Interfaces utilisateurs pour Linux embarqué

(Embedded IHM)

Pierre Ficheux (pierre.ficheux@openwide.fr)

Décembre 2004

Résumé

Cet article présente des solutions d'interfaces homme machine (ou *IHM*) les plus fréquemment utilisées dans un environnement Linux embarqué. Après une brève introduction sur les interfaces graphiques dans les environnements embarqués, l'article traitera des principales couches graphiques bas-niveaux utilisées sur Linux (soit *X11* et le *frame-buffer*). La suite de l'article s'attachera à la présentation de quelques bibliothèques graphiques plus ou moins évoluées disponibles pour le développement d'applications sous Linux embarqué (Qt, GTK+, MicroWindows/Nano-X, LCDproc).

Les IHM et les produits industriels

Traditionnellement, l'IHM n'est pas le sujet de prédilection des développeurs systèmes et encore moins celui des développeurs de solutions industrielles. Ces derniers préfèrent utiliser leur énergie à peaufiner de superbes pilotes de périphériques, quitte à présenter au final un produit à l'interface plutôt ésotérique. Si l'on remonte à l'origine du système UNIX, ancêtre et inspirateur de notre Linux favori, il n'existait pas d'interface graphique mise à part celle offerte indirectement par la connexion d'un terminal externe au travers d'une connexion série (RS-232).

Si l'on considère les équipements industriels en général (et donc pas toujours équipés de Linux, ni d'un système d'exploitation d'ailleurs) ils furent longtemps dépourvus d'interface graphique intuitive. La raison était en partie technique, car l'utilisation de processeurs peu puissants (micro-contrôleurs 8 bits et souvent 4 bits) et d'une faible quantité de mémoire obligeaient les concepteurs à économiser l'octet et donc à fournir à l'utilisateur quelques tristes boutons poussoirs, quelques LED et parfois un afficheur LCD à basse résolution. Elle est aussi historique, il y a une tradition d'austérité dans l'interface des systèmes industriels et un appareil sérieux se doit forcément d'être incompréhensible à l'utilisateur :-)

L'avènement du réseau puis du « web » a quelque peu changé la donne pour certains équipements évolués (donc communicants). Avec l'apparition du « web » et des navigateurs internet, l'utilisation des réseaux de type IP dans certains équipements a permis leur contrôle par des protocoles standards tels que *SNMP* (pour *Simple Network Management Protocol*). Le pilotage du système est réalisé depuis un programme dit *Manager SNMP* dialoguant par IP et UDP avec les *agents SNMP* implantés dans les équipements. Le principe est simple et le manager est souvent graphique et relativement convivial bien que réservé aux initiés. Ce type de pilotage est répandu depuis longtemps dans les équipements réseaux (routeurs, switch, etc.) .

La généralisation du réseau et surtout de l'implantation de systèmes d'exploitations dans les équipements (souvent dérivés d'UNIX) permet aussi depuis longtemps le pilotage à distance ou la mise à jour grâce à des protocoles classiques comme SSH, SFTP, TELNET, FTP. La aussi nous sommes loin de la convivialité d'une interface graphique d'un client lourd type Windows ou MacOS. Bien entendu, le fait d'utiliser une interface graphique n'est pas forcément un gage

d'efficacité. En tant que vieil utilisateur de GNU-Emacs, j'utilise toujours les séquences de touches imaginées par notre ami Richard M. Stallman et jamais ma souris ne va s'aventurer aux abords des menus déroulants de mon éditeur favori. Cependant, le fait est qu'une interface graphique a l'avantage d'être utilisable par un plus grand nombre, et que de ce fait l'apprentissage en est réduit et l'interchangeabilité des opérateurs est bien meilleure.

La généralisation d'Internet, des intranets et autres « super-extranets » a conduit à l'utilisation du navigateur comme interface banalisé de (presque) tout processus informatique. Grâce au navigateur, d'aucun peu configurer son routeur avec la même logique que celle qu'il a utilisé quelques minutes avant pour réserver son billet TGV et ce quelle que soit la plate forme « client » utilisée.

Bien entendu, il n'est pas envisageable de TOUT piloter depuis un client distant tout d'abord parce que le réseau n'est pas toujours disponible sur tous les équipements mais aussi parce que le navigateur a ses limites, en dépit des extensions diverses et variées disponibles à ce jour (Java, Javascript, etc.). De plus certains équipements comme les terminaux en général doivent naturellement disposer d'une interface graphique locale, ce qui nous ramène directement au sujet de l'article.

L'interface graphique X11

L'interface graphique X Window System, appelée également X11 (pour X version 11) ou tout simplement X est le choix naturel des systèmes UNIX donc de Linux. La raison est également historique car ce fut le premier (et quasiment seul) standard d'interface graphique UNIX multi plate-formes, open source de surcroît et ce bien avant Linux puisque les premières versions publiques de X datent du début des années 80. Cependant, la conception de X comme un système graphique réparti fait qu'il est complexe, gourmand en ressources et donc pas forcément adapté à une environnement réduit comme Linux embarqué. Les applications tournent sur une machine en général assez puissante appelée *hôte* (ou *host*). L'affichage graphique s'effectue sur un terminal banalisé appelé *display,* les deux systèmes étant connectés sur un lien (en général un réseau de type IP) supportant le protocole X. La figure ci-dessous schématise l'architecture de X.

