HOWTO Unicode

Bruno Haible, <*haible@clisp.cons.org*> , traduction : Samuel Tribehou, <*samuel-tribehou@mail.cpod.fr*>
v 0.12, 19 Octobre 1999

Ce document explique comment configurer votre système Linux pour qu'il utilise l'encodage de texte UTF-8. Ce document est en cours d'élaboration. Toutes les astuces, suggestions, patchs, URLs sont les bienvenus. NDT : Il a été convenu avec l'auteur que seule la présente version de ce document serait traduite dans un premier temps, le contenu changeant pour l'instant trop souvent pour qu'une traduction suivie puisse être réalisée. Reportez vous au site du LDP pour obtenir la dernière version.

Contents

1 Introduction

1.1 Pourquoi Unicode ?

Les gens de différents pays utilisent différents caractères pour représenter les mots de leur langue natale. De nos jours la plupart des applications, y compris les logiciels de courrier électronique et les navigateurs, traitent correctement les caractères 8-bits. Ils peuvent donc traiter et afficher du texte correctement à condition qu'il soit représenté dans un jeu de caractères 8-bits, comme ISO-8859-1.

Il y a bien plus de 256 caractères dans le monde - pensez au cyrillique, à l'hébreu, à l'arabe, au chinois, au japonais au coréen et au thaï -, et de temps à autres, de nouveaux caractères sont inventés. Les problèmes que cela induit pour les utilisateurs sont les suivants :

- Il est impossible de stocker du texte avec des jeux de caractères différents dans le même document. Par exemple, je peux citer des journaux russes dans une publication allemande ou française si j'utilise TeX, xdvi et Postscript, mais je ne peux pas le faire avec du texte pur.
- Tant que chaque document a son propre jeu de caractères, et que la reconnaissance des jeux de caractères n'est pas automatique, l'intervention manuelle de l'utilisateur est inévitable. Par exemple, pour voir la page d'accueil de la distribution Linux XTeamLinux *http://www.xteamlinux.com.cn/* , je dois dire à Netscape que la page web est codée en GB2312.
- De nouveaux symboles comme l'Euro sont inventés. ISO a publié un nouveau standard ISO-8859-15 qui est en gros identique à ISO-8859-1, excepté qu'il supprime des caractères rarement utilisés, comme le vieux symbole monétaire, remplacé par le signe Euro. Si les utilisateurs acceptent ce standard, ils auront des documents dans différents jeux de caractères sur leur disque, et cela deviendra une préoccupation quotidienne. Mais les ordinateurs devraient simplifier le choses, pas les compliquer.

La solution à ce problème est l'adoption d'un jeu de caractères universel. Ce jeu de caractères est Unicode *http://www.unicode.org/* . Pour plus d'informations sur Unicode, faites `man 7 unicode` (page de man contenue dans le package lpdman-1.20).

1.2 Les encodages d'Unicode

Cela réduit le problème de l'utilisateur (devoir jongler entre différents jeux de caractères) à un problème technique : comment transporter les caractères Unicode en utilisant des octets de 8 bits ? L'unité de 8 bits est la plus petite unité adressable de la plupart des ordinateurs et c'est aussi l'unité utilisée par les connexions réseau TCP/IP. Cependant, l'utilisation d'un octet pour la représentation d'un caractère est un accident de l'histoire dû au fait que le développement de l'ordinateur commença en Europe et aux États-Unis, où l'on pensait que 96 caractères seraient suffisants pour longtemps.

Il y a fondamentalement quatre façons d'encoder des caractères Unicode dans des octets :

UTF-8

128 caractères sont encodés en utilisant 1 octet : les caractères ASCII.
1920 caractères sont encodé en utilisant deux octets : le latin, le grec, le cyrillique, le copte, l'arménien, l'hébreu, les caractères arabes.
63488 caractères sont encodés en utilisant 3 octets, le chinois et le japonais entre autres.
Les 2147418112 caractères restant (non encore assignés) peuvent être encodés en utilisant 4, 5 ou 6 caractères. Pour plus d'informations sur UTF-8, faites `man 7 utf-8` (cette page est contenue dans le package ldpman-1.20).

UCS-2

Chaque caractère est représenté par deux octets. Cet encodage peut représenter seulement les 65536 premiers caractères d'Unicode.

UTF-16

C'est une extension d'UTF-2 qui peut représenter 11144112 caractères Unicode. Les 65536 premiers caractères sont représentés par deux octets, les autres par quatre.

UCS-4

Chaque caractère est représenté par 4 octets.

L'espace nécessaire pour encoder un texte, comparativement aux encodages actuellement en usage (8 bits par caractères pour les langues européennes, plus pour le chinois/japonais/coréen), est le suivant : (Cela a une influence sur l'espace disque, et la vitesse des communications réseau.

UTF-8

Pas de changement pour l'ASCII américain, juste quelques pourcents supplémentaires pour ISO-8859-1, 50 % de plus pour le chinois/japonais/coréen, 100 % de plus pour le grec et le cyrillique.

UCS-2 et UTF-16

Pas de changement pour le chinois/japonais/coréen, augmentation de 100 % pour l'US ASCII et ISO-8859-1, le grec et le cyrillique.

UCS-4

Augmentation de 100% pour le chinois/japonais/coréen. De 300% pour l'US ASCII et ISO-8859-1, le grec et le cyrillique.

Étant donné la pénalité pour les documents américains et européens, causée par UCS-2, UTF-8 et UCS-4, il semble peu probable que ces encodages aient un potentiel pour une utilisation à grande échelle. L'API Win32 Microsoft supporte l'encodage UCS-2 depuis 1995 (au moins), cependant cet encodage n'a pas été largement adopté pour les documents -SJIS demeure prédominant au Japon.

D'un autre côté UTF-8 a le potentiel pour une utilisation à large échelle, puisqu'il ne pénalise pas les utilisateurs américains et européens, et que beaucoup de logiciels de "traitement de texte" (NDT : au sens large) n'ont pas besoin d'être changés pour supporter UTF-8.
Nous allons maintenant expliquer comment configurer votre système Linux pour qu'il utilise UTF-8 comme encodage de texte.

1.2.1 Notes pour les développeurs C/C++

L'approche de Microsoft Win32 rend facile pour les développeurs la production de versions Unicode de leurs programmes : Vous définissez Unicode ("#define UNICODE") au début de votre programme, et changez alors un grand nombre d'occurrences de `char` en `TCHAR` jusqu'à ce que votre programme compile sans Warnings. Le problèmes est que vous avez au final deux versions de votre programme : une qui comprend le texte UCS-2 mais pas les encodages 8-bit, et une autre qui ne comprend que les vieux encodages 8-bit.

En plus, il y a une complication qui affecte UCS-2 et UCS-4 : l'ordre de la représentation interne des nombre ("the endianness"). Le registre de systèmes de codage de caractères de la IANA dit à l'égard de ISO-10646-UCS-2 : "il faut spécifier l'ordre de la représentation interne des nombres à l'intérieur du réseau, le standard ne le spécifie pas". Cette représentation est "big endian" en contexte réseau, alors que Microsoft recommande dans ses outils de développement C/C++ d'utiliser une représention dépendante de la machine (c.a.d. "little endian" sur les processeurs ix86), et d'ajouter une marque de polarité (BOM) au début du document, ou d'utiliser des heuristiques basées sur la statistique.

D'un autre côté l'approche de UTF-8 garde `char*` comme type standard pour le stockage des chaînes en C. Il en résulte que votre programme supportera l'US ASCII, indépendamment de toute variable d'environnement, et supportera les textes encodés en ISO-8859-1 et UTF-8 à condition que la variable d'environnement LANG soit positionnée en conséquence.

