

Informe sobre un servidor e un cliente TCP

Redes

Grupo 03

{nicolassantiago.gomez,marcos.garcia.blanco}@rai.usc.gal

17 de outubro do 2025

I. INTRODUCCIÓN

Con motivo de aprendizaxe respecto ao protocolo TCP, a súa especificación POSIX e a súa implantación nun sistema Linux, o profesorado da materia *Redes* propuxo a creación dun par servidor e cliente usando o devandito protocolo.

Neste documento avaliaranse dous pares: un que simplemente consiste dun servidor que envía unha mensaxe a un cliente que o recibe e imprime, e outro par que converterá a maiúsculas un arquivo subministrado polo cliente e procesado polo servidor. En ambos os dous casos, avaliarase o código empregado para poder efectuar estas funcións e discutirase os resultados obtidos baixo as condicións de cada apartado do exercicio a elaborar.

II. APARTADO 1.C

Neste apartado do informe analizarase se é posible que, empregando unha única chamada á función `recv()`, o programa `cliente.c` poida recibir dúas mensaxes enviadas mediante dúas chamadas á función `send()` no servidor. Analizaremos dous casos diferentes: un empregando a función `sleep()` e outro sen usar esta función. Amosaremos o código correspondente para cada un dos casos e os resultados que se obteñen.

En ambos casos, o código empregado para o envío da mensaxe ao cliente é idéntico e non presenta ningunha modificación con respecto ao desenvolto e entregado no apartado 1.a.

A. Sen empregar `sleep()`

Nesta sección veremos o que sucede se non se emprega unha función `sleep()` no `cliente.c`. O código empregado para este caso é o seguinte:

```
int main(int argc, char const *argv[])
{
    //Definicion de variables

    // Comprobar que se ha pasado un numero de argumentos correcto (IP y puerto)
    if (argc != 3)
    {
        printf("Introducir IP y puerto como argumentos\n");
        exit(EXIT_FAILURE);
    }
    //Copiar IP en una variable
    char ip[INET_ADDRSTRLEN];
    strncpy(ip, argv[1], INET_ADDRSTRLEN); //Copia la IP en la variable ip

    //Convertir IP de texto a binario ya que por comandos sale en formato de cadena de caracteres
    struct in_addr ipBinario;

    //Hace la conversion y comprueba que se ha hecho correctamente
    if (inet_pton(AF_INET, ip, (void *)&ipBinario.s_addr) != 1)
    {
        perror("Error al convertir la IP a formato binario\n");
        exit(EXIT_FAILURE);
    }
    //printf("IP en formato binario: %X\n", ipBinario.s_addr);

    // Convertir puerto a entero y almacenalo en tipo de entero entre 0 y 65535
    // ya que el puerto solo puede estar en este rango de valores
    uint16_t puerto;
    puerto = (uint16_t)atoi(argv[2]);

    //Crear socket cliente TCP
```

```

int socketClient;
socketClient = socket(AF_INET, SOCK_STREAM, 0);
//AF_INET para IPv4, SOCK_STREAM para TCP

if (socketClient < 0)
{
    perror("Error al crear el socket.\n");
    exit(EXIT_FAILURE);
}

//Configuración de la dirección del servidor
struct sockaddr_in direccionServidor;

//Configuración struct sockaddr_in
uint16_t puertoFormatRed = htons(puerto); //Convertir puerto a formato red

direccionServidor.sin_family = AF_INET; //Dirección IPv4
direccionServidor.sin_port = puertoFormatRed; //Puerto en formato red
direccionServidor.sin_addr = ipBinario; //Dirección IP del servidor

//Conectar al servidor usando la función connect
if (connect(socketClient, (struct sockaddr *)&direccionServidor, sizeof(direccionServidor)) != 0)
//Se conecta y comprueba si hay error si lo hay se cierra el socket y se sale del programa
{
    perror("Error al conectar con el servidor.\n");
    close(socketClient);
    exit(EXIT_FAILURE);
}

//Variable para almacenar el mensaje recibido del servidor
char mensajeRecibido[1024];

//Recepción de los datos del servidor
ssize_t n;
size_t bytesMensaje = 1000;
n = recv(socketClient, mensajeRecibido, bytesMensaje, 0);
if (n < 0)
{
    perror("Error al recibir datos del servidor.\n");
}
else if (n == 0)
{
    printf("El servidor ha cerrado la conexión.\n");
}
else
{
    mensajeRecibido[n] = '\0'; //Asegurar que el mensaje recibido es una cadena de caracteres
    printf("Mensaje recibido del servidor: %s\n", mensajeRecibido);
    printf("Bytes recibidos: %zd\n", n);
}

close(socketClient); //Cerrar el socket cliente ya que no se necesita mas
printf("\nEl servidor ha cerrado la conexión.\n");
return 0;
}