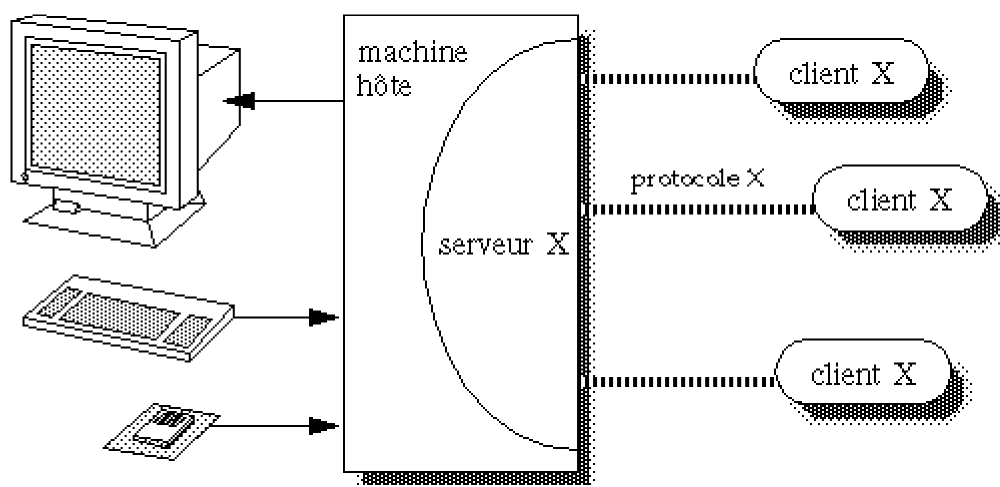

Figure 1. Architecture de X11

Cette architecture puissante mais complexe a le plus souvent peu d'intérêt pour un système embarqué car dans ce cas l'hôte et le display sont dans la même machine et X apporte plus de

lourdeur que d'avantages. Cependant, bon nombres de bibliothèques et composants graphiques commerciaux pour Linux sont uniquement disponibles pour X ce qui force souvent l'utilisation de ce système, malgré ses défauts.

Le paquetage XFree86

Dans le cas de Linux, l'environnement X correspond aux paquetages *XFree86* disponibles sur http://www.xfree86.org mais également fournis dans la plupart des distributions. En plus de problèmes de lourdeur d'exécution, l'empreinte mémoire des paquetages XFree86 est totalement incompatible avec la plupart des architectures réduites (environ 100 Mo). Traditionnellement la distribution Xfree86 est installée sur le répertoire `/usr/X11R6`.

```
$ cd /usr/X11R6/
$ du -s *
10176   bin
5772    include
5236    LessTif
84832   lib
10364   man
488     share
$ du -s .
116872  .
```

Cependant, comme nous pouvons le faire pour le système Linux (noyau, bibliothèques standards, programmes, etc.) il est toujours possible de réduire la distribution XFree86 afin de satisfaire à des contraintes d'encombrement plus raisonnables. Si nous détaillons un peu la structure de XFree86, il apparaît que le système est composé des éléments suivants:

- le serveur X (soit `bin/X`)
- les clients X (également situés sur le répertoire `bin`)
- les bibliothèques X (sur `lib`)
- les polices de caractères (sur `lib/X11/fonts`)
- divers autres fichiers de configuration dont le fichier principal de configuration du serveur (soit `/etc/X11/XF86Config`)

La réduction de XFree86

Le serveur X est quant à lui indispensable mais on peut réduire son empreinte mémoire en appliquant une méthode similaire à l'optimisation du noyau Linux, soit conserver uniquement les pilotes graphiques adaptés au matériel et les « extensions » indispensables à notre application (dans une installation Linux classique, plusieurs pilotes peuvent être installés par défaut). Si nous comparons une version complète à une version réduite, nous pouvons constater que le rapport est d'environ 20.

```
$ du -s /usr/X11R6.gen/lib/modules
1156    /usr/X11R6.gen/lib/modules
$ du -s /usr/X11R6.orig/lib/modules
21004   /usr/X11R6.orig/lib/modules
```

Une forte réduction de la taille du serveur X peut être obtenue en utilisant la carte graphique en mode VESA (pour V*ideo Electronics Standards Association,* voir http://www.vesa.org). Le

consortium VESA a publié un standard de BIOS graphique (*VBE* pour *VESA BIOS Extension*) permettant d'utiliser toutes les cartes graphiques compatibles VBE avec un jeu d'instructions commun. Le serveur XFree86 nécessite des cartes compatibles VBE 2.0 pour fonctionner, ce qui est le cas de la quasi-totalité des contrôleurs graphiques récents. Bien sûr l'utilisation du mode VESA implique une baisse sensible des performances par rapport au mode accéléré spécifique au contrôleur.

Le mode VESA est disponible en installant un minimum de modules pour XFree86.

```
# pwd
/usr/X11R6/lib/modules
# ls -l
total 756
drwxrwxr-x    2 root     root         4096 mai  6 12:40 drivers
drwxrwxr-x    2 root     root         4096 avr 19 13:08 extensions
drwxrwxr-x    2 root     root         4096 avr 19 12:59 fonts
drwxrwxr-x    2 root     root         4096 avr 19 11:17 input
-rwxr-xr-x    1 root     root        27286 avr 19 12:58 libddc.a
-rwxr-xr-x    1 root     root       147402 avr 19 12:58 libfb.a
-rwxr-xr-x    1 root     root       188930 avr 19 12:58 libint10.a
-rwxr-xr-x    1 root     root        62372 avr 19 12:58 libpcidata.a
-rwxr-xr-x    1 root     root        30942 avr 19 12:58 libramdac.a
-rwxr-xr-x    1 root     root       121454 avr 19 12:58 libscanpci.a
-rwxr-xr-x    1 root     root        27262 avr 19 12:58 libshadow.a
-rwxr-xr-x    1 root     root        18494 avr 19 12:58 libshadowfb.a
-rwxr-xr-x    1 root     root        10210 avr 19 12:58 libvbe.a
-rwxr-xr-x    1 root     root        19786 avr 19 12:58 libvgahw.a
drwxrwxr-x    2 root     root         4096 avr 19 11:46 linux
-r--r--r--    1 root     root        26253 jun  6  2001 v10002d.uc
-r--r--r--    1 root     root        35007 jun  6  2001 v20002d.uc
```