1.3 Liens

La liste de ressources de Markus Kuhn (mise à jour très régulièrement) :

- *http://www.cl.cam.ac.uk/˜mgk25/unicode.html*
- *http://www.cl.cam.ac.uk/˜mgk25/ucs-fonts.html*

Un survol d'Unicode, UTF-8 et des programmes fonctionnant avec UTF-8 de Roman Czyborra :
http://czyborra.com/utf/#UTF-8

Des exemples de fichiers UTF-8 :

- Les fichiers `quickbrown.txt`, `utf-8-test.txt`, `utf-8-demo.txt` dans le répertoire `examples` dans le package ucs-fonts de Markus Kuhn
  http://www.cl.cam.ac.uk/˜mgk25/download/ucs-fonts.tar.gz
- *ftp://ftp.cs.su.oz.au/gary/x-utf8.html*
- Le fichier `iso10646` dans le package trans-1.1.1 de Kosta Kosti
  ftp://ftp.nid.ru/pub/os/unix/misc/trans111.tar.gz
- *ftp://ftp.dante.de/pub/tex/info/lwc/apc/utf8.html*
- *http://www.cogsci.ed.ac.uk/˜richard/unicode-sample.html*

2 Configuration de l'affichage

Nous supposons que vous avez déjà adapté votre console Linux et la configuration de X11 à votre clavier et avez positionné correctement la variable de localisation LANG. Ceci est expliqué dans le Danish/International HOWTO, et dans les autres HOWTOS nationaux : Finish, French, German, Italian, Polish, Slovenian, Spanish, Cyrillic, Hebrew, Chinese, Thai, Esperanto. Mais, s'il vous plaît, ne suivez pas le conseil donné dans le Thai-HOWTO vous disant de faire croire que vous utilisez des caractères ISO-8859-1 (U0000..U00FF) alors que vous tapez des caractères Thai (U0E01..U0E5B). Faire cela ne vous causera que des problèmes lorsque vous passerez à Unicode.

2.1 La console Linux

Je ne parle pas tellement de la console ici, parce que je ne l'utilise que pour rentrer mon login, password, et taper `xinit` sur les machines dépourvues de xdm.

Mais revenons à nos moutons : le package kbd-0.99 *ftp://sunsite.unc.edu/pub/Linux/system/keyboards/kbd-0.99.tar.gz* , et une version largement étendue, le package console-tools-0.2.2

ftp://sunsite.unc.edu/pub/Linux/system/keyboards/console-tools-0.2.2.tar.gz , contiennent dans le répertoire kbd-0.99/src/ (ou console-tools-0.22/screenfonttools/) deux programmes : «unicode_start» et «unicode_stop». Quand vous appelez «unicode_start», la sortie de la console est interprétée comme de l'UTF-8. De plus, le clavier est mis en mode Unicode (voir "man kbd_mode"). Dans ce mode, les caractères Unicode tapés par Alt-x1...Alt-xn (où x1...xn sont les chiffres de pavé numérique) seront émis en UTF-8. Si votre clavier, ou plus précisément, votre keymap a des touches correspondant à des caractères non ASCII(comme le Umlaute allemand), que vous souhaiteriez pouvoir «CapsLocker», vous devez appliquer au kernel le patch *linux-2.2.9-keyboard.diff* ou *linux-2.3.12-keyboard.diff* .

Vous voudrez probablement afficher des caractères de différents alphabets sur le même écran. Pour cela, vous aurez besoin d'une fonte Unicode pour la console.
Le package *ftp://sunsite.unc.edu/pub/Linux/system/keyboards/console-data-1999.08.29.tar.gz*

contient une fonte (LatArCyrHeb-{08,14,16,19}.psf), qui couvre les lettres pour le latin, le cyrillique, l'hébreu, et l'arabe. Il supporte les parties 1, 2, 3, 4, 5, 6, 8, 9, 10 d'ISO-8859 à lui tout seul. Pour l'installer, copiez le dans /usr/lib/kbd/consolefonts/, et exécutez
`/usr/bin/setfont /usr/lib/kbd/consolefonts/LatArCyrHeb-14.psf`.

Si vous voulez que le copier-coller marche avec les consoles UTF-8, vous aurez besoin du patch *linux-2.3.12-console.diff*

d'Edmund Thomas, Grimley Evans et Stanislav Voronyi.

2.2 Fontes étrangères pour X11

N'hésitez pas à installer des fontes cyrilliques, chinoises, japonaises, etc. Même si ce ne sont pas des fontes Unicodes, elles aideront à afficher des documents Unicode : au moins Netscape Communicator 4 et Java feront usage des fontes étrangères si elles sont disponibles.

Les programmes suivants sont utiles pour installer des fontes :

- "mkfontdir répertoire" prépare un répertoire de fontes utilisables par le serveur X. Il doit être exécuté après avoir installé les fontes dans un répertoire.
- "xset fp+ répertoire" ajoute un répertoire au chemin de fontes actuel du serveur X. Pour que ce soit permanent, ajoutez une ligne "FontPath" à votre fichier /etc/XF86Config, dans la section "Files".
- "xset fp rehash" doit être exécuté après avoir appelé mkfontdir sur un répertoire qui est déjà contenu dans le chemin de fontes actuel du serveur X.
- "xfontsel" vous permet de parcourir les fontes installées en filtrant selon les diverses propriétés des fontes.
- "xlsfonts -fn motif-de-recherche" liste toutes les fontes qui correspondent à un motif de recherche. Il affiche aussi diverses propriétés des fontes. En particulier, "xlsfonts -ll -fn fonte" liste les propriétés de la fonte CHARSET_REGISTRY et CHARSET_ENCODING, qui ensemble déterminent l'encodage de la fonte.
- "xfd -fn fonte" affiche une fonte page par page.

Les fontes suivantes sont disponibles gratuitement (liste non exhaustive) :

- Celle contenues dans XFree86, quelquefois contenues dans un package séparé. Par exemple, la SuSE a seulement les fontes 75 dpi normales dans le package "xf86" de base. Les autres fontes sont dans les packages "xfnt100", "xfntbig", "xfntcyr", "xfntscl".

- Les fontes internationales pour Emacs, *ftp://ftp.gnu.org/pub/gnu/intlfonts/intlfonts-1.1.tar.gz* . Comme il a été mentionné précédemment, elles sont utiles même si vous préférez XEmacs à GNU Emacs, ou même si vous n'utilisez pas Emacs du tout.

2.3 Les fontes Unicode pour X11

Les applications qui souhaitent pouvoir afficher du texte utilisant différentes alphabets (comme le cyrillique et le grec) en même temps, peuvent le faire en utilisant les différentes fontes X pour chaque partie de texte. C'est ce que font Netscape Communicator et Java. Cependant, cette approche est plus compliquée, parce que au lieu de travailler avec "Font" et "XFontStruct", le programmeur devra utiliser "XFonSet", et aussi parce que toutes les fontes dans le jeu de fontes doivent avoir les mêmes dimensions.

- Markus Kuhn a assemblé des fontes à largeur fixe (fixed width) de 75 dpi avec Unicode couvrant le latin, le grec, le cyrillique, l'arménien, le géorgien, l'hébreu, et les écritures symboliques. Elles couvrent les parties 1 à 10 et 13 à 15 de ISO-8859 en un seul jeu de fontes. Cette fonte est nécessaire pour utiliser un xterm en mode UTF-8. *http://www.cl.cam.ac.uk/˜mgk25/download/ucs-fonts.tar.gz*
- Roman Czyborra a assemblé une fonte 8x16/16x16 75 dpi avec l'encodage Unicode couvrant une partie énorme d'Unicode. Teléchargez unifont.hex.gz et hex2bdf depuis *http://czyborra.com/unifont/*

  Elle n'est pas à largeur fixe : 8 pixels de large pour les caractères européens, 16 pour les caractères chinois. Instructions d'installation :

  ```
  $ gunzip unifont.hex.gz
  $ hex2bdf < unifont.hex > unifont.bdf
  $ bdftopcf -o unifont.pcf unifont.bdf
  $ gzip -9 unifont.pcf
  $ cp unifont.pcf.gz /usr/X11R6/lib/X11/fonts/misc
  $ cd usr/X11R6/lib/X11/fonts/misc
  $ mkfontdir
  $ xset fp rehash
  ```
- Primoz Peterlin a assemblé un famille de fontes ETL couvrant le latin, le grec, le cyrillique, l'arménien, le géorgien, et l'hébreu. *ftp://ftp.x.org/contrib/fonts/etl-unicode.tar.gz* . Utilisez le programme "bdftopcf" pour l'installer.

2.4 Unicode xterm

xterm fait partie de X11R6 et XFree86, mais il est maintenu séparément par Tom Dickey. *http://www.clark.net/pub/dickey/xterm/xterm.html* . Les nouvelles versions (patch niveau 109 et plus) supportent la conversion des touches (keystrokes) en UTF-8 avant de les envoyer à l'application qui tourne dans le xterm, et l'affichage des caractères Unicode que l'application renvoie comme une séquence d'octets UTF-8.

