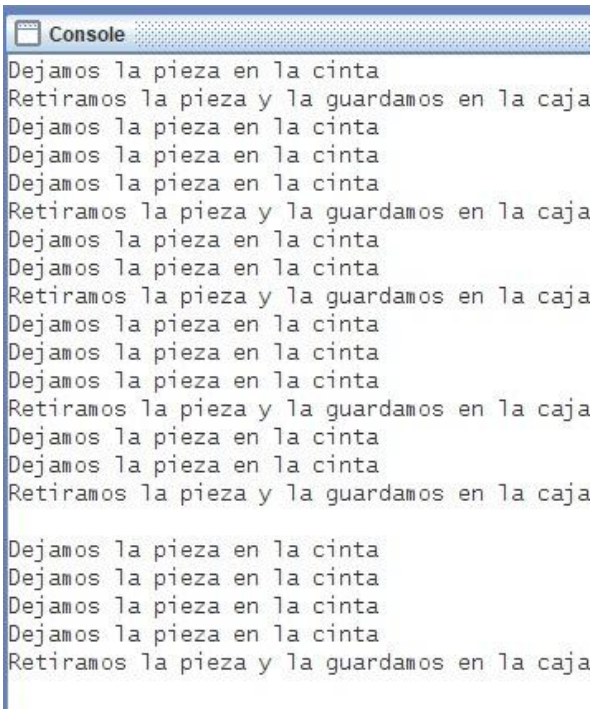
En una nave industrial hay una máquina de inyectar que produce piezas de una en una y las deja en una cinta transportadora de tamaño 10. Además, hay un robot que coge las piezas de una en una y las deja en la caja de embalaje. Implementar mediante semáforos la solución al problema suponiendo que la máquina produce 100 piezas y que la cinta es la sección crítica.

Este programa inicializa los valores de la cantidad de piezas tanto en la cinta como en la caja a 0. La función main del programa llama a ambas funciones que se ejecutan en paralelo mediante cobegin.
La función inyectar se encarga de la máquina que crea piezas y las deja en la cinta, esta comprueba que esta acción se realiza un máximo de 100 veces y que sólo se pueden colocar como máximo 10 piezas en la cinta. Se espera a que la señal para poner piezas sea 1 y entonces se aumenta en 1 el valor de la cinta. Adicionalmente, se muestra por consola la acción y se aumenta en 1 el contador de máximo de veces que se repite la acción.Por último se lanza la señal con el valor de turno.
La función robot representa al robot que se encarga de meter piezas en la caja, esta comprueba que esta acción se realiza un máximo de 100 veces y que deben haber piezas en la cinta. Una vez comprobado todo se disminuye en 1 el número de piezas que hay en la cinta y se aumenta en 1 el número de piezas que hay en la caja. Se muestra por consola la acción.
Finalmente, se muestra por consola el valor de cinta y de caja.

```
int cinta = 0; //Variable para contar el número de piezas en la cinta
int caja = 0; //Variable para ir añadiendo piezas en la caja
binarysem turno = 1; //Variable para el semáforo para controlar la producción
void inyectar() { //Función que representa a la máquina que crea las piezas
        int contador; //Variable para controlar que no se crean más de 100 piezas
        while(contador<=100){ //Comprobamos que se hacen como mucho 100 piezas
                if(cinta<10){ //Comprobamos que no hay más de 10 piezas en la cinta
                        wait(turno); //Esperamos a que el semáforo sea 1 para que sea turno de crear la pieza
                        cinta = cinta +1; //Sumamos una unidad al número de piezas que hay en la cinta
                        cout << "Dejamos la pieza en la cinta" <<endl; //Mostramos por consola que hemos creado la pieza y la hemos dejado en la cinta
                        contador++; //Aumentamos el contador
                }else{
                        contador--; //Si hay más de diez piezas el contador disminuye
                }
                signal(turno); //Mandamos la señal para cambiar el binario de turno
        }
}
void robot() { //Función que representa al robot que guarda las piezas en la caja
        int contador2; //Variable que permite controlar la cantidad de veces que se realiza la acción
        while(contador2 <= 100){ //Se repite la acción hasta las 100 veces
                if(cinta >0){//Se comprueba que que en la cinta hay piezas
                        contador2++; //Aumentamos el contador
                        cinta--; //Restamos piezas de la cinta
                        caja= caja + 1; //Aumntamos las piezas en la caja
                        cout << "Retiramos la pieza y la guardamos en la caja" <<endl; //Mostramos por consola que hemos guardado la pieza en la caja
                }
        }
}

void main(){
        cobegin{ //Llamamos a ambas funciones
                inyectar();
                robot();
        }
        cout << cinta <<endl; //Mostramos por pantalla el valor de la cinta
        cout << caja <<endl; //Mostramos por pantalla el valor de la caja
}
```

Se muestra aquí el resultado:


Console
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Dejamos la pieza en la cinta
Retiramos la pieza y la guardamos en la caja