La partie driver elle-même étant réduite à la liste qui suit:

```
# ls -l drivers/
total 40
-rwxr-xr-x    1 root     root        21134 jun  6  2001 vesa_drv.o
-rwxr-xr-x    1 root     root        14809 jun  6  2001 vga_drv.o
```

De même, concernant le fichier `XF86Config`, on sélectionnera le pilote en configurant la section *Devices* comme suit:

```
Section "Device"
    Identifier  "MyVidCard"
    Driver      "vesa"
    VideoRam    2048
    # Insert Clocks lines here if appropriate
EndSection
```

Afin de compléter l'optimisation, il sera également nécessaire de réduire au strict minimum, voire au néant, la liste des clients X installés sur la cible. Une autre optimisation importante est la réduction du nombre de polices de caractères (ou *fonts*) car le système embarqué n'aura pas forcément besoin d'utiliser 20 tailles de polices ni d'être compatible avec les idéogrammes chinois, japonais ou coréens pour lesquels les polices de caractères pèsent plusieurs centaines de Ko. Bien sûr les modifications comme la suppression d'un répertoire de polices devront être répercutées dans le fichier de configuration du serveur de polices (soit `/etc/X11/fs/config`) ou directement dans le fichier de configuration du serveur si `xfs` n'est pas utilisé. Dans le cas d'un système réduit, ce

dernier cas sera probablement le plus fréquent et nous pourrons intervenir au niveau du fichier `XF86Config`. Dans l'exemple qui suit, nous avons limité la liste des polices à la résolution 75dpi.

```
FontPath   "/usr/X11R6/lib/X11/fonts/misc/"
FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/:unscaled"
FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/"
```

Au niveau d'un répertoire de polices, chaque fichier `.pcf.gz` correspond à une police de caractère utilisable. Dans chaque répertoire on trouve également le fichier `fonts.dir` indiquant la liste des polices du répertoire et le fichier `fonts.alias` qui définit des alias plus simples aux noms des polices X, exemple:

```
fixed        -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
```

Il est important de noter que l'alias *fixed* correspond au nom de la police par défaut utilisée par le serveur X, si il ne doit rester qu'une police celle-ci devra être définie comme police *fixed*. De même, si l'on supprime des polices, donc des fichiers `.pcf.gz` dans un répertoire de polices, il est nécessaire de reconstruire le fichier `fonts.dir` avec l'utilitaire `mkfontdir`.

```
# mkfontdir .
```

Pour terminer l'optimisation de XFree86 il conviendra de sélectionner soigneusement les bibliothèques X utilisées (coté client, donc coté application) en utilisant une méthode basée sur l'utilisation des scripts `mklibs.sh` ou `mklibs.py` déjà cités dans plusieurs publications de l'auteur.

Au terme de cette optimisation, on peut espérer réduire la taille de la distribution X en dessous de 10 Mo, ce qui n'est déjà pas si mal.

Le serveur XkDrive

Indépendamment de l'empreinte mémoire sur le disque, les versions complètes de X sont naturellement consommatrices de mémoire vive. Les sources de XFree86 incluent cependant un serveur très optimisé tant au niveau de l'empreinte mémoire que de la RAM utilisée à l'exécution. Ce serveur *Xkdrive* implique quelques limitations, comme l'absence de support des polices vectorielle et le nombre limité de contrôleurs graphiques supportés. Il fonctionne cependant très bien en mode VESA ou bien en utilisant le *frame-buffer* Linux qui sera décrit plus loin dans l'article. Le résultat se présente comme un exécutable autonome qui sera parfois mieux adapté à une architecture réduite.

Le fichier exécutable serveur *Xkdrive* n'est pas disponible directement dans les distributions Linux et il sera nécessaire de le compiler à partir des sources de Xfree86. Pour ce faire, il faudra récupérer les sources par le site http://www.xfree86.org puis créer un fichier `host.def` dans le répertoire `xc/config/cf`. Ce fichier aura l'allure suivante:

```
#define BuildServersOnly YES
#define KDriveXServer YES
#define TinyXServer YES
#define XfbdevServer NO
#define XvesaServer YES
#define BuildType1 YES
```

On lance ensuite la compilation en exécutant la commande:

```
make World
```

Lorsque la compilation est terminée avec succès, on doit obtenir le message suivant:

```
Full build of Release 6.6 of the X Window System complete.
```

Le fichier `Xvesa` doit également être présent.

```
$ ls -l programs/Xserver/Xvesa
-rwxrwxr-x    1 pierre   pierre     838540 mai  3 14:31 programs/Xserver/Xvesa
```

On note la taille relativement modeste de cet exécutable qui occupe « seulement » 800 Ko par rapport aux 2 Mo au du serveur standard (sans compter les modules associés). On peut tester le serveur en tapant la commande:

```
# Xvesa -screen 1024x768x16 &
```

Dans ce cas nous utiliserons une résolution de 1024 sur 768 pixels en mode 16 bits (soit 65536 couleurs).