Pour obtenir un xterm UTF-8 fonctionnel, vous devez :

- Rapatrier *http://www.clark.net/pub/dickey/xterm/xterm.tar.gz* .
- Le configurer en exécutant "./configure -enable-wide-chars...", puis le compiler et l'installer.
- Avoir une fonte Unicode à largeur fixe installée. ucs-fonts.tar.gz de Markus Kuhn (voir ci-dessus) est fait pour ça.
- Lancer "xterm -u8 -fn fixed". L'option "-u8" enclenche le support d'Unicode et d'UTF-8. La fonte "fixed" est celle de Markus Khun.

- Jeter un oeil aux fichiers-exemples contenus dans le package ucs-fonts de Markus Khun :

```
$ cd .../ucs-fonts
$ cat quickbrown.txt
$ cat utf-8-demo.txt
```

  Vous devriez voir (entre autre) des caractères grecs et cyrilliques.

- Pour configurer xterm pour qu'il utilise le support UTF-8 dès le lancement, ajoutez la ligne "XTerm*utf8:1" à votre $HOME/.Xdefaults (pour vous seul). Je ne recommande pas de changer directement /usr/X11R6/lib/X11/app-defauts/XTerm, parce que vos changements seront effacés lorsque vous installerez une nouvelle version de XFree86.

- Si vous avez aussi changé le nom de la fonte, vous aurez besoin d'une ligne "*VT100*font: votre-fonte" dans votre fichier $HOME/.Xdefaults. Pour les fontes "fixes" ce n'est pas nécessaire, puisque "fixed" est la valeur par défaut de toute façon.

2.5 Divers

Un petit programme qui teste si une console Linux est en mode UTF-8 peut être trouvé dans le package *ftp://sunsite.unc.edu/pub/Linux/system/keyboards/x-lt-1.18.tar.gz*

de Ricardas Cepas. Ce sont les fichiers testUTF-8.c et testUTF8.c.

3 Configuration des locales

3.1 Les fichiers et le kernel

Vous pouvez maintenant utiliser n'importe quel caractère Unicode dans les noms de fichiers. Ni le kernel ni aucun utilitaire système ne nécessite de modifications. Ceci parce que les noms de fichiers dans le kernel peuvent être n'importe quoi qui ne contient ni octet nul, ni '/' (utilisé pour délimiter les sous-répertoires). Quand ils sont encodés en utilisant UTF-8, les caractères non-ASCII ne seront jamais encodés en utilisant un octet nul ou un slash. La seule conséquence est que les noms de fichiers et de répertoires occupent plus d'octets qu'ils ne contiennent de caractères. Par exemple, un nom de fichier contenant cinq caractères grecs apparaîtra pour le kernel comme un nom de fichier de 10 octets. Le kernel ne sait pas (et n'a pas besoin de savoir) que ces octets sont affichés en grec.

C'est une théorie générale, qui est vraie tant que vos fichiers restent sur un système Linux. Sur les systèmes de fichiers utilisés depuis d'autres systèmes d'exploitation, `mount` possède des options pour contrôler la conversion des noms de fichiers de/vers UTF-8 :

- Les systèmes de fichiers "vfat" ont une option "utf8".
  Voir le fichier *file:/usr/src/linux/Documentation/filesystems/vfat.txt* . Quand vous donnez à mount une option "iocharset" différente de celle utilisée par défaut (qui est "iso8859-1"), les résultats avec et sans l'option "utf8" ne sont pas cohérents. Par conséquent, je ne conseille pas d'utiliser l'option "iocharset" de mount.

- Les systèmes de fichiers "msdos", "umsdos" ont la même option de montage, mais il semble qu'elle n'ait pas d'effet.

- Le système de fichiers "iso9660" a une option de montage "utf8".
  Voir *file:/usr/src/linux/Documentation/filesystems/isofs.txt*

- À partir des kernels Linux 2.2.x, le système de fichier "ntfs" a une option de montage"utf8".
  Voir *file:/usr/src/linux/Documentation/filesystems/ntfs.txt*

Les autres systèmes de fichiers (nfs, smbfs, ncpfs, hpfs, etc.) ne convertissent pas les noms de fichiers. Par conséquent ils accepteront les noms de fichier Unicode encodés en UTF-8 seulement si l'autre système d'exploitation les supporte. Rappelez vous que pour qu'une option de montage soit appliquée aux prochains montages, vous devez l'ajouter à la quatrième colonne de la ligne correspondante dans /etc/fstab.

3.2 Le kernel et les ttys

Les ttys sont en quelque sorte des tubes bidirectionnels entre deux programmes, autorisant des choses comme la répétition (echoing) ou l'édition de la ligne de commande. Quand dans un xterm, vous exécutez la commande "cat" sans arguments, vous pouvez entrer et éditer autant de lignes que vous voulez, elles seront répétées en retour ligne par ligne. Les actions d'édition du kernel ne sont pas correctes, en particulier les touche Backspace et Tab ne seront pas traitées correctement.

Pour résoudre ce problème, vous devez :

- Appliquer le patch *linux-2.0.35-tty.diff*

  ou *linux-2.2.9-tty.diff*

  ou *linux-2.3.12-tty.diff*

  et recompiler votre kernel.

- Si vous utilisez la glibc2, appliquer le patch *glibc211-tty.diff*

  et recompiler votre libc. Si vous n'êtes pas aussi aventureux, il suffit de patcher une version déjà installée avec le fichier inclus : *glibc-tty.diff*

- Appliquer le patch *stty.diff*

  à GNU sh-utils-1.16b, et recompiler le programme `stty`. Testez le ensuite en utilisant `stty -a` et `stty iutf8`.

- Ajouter la commande `stty iutf8` au script unicode_start, et la commande `stty -iutf8` au script unicode_stop.

- Appliquer le patch *xterm.diff*

  à xterm-109, et recompiler "xterm", puis le tester en lançant `xterm -u8` / `xterm +u8` et en lançant `stty -a` et un `cat` interactif à l'intérieur.

Pour que ce changement soit persistant même à travers rlogin et telnet, vous devrez aussi :

- Définir des nouvelles valeurs pour la variable d'environnement TERM, "linux-utf8" comme alias pour "linux", et "xterm-utf8" comme alias pour "xterm". Si vous avez la bibliothèque ncurses sur votre système et la base de données /usr/lib/terminfo (ou /usr/share/terminfo), faites cela en éxecutant

  ```
  $ tic linux-utf8 . terminfo
  $ tic xterm-utfu . terminfo
  ```

  sur un compte utilisateur (cela créera les entrées terminfo dans votre repertoir $HOME/.terminfo). Voilà *linux-utf8.terminfo* et *xterm-utf8.terminfo* .
  Je ne recommande pas de lancer ces commandes en tant que root, parce que cela créera les entrées terminfo dans /urs/lib/terminfo, où, elles seront probablement effacées lors de la prochaine mise à jour de votre système. Si votre système possède un fichier /etc/termcap, vous devriez aussi éditer ce fichier :

copiez les entrées linux et xterm, et donnez leur les nouveaux noms "linux-utf8" et "xterm-utf8". Le fichier *termcap.diff*

contient un exemple.

- À chaque fois que vous appelez "unicode_start" et "unicode_stop" depuis la console, exécutez aussi "export TERM =linux-utf8", ou "export TERM=linux", respectivement.
- Appliquer le patch *xterm2.diff*

  à xterm-0.9, recompiler "xterm", et et enlever toutes les lignes "XTerm*termName" des fichiers /usr/X11R6/lib/X11/app-defaults/XTerm et $HOME/.Xdefaults. Maintenant xterm donne à TERM la valeur "xterm-utf8" plutôt que "xterm".
- Appliquer les patches *netkit.diff* , *netkitb.diff* et *telnet.diff*

  puis recompiler "rlogind" et "telnetd". Maintenant rlogin et telnet mettent tty en mode d'édition UTF-8 à chaque fois que la variable d'environnement TERM est "linux-ut8" ou "xterm-utf8".

3.3 Conversion de données générales

Vous aurez besoin d'un programme pour convertir vos fichiers texte locaux (probablement ISO-8859-1) en UTF-8. L'alternative serait de continuer à utiliser des textes qui utilisent différents encodages sur la même machine, mais ce n'est pas une bonne solution sur le long terme. Un de ces programmes est "iconv", qui est livré avec la glibc-2.1. Tapez simplement

```
icon --from-code=ISO-8859-1 --to-code=UTF-8 < vieux_fichier > nouveau_fichier
```

Voici deux scripts shell très pratiques, appelés "i2u" *i2u.sh* (pour conversion de ISO à UTF) et "u2i" *u2i.sh* (pour conversion de UTF à ISO). [skip adapt..]