```

Como se puede observar en el código, no se emplea ninguna llamada a la función `sleep()` y se utiliza una única llamada a `recv()`. Al ejecutar este código junto con el del servidor, comprobamos que, independientemente del número de veces que se ejecute, únicamente se recibe una de las dos mensajes enviadas por el servidor, a la primera de ellas. Este resultado puede corroborarse en la siguiente imagen que muestra la ejecución paralela de cliente y servidor:

```

# Left Terminal (Server):
$ gcc -Wall -o servidor.o servidor.c -L .
$ gcc -Wall -o cliente.o cliente.c -L .
$ ./servidor.o 7777
Servidor escuchando
Servidor conectado al cliente
IP conectada: 127.0.0.1
Socket pechado
$ ./servidor.o 8888
Servidor escuchando
Servidor conectado al cliente
IP conectada: 127.0.0.1
Socket pechado
$

# Right Terminal (Client):
$ gcc -Wall -o servidor.o servidor.c -L .
$ gcc -Wall -o cliente.o cliente.c -L .
$ ./cliente.o 127.0.0.1 7777
Mensaje recibido del servidor: Mensaje genérica 1
Bytes recibidos: 19
El servidor ha cerrado la conexión.
$ ./cliente.o 127.0.0.1 8888
Mensaje recibido del servidor: Mensaje genérica 1
Bytes recibidos: 19
El servidor ha cerrado la conexión.

```

Figura 1: Salida sin `sleep()`

B. Empleando `sleep()`

Ahora analizaremos lo que sucede cuando se utiliza la función `sleep()` para esperar a que el servidor envíe ambos mensajes. Como se puede apreciar en el siguiente código, se emplea un `sleep` de 10 segundos (`sleep(10)`) antes de ejecutar el `recv()`:

```

//Variable para almacenar el mensaje recibido del servidor
char mensajeRecibido[1024];
sleep(10); //Esperar 10 segundos para asegurarse de que el servidor ha enviado el mensaje
//Recepción de los datos del servidor
ssize_t n;
size_t bytesMensaje = 1000;

```

```

n = recv(socketClient, mensajeRecibido, bytesMensaje, 0);
if (n < 0)
{
    perror("Error al recibir datos del servidor.\n");
}
else if (n == 0)
{
    printf("El servidor ha cerrado la conexion.\n");
}
else
{
    mensajeRecibido[n] = '\0'; //Asegurar que el mensaje recibido es una cadena de caracteres
    printf("Mensaje recibido del servidor: %s\n", mensajeRecibido);
    printf("Bytes recibidos: %zd\n", n);
}
}

```

Co uso da función de espera, ao ser unha conexión de tipo TCP, é dicir que funciona a base dun fluxo de bytes, o cliente cun único `recv()` e dándolle tempo ao servidor a enviar ambas mensaxes é capaz de recibir as dúas frases. Este resultado pódese comprobar na seguinte imaxe que contén a saída do programa desta versión:

Figura 2: Saída con `sleep()`

C. Resultados do apartado 1 c

Como resultados para este apartado, como era de esperar, sen empregar o `sleep` e un único `recv` o cliente só e capaz de recibir unha mensaxe, mentres que se se emprega a función de espera no cliente este recibe e imprime ambas sentencias.

III. APARTADO 1.D

O obxectivo é analizar como un cliente TCP pode recibir dúas mensaxes enviadas consecutivamente por un servidor mediante unha única conexión empregando un bucle `while(recv(...))`, sen necesidade de esperar explicitamente empregando `sleep()`. Ademais, comprobarase o efecto de variar o tamaño de bytes a recibir en cada execución.

