El paquete de comunicaciones Lower_Layer_UDP

Programación de Sistemas de Telecomunicación
Informática II

Departamento de Sistemas Telemáticos y Computación (GSyC)

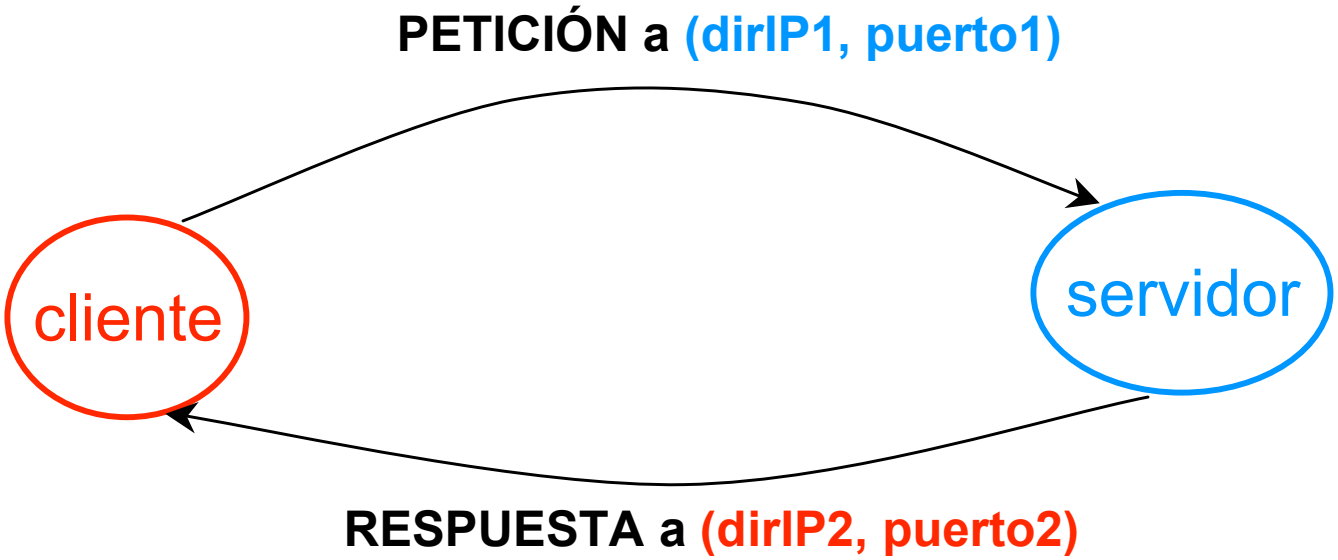
Octubre de 2019

Contenidos

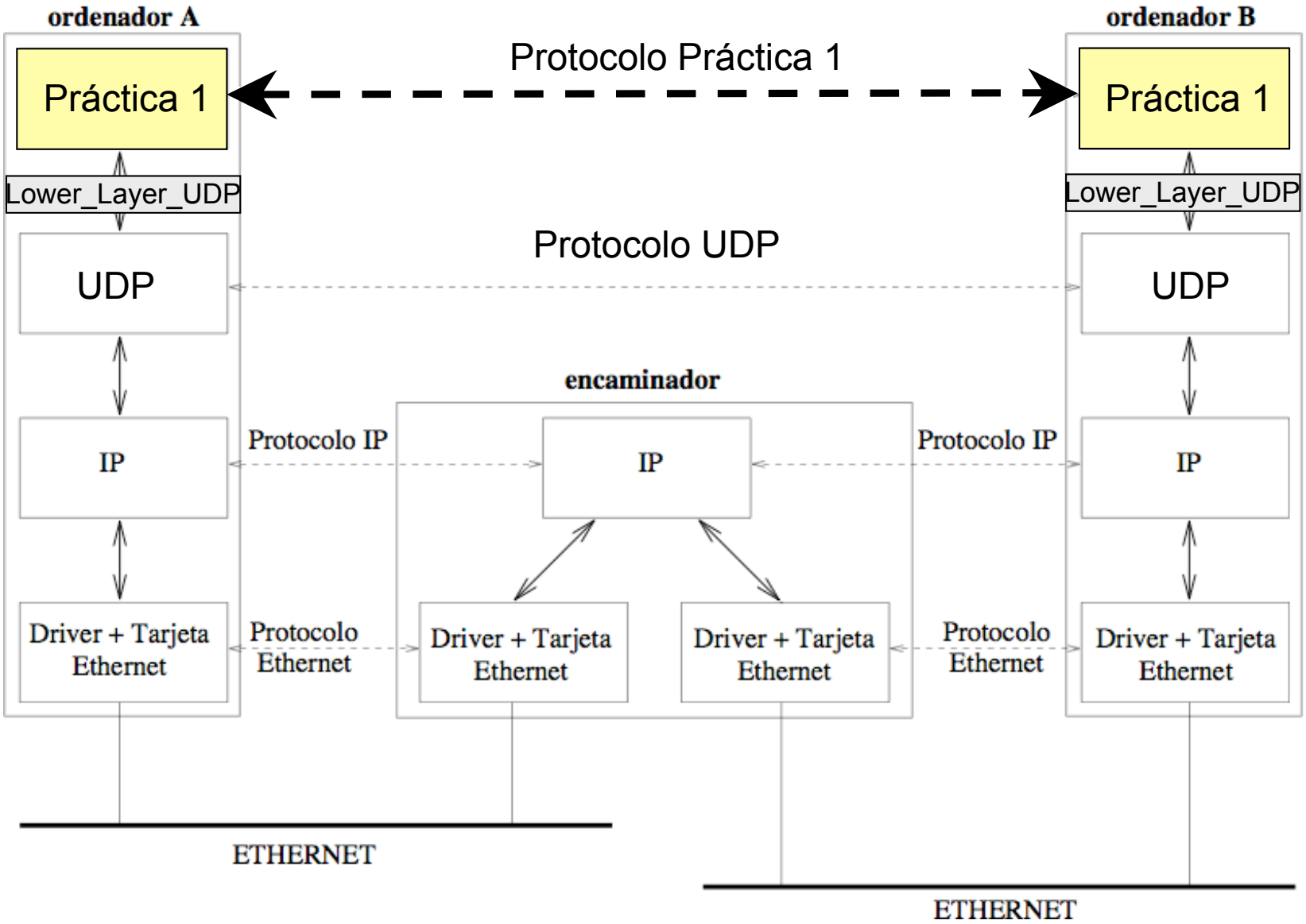
Contenidos

Introducción: Modelo Cliente-Servidor

- En el modelo cliente-servidor hay dos roles diferenciados para los procesos que intercambian mensajes a través de la red:
  - El servidor espera a recibir un mensaje, y entonces contesta enviando una respuesta
  - El cliente envía mensajes de petición al servidor, esperando recibir un mensaje de respuesta.



Introducción: LLU



Lower_Layer_UDP

- Es un paquete de comunicaciones que ofrece un servicio de mensajes en el nivel de transporte:
  - extremo a extremo
  - no orientado a conexión
  - no fiable
- Ofrece un servicio similar a UDP con una interfaz más fácil de usar que permite:
  - Construir `End_Points` (dirección IP + puerto)
  - Construir y procesar mensajes
  - Enviar y recibir mensajes
- A partir de ahora utilizaremos `LLU` para hacer referencia a los tipos, funciones y procedimientos de este paquete:

```
package LLU renames Lower_Layer_UDP;
```

Contenidos

End_Point

- Un `End_Point` es el destino de los datos que envía un proceso de una máquina a otro proceso de otra máquina.
- Se construye a partir de la **Dirección IP** de la máquina y el **puerto** elegido por el proceso al que van dirigidos los datos.
- Un `End_Point` identifica de forma unívoca al proceso al que van dirigidos los mensajes que transportan datos.
- Se construye de la siguiente forma con la función `Build`:

```
Server_EP: LLU.End_Point_Type;

 ...

Server_EP := LLU.Build("212.128.4.1", 6002);
```

