Architecture et Système

Stefan Schwoon

Cours L3, 2018/19, ENS Cachan

Codage et traitement des caractères

Comme d'autres données, les textes sont représentés par des suites d'octets.
Cependant, ce sujet a tendance à poser des difficultés particulières :

- beaucoup de standards différents et partiellement (in)compatibles
- support insuffisant dans les langages de programmation
- ignorance des programmeurs/utilisateurs/institutions
- difficile à debugger

Resumé du suivant et du TP

Survol des standards existants

Traitement dans quelques applications importantes (terminal, LaTeX, web, mail)

Repérer (et corriger) les erreurs les plus fréquentes

Leçons à retenir :

Un caractère n'égale pas (toujours) à un octet

Pour traiter du texte il faut connaître son codage de caractères.
(et cette information manque souvent !)

C'est un enjeu socio-culturel.

Débuts historiques

Terminal : à l'origine, machine "bête" branchée sur un ordinateur partagé

Dispose d'un clavier et d'un écran

Affichage des caractères saisis sur clavier (si en mode echo)
+ transfert vers l'ordinateur

Affichage des données venues de l'ordinateur

À l'époque : un caractère équivaut un octet, chaque caractère à son code

Certains Caractères sont imprimables (a..z,0..9), d'autres sont dits de contrôle (retour chariot CR, saut de ligne LF, tabulation, appel BEL, retour arrière BS, . . .)

Aujourd'hui : *émulation* d'un terminal dans, p.ex., X11

Parenthèse : gestion des touches

Qu'est-ce qui se passe quand l'utilisateur tape un touche ?

Le clavier signale une interruption auprès du processeur central.
(une interruption pour déprimer la clé, une autre pour la relâcher)

Pour chaque type d'interruption, le processeur connaît une adresse mémoire à appeler pour gérer l'interruption. Ce gestionnaire fait partie du système.

Lorsque le processeur est prêt pour traiter l'interruption, il transfert donc le contrôle au gestionnaire qui obtient le code de touche.

Les codes de touche sont spécifiques au clavier. P.ex. un clavier pourrait envoyer 11 pour la touche 'espace', un autre 25.

Sous X11, le gestionnaire fait appel a un pilote qui traduit le code de touche vers un symbole (p.ex, A, a-accent-aigu, . . .), en prenant compte des *modificateurs* actuels (Shift, Ctrl, Alt).

Finalement, le gestionnaire génère un évènement auprès de l'application de premier plan qui contient le nom de touche et le symbole (`xev` laisse observer ces évènements).

L'application recevant l'évènement est ensuite libre de le traiter à sa discétion. P.ex., un terminal traduit d'abord le symbole selon son encodage en vigueur. Ensuite, il le transfert vers son *processus de premier plan* (typiquement, le shell).

Détails sur le terminal

Normalement, le terminal est en mode echo. Dans ce mode, il affiche immédiatement tout symbole reçu par le système. L'echo peut être désactivé, p.ex. pour les mots de passe. (commande : `stty -echo`)

En mode canonique, les codages des caractères sont envoyés au processus du premier plan (dans le terminal) lorsque l'utilisateur saisit "nouvelle ligne". En mode non-canonique, ils sont envoyés toute de suite.

En plus, le terminal reçoit des séquences d'octets de la part des processus qui lui sont attachés. Ces séquences seront décodées vers des caractères qui seront affichés sur l'écran.

Séquences de contrôle dans le terminal

Certains caractères non-imprimables ont une interprétation spéciale dans le terminal :

des contrôles simples (nouvelle ligne etc)

des séquences commençant par ‘escape’ (27), p.ex.:
ESC[31m : afficher texte en rouge
ESC[42m : arrière-plan en vert
ESC[0m : arrière-plan en noir
ESC[2J : effacer l’écran
ESC[10;20H : mettre le cursor sur position (10,20)

Le caractère ESC correspond à `\e` en C. Dans le shell, `echo -e "\e..."`

Parenthèse : nouvelle ligne

Grâce à des raisons historiques, il existe deux standards pour représenter une nouvelle ligne :

format Unix : un seul caractère (10 décimal) entre deux lignes

format DOS: nouvelle ligne représentée par CR+LF = caractères 13+10

La plupart des éditeurs de texte savent traiter les deux formats.

Conversion explicite : `dos2unix`, `unix2dos`

En `vi`: option `fileformat` (=unix,dos)

Codage caractères : Terminologie

Pour échanger des caractères, il faut se mettre d'accord sur leur codage.
Les distinctions suivantes seront utiles :

jeu de caractères : l'ensemble (non-ordonné) des caractères (graphèmes) dont on dispose
Notons $A \subseteq B$ si un jeu de caractères A est un sous-ensemble de B.

jeu de caractères codés : fonction qui affecte à chaque caractère un *code* numérique.
Notons $A \prec B$ si $A \subseteq B$ et que A affecte tous ses caractères aux mêmes codes que B.

Terminologie

Un chaîne de caractères sera donc représentée par une suite de codes.