O código empregado para o servidor é exactamente o mesmo que para o apartado 1 c, non se lle fai ningún cambio. Por outra banda o código do cliente si que se modifica bastante, engádese o seguinte bucle `while` e elimínase a parte do `recv()` anterior:

```

char mensaje[1024];
ssize_t n;
size_t bytesMensaje = 1024; //bytes a recibir

while ((n = recv(socketClient, mensaje, bytesMensaje, 0)) > 0) {
    mensaje[n] = '\0';
    printf("Mensaje recibido del servidor: %s\n", mensaje);
    printf("Bytes recibidos: %zd\n", n);
}

if (n < 0) {
    perror("Error al recibir el mensaje");
}
}

```

Vese que o cambio máis significativo é ese bucle `while` no que podemos modificar a variable `bytesMensaje` a diferentes valores (10, 20 e 1024 nas nosas execucións). Ao executar o código

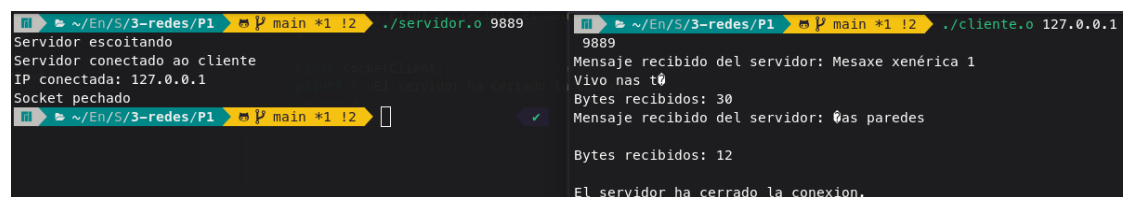
cambiando o número de bytes da mensaxe obtemos os seguintes resultados:



```
~/En/S/3-redes/P1 main *1 !2 ./servidor.o 7777
Servidor escoitando
Servidor conectado ao cliente
IP conectada: 127.0.0.1
Socket pechado

~/En/S/3-redes/P1 main *1 !2 ./cliente.o 127.0.0.1 7777
Mensaje recibido del servidor: Mesaxe xen
Bytes recibidos: 10
Mensaje recibido del servidor: érica 1
V
Bytes recibidos: 10
Mensaje recibido del servidor: ivo nas t0
Bytes recibidos: 10
Mensaje recibido del servidor: 0as parede
Bytes recibidos: 10
Mensaje recibido del servidor: s
Bytes recibidos: 2
El servidor ha cerrado la conexion.
```


Figura 3: Saída con 10 bytes



```
~/En/S/3-redes/P1 main *1 !2 ./servidor.o 9889
Servidor escoitando
Servidor conectado ao cliente
IP conectada: 127.0.0.1
Socket pechado

~/En/S/3-redes/P1 main *1 !2 ./cliente.o 127.0.0.1 9889
Mensaje recibido del servidor: Mesaxe xenérica 1
Vivo nas t0
Bytes recibidos: 30
Mensaje recibido del servidor: 0as paredes
Bytes recibidos: 12
El servidor ha cerrado la conexion.
```

Figura 4: Saída con 30 bytes



```
~/En/S/3-redes/P1 main *1 !2 ./servidor.o 8888
Servidor escoitando
Servidor conectado ao cliente
IP conectada: 127.0.0.1
Socket pechado

~/En/S/3-redes/P1 main *1 !2 ./cliente.o 127.0.0.1 8888
Mensaje recibido del servidor: Mesaxe xenérica 1
Vivo nas túas paredes
Bytes recibidos: 42
El servidor ha cerrado la conexion.
```

Figura 5: Saída con 1024 bytes

Como se pode ver nas imaxes anteriores, ao facer o bucle *while* a mensaxe envíase por partes de *n* en *n* bits, polo tanto se poñemos un buffer de *bytes* recibidos grande recibiríamos todo o mensaxe cunha única iteración.

Este resultado débese a que o protocolo TCP, como xa se explicou no apartado anterior, envía os datos nun fluxo continuo de *bytes*, polo tanto o bucle do cliente recibe en bloques a mensaxe. Este bucle continua executándose ata que `recv()` deixa de recibir datos do servidor e devolve como resposta un 0.

Analizando esta versión pódese apreciar que se recibe sempre a mensaxe completa, da igual de que tamaño sexa; ao empregarse o bucle se a variable que indica o número de bytes enviados é pequena, entón a mensaxe recíbese fragmentada. Por asegurar a menos que ocorra un erro de transmisión a recepción da mensaxe completo podemos asegurar que é unha mellor implantación ca do apartado c.