Funciones auxiliares: Get_Host_Name

- La función Get_Host_Name devuelve el nombre de la máquina donde se está ejecutando un programa.

```
function Get_Host_Name return String;
```

- Ejemplo de uso:

```
Maquina : ASU.Unbounded_String;
...
Maquina := ASU.To_Unbounded_String (LLU.Get_Host_Name);
Ada.Text_IO.Put_Line ("Estoy en: " & ASU.To_String(Maquina));
```

Funciones auxiliares: To_IP

- La función To_IP devuelve la direccción IP del nombre de la máquina que se le pasa como argumento.

```
function To_IP (Name: in String) return String;
```

- Si el argumento que se le pasa a To_IP es una dirección IP, la función devuelve la misma dirección IP.
- Ejemplo de uso:

```
Maquina : ASU.Unbounded_String;
Dir_IP : ASU.Unbounded_String;
...
Maquina := ASU.To_Unbounded_String ("f-l3209-pc09");
Dir_IP := ASU.To_Unbounded_String (LLU.To_IP(ASU.To_String(Maquina));
Ada.Text_IO.Put_Line ("Dirección IP:" & ASU.To_String(Dir_IP));
```

Ejercicio

- Utiliza el comando Unix `hostname` en tu máquina del laboratorio para averiguar el nombre del ordenador.
- Utiliza ahora `ifconfig` para identificar la dirección IP del ordenador. En caso de que haya varias, fíjate en cuál es la dirección IP asignada a un dispositivo físico. Debes ignorar las direcciones de dispositivos tipo *loopback* (cuyo nombre comienza por `lo`), de máquinas virtuales como docker, etc. El nombre de dispositivos físicos conectados a una red Ethernet suele comenzar por `en`.
- Escribe un programa Ada llamado `Host` que muestre tanto el nombre de la máquina como su dirección IP.
  - ¿Coinciden los resultados?
  - ¿Qué sucede cuando termina el programa?

Contenidos

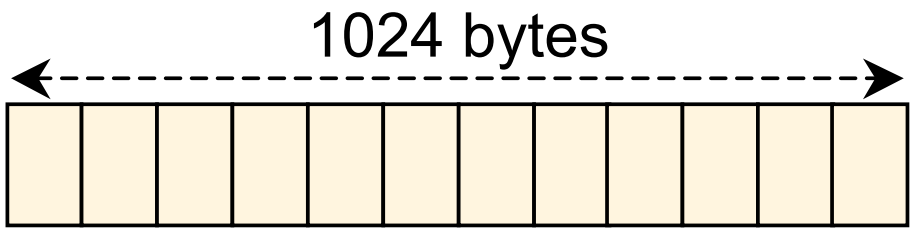
Buffer

- Un *buffer* es un fragmento de posiciones de memoria contiguas que utilizaremos en:
  - **Envío**: para almacenar en el *buffer* los datos que queremos enviar. Una vez almacenados, realizaremos el envío.
  - **Recepción**: para recibir los datos que nos estén enviando. El *buffer* inicialmente estará vacío y después de realizar la recepción contendrá los datos que nos hayan enviado.
- Accederemos al *buffer* según el modelo FIFO (*First In First Out*).

Declaración de un *Buffer*

- Utilizaremos el tipo `LLU.Buffer_Type`.
- Para declarar un *buffer* es necesario proporcionale el tamaño en bytes que se reservará para almacenar datos en él:
  - Es necesario que el tamaño del *buffer* sea suficiente como para contener todos los datos que vamos a almacenar en él.

```
Buffer: aliased LLU.Buffer_Type(1024);
```

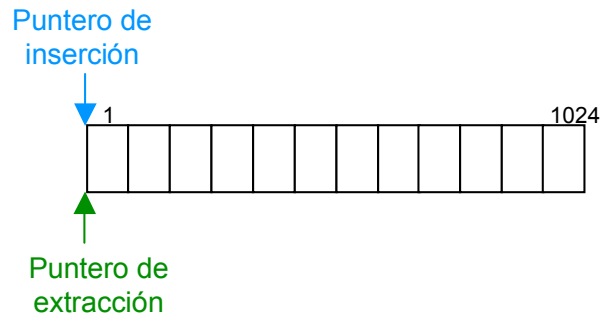


- Utilizaremos el atributo 'Access del `Buffer` para meter y sacar datos en él. Por este motivo, es necesario que lo declaremos como `aliased`.

