Informe sobre un servidor e un cliente UDP

Nicolás Santiago Gómez Forján, Marcos García Blanco

Redes

Grupo 03

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

18 de novembro do 2025

I. Introducción

Este informe presenta o desenvolvemento e análise dunha práctica sobre o protocolo UDP (User Datagram Protocol) na que foron implementadas diferentes programas para explorar as características fundamentais deste protocolo non orientado a conexión. A práctica estrutúrase en tres bloques principais: programas básicos de envío e recepción de mensaxes (`emisor.c` e `receptor.c`), un sistema cliente-servidor para conversión de texto a maiúsculas (`clienteUDP.c` e `servidorUDP.c`), e probas de concorrencia. O obxectivo é comprender o comportamento de UDP ante diferentes escenarios: limitacións no tamaño do buffer de recepción, transmisión de tipos de datos estruturados como vectores de `float`, e capacidade de atención simultánea de múltiples clientes sen mecanismos explícitos de concorrencia.

II. Apartado 1

Neste apartado, considérase unha implantación básica do protocolo UDP, onde créase un par `emisor.c`, o cal emprega a función `sendto()` para enviar datos sen conexión e `receptor.c`, o cal emprega `recvfrom()` para poder recibir estes datos sen conexión.

A. Apartado C

Nesta sección, descríbense as modificacións levadas a cabo para reducir drasticamente a cantidade de *bytes* que o receptor pode recibir, e as consecuencias que isto produce no programa.

Comézase modificando a función dentro de `receptor.c` `recvfrom()`, previamente co formato

```
recvfrom(nsocket, mensaxe, (size_t) MAX, 0, (struct sockaddr*) &socket,
                                &tamsocket)
```

onde `MAX` $= 1024$ ao formato

```
recvfrom(nsocket, mensaxe, (size_t) MIN, 0, (struct sockaddr*) &socket,
                                &tamsocket)
```

onde `MIN` $= 8$, colapsando o tamaño máximo a recibir no socket por un factor de cento vinte oito. A función de envío, `sendto(sockEmisor, arrayFloats, sizeof(arrayFloats), 0, (struct sockaddr*) &socketReceptor, tamSocket)` permanece sen cambios.

En tanto aos resultados obtidos, na maioría dos casos onde a mensaxe enviada supera a lonxitude `MIN`, esta non é recibida. Debido a que UDP é un protocolo sen conexión, a mensaxe que non foi transmitida é perdida permanentemente salvo reenvío explícito, posto a que UDP coma protocolo sen conexión non posúe ningún método na capa de transporte para a verificación da integridade dos datos do receptor.

B. Apartado D

Este apartado cubre a modificación dos programas `receptor.c` e `emisor.c` en tanto ao cambio do envío dunha cadea de texto de tipo `char*` a un vector que contén unha cantidade arbitraria de tipos de datos de punto flotante.

Comezando coas modificacións realizadas no programa emisor `emisor.c`, cabe destacar a creación dun vector de tipo de datos de punto flotante, que, para os propósitos do exercicio encomendado, foi enchido con valores escollidos sen importancia a súa semántica coa liña

`float arrayFloats[] = {1.1f, 2.2f, 3.14159f, 40.5f, -500.99f}.`

Para acomodar este cambio, debeuse modificar a función `sendto()` para enviar estes datos, substituindo os campos `size_t __n` e `const void *__buf` polas estruturas correctas:

```
sendto(sockEmisor, arrayFloats, sizeof(arrayFloats), 0, struct sockaddr*)
                        &socketReceptor, tamSocket).
```

Dentro do ficheiro de `receptor.c`, producíronse múltiples cambios para axeitar o novo tipo de datos. Foi substituído en primeiro lugar o tipo de datos correspondente a mensaxe dende unha cadea de texto (`char*`) a un vector de tipo de datos de punto flotante (`float []`).

Para calcular a cantidade de datos recibidos, emprégase a fórmula `nbytes/sizeof(float)`, onde `nbytes` é un `ssize_t` que describe a cantidade de bytes que foron obtidos pola función `recvfrom()`. Resumidamente, isto determina a talla enviada ao dividir a cantidade total de *bytes* polo tamaño en *bytes* de cada elemento.

