Sockets TCP/IP en C - Protocolos

Di Paola Martín
martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

De qué va esto?

Protocolos y formatos

Protocolos y formatos

Binario o Texto

- Protocolos en Texto: son la contracara de los protocolos binarios, son lentos, ineficientes y más difíciles de parsear pero más fáciles de debuggear. Son independientes del endianess, padding y otros pero dependen del encoding del texto y que caracteres se usan como delimitadores.
- Protocolos en Binario: son simples y eficientes en terminos de memoria y velocidad de procesamiento. Son más difíciles de debuggear. Es necesario tomar en consideración el endianess, el padding, los tamaños y los signos.

HTTP

```
1 GET /index.html HTTP/1.1\r\n
2 Host: www.fi.uba.ar\r\n
3 \r\n
```

- En HTTP el fin del mensaje esta dado por una línea vacia; cada línea esta delimitada por un `\r\n`
- Cuantos bytes reservarían para contener dicho mensaje o alguna línea?
- Que pasa si el delimitador `\r\n` aparece en el medio de una línea, como lo diferenciarían?

- Habitualmente en protocolos en texto se usa uno o una secuencia de caracteres como delimitadores.
- En la cabezera de HTTP se usa `\r\n`
- En C/C++, los fin de strings son marcados con `\0`
- Aunque simple, no es trivial saber cuantos bytes hay hasta el delimitador.
- Tampoco es trivial el caso de que el texto contenga al delimitador meramente por que es parte de su contenido.
- Hay dos opciones, o se opta por otro protocolo o se usa una secuencia de escape para que el delimitador sea considerado un literal y no un delimitador.
- Y si la secuencia de escape es parte del contenido? Hay que escapear la secuencia de escape con otra secuencia de escape.
- Por ejemplo, el compilador de C/C++ ve `"\1"` como un string con el byte 1 (a pesar de haber 2 caracteres). Si se quisiera literalmente poner una barra y un 1 hay que escapear la barra: `"\\1"`

TLV

```
1  struct Msj {
2      unsigned short type;
3      unsigned short length;
4      char* value;
5  };
6
7  read(fd, &msj.type, sizeof(unsigned short) * 2);
8  msj.value = (char*) malloc(msj.length);
9  read(fd, msj.value, msj.length);
```

- Los primeros 4 bytes indican la longitud y tipo del valor; el resto de los bytes son el valor en sí.
- Por qué es importante usar `unsigned short` y no solamente `short`? Qué pasa si `sizeof(unsigned short)` no es 2?
- Que pasa si el endianess no coincide? y si hay padding entre los dos primeros campos?

- Prefijar la longitud del mensaje soluciona varios problemas pero trae otros.
- Si `sizeof(unsigned short)` vale 4 estaríamos enviando 8 bytes con las longitud y tipo pero la máquina que recibe el mensaje puede esperar 4.
- Hay que definir y forzar un endianess y reglas de padding.