Inicialización de un *Buffer*

- El procedimiento `Reset` vacía los datos que hay en un *buffer*.

```
Buffer: aliased LLU.Buffer_Type(1024);

...

LLU.Reset(Buffer);
```



- Utilizaremos `Reset` en las siguientes situaciones:
  - **Envío**: antes de utilizar un *buffer* para componer un mensaje y meter los datos que queremos enviar.
  - **Recepción**: antes de utilizar un *buffer* para recibir un mensaje.

Inserción de datos en un *Buffer*

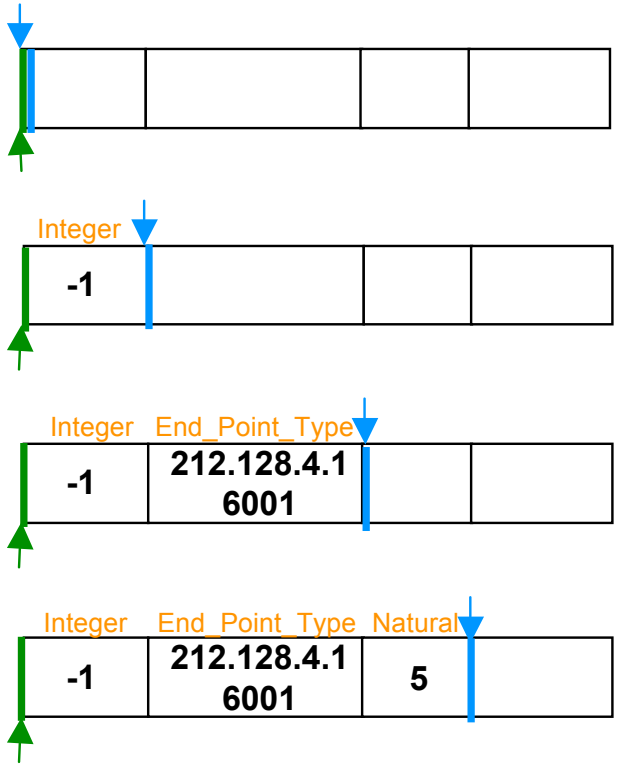
- Se invoca el atributo `'Output` del tipo del dato que queremos introducir en el *buffer*.

```
Buffer: aliased LLU.Buffer_Type(1024);
I: Integer;

...

I:=5;
LLU.Reset(Buffer);
Integer'Output(Buffer'Access, I);

...
 <Enviar los datos del buffer>
...
```

- Después de introducir un dato en un *buffer*, el siguiente dato que se introduzca se almacenará a continuación del anterior.
- Si intentamos introducir un dato en un *buffer* y no hay espacio suficiente en el *buffer* se elevará la excepción `End_Error`.

Ejemplo de inserción de datos en un *Buffer*

```
Buffer: aliased LLU.Buffer_Type(1024);
I: Integer;
EP: LLU.End_Point_Type;
N: Natural;

...

LLU.Reset(Buffer);

I := -1;
Integer'Output(Buffer'Access, I);

EP := LLU.Build("212.128.4.1", 6001);
LLU.End_Point_Type'Output(Buffer'Access, EP);

N := 5;
Natural'Output(Buffer'Access, N);

...
 <Enviar los datos del buffer>
...
```