IV. APARTADO 3

Esta sección cubrirá o apartado tres do exercicio proposto, o cal comprende un outro servidor e cliente. O cliente lee un arquivo cuxa ruta é subministrada pola liña de comandos e envía os contidos deste a un servidor especificado asemade na liña de comandos após conectarse ao mesmo. O servidor recibe os datos proporcionados polo cliente e, empregando a función `toupper()`, converte o texto recibido en maiúsculas e o envía de volta ao cliente. O cliente recibe

esta liña en todo maiúsculas e imprímela nun arquivo cuxo nome é o mesmo que o orixinal pero en todo maiúsculas. O cliente repite o ciclo de envío e recepción ate que non haia máis que ler ou escribir nos arquivos correspondentemente.

A. Perspectiva do cliente

```
if ((arquivoLectura = fopen(argv[1], "r")) != NULL && (arquivoEscritura = fopen(arquivosMaiusculas(argv[1], "w+")) != NULL){
    char linha[1024];

    // Conectarse ao servidor
    if (connect(nsocketRecibir, (struct sockaddr*)&socketEnviar, sizeof(socketEnviar)) == -1){
        perror("Erro ao conectarse ao servidor");
        close(nsocketRecibir);
        return EXIT_FAILURE;
    }

    printf("Cliente conectado a %s\n", argv[2]);

    while (fgets(linha, sizeof(linha), arquivoLectura) != NULL) {
        // Enviar os datos ao servidor
        if (send(nsocketRecibir, linha, (size_t) strlen(linha), 0) != (ssize_t) strlen(linha))
            printf("Erro enviado os datos ao servidor\n");

        // Recibir os datos menos o último posterior, que poñemos cun \0 para que sexa con fin nulo
        ssize_t nrecv;
        if (nrecv = recv(nsocketRecibir, linha, sizeof(linha) - 1, 0) <= 0)
            printf("Erro ao recibir os datos do servidor\n");

        linha[nrecv] = '\0';

        fprintf(arquivoEscritura, "%s", linha);
    }
}
```

Código de envío de datos de *clientemay.c*

O cliente non presenta funcionalidade especial respecto a un cliente TCP: usando uns *socket* tenta conectarse a un servidor especificado mediante a instrución `connect()`. Se non falla, informa ao usuario e recupera os datos do arquivo, en paquetes de liñas de texto de ate mil vinte e catro caracteres, ate que non haxa máis que ler (NULL) mediante a instrución `fgets()`. Envía esta liña ao servidor con `send()` e cando fina a instrución, inmediatamente comeza a recibilos con `recv()`. Finalmente, imprime o que fose recibido dende o servidor no arquivo usando `fprintf()`, comezando unha nova lectura dunha liña se procede.

O cliente en ningún momento pode percibir que houbese paralelismo no servidor onde se procesan as instrucións, nin sabe o número de conexións deste. Simplemente envía o lido do ficheiro e recibe o que o servidor envíalle, sexa correcto ou non.

B. Perspectiva do servidor

```
socketDatos = accept(socketServidor, (struct sockaddr*)&datos, &tamanho); //Aceptar conexiones entrantes

if(socketDatos < 0) //comprobar que no hay errores en accept
{
    perror("Error en accept.\n");
    close(socketServidor);
    exit(EXIT_FAILURE);
}

inet_ntop(AF_INET, &(datos.sin_addr), ipCliente, INET_ADDRSTRLEN); //pasa la ip recibida en binario a formato textual
printf("Dirección IP cliente conectado: %s: %d\n", ipCliente, ntohs(datos.sin_port));
//ntohs para convertir del entero de orden de red a entero de orden de host

char linea[1024];
int n, msgEnv;

//Recibir datos del cliente hasta que se cierre la conexion
while ((n = recv(socketDatos, linea, sizeof(linea) - 1, 0)) > 0)
{
    sleep(3);
    //n = numero de bytes recibidos -1
    linea[n] = '\0'; //Se añade el caracter nulo al final de la cadena recibida para asegurar que es una cadena

    //Pasar a mayusculas
    for (int i = 0; i < n; i++){
        linea[i] = toupper(linea[i]);
    }

    //Enviar datos al cliente
    msgEnv = send(socketDatos, linea, n, 0);
}
```

```

if (msgEnv < 0) //comprobar que no hay errores en send
{
    perror("Error en send.\n");
    close(socketDatos);
    close(socketServidor);
    exit(EXIT_FAILURE);
}
}
if (n<0) //comprobar que no hay errores en recv
{
    perror("Error en recv.\n");
    close(socketDatos);
    close(socketServidor);
    exit(EXIT_FAILURE);
}
}

```

Código de entrada e saída de datos de *servidormay.c*

O servidor está deseñado coma un servidor TCP básico que após inicializar as súas variables escoita calquera *host* que desexe conectarse a el. Comeza aceptando a petición de conexión dun cliente, e en caso que poda establecela correctamente, comeza a recibir liñas de texto. Estas liñas son postas con `toupper()` a maiúsculas e enviadas á conexión do cliente coa operación `()`. Este proceso repítese ate que non haxa ningunha liña de texto a recibir dende a IP.