Le frame-buffer de Linux

La frame-buffer permet de piloter les modes graphiques en mode haute résolution directement depuis le noyau Linux. Notons que ce n'est pas le cas de la majorité des serveurs X qui pilotent les cartes graphiques en mode utilisateur! Une des applications principales du frame-buffer est la possibilité d'afficher un logo au démarrage du noyau (souvent un pingouin) mais nous allons voir que cette utilisation est quelque peu réductrice.

Configuration du noyau pour le support frame-buffer

Le support du frame-buffer doit être activé à travers la procédure de configuration du noyau Linux. Pour le noyau 2.4, la configuration s'effectue dans dans le menu *Console drivers/Frame-buffer* comme décrit dans la figure ci-dessous. On notera que le support VESA est bien entendu disponible ainsi que celui de nombreux contrôleurs graphiques. Dans le cas du support VESA, le choix du mode frame-buffer s'effectue obligatoirement au démarrage du système dans un programme de démarrage type LILO ou GRUB, ce mode graphique ne pouvant être modifié ultérieurement sans un redémarrage du système. Dans le cas d'un module dédié à un contrôleur graphique, le passage en mode frame-buffer prendra effet lors du chargement du module associé via la commande `modprobe` et l'on pourra modifier le mode graphique dynamiquement grâce à l'utilitaire `fbset` qui permet de manipuler le point d'entrée du frame-buffer soit par défaut `/dev/fb0`.

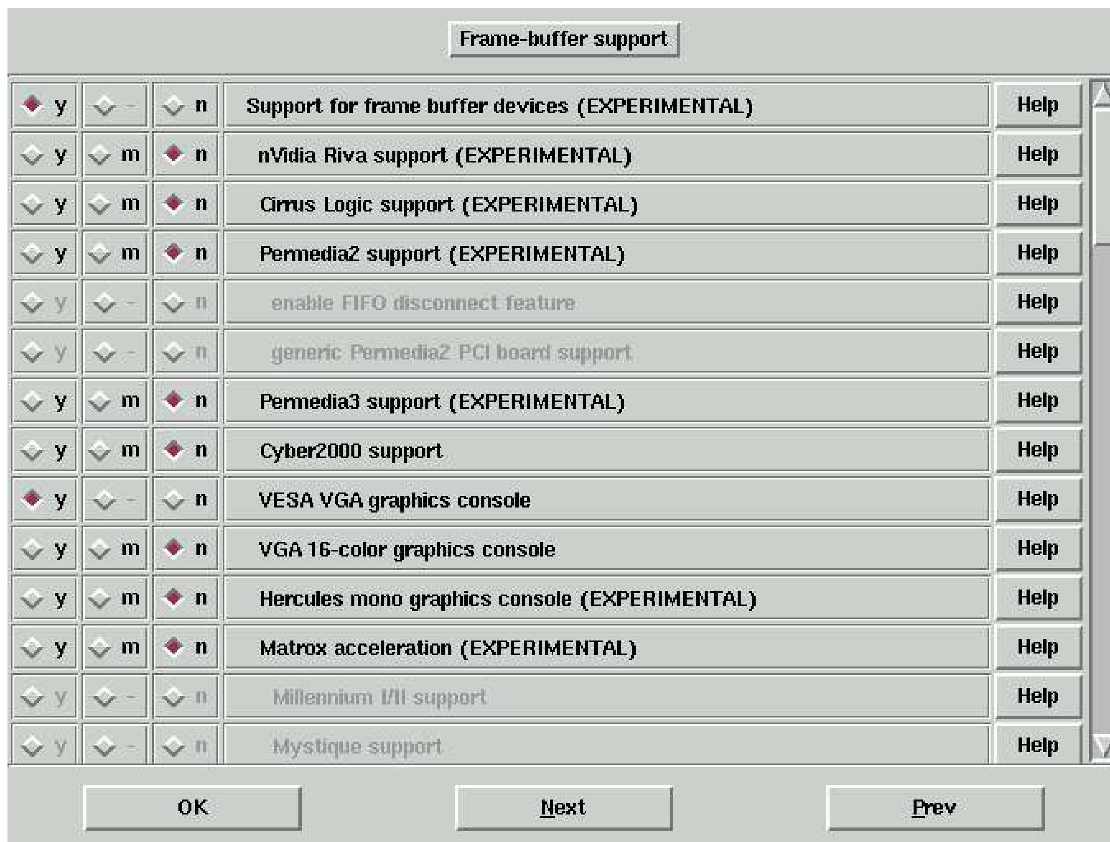

Figure 2. Configuration du frame-buffer pour le noyau 2.4

Pour le noyau 2.6, le configuration s'effectue dans le menu *Device Drivers/Graphics Support* comme décrit dans la figure ci-dessous.