Extracción de datos de un *Buffer*

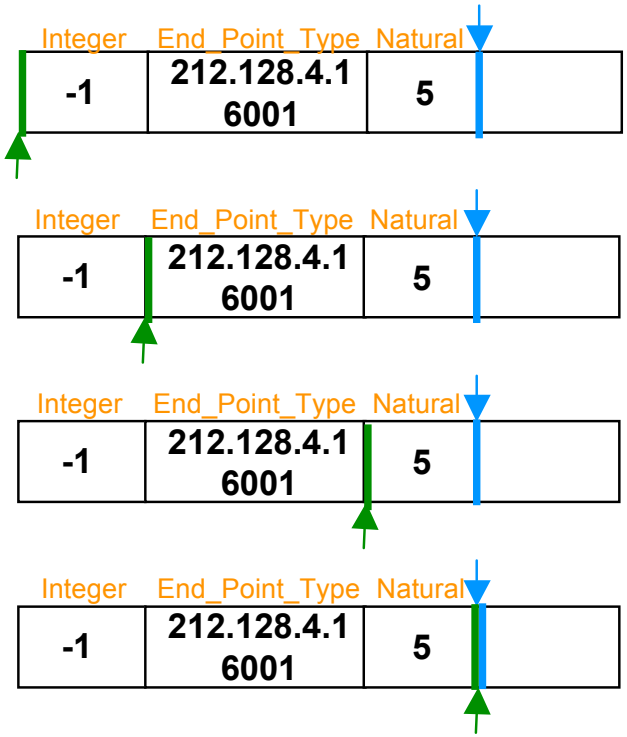
- Se invoca el atributo `'Input` del tipo del dato que queremos extraer del *buffer*.

```
Buffer: aliased LLU.Buffer_Type(1024);
I: Integer;

...

LLU.Reset(Buffer);

...
 <Recibir datos en el buffer>
...

I := Integer'Input(Buffer'Access);
```

- Después de extraer un dato de un *buffer*, el siguiente dato que se extraiga será el que se encuentre a continuación del anterior.
- Si intentamos extraer un dato de un *buffer* en el que no hay más datos, se elevará la excepción `End_Error`.
- Si el tipo de dato que hay en el *buffer* no coincide con el tipo de datos que se está intentando extraer puede ocurrir:
  - La extracción se realiza y la variable destino toma un valor sin sentido.
  - Se eleva alguna excepción, ej: `Constraint_Error`
  - Se produce un fallo de segmentación: `Segmentation Fault`