Si vous n'avez pas la glibc-2.1 et iconv installés, vous pouvez utiliser GNU recode 3.5 à la place. "i2u" *i2u_recode.sh* est "recode ISO-8859-1..UTF-8" "u2i" *u2i_recode.sh* est "recode UTF-8..ISO-8859-1".
ftp://ftp.iro.umontreal.ca/pub/recode/recode-3.5.tar.gz
ftp://ftp.gnu.org/pub/gnu/recode/recode-3.5.tar.gz
Notes : vous devez utiliser GNU recode 3.5 ou plus. Pour compiler GNU recode sur des plateformes sans glibc-2 (c'est à dire sur toutes les plateformes sauf les systèmes Linux récents), vous devez le configurer avec "–disable-nls", autrement l'édition des liens échouera.

Sinon, vous pouvez aussi utiliser CLISP à la place. Voici "i2u" et "u2i" en version lisp : *i2u.lsp* et *u2i.lsp* .
Note : Vous devez avoir une version de CLISP qui date au plus de juillet 1999.
ftp://clisp.cons.org/pub/lisp/clisp/source/clispsrc.tar.gz
D'autres programmes de conversion de données existent, mais ils sont moins puissants que GNU recode. Ce sont

- "trans"
  ftp://ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/trans113.tar.gz
- "tcs" qui vient du système d'exploitation Plan9 :
  ftp://ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/tcs.tar.gz
- et "utrans/uhtrans/hutrans" par G.Adam Stanislav <*adam@whizkidtech.net*>
  ftp://ftp.cdrom.com/pub/FreeBSD/distfiles/i18ntools-1.0.tar.gz

3.4 Les variables d'environnement locales

Vous pouvez avoir les variables d'environnement suivantes positionnées, contenant les noms de locales :

LANGUAGE

Remplacement pour LC_MESSAGES. Seulement utilisé par GNU gettext.

LC_ALL

Remplacement pour toute les autres variables LC_* :

LC_CTYPE, LC_MESSAGES, LC_COLLATE, LC_NUMERIC, LC_MONETARY, LC_TIME

Ce sont des variables individuelles pour : le type des caractères et leur encodage, les messages en langue maternelle, les règles de classement, le format des nombres, le format des montants monétaires, l'affichage de la date et de l'heure.

LANG

Valeur par défaut pour toutes les variables LC_*

(Voir '`man 7 locale`' pour une description détaillée.)

Chacune des variables LC_* et LANG peuvent contenir un nom de locale de la forme suivante :

language[_territory[.codeset]][@modifier]

Où *language* est un code de langue *ISO 639*

(en minuscules), *territory* est un code de pays *ISO 3166*

(en majuscules), *codeset* désigne une table de caractères, et *modifier* d'autres attributs particuliers (par pour exemple indiquer un dialecte particulier d'une langue, ou une orthographe non standard).

LANGUAGE peut contenir plusieurs noms de locale, séparés par deux points (:).

Pour dire à votre système et à toutes les applications que vous utilisez UTF-8, vous devez ajouter un suffixe d'UTF-8 à vos noms de locales. Par exemple, si vous utilisiez

```
LANGUAGE=de:fr:en
LC_CTYPE=de_DE
```

vous le changeriez en

```
LANGUAGE=de.UTF-8:fr.UTF-8:en.UTF-8
LC_CTYPE=de_DE.UTF-8
```

3.5 Créer les fichiers pour le support des locales

Si vous avez la glibc-2.1 ou glibc-2.1.1 ou glibc-2.1.2 installée, vérifiez d'abord en utilisant "localedef -help" que le répertoire système pour le tables de caractères est /usr/share/i18n/charmaps. Puis appliquez au fichier /usr/share/i18n/charmaps/UTF8 le patch *glibc21.diff*

ou *glibc211.diff*

ou *glibc212.diff* , respectivement. Puis créez les fichiers de support pour toute les locales UTF-8 que vous voulez utiliser, par exemple :

```
$ localedef -v -c -i de_DE -f UTF8 /usr/share/locale/de_DE.UTF-8
```

Généralement vous n'avez pas besoin de créer des variables appelées "de" ou "fr" sans suffixe pour le code du pays, parce que ces locales sont normalement utilisées seulement par la variable LANGUAGE, et pas par les variables LC_*. De plus LANGUAGE est seulement utilisé en remplacement de LC_MESSAGES.

3.6 Ajouter le support pour la bibliothèque C

La glibc-2.2 supportera les locales multi-octets (de plusieurs octets), en particulier les locales UTF-8 créées plus haut. Mais les glibc-2.1 et 2.1.1 ne la supportent pas réellement. Par conséquent le seul effet réel de la création des fichiers /usr/local/share/de_DE.UTF-8/* ci dessus est que `setlocale(LC_ALL,"")` retournera "de_DE.UTF-8", conformément à vos variables d'environnement, au lieu d'enlever le suffixe "UTF-8".

Pour ajouter le support pour la locale UTF-8, vous devriez compiler et installer la bibliothèque 'libutf8_plug.so', depuis *libutf8-0.5.2.tar.gz* . Puis vous pouvez positionner la variable d'environnement LD_PRELOAD pour qu'elle pointe sur la bibliothèque installée :

```
export LD_PRELOAD=/usr/local/lib/libutf8_plug.so
```

Alors, dans chaque programme lancé avec cette variable d'environnement positionnée, les fonctions de libutf8_plug.so seront appelées à la place des originales dans /lib/libc.so.6. Pour plus d'informations sur LS_PRELOAD, voyez "man 8 ld.so".

Tout cela ne sera plus nécessaire quand la glibc-2.2 sortira.

3.7 Conversion des catalogues de messages

Maintenant ajoutons un contenu à ces nouvelles locales. Les commandes /bin/sh suivantes convertiront vos catalogues de messages au format UTF-8. Elles doivent être lancées en tant que root, et nécessitent les programmes 'msgfmt' et 'msgunfmt' de GNU gettext-0.10.35 *convert-msgcat.sh* .
Ceci non plus ne sera plus nécessaire une fois que la glibc-2.2 sera sortie, parce qu'alors la fonction gettext convertira les chaînes de caractères de façon appropriée depuis la table de caractères du traducteur vers la table de caractères de l'utilisateur, en utilisant soit iconv soit librecode.

4 Applications spécifiques.

4.1 Le réseau

4.1.1 rlogin

Marche bien avec les patches mentionnés précédemment.

4.1.2 telnet

Telnet n'est pas ne traite pas les caractères 8-bits (il n'est pas"8-bit-clean") par défaut. Pour pouvoir envoyer des codes de touches Unicode à un hôte distant, vous devez mettre telnet en mode "outbinary". Il y a deux façons de faire cela :

```
$ telnet -L <hote>
```

et

```
$telnet
telnet> set outbinary
telnet> open <hote>
```

en outre, utilisez les patches mentionnés précédemment.

4.2 Les navigateurs

4.2.1 Netscape

Netscape 4.05 ou plus peut afficher des documents HTML en encodage UTF-8. Tout ce qu'un document nécessite est la ligne suivante, située entre les tags <head> et </head>

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Netscape 4.05 ou plus peut aussi afficher du HTML et du texte en encodage UCS-2 avec le [byte-order mark].
http://www.netscape.com/computing/download/

4.2.2 Lynx

Lynx 2.8 a un écran d'options (touche 'O') qui permet de sélectionner la table de caractères utilisée à l'affichage. Quand il, tourne dans un xterm ou depuis une "Linux console" en mode UTF-8, sélectionnez "UNICODE UTF-8".

Maintenant, encore une fois, tout ce qu'un document nécessite est la ligne suivante, entre les tags <head> et </head> :

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Quand vous visualisez des fichiers texte encodés en UTF-8, vous devez aussi passer à la ligne de commande l'option "-assume_local_charset=UTF-8" (affecte seulement les URLs de type file:/...) ou "-assume_charset=UTF-8" (affecte toute les URLs). Sinon, dans Lynx-2.2.8, vous pouvez, dans l'écran d'options, changer le jeu de caractères supposé par défaut ("assumed document character set") en "utf-8".