O servidor descrito en *servidormay.c* non fai transaccións das peticións doutros clientes mentres que o cliente actual non se desconecte. Os clientes pendentes para a conexión pódense conectar grazas a función `connect()`, mais serán postos nunha cola. Perante que non se volva a executar novamente `connect()`, o `socket` fará referencia ao membro máis posteriormente conectado en todas as súas chamadas, sexan de `recv()` ou ben de `send()`.

C. Resultados da conexión

```

RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ ./clientenay.o texto.txt 1
72.25.45.104 5555
Cliente conectado a 172.25.45.104
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ cat texto.txt
I. DISPOSICIONES XERAIS
MINISTERIO DE FACENDA E FUNCIÓN PÚBLICA
15861 Orde HFP/1031/2021, do 29 de setembro, pola que se establecen o procedemento e o formato da información que proporcionarán as entidades do sector público estatal, autonómico e local para o seguimento do cumprimento de fitos e obxectivos e de execución orzamentaria e contable das medidas dos compoñentes do Plan de recuperación, transformación e resiliencia.

O Plan de recuperación transformación e resiliencia, en diante, PRTR, configúrase como un instrumento promovido no ámbito da Unión Europea orientado a mitigar os impactos da pandemia de COVID-19, así como a transformar a sociedade, cos obxectivos de modernizar o tecido produtivo, impulsar a «descarbonización» e o respecto ao ambiente, fomentar a dixitalización e a mellora das estruturas e recursos destinados a investigación e formación, e acadar, en última instancia, unha maior capacidade da sociedade para superar problemas como a pandemia, conforme o marco establecido no Regulamento (UE) 2021/241 do Parlamento Europeo e do Consello, do 12 de febreiro de 2021, polo que se establece o mecanismo de recuperación e resiliencia.

O artigo 8 do citado regulamento, relativo á execución do PRTR, determina que «a Comisión executará o mecanismo en réxime de xestión directa de conformidade coas normas pertinentes adoptadas en virtude do artigo 322 do TFUE, en particular o Regulamento financeiro e o Regulamento (UE, Euratom) 2020/2092 do Parlamento Europeo e do Consello», e os Estados membros, conforme o artigo 22 do Regulamento (UE) 2021/241, do 12 de febreiro, terán a condición de beneficiarios ou prestameiros de fondos no marco do mecanismo.
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ cat TEXTO.TXT
I. DISPOSICIONES XERAIS
MINISTERIO DE FACENDA E FUNCIÓN PÚBLICA