Ejemplo de extracción de datos en un *Buffer*

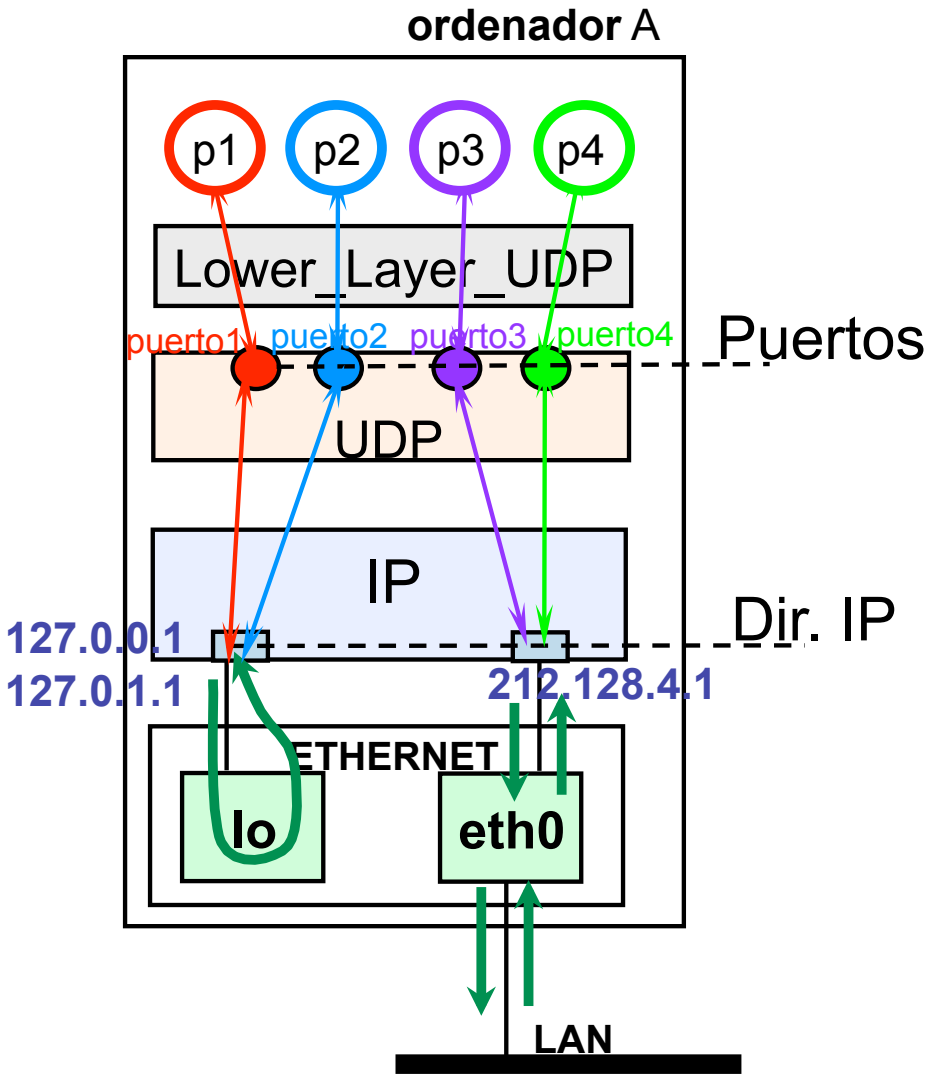
```
Buffer: aliased LLU.Buffer_Type(1024);
I: Integer;
EP: LLU.End_Point_Type;
N: Natural;
...

LLU.Reset(Buffer);

...
 <Recibir datos en el buffer>
...

I := Integer'Input(Buffer'Access);
EP := LLU.End_Point_Type'Input(Buffer'Access);
N := Natural'Input(Buffer'Access);
```



Contenidos

Enlazarse a un End_Point (I)

- Para que un proceso pueda recibir datos de otros, es necesario que previamente se enlace a un `End_Point`. De esta forma "escucha" los mensajes dirigidos a ese `End_Point`.
- La operación de enlazarse a un `End_Point` tiene sentido en el ámbito local de una máquina:
  - Cuando un proceso que se ejecuta en una máquina se ata a un `End_Point`, el proceso queda ligado a una dirección IP de esa máquina y a un puerto de esa máquina.



Enlazarse a un End_Point (II)

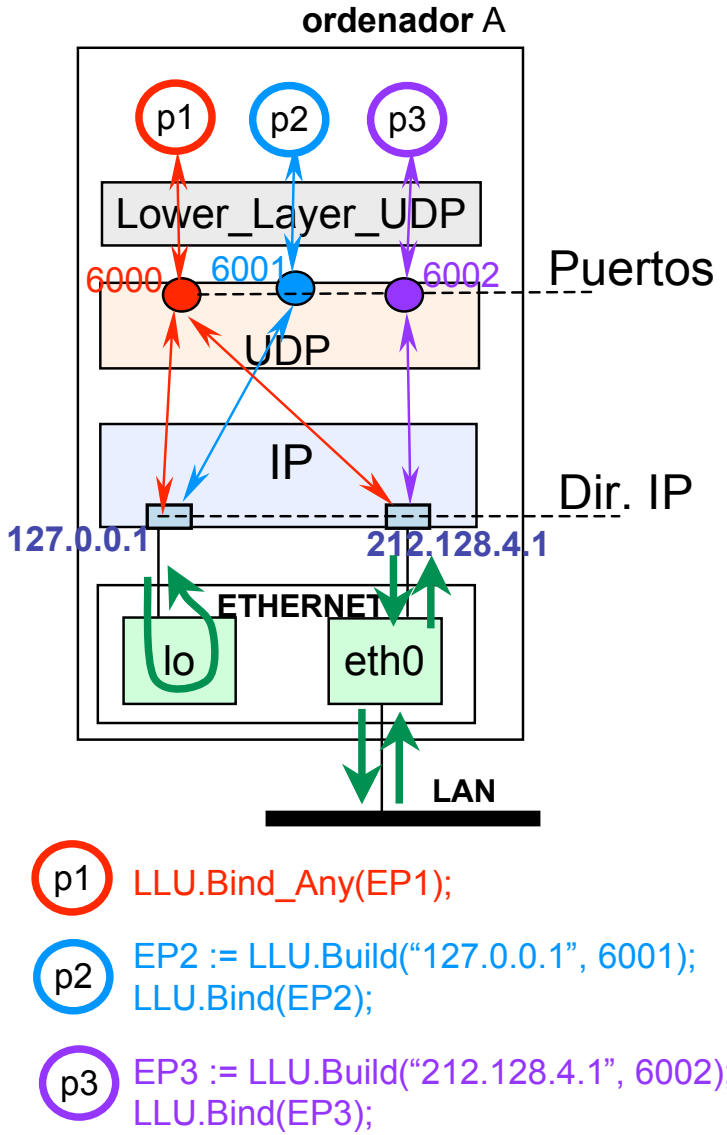
Dos formas:

1. **Enlazándose a un puerto concreto:**

```
Server_EP: LLU.End_Point_Type;

...

Server_EP := LLU.Build("212.128.4.1", 6002);

LLU.Bind(Server_EP);
```

2. **Enlazándose a un puerto libre cualquiera:**

```
Client_EP: LLU.End_Point_Type;

...

LLU.Bind_Any(Client_EP);
```



Motivos por los que puede fallar Bind

```
Server_EP: LLU.End_Point_Type;

...

Server_EP := LLU.Build("212.128.4.1", 6002);

LLU.Bind(Server_EP);
```

- La dirección IP que se usa en `Build` no pertenece a la máquina donde se ejecuta el código.
- El número de puerto que se usa en `Build` está reservado (0-1023). Los puertos reservados los utilizan protocolos conocidos o procesos con privilegios.
- El número de puerto que se usa en `Build` ya se está utilizando y no se encuentra libre.

Contenidos

Envío de mensajes

- Para enviar datos es necesario conocer el `End_Point` del proceso receptor, de alguna de estas dos maneras:
  1. El receptor escucha en un `End_Point` formado por una dirección IP y puerto conocidos por el emisor.
  2. El receptor le ha pasado previamente al emisor su `End_Point` dentro de un mensaje.
- Para enviar datos se utiliza el procedimiento `Send`:

```
procedure Send(To :    in       End_Point_Type;
               Data:   access   Buffer_Type);
```

- Ejemplo de uso:

```
LLU.Send(Dest_EP, Buffer'Access);
```

Recepción de mensajes

- Para que un proceso reciba datos es necesario que construya un `End_Point` (con la dirección IP de su máquina y un puerto libre) y se enlace a él.
- Para recibir datos se utiliza el procedimiento `Receive`:

```
procedure Receive(To :      in      End_Point_Type;
                  Data:     access  Buffer_Type;
                  Timeout: in       Duration;
                  Expired: out      Boolean;)
```

- La llamada `Receive` es bloqueante hasta que se recibe algo o vence el plazo (`Timeout`).
- Ejemplo de uso:

```
LLU.Reset(Buffer);
LLU.Receive(My_EP, Buffer'Access, 2.0, Expired);
if Expired then
    Put_Line ("Expiró el plazo");
else
    I := Integer'Input (Buffer'Access);
end if;
```

Contenidos

Terminación del uso de Lower_Layer

- Antes de terminar un programa que utilice `Lower_Layer` es necesario ejecutar:

```
LLU.Finalize;
```

- Si no se ejecuta `Finalize` el programa no terminará nunca.
- Una vez ejecutado `Finalize` el programa ya no podrá utilizar las funciones de envío/recepción de mensajes de LLU.