Il y a aussi, dans l'écran d'options, une option permettant de choisir le jeu de caractères par défaut ("preferred document character set"), mais cela n'a pas d'effet, au moins avec les URLs file:/... et http://... servies par apache-1.3.0.

Cependant, il y a un problème concernant les espaces et les fins de lignes : regardez à la section russe de x-utf-8.html, ou utf-8-demo.txt.

Notez aussi que dans Lynx-2.8.2, configuré avec l'option -enable-prettysrc, les jolies combinaisons de couleurs ne marchent plus correctement quand le jeu de caractères pour l'affichage à été positionné à "UNICODE-UTF-8". Ce problème est résolu par le patch *lynx282.diff* .

D'après les développeurs de lynx : "Pour une utilisation sérieuse de l'affichage UTF-8 avec Lynx, il est toujours recommandé de compiler avec la bibliothèque slang et -DSLANG_MBCS_HACK."
ftp://ftp.gnu.org/pub/gnu/lynx/lynx-2.8.2.tar.gz
http://lynx.browser.org/
http://www.slcc.edu/lynx/
ftp://lynx.isc.org/

4.2.3 Pages de test

Des pages de test pour les navigateurs peuvent être trouvées sur les pages d'Alan Wood et James Kass : *http://www.hclrss.demon.co.uk/unicode/#links* , *http://home.att.net/˜jameskass/*

4.3 Les éditeurs

4.3.1 yudit

Le programme `yudit` de Gáspár Sinai *http://czyborra.com/yudit/*

est un éditeur de texte unicode de premier ordre pour le système X Window. Il supporte le traitement simultané de beaucoup de langages, méthodes d'entrée, conversions de caractères locaux standards. Il est équipé pour supporter l'entrée de texte dans tous les langages avec seulement un clavier anglais, en utilisant des tables de configuration du clavier.

Il peut être compilé en 3 versions : interface graphique Xlib , KDE, ou Motif.

Il peut être personnalisé très facilement. Typiquement, vous voudrez modifier votre fonte. Depuis le menu font je choisis "Unicode". Puis, puisque la commande "xlsfonts '*_*-iso10646-1" donnait toujours un affichage ambigu, je choisis un taille de fonte de 13 (pour correspondre à la fonte fixe de 13 pixels de Markhus Kuhn).

Ensuite, vous personnaliserez votre méthode d'entrée. Les méthodes "Straight", "Unicode" et "SGML" sont les plus intéressantes. Pour avoir des détails sur les autres méthodes d'entrées incorporées, regardez dans "/usr/local/share/yudit/data/".

Pour qu'un changement devienne un réglage par défaut pour les prochaines sessions, éditez votre fichier $HOME/.yuditrc.

Les fonctionnalités générales de l'éditeur sont limitées à l'édition, le copier-coller, et le "chercher-remplacer" (search&replace). Pas de fonction d'annulation (undo).

4.3.2 mined98

`mined98` est un petit éditeur de texte de Michiel Huisjes, Achim Müller et Thomas Wolff : *http://www.inf.fu-berlin.de/˜wolff/mined.html* . Il vous permet d'éditer des fichier encodés en UTF-8 ou 8 bits, dans un xterm UTF-8 ou 8-bits. Il dispose aussi de puissantes fonctionnalités pour entrer les caractères Unicode.

Quand mined est lancé dans un xterm ou une console Linux en mode UTF-8, vous devriez positionner la variable d'environnement `utf8_term`, ou appeler mined avec l'option `-U`.

mined vous permet d'éditer des fichiers encodés aussi bien en UTF-8 qu'en 8-bits. Par défaut il utilise un heuristique d'auto-détection. Si vous ne voulez pas vous reposer sur des heuristiques, passez l'option `-u` à la ligne de commande lorsque vous éditez un fichier UTF-8, ou `+u` lorsque vous éditez un fichier encodé en 8 bits. Vous pouvez changer l'interprétation à n'importe quel moment depuis l'éditeur : il affiche l'encodage ("L:h" pour du 8-bits, "U:h" pour de l'UTF-8) dans la ligne de menu. Vous pouvez cliquer sur le premier de ces caractères pour le changer.

Quelques bémols :

- Le binaire Linux dans la distribution est dépassé et ne supporte pas UTF-8. Vous devrez recompiler un binaire à partir des sources. Installez ensuite src/mined dans /usr/local/bin/mined et doc/mined.help dans /usr/local/man/cat1/mined.1, de façon à ce que `ESC h` le trouve.

- mined ignore votre réglage "stty erase". Quand votre touche backspace renvoie un code ASCII 127, et que vous avez positionné "stty erase ^?" - ce qui est finalement le réglage le plus sûr - vous devez appeler mined avec l'option `-B` de façon à ce que la touche backspace efface le caractère à gauche du curseur.
- Les touches "Home", "End", "Delete" ne marchent pas.

4.3.3 vim

`vim` (depuis la version 5.4m) supporte les locales multi-octets, mais seulement si la bibliothèque X a le même support, et seulement pour les encodages avec au moins deux octets par caractères, i.e les encodages ISO-2022. Il ne supporte pas l'UTF-8.

4.3.4 emacs

Avant tout, vous devriez lire la section "International Character Set Support" (noeud "International") dans le manuel d'Emacs. En particulier, notez que vous devez démarrer Emacs avec la commande

```
$ emacs -fn fontset-standard
```

pour qu'il utilise un jeu de fontes comprenant beaucoup de caractères internationaux.

À court terme, le package emacs-utf *http://www.cs.ust.hk/faculty/otfried/Mule/*

d'Otfried Cheong procure un "unicode-utf-8" pour Emacs. Après avoir compilé le programme "utf2mule" et l'avoir installé quelque part dans votre $PATH, installez aussi unicode.el, muleuni-1.el, unicode_char.el quelque part, et ajoutez les lignes

```
(setq load-path (cons "/home/user/somewhere/emacs" load-path))
(if (not (string-match "XEmacs" emacs-version))
  (progn
    (require 'unicode)
    (if (eq window-system 'x)
      (progn
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-12-*-*-*-*-*-fontset-standard")
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-13-*-*-*-*-*-fontset-standard")
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-14-*-*-*-*-*-fontset-standard")
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-15-*-*-*-*-*-fontset-standard")
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-16-*-*-*-*-*-fontset-standard")
        (create-fontset-from-fontset-spec
          "-misc-fixed-medium-r-normal-*-18-*-*-*-*-*-fontset-standard")))))
```

à votre $HOME/.emacs. Pour activer un des jeux de fontes, utilisez l'item "Set Font/FonSet" du menu mule, ou Shift-down-mouse-1. Pour l'instant les jeux de fontes avec des hauteurs de 15 et 13 ont le meilleur support Unicode, grâce aux fontes 9x15 et 6x13 de Markus Kuhn. Pour ouvrir un fichier encodé en UTF-8, vous pouvez taper

```
M-x universal-coding-system-argument unicode-utf8 RET
M-x find-file filename RET
```

ou

```
C-x RET c unicode-utf8 RET
C-x C-f filename RET
```

Notez que cela marche avec Emacs seulement en mode fenêtre, pas en mode terminal.

Richard Stallman prévoit à long terme d'ajouter un support d'UTF-8 intégré à Emacs.

4.3.5 Xemacs

(Cette section est écrite par Gilbert Baumann.)

Voici comment apprendre à XEmacs (20.4 configuré avec MULE) l'encodage UTF-8. Malheureusement vous aurez besoin des sources pour pouvoir le patcher.

D'abord vous aurez besoin de ces fichiers fournis par Tomohiko Morioka :

http://turnbull.sk.tsukuba.ac.jp/Tools/XEmacs/xemacs-21.0-b55-emc-b55-ucs.diff

http://turnbull.sk.tsukuba.ac.jp/Tools/XEmacs/xemacs-ucs-conv-0.1.tar.gz

Le .diff est un diff pour les sources C. Le tarball contient du code elisp, qui fournit beaucoup de tables de code qui permettent la conversion depuis et vers Unicode. Comme le nom du diff le suggère, il est prévu pour XEmacs-21. J'ai eu besoin d'aider un peu 'patch'. La différence la plus notable avec mes sources XEmacs-20.4 est que file-coding.[ch] était appelé mule-coding.[ch]

Pour ceux qui connaissent peu XEmacs-MULE (comme moi) voici un guide rapide :

Ce que nous appelons un encodage est appelé par MULE "coding system". Les commandes les plus importantes sont :

```
M-x set-file-coding-system
M-x set-buffer-process-coding-system [comint buffer]
```

et la variable 'file-coding-system-alist', qui guide 'find-file' pour qu'il trouve l'encodage utilisé. Une fois que tout marchait, la première chose que j'ai faite fut

ceci `gb-hacks.el` .