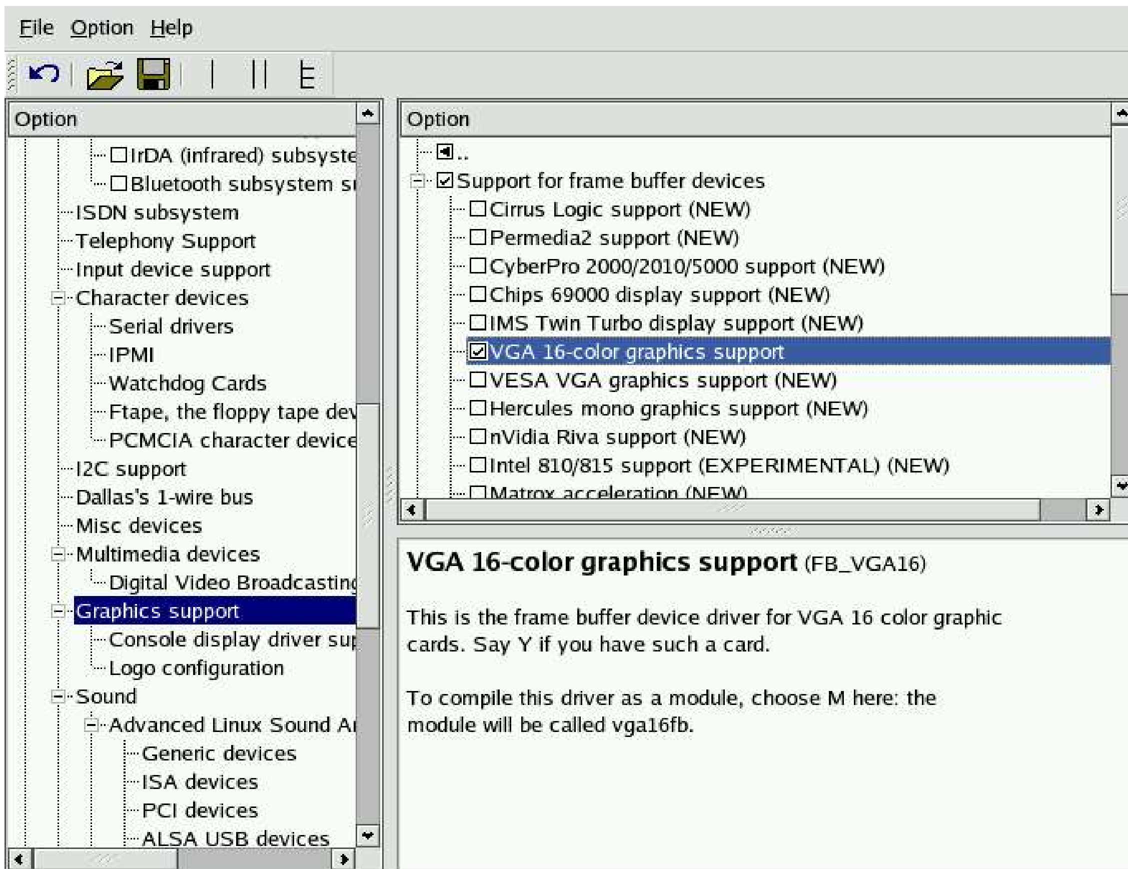

Figure 3. Configuration du frame-buffer pour le noyau 2.6

Lorsque la configuration est sauvée, le nouveau noyau est compilé par la procédure habituelle soit en version 2.4 :

```
# make dep clean bzImage modules
```

et en version 2.6 :

```
# make clean bzImage modules
```

Au niveau du programme de démarrage, il faut ensuite indiquer la sélection d'un mode graphique ou bien donner la possibilité à l'utilisateur de sélectionner ce mode en saisissant le code associé lors du démarrage. Dans ce cas on placera simplement une ligne définissant le mode graphique dans le fichier `/etc/lilo.conf`.

```
image=/boot/bzImage-2.4.27
        label=linux
        vga = ASK
        ...
```

Dans le cas de GRUB, la même option sera ajoutée dans le fichier `/etc/grub.conf` au niveau des paramètres du noyau.

```
kernel /bzImage-2.4.27 vga=ASK ro root=LABEL=/
```

Si l'on veut sélectionner un mode graphique particulier (exemple: 1024x768 en mode 64K couleurs), il suffit de saisir le code correspondant au démarrage du système. Si le mode est utilisé en permanence, on le donnera comme paramètre à la directive `vga` en fonction du tableau décrit dans la documentation du noyau dans le fichier *Documentation/fb/vesafb.txt*.

	640x480	800x600	1024x768	1280x1024
256	0x301	0x303	0x305	0x307
32k	0x310	0x313	0x316	0x319
64k	0x311	0x314	0x317	0x31A
16M	0x312	0x315	0x318	0x31B

Tableau 1. Valeurs des codes VESA

En fonction du tableau ci-dessus, le mode 1024x768x64k s'obtiendra en tapant le code 317 au niveau du démarrage.

Quelques utilitaires pour le frame-buffer

Une fois le système démarré, on peut vérifier le mode graphique en utilisant la commande `fbset`.

```
# fbset

mode "1024x768-76"
    # D: 78.653 MHz, H: 59.949 kHz, V: 75.694 Hz
    geometry 1024 768 1024 768 16
    timings 12714 128 32 16 4 128 4
    rgba 5/11,6/5,5/0,0/0
endmode
```

Par contre une tentative de modification se soldera comme prévu par un échec car nous sommes en mode VESA.

```
# fbset -xres 640 -yres 480 -depth 16
Vesafb does not support changing the video mode
ioctl FBIOPUT_VSCREENINFO: Invalid argument
```

On peut facilement effectuer des copies d'écran du frame-buffer vers un fichier en utilisant les programmes `fbdump` ou `fbgrab`. La commande `fbdump` génère du format PPM (pour *Portable PixMap*) alors que `fbgrab` génère directement du format PNG.

```
# fbdump > /tmp/mon_fichier.ppm
# fbgrab /tmp/mon_fichier.png
```

Le programme `fbi` permet d'afficher dans le frame-buffer des images aux formats JPEG, PPM, GIF, TIFF, XWD, BMP, PNG et Photo-CD.