Selon les besoins, les mêmes codes peuvent être représentés différemment, on distingue donc deux couches de représentation :

codets : découpage du code en une ou plusieurs unités, généralement des valeurs entre 0 et 255

forme de codage : comment représenter un codet (typiquement binaire)

Les standards les plus importants

ISO-646 in 1963/1972, version américaine : ASCII

codes à 7 bits, caractères de contrôle avec codes < 32

Quelques codes à 8 bits :

Codepage 437 (et d'autres), dans les IBM PC (1981)

ISO-8859 (à partir de 1985) : diverses variants, le plus important étant
ISO-8859-1 (= Latin-1), avec encore quelques positions inutilisées ;
ISO-8859-15 (= Latin-9) encode tous le caractères français

Windows-1252 (arrivé avec MS Windows) : extension de Latin-1

$\text{ASCII} \prec \text{Latin-1} \prec \text{Windows-1252}$

Unicode/ISO 10646

(à partir de 1991) effort pour définir un codage *universelle*, c'est à dire pour pouvoir représenter tous les langages du monde.

Version actuelle connaît 1 million de caractères différents

On affecte un point de code hexadécimal à chaque caractère :
p.ex. A = U+0041, é = U+00E9, etc

$\text{Latin-1} \prec \text{Unicode}$ et $\text{Windows-1252} \subseteq \text{Unicode}$

Usage actuel :

plus de 80% des pages web modiales en Unicode, 10% en Latin-1

Découpage en codets

ASCII/Latin-1/Windows : facile, un caractère égale un octet

Unicode : plusieurs standards différents

- Coder chaque caractère avec 4 octets = 32 bits (UTF-32)
- Coder les caractères ‘bas’ avec 16 bits (UTF-16)
- Codage vers des codets ‘imprimables’ en ASCII (UTF-7)
- Le plus important : UTF-8 (codage de longueur variable)

Découpage UTF-8

Un code représenté par un ou plusieurs *codets*.

Pour les caractères ASCII: un seul codet entre 0..127.

Pour les autres : 2 à 4 codets entre 128..255. Exemple :

> Les codes entre 128 (= 0x80) et 2047 (= 0x7FF) prennent deux octets.
>
> Le code pour “é” égale 233, en binaire avec 11 positions: 00011101001
>
> Le premier octet est composé de 110 suivi par les cinq premiers bits du code, le deuxième de 10 suivi par les six autres bits.
>
> Ça donne 11000011 10101001 = 0xC3 0xA9.

Traitement de text sous Unix

La plupart des outils (`grep` etc) ignorent les codages ;
ils traitent des séquences d'octets.

Des éditeurs de texte (`vi`, `emacs`) ou navigateurs (firefox) encodent selon les options en vigueur (ou selon leur propre 'intelligence'.

`file` essaie à déterminer le codage des fichiers texte ;
(sinon, `hexdump -C` s'avère utile. . .)

`iconv` sait convertir entre une grande variété d'encodages.

Faire connaître le codage d'un document

En HTML ou XML, dans les entêtes :

HTML : `<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"/>`

depuis HTML5 : `<meta charset="ISO-8859-1">`

XML: `<?xml version="1.0" encoding="UTF-8"?>`

Pour des fichiers texte : pas de standard, il faut spécifier autrement !

Unicode : formes composées et décomposées

Problème spécifique dans Unicode : un même caractère peut être codé sous plusieurs formes !

P.ex., é = U+00E9 (forme composée) ou U+0065 U+0301 (forme décomposée)

En fait, la partie U+03xx de Unicode contient des diacritiques qui se combinent avec le caractère précédent.

Une application conforme aux standards Unicode est censée traiter ces séquences identiquement !

Les équivalences et formes normales

Unicode connaît deux formes d'équivalence entre (séquences de) caractères :

Canonical equivalence (équivalence forte) – deux chaînes fortement équivalent sont à traiter identiquement dans tous les contextes

Compatibility equivalence (équivalence faible) – équivalence entre caractères qui ont la même apparence mais une semantique différente selon le contexte (p.ex. 2 normal et ^{2} superscript)

Formes normales canoniques : NFC/NFD (composé/décomposé)

Formes normales de compatibilité : NFKC/NFKD (composé/décomposé)

L'utilitaire `uconv` sait convertir un texte aux formes normales.

Forme de codage

Les codets sont typiquement sous la forme d'un valeur 0..255.

Normalement, on représente un codet dans sa forme binaire dans un octet.

Mais parfois, on souhaite une représentation "imprimable" :

> P.ex., les protocoles Internet (mail, web) ne permettent que des caractères imprimables dans les entêtes, dans les adresses, ...
>
> Les formes imprimables résistent mieux aux erreurs de traductions car tout codage habituel est compatible avec ASCII.

Exemples de forme de codage non-trivial

Codage ‘pourcent’ dans les adresses web, p.ex. :

`https://fr.wikipedia.org/wiki/Caract%C3%A8re`

Codage ‘quoted-printable’, p.ex. dans les entêtes mail :

`Subject:  =?utf-8?Q?T=C3=A9st?=`

(donne ‘Tést’ comme sujet du mail)

Codage ‘base64’, pour le même contenu :

`Subject:  =?utf-8?B?VMOpc3Q=?=`

Dans base64, on se donne 64 symboles (A-Za-z0-9+/) qui code des séquences de 6 bits ; quatre symboles font trois codets.