Ce code cherche une ligne de mode spéciale commençant par _*_ quelque part dans les 600 premiers octets du fichier qui va être ouvert. Si cette ligne contient un champ "Encoding: xyz;" et que l'encodage xyz ("coding system" dans le langage d'Emacs) existe, il le sélectionne. Donc maintenant vous pouvez utiliser :

```
;;;  _*_ Mode: Lisp; Syntax: Common-Lisp; Package: CLEX; Encoding:
utf-8; _*_
```

et Emacs entrera en mode utf-8 à partir de là.

Une fois que tout marchait, j'ai défini une macro pour \u03BB (lambda grec) comme ceci :

```
(defmacro \u03BB (x) '(lambda .,x))
```

4.3.6 nedit

4.3.7 xedit

En théorie, `xedit` devrait être capable d'éditer des fichiers UTF-8 si vos locales sont configurées en conséquence (voir au dessus), et que vous ajoutez la ligne "Xedit*international: true" à votre fichier

$HOME/.Xdefaults. En pratique, il reconnaîtra les encodages UTF-8 des caractères non ASCII, mais il les affichera comme des séquences de caractères "@".

4.3.8 axe

En théorie, `axe` devrait être capable d'éditer des fichiers UTF-8 si vos locales sont configurées en conséquence (voir au dessus), et que vous ajoutez la ligne "Axe*International: true" à votre fichier $HOME/.Xdefaults. En pratique, il vous laissera simplement un joli fichier core.

4.3.9 pico

4.3.10 TeX

Les distributions de teTex 0.9 (et supérieures) contiennent une adaptation Unicode de TeX, appelée Omega (*http://www.gutenberg.eu.org/omega/*

ftp://ftp.ens.fr/pub/tex/yannis/omega), mais est-ce que quelqu'un connaîtrait un tutorial sur ce système ?

Autres liens peut-être en rapport :
http://www.dante.de/projekte/nts/NTS-FAQ.html ,
ftp://ftp.dante.de/pub/tex/language/chinese/CJK/

4.4 Les logiciels de courrier électronique

MIME : RFC 2279 définit les jeux de caractères UTF-8 et MIME, qui peuvent être convertis en encodage 8-bits, quoted-printable ou base64. L'ancienne norme MIME UTF-7 (RFC 2152) est considérée comme étant obsolète ("deprecated") et ne devrait plus être utilisée.

Les clients mail sortis après le 1er janvier 1999 devraient être capables d'envoyer et d'afficher des courriers encodés en UTF-8, sous peine d'être considérés comme incomplets. Mais ces courriers doivent contenir les étiquettes (labels) MIME :

```
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8-bit
```

Envoyer simplement un fichier UTF-8 vers "mail" via un *pipe* sans faire attention aux labels MIME ne fonctionnera pas. Les gens qui implémentent des clients de courrier devraient jeter un oeil à *http://www.imc.org/imc-intl/*

et *http://www.imc.org/mail-i18n.html*

Parlons maintenant des clients mail individuels (ou "mail user agents") :

4.4.1 pine

La situation pour une version non patchée de pine 4.10 est la suivante.

Pine ne fait pas de conversion de jeu de caractères. Mais il vous permet de voir des courriers UTF-8 dans une fenêtre texte UTF-8 (console Linux ou xterm).

Normalement, Pine se plaindra du fait qu'il y a différents jeux de caractères à chaque fois que vous visualiserez un courrier encodé en UTF-8. Pour vous débarrasser de cet avertissement, choisissez S (setup), puis C (config), et changez la valeur de "character set" à UTF-8. Cette option ne fera rien à part réduire le nombre d'avertissements, puisque Pine ne connaît pas UTF-8 en interne.

Notez aussi que pour Pine la notion de caractères Unicode est très limitée : il affichera les caractères latins et grecs, mais ce sont les seuls types de caractères Unicode qu'il connaît.

Un patch de Robert Brady *http://www.ents.susu.soton.ac.uk/˜robert/pine-utf8-0.1.diff*

ajoute à Pine un support UTF-8. Avec ce patch, il peut décoder et afficher les entêtes et le corps des messages correctement. Ce patch nécessite GNOME libunicode *http://cvs.gnome.org/lxr/source/libunicode/* .

4.4.2 Kmail

Kmail (comme tout KDE 1.0) ne contient absolument aucun support pour les mails en UTF-8.

4.4.3 Netscape Communicator

Le Messenger de Netscape Communicator peut envoyer et afficher des mails encodés en UTF-8, mais cela nécessite quelques interventions manuelles de l'utilisateur.

Pour envoyer un mail encodé en UTF-8 : après avoir ouvert la fenêtre "Compose", mais avant de commencer à composer le message, sélectionnez dans le menu "View -> Character Set ->Unicode (UTF-8)" puis composez votre message et envoyez le.

Quand vous recevez un courrier encodé en UTF-8, Netscape ne l'affiche malheureusement pas en UTF-8 directement, et ne donne même pas un indice visuel montrant que le courrier a été encodé en UTF-8. Vous devez sélectionner manuellement l'option adéquate dans "View -> Character Set -> Unicode (UTF-8)". Pour afficher les courriers UTF-8, Netscape utilise des fontes différenciées. Vous pouvez ajuster la fonte utilisée dans la boîte de dialogue "Edit -> Preferences -> Fonts". Choisissez la catégorie de fontes "Unicode".

4.4.4 Emacs (rmail, vm)

4.5 Autres applications en mode texte

4.5.1 less

Teléchargez *ftp://ftp.gnu.org/pub/gnu/less/less-340.tar.gz*

et appliquez le patch *less-340-utf-2.diff*

de Robert Brady <*rwb197@ecs.soton.ac.uk*> . Puis positionnez la variable d'environnement LESSCHARSET :

```
export LESSCHARSET=utf-8
```

Si vous avez positionné une variable d'environnement LESSKEY, vérifiez aussi que le fichier vers lequel elle pointe ne définit pas LESSCHARSET. Si nécessaire, régénérez ce fichier en utilisant la commande 'lesskeys', ou enlevez la variable d'environnement LESSKEYS.

4.5.2 expand, wc

Procurez vous GNU textutils-2.0 et appliquez le patch *textutils-2.0.diff* , puis lancez configure.
Ajoutez "#define HAVE_MBRTOWC 1", "#define HAVE_FPUTWC 1" à config.h. Dans src/Makefile, modifiez CFLAGS et LDFLAGS pour qu'ils incluent les répertoires où libutf8 est installé, puis recompilez.

4.5.3 col, colcrt, colrm, column, rev, ul

Procurez vous le package util-linux-2.9y, configurez le, puis définissez ENABLE_WIDECHAR dans defines.h , changez le "# if 0" en "# if 1" dans lib/widechar.h. dans text-utils/Makefile, modifiez CFLAGS et LDFLAGS pour qu'ils incluent les répertoires où libutf8 est installé. Puis recompilez.

4.5.4 figlet

Figlet 2.2 a une option pour gérer l'entrée en UTF-8 : "figlet -C utf-8".

4.5.5 kermit

Le programme de communication série C-Kermit *http://www.columbia.edu/kermit/* , dans les versions 7.0beta10 et supérieures, comprend les encodages de fichier et de transfert UTF-8 et UCS-2, et l'encodage de terminal UTF-8. La documentation de ces caractéristiques peut être trouvée à *ftp://kermit.columbia.edu/kermit/test/text/ckermit2.txt*

4.6 Autres applications X11

La Xlib de X11 ne peut malheureusement pas encore localiser UTF-8, cela devrait être travaillé prochainement.

5 Comment faire pour que vos programmes comprennent Unicode

5.1 C/C++

Le type C '`char`' est 8-bits et restera 8-bits parce qu'il désigne la plus petite unité de données adressable. Divers aménagements sont disponibles :

5.1.1 Pour le traitement de texte normal

Le standard ISO/ANSI C contient, dans une correction qui fut ajoutée en 1995, un type de caractère codé sur 16 bits '`wchar_t`', un ensemble de fonctions comme celles contenues dans <`string.h`> et <`ctype.h`> (déclarées respectivement dans <`wchar.h`> et <`wctype.h`>), et un ensemble de fonctions de conversion entre '`char *`' et '`wchar_t *`' (déclarées dans <`stdlib.h`>).