Les bibliothèques graphiques

Il existe pas mal de bibliothèques graphiques (appelées également *toolkits*) dans l'environnement Linux. Un bon nombre d'entre-elles sont avant tout dédiées à l'interface X11. Cependant, certaines bibliothèques comme Qt ou GTK+ sont également disponibles pour le frame-buffer Linux. Ces bibliothèques sont relativement complexes et fournissent un grand nombre de composants graphiques évolués (menus, boutons, etc.). De ce fait elles peuvent être sur-dimensionnées pour des environnements réduits ne nécessitant pas une grande complexité graphique ou disposant de moyens d'affichage réduits. C'est la raison pour laquelle nous présenterons aussi des bibliothèques beaucoup plus simples (*MicroWindows/Nano-X* et *LCDproc*) mais également beaucoup moins gourmandes en ressources.

La bibliothèque Qt-Embedded

La bilbliothèque Qt est développée depuis plusieurs années par la société norvégienne *Trolltech* (http://www.trolltech.com). Elle est écrite en C++ et a l'avantage d'être multi plate-forme. Le code source pouvant être compilé pour Linux, Windows ou MacOS. Cette bibliothèque a été utilisée entre-autres pour le navigateur OPERA et le bureau KDE. Elle est disponible sous double licence GPL et propriétaire. L'utilisation de la version GPL implique bien sûr que le projet associé soit diffusé sous GPL. En cas de diffusion sous licence propriétaire, il sera nécessaire de payer une licence pour Qt. Cependant, ce principe de double licence permet d'évaluer totalement le produit puisqu'il n'y a pas de perte de fonctionnalités entre la version GPL et la version propriétaire.

La bibliothèque Qt a une excellente réputation de fiabilité mais elle est relativement consommatrice en ressources (elle est écrite en C++). La version GPL de Qt-Embedded est disponible en télé-chargement à l'adresse http://www.trolltech.com/download/qt/embedded.html. Le plus simple est de construire le programme de démonstration qui donne un bon aperçu des fonctionnalités de Qt.

La procédure de compilation de la bibliothèque est assez simple:

Il faut tout d'abord extraire l'archive télé-chargée depuis le site de Trolltech, soit:

```
$ tar xjvf qt-embedded-free-3.3.3.tar.bz2
```

Il faut ensuite se placer dans le répertoire ainsi créé et positionner les variables d'environnements nécessaires:

```
$ export QTDIR=`pwd`
$ export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
```

Il faut ensuite utiliser le script configure pour générer l'environnement de compilation de la bibliothèque soit:

```
$ ./configure -depths 16,24,32
```

La dernière étape est de compiler la bibliothèque, ce qui prend pas mal de temps!

```
$ make
```

Lorsque la compilation s'est terminée sans erreur, on peut alors exécuter le programme de démonstration:

```
# cd examples/launcher
# ./start_demo
```

La fenêtre affichée a l'allure suivante. Nous pouvons voir que Qt fournit pas mal de composants graphiques et que ceux-ci sont totalement fonctionnels dans l'environnement frame-buffer.

