2023/2024

PRÁCTICA 3

SISTEMAS OPERATIVOS

JAIME HERNANDEZ DELGADO
UNIVERSIDAD DE ALICANTE

Ejercicio 1:

El servidor se lanzará con la orden Servidor 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 para la transferencia del archivo. El servidor debe estar en una máquina distinta al cliente.

```c
int KBUFFER = 1024;

int main(int argc, char const *argv[]) {
    struct sockaddr_in client, server;
    int fd1, fd2;
    int fichero;
    int tam;
    char buffer[KBUFFER];

    fd1 = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = htons(9999);
    server.sin_addr.s_addr = inet_addr(argv[1]);

    connect(fd1, (struct sockaddr *)&server, sizeof(struct sockaddr_in));

    do{
        tam = read(fd1, buffer, KBUFFER - 1);
        if(tam > 0){
            buffer[tam] = '\0';
            printf("%s", buffer);
        }
    }
    while(tam > 0);
    close(fd1);

    return 0;
}
```

Este código en C implementa un cliente de socket que se conecta a un servidor a través del protocolo TCP. Primero, se crea un socket utilizando la función socket, y se configuran los detalles de la dirección del servidor, como la familia de direcciones (IPv4), el puerto (9999), y la dirección IP del servidor proporcionada como argumento de línea de comandos.

Luego, el programa establece una conexión al servidor mediante la función connect. Después de la conexión, entra en un bucle de lectura donde lee datos del socket en bloques de hasta 1023 bytes (definido por la constante KBUFFER). Los datos leídos se imprimen en la consola. El bucle continúa hasta que la función read devuelve 0, indicando el final de la transmisión.

Finalmente, una vez completada la lectura de datos, el programa cierra el descriptor de archivo del socket con close y termina la ejecución. Este código es una implementación básica de un

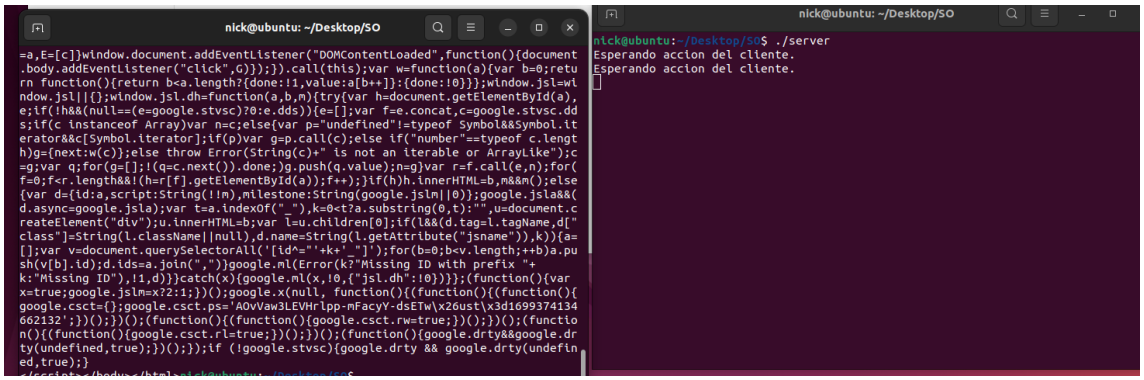
cliente de socket TCP en C, diseñado para conectarse a un servidor y recibir datos de manera continua hasta que la transmisión se completa.

```c
int KBUFFER = 1024;

int main(int argc, char const *argv[]) {
    struct sockaddr_in client, server;
    int fd1, fd2;
    int fichero;
    pid_t pid; // puede que no lo necesite
    char buffer[KBUFFER];
    int tam;

    fd1 = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = htons(9999);
    server.sin_addr.s_addr = INADDR_ANY;

    bind(fd1, (struct sockaddr *)&server, sizeof(struct sockaddr_in));
    listen(fd1, 5);

    do{
        if(fork() == 0){
            tam = sizeof(struct sockaddr_in);
            fd2 = accept(fd1, (struct sockaddr *)&client, &tam);
            fichero = open("Google.html", O_RDONLY);
            do{
                tam = read(fichero, buffer, KBUFFER - 1);
                if(tam > 0) write(fd2, buffer, tam);
            }
            while(tam > 0);
            close(fd2);
            close(fichero);
        }
```

En este bucle principal, el servidor acepta conexiones entrantes utilizando accept. Cuando se establece una conexión, se crea un nuevo proceso hijo utilizando fork(). El proceso hijo lee el contenido del archivo "Google.html" y lo envía al cliente a través del socket. El proceso padre espera la finalización del proceso hijo antes de continuar.

Este código implementa un servidor que puede manejar múltiples clientes concurrentemente mediante la creación de procesos hijos para cada conexión entrante. Cada proceso hijo se encarga de enviar el contenido del archivo "Google.html" al cliente correspondiente.

El resultado es el siguiente:



Ejercicio 2:

El código implementa el problema clásico del productor-consumidor utilizando semáforos y una interfaz gráfica en un entorno específico. Se definen semáforos (accesoBuffer, buffer, consumer) para sincronizar la producción y el consumo de elementos en un búfer limitado. La interfaz gráfica crea objetos representativos de los elementos en el búfer, productores y consumidores. Las funciones add y take simulan la producción y el consumo, cambiando los colores de los objetos gráficos. Las funciones productor y consumidor ejecutan en bucles infinitos, esperando y señalizando semáforos para garantizar la exclusión mutua y la sincronización entre productores y consumidores.

El código utiliza funciones gráficas específicas (create, makevisible, changecolor) que deben estar disponibles en el entorno de ejecución. Aunque las funciones producir y consumir están definidas, no parecen realizar acciones específicas en este código. El programa se inicia en la función main, inicializando semáforos, creando la interfaz gráfica y ejecutando concurrentemente las funciones productor y consumidor utilizando cobegin.