12 en binaire : 1100 petit bout
0011 grand bout

TD 4 – Représentation des données

1. Petit boutiste et grand boutiste

Le but de cet exercice est de mettre en lumière les différences de stockage des données entre les formats petit boutiste (*little endian*) et grand boutiste (*big endian*). Ce problème acquiert son importance dès lors que l'on cherche à transférer de l'information entre des clients et des serveurs qui ont fait des choix de représentation de données différents : il faut que le protocole s'assure de la bonne conversion des données.
On considère un flux d'entrée/sortie en mode binaire (*i.e.*, constitué d'octets) dans lequel on veut envoyer des mots de 16 bits. Chaque mot se décompose en octet de poids fort et octet de poids faible (la valeur du mot est 256*octet poids fort + octet poids faible).

- le format petit boutiste consiste à écrire d'abord l'octet de poids faible puis l'octet de poids fort,
- le format grand boutiste consiste à écrire d'abord l'octet de poids fort puis l'octet de poids faible.

Remarque : ces définitions se généralisent lorsqu'il s'agit d'écrire des mots de plus de 16 bits (par exemple 32 ou 64 bits).

1. On considère la chaîne de caractères « hello world » codée de la façon suivante :
- 2 octets pour sa longueur,
- 1 octet par caractère.

Soit A une machine utilisant la convention petit boutiste et B une machine utilisant la convention grand boutiste. Représenter l'ordre de stockage des octets dans les deux cas.

11 caractères dans la chaîne (y compris l'espace). Les caractères sur 1 octet ne sont pas affectés par le format (seuls les mots de 2 octets le sont).

longueur données

```
A          11 00 hello world
B          00 11 hello world
```

2. Que se passe-t-il si la machine A envoie la chaîne de caractères telle quelle à la machine B ?

B extrait la longueur de la chaîne, et étant grand boutiste, considère que le 1er octet (11) est l'octet de poids fort, tandis que le 2nd (00) est l'octet de poids faible. Elle s'attend donc à trouver à la suite 11*256 + 0 caractères (ce qui bien évidemment n'est pas le cas).

3. Dans un deuxième temps, on décide d'inverser l'ordre des octets pour chaque couple d'octets arrivant sur B. Que se passe-t-il ?

Après inversion, la machine B obtient la suite `00 11 ehll oowlr d`. Le nombre de caractères attendu est bien de 11 cette fois, mais les caractères sont inversés 2 à 2.

Conclusion : un mécanisme d'encodage/désencodage des données ne peut se contenter de faire des conversions automatiquement lorsqu'il s'agit de transférer de l'information entre machines ayant des conventions de représentation différentes. Le type des données (ici entier ou caractère) doit être pris en compte pour savoir quelle règle appliquer. Les mécanismes