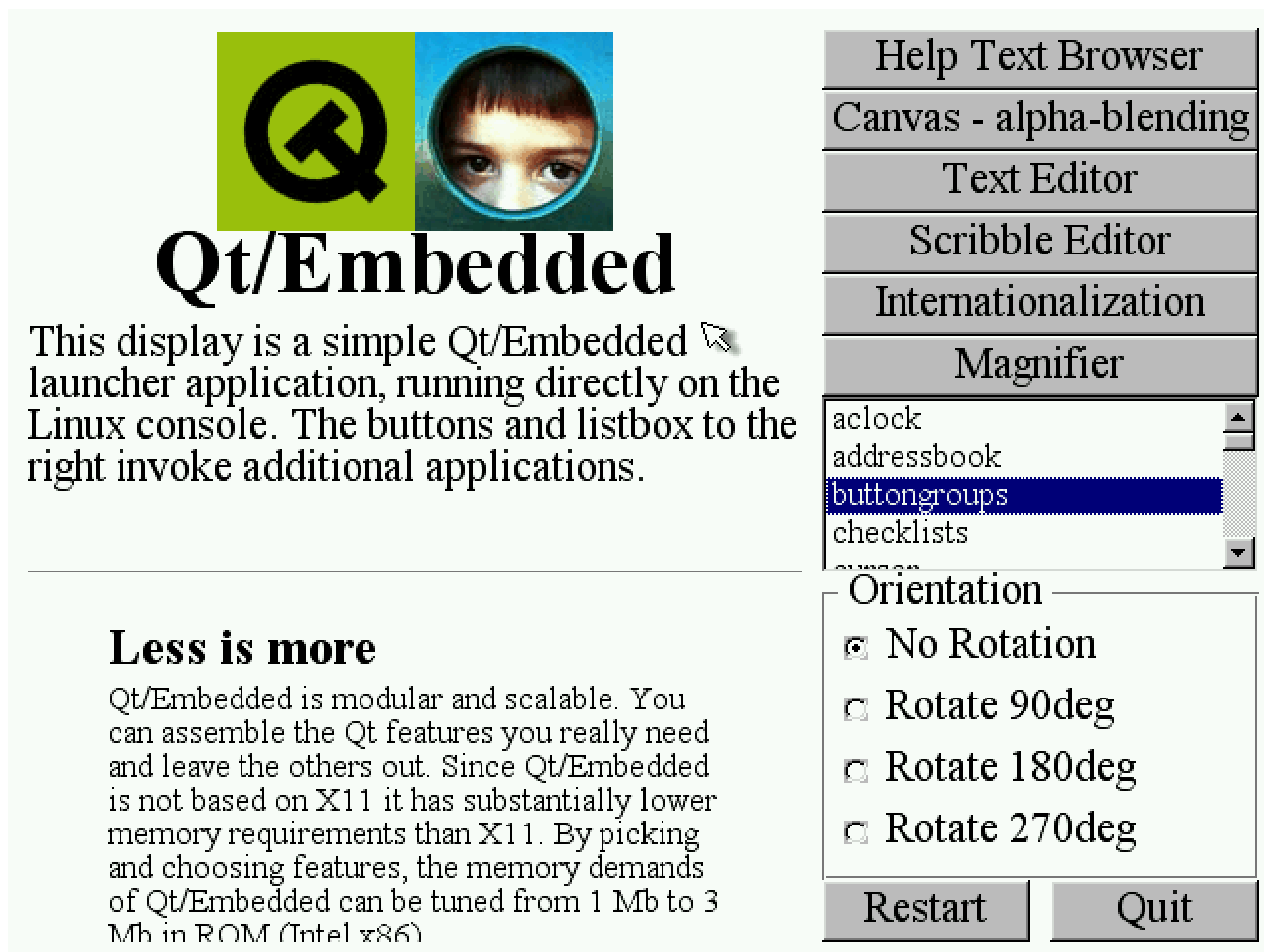

Figure 4. Démonstration de Qt-Embedded

La bibliothèque GTK+/Embedded

Cette bibliothèque a pour origine l'outil d'édition graphique GIMP. A l'époque elle fut créée comme alternative libre (et plus performante) à la bibliothèque OSF-Motif qui était alors propriétaire. Depuis elle a servie de base à l'environnement GNOME. Cette bibliothèque est écrite en C et elle est diffusée sous licence LGPL, ce qui permet de l'utiliser pour ses propres développements sans obligatoirement placer son propre code sous GPL. Au niveau des fonctionnalités, elle est relativement proche de Qt et sera donc réservée aux applications graphiques relativement complexes.

La compilation de GTK+ est plus complexe que celle de Qt car elle nécessite la présence de plusieurs bibliothèques annexes pour fonctionner (soit *Glib*, *Pango* et *ATK*). Ces bibliothèques sont disponibles en télé-chargement sur le site de GTK+ (http://www.gtk.org). Pour ce test, nous avons utilisé les versions suivantes des bibliothèques:

- atk-1.8.0

- glib-2.4.8
- gtk+-2.4.14
- pango-1.4.1

Il faut tout d'abord compiler la bibliothèque *glib* en utilisant la procédure classique, soit:

```
$ ./configure –prefix=/usr
$ make
# make install
```

On peut ensuite faire de même pour les bibliothèques *Pango* et *ATK*.

Pour la bibliothèque GTK+, il faut indiquer que l'on utiliser le frame-buffer et non la sortie X11 qui est la configuration par défaut. Avant cela, il faut appliquer un petit *patch* à la bibliothèque car il semble y avoir un problème de fonction non définie si l'on utilise la sortie frame-buffer. La référence du *patch* à télé-charger est disponible dans la bibliographie de l'article. Il est inspiré d'un patch diffusé pour GTK+-2.4.0 par Frederic Crozat (fcrozat@mandrakesoft.com) mais il semblerait que celui-ci n'ait pas été pris en compte par les développeurs de GTK+. La procédure de compilation est donc la suivante, tout d'abord l'application du *patch* à partir du répertoire des sources.

```
$ patch -p0 < /tmp/gtk_2.4.4_linux-fb.patch
patching file gdk/linux-fb/gdkdrawable-fb2.c
patching file gdk/linux-fb/gdkfont-fb.c
patching file gdk/linux-fb/gdkwindow-fb.c
```

Ensuite viennent la génération de l'environnement via `configure` en précisant la sortie frame-buffer, puis la compilation et enfin l'installation:

```
$ ./configure --prefix=/usr --with-gdktarget=linux-fb
$ make
# make install
```

Lorsque tout est installé on peut utiliser le programme de démonstration présent dans le répertoire `demos/gtk-demo` des sources de GTK+. Ce programme doit bien entendu être appelé depuis la console frame-buffer Linux en ayant pris soin de stopper le service `gpm` (qui gère la souris en mode texte) si ce dernier était actif.

```
# cd demos/gtk-demo
# ./gtk-demo
```

L'exécution du programme de test provoque l'affichage de la fenêtre suivante:

Figure 5. Démonstration de GTK+/Embedded

On voit donc que GTK+ est assez riche au niveau fonctionnalités mais le support frame-buffer semble moins bien maintenu.

La bibliothèque MicroWindows/Nano-X

Contrairement aux autres bibliothèques, *MicroWindows* et *Nano-X* ont été développées spécifiquement pour des applications réduites. Elle sont donc moins riches en fonctionnalités mais aussi beaucoup moins gourmandes en ressources. Pour donner une idée, un application de démonstration *MicroWindows* occupe environ 300 Ko en version statique ce qui est presque 10 fois moins qu'un programme Qt ou GTK+ dans les même conditions. *MicroWindows* est conçu pour être proche de l'API de programmation graphique Win32 utilisée sous *MS Windows*. Au contraire, *Nano-X* utilise un principe de client/serveur proche de X11.