15861 Orde HFP/1031/2021, DO 29 DE SETEMBRO, POLA QUE SE ESTABLECEN O PROCEDIMIENTO E O FORMATO DA INFORMACIÓN QUE PROPORCIONARÁN AS ENTIDADES DEL SECTOR PÚBLICO ESTATAL, AUTONÓMICO Y LOCAL PARA O SEGUIMIENTO DO CUMPRIMENTO DE FITOS Y OBJETIVOS Y DE EJECUCIÓN ORZAMENTARIA Y CONTABLE DAS MEDIDAS DOS COMPONENTES DO PLAN DE RECUPERACIÓN, TRANSFORMACIÓN E RESILIENCIA.

O PLAN DE RECUPERACIÓN TRANSFORMACIÓN E RESILIENCIA, EN DIANTE, PRTR, CONFIGURASE COMO UN INSTRUMENTO PROMOVIDO NO ÁMBITO DA UNIÓN EUROPEA ORIENTADO A MITIGAR OS IMPACTOS DA PANDEMIA DA COVID-19, ASÍ COMO A TRANSFORMAR A SOCIEDADE, COS OBJETIVOS DE MODERNIZAR O TECIDO PRODUTIVO, IMPULSAR A «DESCARBONIZACIÓN» E O RESPECTO AO AMBIENTE, FOMENTAR A DIXITALIZACIÓN E A MELLORA DAS ESTRUCTURAS E RECURSOS DESTINADOS A INVESTIGACIÓN E FORMACIÓN, E ACADAR, EN ÚLTIMA INSTANCIA, UNA MAIOR CAPACIDADE DA SOCIEDADE E PARA SUPERAR PROBLEMAS COMO A PANDEMIA, CONFORME O MARCO ESTABLECIDO NO REGULAMENTO (UE) 2021/241 DO PARLAMENTO EUROPEO E DO CONSELLO, DO 12 DE FEBREIRO DE 2021, POLO QUE SE ESTABLECE O MECANISMO DE RECUPERACIÓN E RESILIENCIA.

O ARTIGO 8 DO CITADO REGULAMENTO, RELATIVO Á EXECUCIÓN DO PRTR, DETERMINA QUE «A COMISIÓN EXECUTARÁ O MECANISMO EN RÉXIME DE XESTIÓN DIRECTA DE CONFORMIDADE COAS NORMAS PERTINENTES ADOPTADAS EN VIRTUDE DO ARTIGO 322 DO TFUE, EN PARTICULAR O REGULAMENTO FINANCIERO E O REGULAMENTO (UE, EURATOM) 2020/2092 DO PARLAMENTO EUROPEO E DO CONSELLO», E OS ESTADOS MIEMBROS, CONFORME O ARTIGO 22 DO REGULAMENTO (UE) 2021/241, DO 12 DE FEBREIRO, TERÁN A CONDICIÓN DE BENEFICIARIOS OU PRESTAMEIROS DE FONDOS NO MARCO DO MECANISMO.
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$

RAI@nicolassantiago.gome@220a045h104l: ~
RAI@nicolassantiago.gome@220a045h104l: ~$ ssh 172.25.45.104
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

O mantemento de seguranza ampliado para Applications non está activado.

É posíbel aplicar 786 actualizacións inmediatamente.
565 destas actualizacións son actualizacións de seguranza normais.
Para consultar estas actualizacións adicionais execute: apt list --upgradable

Pódense aplicar 87 actualizacións de seguranza adicionais con ESM Apps.
Saiba máis sobre como activar o servizo de ESM Apps at https://ubuntu.com/esm.

Last login: Tue Oct  7 11:30:01 2025 from 172.25.45.103
$ bash
RAI@nicolassantiago.gome@220a045h104l: ~$ cd
RAI@nicolassantiago.gome@220a045h104l: ~$ cd P2/
RAI@nicolassantiago.gome@220a045h104l: ~/P2$ ./servidormay.o 5555
Servidor escuchando...
Dirección IP cliente conectado: 172.25.45.103:39576
Datos enviados al cliente.
Cliente desconectado: 172.25.45.103:39576

.....
Puerto: 5555
Servidor escuchando...

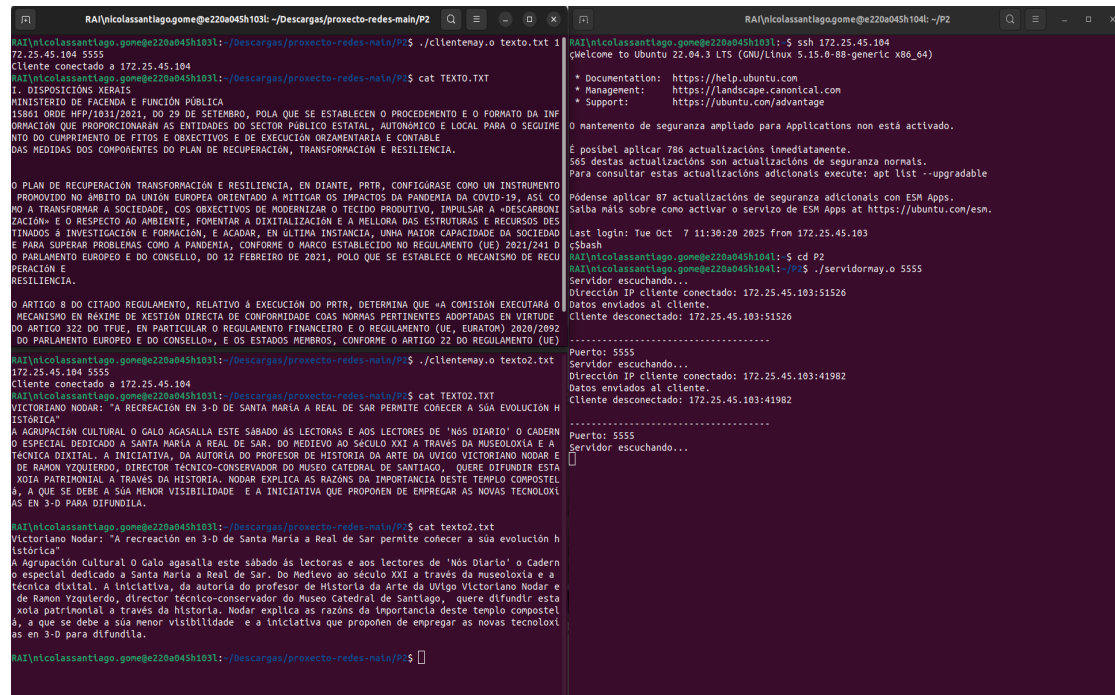
```