Voici de bonnes réferences pour cette interface de programmation :

- Le manuel de GNU libc-2.1, chapitres 4 "Characters Handling" et 6 "Character Set Handling"
- Les pages de manuel *man-mbswcs.tar.gz*
- La référence de la bibliothèque C Dinkumware *http://www.dinkumware.com/htm_cl/*
- La spécification Single Unix d'OpenGroup *http://www.UNIX-systems.org/online.html*

Avantages de cette interface de programmation :

- C'est un standard non propriétaire.

- Ces fonctions font ce qu'il faut, selon les locales de l'utilisateur. Tout ce qu'un programme doit faire est d'appeler `setlocale(LC_ALL,"");`.

Inconvénients de cette interface de programmation :

- Certaines de ces fonctions ne sont pas "multithread-safe" parce qu'elles conservent un état interne caché entre les appels de fonction.
- Il n'y a pas de type de donnée de première classe. Par conséquent cette API ne peut pas être utilisée raisonnablement pour tout ce qui nécessite plus d'une locale ou d'un jeu de caractères au même moment.
- La plupart des systèmes d'exploitation ont un mauvais support de cette interface de programmation.

Notes concernant la portabilité Une variable '`wchar_t`' peut être encodée en Unicode ou non. Ceci dépend de la plateforme et quelquefois aussi des locales. Une séquence multi-octets '`wchar_t`' peut être encodée en UTF-8 ou non selon la plateforme, et parfois selon les locales.

En détails, voici ce que la

Single Unix specification `http://www.UNIX-systems.org/online.html`

dit à propos du type '`wchar_t`' :

Tous les codes de caractères 16 bits dans un processus donné consistent en un nombre égal de bits. Ceci en contraste avec les caractères, qui peuvent être constitués d'un nombre variable d'octets. L'octet ou la séquence d'octets qui représentent un caractère peuvent aussi être représentés comme un code de caractère 16 bits. Les codes de caractères 16 bits fournissent donc une taille uniforme pour manipuler les données textuelles. Un code de caractère 16 bits ayant tous les bits à 0 est un code de caractère 16 bits nul (null), et termine une chaîne. La valeur des caractères larges pour chaque membre du "Portable Character Set" (i.e ASCII)*est égale quand il est utilisé en tant que seul caractère dans un caractère entier (integer) constant. Les codes de caractères 16 bits pour les autres caractères dépendent des locales et de l'implémentation. Les octets modificateurs d'état n'ont pas de représentation en code de caractère 16 bits.*

Une conséquence notable est que dans des programmes portables vous ne devriez pas utiliser des caractères non-ASCII dans des chaînes littérales. Cela signifie que même si vous savez que les doubles guillemets ont les codes U+201C et U+201D, vous ne devriez pas écrire une chaîne littérale `L"\u201cBonjour\u201d, dit il"` ou `"\xe2\x80\x9cBonjour\xe2\x80\x9d, dit il"` dans des programmes C. Utilisez plutôt GNU gettext comme cela : `gettext ("'Bonjour', dit il")`, et créez une base de données de messages en.UTF-8.po qui traduit "'Bonjour' dit il" en `"\u201cBonjour\u201d, dit il"`.

Voici une étude de la portabilité des aménagements ISO/ANSI C sur diverses implémentations d'Unix. La GNU glibc-2.2 les supportera tous, mais pour l'instant nous avons le tableau suivant.

GNU glibc-2.0.x, glibc-2.1.x

- <wchar.h> et <wctype.h> existent.
- Possède les fonctions wcs/mbs, mais pas fgetwc/fputwc/wprintf.
- Pas de locales UTF-8.
- mbrtowc retourne EILSEQ pour les octets >= 0x80.

Solaris 2.7

- <wchar.h> et <wctype.h> existent.
- Possède toutes les fonctions wcs/mbs, fgetwc/fputwc/wprintf.

- Supporte les locales UTF-8 suivantes : en_US.UTF-8, de.UTF-8, es.UTF-8, fr.UTF-8, it.UTF-8, sv.UTF-8.
- mbrtowc retourne EILSEQ pour les octets >= 0x80.

OSF/1 4.0d

- <wchar.h> et <wctype.h> existent.
- Possède toutes les fonctions wcs/mbs, fgetwc/fputwc/wprintf.
- A en plus universal.utf8@ucs4 locale, voir "man 5 unicode".
- mbrtowc ne connaît pas UTF-8.

Irix 6.5

- <wchar.h> et <wctype.h> existent.
- Possède les fonctions wcs/mbs et fgetwc/fputwc, mais pas wprintf.
- N'a pas de locales multi-octets.
- A seulement un simulacre de définition pour mbstate_t.
- N'a pas mbrtowc.

HP-UX 11.00

- <wchar.h> existe, mais pas <wctype.h>.
- Possède les fonctions wcs/mbs et fgetwc/fputwc, mais pas wprintf.
- A une locale C.utf8.
- N'a pas mbstate_t.
- N'a pas mbrtowc.

AIX 4.2

- <wchar.h> existe, mais pas <wctype.h> - utilisez à la place <ctype.h> et <wchar.h>.
- Possède les fonctions wcs/mbs et fgetwc/fputwc, mais pas wprintf.
- Possède les locales UTF-8 suivantes : ET_EE.UTF-8, LT_LT.UTF-8, LV_LV.UTF-8, ZH_CN.UTF-8.
- N'a pas mbstate_t.
- N'a pas mbrtowc.

Par conséquent je recommande l'utilisation des fonctions redémarrables et multithread-safe wcsr/mbsr. Oubliez les systèmes qui ne les ont pas (Irix, HP-UX, Aix), et utilisez le plug-in qui permet d'utiliser des locales UTF-8, libutf8_plug.so (voir ci dessous) sur les systèmes qui vous permettent de compiler des programmes qui utilisent ces fonctions wcrs/mbsr (Linux, Solaris, OSF/1).

Un avis similaire, donné par Sun dans *http://www.sun.com/software/white-papers/wp-unicode/* , section "Internationalized Applications with Unicode", est :

Pour internationaliser correctement une application utilisez les indications suivantes :

1. *Évitez l'accès direct à Unicode. Ceci est la tâche de la couche d'internationalisation de la plateforme.*
2. *Utilisez le modèles POSIX pour les interfaces multi-octets et à caractères 16 bits.*

3. *Appelez seulement les fonctions de l'API que la couche d'internationalisation fournit pour la langue et les opération spécifiques à la culture.*
4. *Restez indépendant de l'encodage.*

Si, pour une raison quelconque, vous devez vraiment supposer que 'wchar_t' est Unicode dans un morceau de code (par exemple, si vous voulez faire un traitement spécial de certains caractères Unicode), vous devriez rendre ce bout de code conditionnel selon le résultat de `is_locale_utf88()`. Autrement vous allez mettre la pagaille dans le comportement de votre programme sur d'autres plateformes, ou si d'autres locales sont utilisées. La fonction `is_locale_utf8()` est déclarée dans *utf8locale.h*

et définie dans *utf8locale.c* .

La bibliothèque libutf8 Une implémentation portable de l'API ISO/ANSI C, qui supporte les locales 8-bits et les locales UTF-8, peut être trouvée dans *libutf8-0.5.2.tar.gz*

Avantages :

- Dès maintenant un support pour Unicode UTF-8 portable, même sur les systèmes d'exploitation dont le support des caractères multi-octets ne marche pas, ou qui n'ont pas du tout de support pour les caractères multi-octets/larges.
- Le même binaire marche pour toutes les locales 8-bit et les locales UTF-8 supportées par le système.
- Quand un système d'exploitation fournit un vrai support pour les caractères multi-octets, vous pouvez en tirer avantage simplement en recompilant sans l'option du compilateur -DHAVE_LIBUTF8.

La méthode Plan9 Le système d'exploitation Plan9, une variante d'Unix, utilise UTF-8 comme encodage dans toutes ses applications. Son type de caractère large est appelé '`Rune`', pas '`wchar_`'. Des parties ce ses bibliothèques, écrites par Rob Pike et Howard Trikey, sont disponibles à *ftp://ftp.cdrom.com/pub/netlib/research/9libs/9libs-1.0.tar.gz* . Une autre bibliothèque similaire, écrite par Alistair G. Crook, est à *ftp://ftp.cdrom.com/pub/NetBSD/packages/distfiles/libutf-2.10.tar.gz* .
En particulier, chacune de ces bibliothèques contient un moteur d'expressions rationnelles qui comprend l'UTF-8.