Para imprimir o vector recibido, imprímese unha cabeceira cunha breve información, e posteriormente itérase o número de elementos (calculados tal coma descrito) salvo o último que será tratado especialmente. Finalmente, imprímese este último elemento illadamente, para evitar calquera adorno que se deba facer no proceso iterativo, e un pé para poder engadir información sobre o traspase de datos.

```
printf("Mesaxe recivida dende %s:%s: ", ipRecividoTexto, argv[1]);

// Se temos 20 bytes e sizeof(float) = 4, recibimos 5 elementos en total e iteramos sobre 4
for (int i = 0; i < nbytes/sizeof(float) - 1; i++)
{
    printf("%f, ", mensaxe[i]);
}

printf("%f (%ld bytes, %ld elementos)\n", mensaxe[nbytes/sizeof(float) - 1], nbytes, nbytes/sizeof(float));
```

Código modificado para a impresión do vector de números de punto flotante

III. Apartado 3

Neste apartado explicarase o que sucede cando se é engadido un `sleep()` no bucle do cliente e inténtanse conectar varios clientes de forma simultánea. Como xa foi nomeado con anterioridade, o único cambio que se fai no cliente é o engadido dun `sleep()` no final do bucle `while (fgets(linea, sizeof(linea), ficheroEntrada) != NULL)`.

Ao engadir esta función e tentar varios clientes de maneira simultánea podemos observar que o servidor recibe unha liña do primeiro cliente e espera uns segundos. Ao executar o segundo cliente vese que recibe a primeira liña deste sen ter en conta o `sleep()` do primeiro. Isto débese a que ao ser un protocolo non orientado a conexión, UDP non mantén ningún tipo de sesión ou estado entre o servidor e cada cliente. O servidor simplemente procesa cada datagrama que chega de forma independente, sen importar de que cliente provén nin se hai outros clientes esperando. Cada chamada a `recvfrom()` obtén o seguinte datagrama dispoñible na cola do socket, procésao e responde ao cliente correspondente mediante `sendto()`, utilizando a dirección de orixe que se obtivo no propio `recvfrom()`. Deste xeito, mentres o primeiro cliente está bloqueado polo `sleep()`, o servidor queda libre para recibir e procesar

datagramas doutros clientes, permitindo así a atención simultánea de múltiples clientes sen necesidade de implementar mecanismos adicionais.

Todo isto pódese visualizar na seguinte imaxe, a cal é un exemplo de execución do código deste apartado.

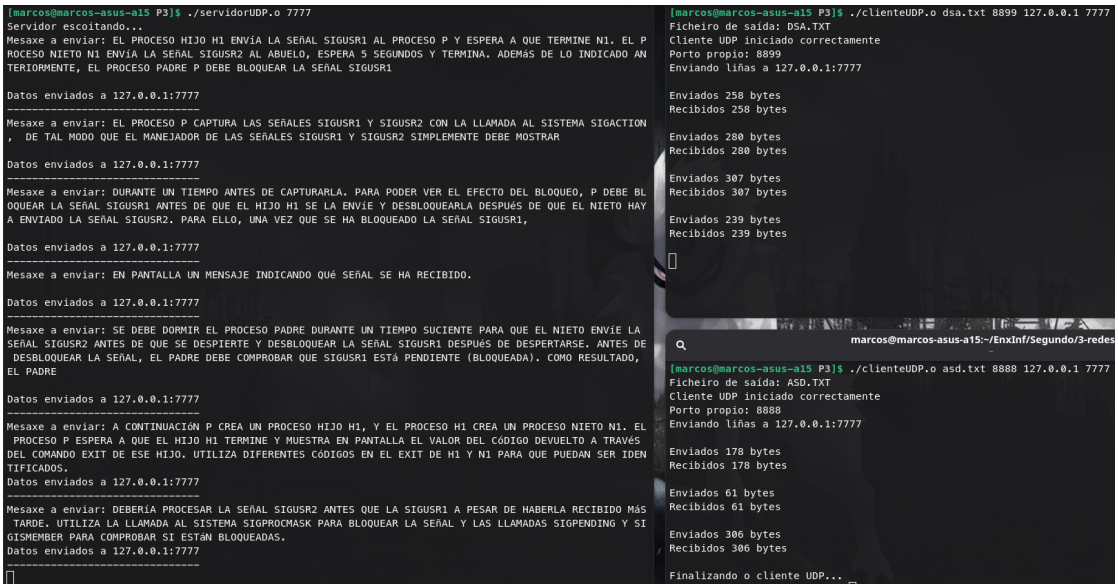

Figura 1: *Execución do código do apartado 3*

IV. Conclusións

As probas realizadas ao longo desta práctica demostraron as características esenciais do protocolo UDP como protocolo non orientado a conexión. Verificouse que a perda de datos é unha consecuencia directa cando o buffer de recepción é insuficiente para almacenar un datagrama completo, sen posibilidade de recuperación dos bytes restantes. Ademais, comprobouse que UDP permite a transmisión de calquera tipo de datos estruturados, sempre que ambos extremos interpreten correctamente a información binaria recibida. Finalmente, confirmouse que a natureza sen estado de UDP permite ao servidor atender múltiples clientes de forma simultánea sen necesidade de implementar threads ou procesos adicionais, xa que cada datagrama é procesado de xeito independente. Estas características fan de UDP un protocolo idóneo para aplicacións onde prima a velocidade e simplicidade sobre a fiabilidade garantida.