Figura 6: Resultados cun só cliente

O cliente conéctase ao servidor con `connect()`, e este acepta a conexión con `accept()`. O cliente lee o arquivo con `fgets()` e este enviase ao servidor, que o recibe con `recv()`. O servidor

procesado con `toupper()` e envíao de volta ao cliente con `send()`. O cliente queda parado ate que recibe os datos e non lee outra liña ate recibir a previa. O proceso de envío e recepción repítense entre ambos ate que non haxa máis fluxo de datos, e o servidor volver a estar listo a aceptar unha nova conexión.

D. Resultados da conexión



```
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ ./clientenay.o texto.txt 1
72.25.45.104 5555
Cliente conectado a 172.25.45.104
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ cat TEXTO.TXT
1. DISPOSICIÓN XERAIS
MINISTERIO DE PACENCIA E FUNCIÓN PÚBLICA
15861 ORDE HFP/1031/2021, DO 29 DE SETEMBRO, POLA QUE SE ESTABLECEN O PROCEDIMENTO E O FORMATO DA INFORMACIÓN QUE PROPORCIONARÁN AS ENTIDADES DO SECTOR PÚBLICO ESTATAL, AUTÓNOMICO E LOCAL PARA O SEGUIMIENTO DO CUMPRIMENTO DE PLANES E OBJETIVOS DE EXECUCIÓN ORZAMENTARIA E CONTABLE E DAS MEDIDAS DOS COMPONENTES DO PLAN DE RECUPERACIÓN, TRANSFORMACIÓN E RESILIENCIA.
O PLAN DE RECUPERACIÓN TRANSFORMACIÓN E RESILIENCIA, EN DIANTE, PRTR, CONFIGURASE COMO UN INSTRUMENTO PROMOVIDO NO ÁMBITO DA UNIÓN EUROPEA ORIENTADO A MITIGAR OS IMPACTOS DA PANDEMIA DA COVID-19, ASÍ COMO A TRANSFORMAR A SOCIEDADE, COS OBJETIVOS DE MODERNIZAR O TECIDO PRODUTIVO, IMPULSAR A «DESCARBONIZACIÓN» E O RESPECTO AO AMBIENTE, FOMENTAR A DIXITALIZACIÓN E A MELLORA DAS ESTRUTURAS E RECURSOS DESTINADOS A INVESTIGACIÓN E FORMACIÓN, E ACADAR, EN ÚLTIMA INSTANCIA, UNHA MAIOR CAPACIDADE DA SOCIEDADE PARA SUPERAR PROBLEMAS COMO A PANDEMIA, CONFORME O MARCO ESTABLECIDO NO REGULAMENTO (UE) 2020/2282 DO PARLAMENTO EUROPEO E DO CONSELLO, DO 12 DE FEVEREIRO DE 2021, POLO QUE SE ESTABECE O MECANISMO DE RECUPERACIÓN E RESILIENCIA.
O ARTIGO 8 DO CITADO REGULAMENTO, RELATIVO Á EXECUCIÓN DO PRTR, DETERMINA QUE «A COMISIÓN EXECUTARÁ O MECANISMO EN MÁXIMA DE XESTIÓN DIRECTA DE CONFORMIDADE COAS NORMAS PERTINENTES ADOPTADAS EN VIRTUDE DO ARTIGO 322 DO TRFUE, EN PARTICULAR O REGULAMENTO FINANCEIRO E O REGULAMENTO (UE) EUROPEO 2020/2092 DO PARLAMENTO EUROPEO E DO CONSELLO», E OS ESTADOS MIEMBROS, CONFORME O ARTIGO 22 DO REGULAMENTO (UE) VICTORIANO NODAR: «A RECREACIÓN EN 3-D DE SANTA MARÍA A REAL DE SAR PERMITE COÑECER A SÚA EVOLUCIÓN HISTÓRICA»
A AGRUPACIÓN CULTURAL O GALO AGASALLA ESTE SÁBADO ÁS LECTORAS E AOS LECTORES DE 'NÓS DIARIO' O CADERN O ESPECIAL DEDICADO A SANTA MARÍA A REAL DE SAR. DO MEDIEVO AO SÉCULO XXI Á TRAVÉS DA MUSEOLOXÍA E A TÉCNICA DIXITAL. A INICIATIVA, DA AUTORA DO PROFESOR DE HISTORIA DA ARTE DA UNIVG VICTORIANO NODAR E DE RAMON YZQUIERDO, DIRECTOR TÉCNICO-CONSERVADOR DO MUSEO CATEDRAL DE SANTIAGO, QUERE DIFUNDIR ESTA XOIA PATRIMONIAL Á TRAVÉS DA HISTORIA. NODAR EXPLICA AS RAZÓN DA IMPORTANCIA DESTA TEMPLO COMPOSTELÁ, Á QUE SE DEBE A SÚA MENOR VISIBILIDADE E A INICIATIVA QUE PROPONEN DE EMPREGAR AS NOVAS TECNOLOXÍAS EN 3-D PARA DIFUNDILÁ.