Désavantages de cette API :

- UTF-8 est compilé dans la bibliothèque, pas optionnel. Les programmes compilés dans cet univers perdent le support des encodages 8-bits qui sont toujours utilisés fréquemment en Europe.

5.1.2 Pour les interfaces utilisateur graphiques

La bibliothèque QT-2.0 *http://www.troll.no/*

contient la classe QString qui est totalement Unicode. Vous pouvez utiliser les fonctions membres QString::utf8 et QString::fromUtf8 pour convertir depuis/vers un texte encodé en UTF-8. Les fonctions membres QString::ascii et QString::latin1 ne devraient plus être utilisées.

5.1.3 Pour la manipulation de texte avancée

Les bibliothèques mentionnées précédemment implémentent des versions des concepts ASCII qui comprennent Unicode. Voici des bibliothèques qui traitent des concepts Unicode, comme titlecase (Une troisième casse de lettres, différente des majuscules et des minuscules), la distinction entre la ponctuation et les symboles, la décomposition canonique, les classes combinables, le classement canonique et d'autres choses du même genre.

ucdata-1.9

La bibliothèque ucdata de Mark Leisher *http://crl.nmsu.edu/˜mleisher/ucdata.html*

s'occupe des propriétés des caractères, de la conversion de la casse, de la décomposition, des classes combinées.

ICU

Ce sont les classes IBM pour Unicode. *http://www.alphaworks.ibm.com/tech/icu/* . Une bibliothèque très détaillée comprenant des chaînes de caractères Unicode, des paquets de ressources, des formateurs de nombres, de date, d'heure et de messages, des assemblages, des assembleurs de messages et plus encore. Beaucoup de locales sont supportées. Cette bibliothèque est portable pour Unix et Win32, mais compilera sans intervention ("out of the box") seulement avec la libc6, pas la libc5.

libunicode

La librairie Unicode de GNOME *http://cvs.gnome.org/lxr/source/libunicode/*

de Tom Tromey entre autres. Elle couvre la conversion du jeu de caractères, les propriétés des caractères, la décomposition.

5.1.4 Pour la conversion

Deux bibliothèques de conversion qui supportent UTF-8 et un grand nombre de de jeux de caractères 8-bits, sont disponibles :

L'implémentation iconv d'Ulrich Drepper, contenue dans la GNU glibc-2.2.1 *ftp://ftp.gnu.org/pub/gnu/glibc/glibc-2.1.1.tar.gz* .

Avantages :

- iconv est conforme au standard POSIX, les programmes qui l'utilisent pour la conversion depuis/vers UTF-8 tourneront aussi sous Solaris. Cependant le nom des jeux de caractères diffère entre les plateformes. Par exemple, "EUC-JP" sous glibc devient "eucJP" sous HP-UX. (Le nom INIA officiel pour ce jeu de caractères est "EUC-JP". Il s'agit donc clairement d'une déficience de HP-UX.)
- Aucune bibliothèque supplémentaire n'est nécessaire.

librecode par François Pinard *ftp://ftp.gnu.org/pub/gnu/recode/recode-3.5.tar.gz* .

Avantages :

- Support pour la translittération, i.e conversion de caractères non-ASCII en séquences de caractères ASCII de façon à préserver la lisibilité pour les humains, même lorsqu'une transformation sans pertes est impossible.

Problèmes :

- Cette API est non standard.

5.1.5 Les autre approches

libutf-8

libutf-8, de G.Adam.Stanislav <*adam@whizkidtech.net*>

contient quelques fonctions pour la conversion depuis/vers des flux "FILE*". *http://www.whizkidtech.net/i18n/libutf-8-1.0.tar.gz*
Avantages :

- Très petit.

Problèmes :

- API non standard ;
- UTF-8 est compilé dans la bibliothèque, pas optionnel. Les programmes compilés dans cet univers perdent le support des encodages 8-bits qui sont toujours utilisés fréquemment en Europe ;
- L'installation n'est pas évidente : le Makefile doit être modifié. Pas d'auto-configuration.

5.2 Java

Java supporte Unicode en interne. Le type 'char' désigne un caractère Unicode, et la classe 'java.lang.String' désigne une chaîne de caractères construite à partir de caractères Unicode.

Java peut afficher n'importe quel caractère à travers son système de fenêtrage AWT, à condition que

1. vous positionniez la propriété système "user.language" de façon appropriée.
2. Les définitions de jeux de fontes /usr/lib/java/lib/font.properties.*language* soient appropriées, et
3. Le fontes spécifiées dans ce fichier soient installées.

Par exemple, pour afficher du texte contenant des caractères japonais, vous devriez installer des fontes japonaise, et lancer "java -Duser.language=ja ...". Vous pouvez combiner les jeux de fontes : pour pouvoir afficher des caractères d'Europe de l'ouest, grecs et japonais simultanément, vous devriez créer une combinaison des fichiers "font.properties" (couvre ISO-8859-1), "font.properties.el" (couvre ISO-8859-7) et "font.properties.ja" dans un seul fichier. ??Ceci n'a pas été testé??

Les interfaces java.io.DataInput et java.io.DataOutput contiennent des méthodes appelées 'readUTF', et 'writeUTF' respectivement. Mais notez qu'elles n'utilisent pas UTF-8 ; elles utilisent un encodage UTF-8 modifié : le caractère NUL est encodé dans une séquence de deux octets 0xC0 et 0x80 à la place de 0x00, et un octet 0x00 est ajouté à la fin. Encodées de cette façon, les chaînes peuvent contenir des caractères NUL mais elles doivent néanmoins être préfixées par un champ de taille. Les fonctions C <string.h> comme strlen() et strcpy() peuvent être utilisées pour les manipuler.

5.3 Lisp

Le standard Lisp ordinaire détermine deux types de caractères : 'base-char' et 'character'. C'est à l 'implémentation d'ajouter un support Unicode ou non. Ce langage détermine aussi un mot-clef argument ':external-format' pour 'open' comme place naturelle pour spécifier un jeu de caractères ou un encodage.

Parmi les implémentation gratuites du lisp standard, seul CLISP *http://clisp.cons.org/*

supporte Unicode. Vous aurez besoin d'une version de CLISP datant de juillet 99 ou plus récente. *ftp://clisp.cons.org/pub/lisp/clisp/source/clispsrc.tar.gz*

Les types "base-char" et "character" sont tous équivalents au 16-bits Unicode. L'encodage utilisé pour le fichier ou l'I/O socket/pipe peut être spécifié par l'argument ':external-format'. Les encodages utilisés pour les opérations d'entrée/sortie sur des ttys et l'encodage par défaut pour les I/O file/socket dépendent des locales.

Parmi les implémentations commerciales du Lisp standard, seule Eclipse *http://www.elwood.com/eclipse/eclipse.htm*

supporte Unicode. Voir *http://www.elwood.com/eclipse/char.htm*

Le type 'base-char' est équivalent à ISO-8859-1, et le type 'character' contient tous les caractères Unicode. L'encodage utilisé pour les entrées/sorties sur un fichier peut être défini à travers une combinaison des arguments de 'open' ':element-type' et :'external.format'.
Limitations : les fonctions d'attributs de caractères sont dépendantes des locales. Les sources et les fichiers de sources compilés ne peuvent pas contenir des chaînes Unicode littérales. L'implémentation commerciale du Lisp standard Allegro CL ne contient pas encore de support Unicode, mais Erik Naggum y travaille.

5.4 Ada95

Ada95 a été conçu pour supporter Unicode, et la bibliothèque standard Ada95 contient les types de données spéciaux ISO 10646-1 Wide_Character et Wide_String, ainsi que de nombreuses procédures et fonctions associées. Le compilateur Ada95 GNU (gnat-3.11 ou plus) supporte UTF-8 comme encodage externe des caractères 16 bits. Cela vous autorise à utiliser UTF-8 à la fois dans le code source et dans les entrées/sorties de l'application. Pour l'activer dans l'application, utilisez "WCEM=8" dans la chaîne FORM quand vous ouvrez un fichier, et utilisez l'option du compilateur "-gnatW8" si le code source est UTF-8. Pour plus de détails, voyez les manuels de référence GNAT et Ada95.