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$ cat texto2.txt
Victoriano Nodar: «A recreación en 3-D de Santa María a Real de Sar permite coñecer a súa evolución histórica»
A Agrupación Cultural O Galo agasalla este sábado ás lectoras e aos lectores de 'Nós Diario' o Cadern o especial dedicado a Santa María a Real de Sar. Do Medievo ao século XXI á través da museoloxía e a técnica dixital. A iniciativa, da autora do profesor de Historia da Arte da Univg Victoriano Nodar e de Ramon Yzquierdo, director técnico-conservador do Museo Catedral de Santiago, quere difundir esta xoia patrimonial a través da historia. Nodar explica as razóns da importancia deste templo compostelá, a que se debe a súa menor visibilidade e a iniciativa que propoñen de empregar as novas tecnoloxías en 3-D para difundila.
RAI@nicolassantiago.gome@220a045h103l: ~/Descargas/proxecto-redes-main/P2$
```

```
RAI@nicolassantiago.gome@220a045h104l: ~/P2
RAI@nicolassantiago.gome@220a045h104l: $ ssh 172.25.45.104
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
0 mantemento de seguraza ampliado para Applications non está activado.
É posíbel aplicar 786 actualizacións inmediatamente.
565 destas actualizacións son actualizacións de seguraza normais.
Para consultar estas actualizacións adicionais execute: apt list --upgradable
Pódense aplicar 87 actualizacións de seguraza adicionais con ESM Apps.
Salba máis sobre como activar o servizo de ESM Apps at https://ubuntu.com/esm.
Last login: Tue Oct 7 11:30:20 2025 from 172.25.45.103
c$bash
RAI@nicolassantiago.gome@220a045h104l: $ cd P2
RAI@nicolassantiago.gome@220a045h104l: ~/P2$ ./servidormay.o 5555
Servidor escuchando...
Dirección IP cliente conectado: 172.25.45.103:51526
Datos enviados al cliente.
Cliente desconectado: 172.25.45.103:51526
-----
Puerto: 5555
Servidor escuchando...
Dirección IP cliente conectado: 172.25.45.103:41982
Datos enviados al cliente.
Cliente desconectado: 172.25.45.103:41982
-----
Puerto: 5555
Servidor escuchando...

```

Figura 7: Resultados cun só cliente

No caso de ter outro cliente esperando, o sistema operativo pon a petición do novo cliente en espera nunha cola tal como é definido na especificación POSIX de `listen()`. O servidor concéntrase na conexión a comezo da cola, e o resto de peticións quedan nesta bloqueados.

V. CONCLUSIÓN

Neste traballo, foron creados dous pares de servidor e cliente para servir dous tipos de casos: un cliente que simplemente recibe dúas frases dun servidor, e outro que intercambia datos entre un cliente que os fornece e os recibe transformados. Ambos estes dous conxuntos de clientes e servidores foron desenvolvidos nun repositorio Git.

Segundo os resultados experimentais da implantación escollida para os dous problemas, o programa consegue completar a funcionalidade proposta, e segue as convencións especificadas en POSIX, tales coma o encolado de peticións, e no propio protocolo TCP, como son a creación de *sockets* orientados a conexión, que permiten observar cal é a IP do cliente dende o servidor.