On peut facilement tester ces bibliothèques dans un environnement classique type x86 sous X11 ou frame-buffer mais elles sont prévues pour la compilation croisée vers des processeurs plus dédiés aux environnements embarqués comme l'ARM.

Pour compiler la bibliothèque, il faut tout d'abord extraire l'archive télé-chargée depuis le site http://www.microwindows.org. A la date de l'écriture de ce document, la dernière version est la

0.90.

```
$ tar xzvf microwindows-0.90.tar.gz
```

Il faut ensuite se placer dans le répertoire des sources soit:

```
$ cd microwindows-0.90/src
```

MicroWindows n'utilise pas le système *autoconf* et la configuration en fonction de la cible doit s'effectuer à la main. Des exemples de fichiers de configuration sont disponibles dans le répertoire `Configs`. Dans notre cas nous pouvons effectuer un test dans l'environnement du frame-buffer Linux en utilisant le fichier de configuration `Configs/config.fb` soit:

```
$ cp Configs/config.fb config
```

Il reste à compiler la bibliothèque et les exemples en utilisant la commande:

```
$ make
```

Le temps de compilation est beaucoup plus court que pour Qt ou GTK+ ce qui indique bien que cette bibliothèque est beaucoup plus légère. Pour exécuter le programme de test de la bibliothèque, il faut tout d'abord exécuter le script `mouse.sh` qui sous Linux lance simplement le démon `gpm`:

```
# kill old mouse driver
gpm -k
# start mouse driver in repeater mode
gpm -R -t ps2
```

Il suffit ensuite d'exécuter le script `demo.sh` qui correspond au lancement du programme `mdemo`:

```
$ ./demo.sh
```

Si tout se passe bien, on doit obtenir la fenêtre suivante:

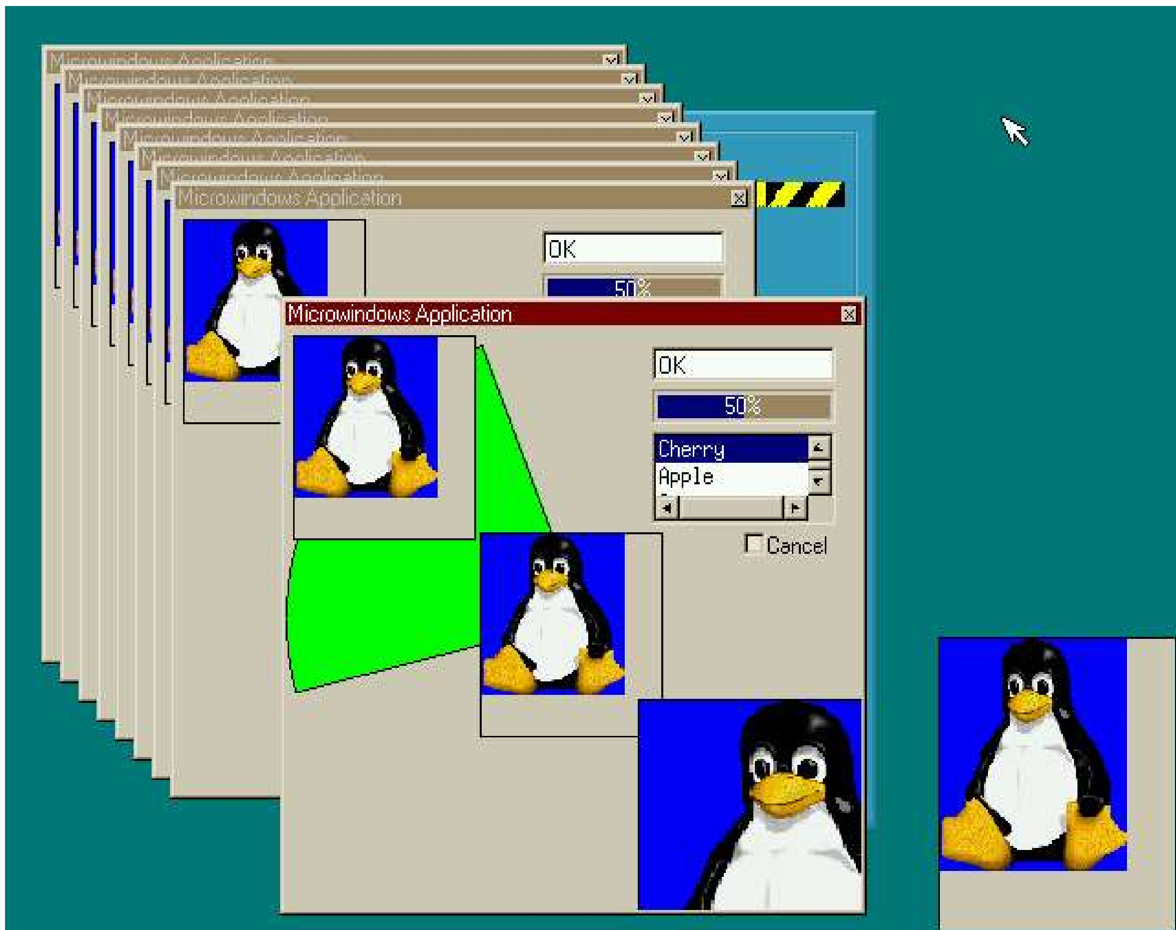

Figure 6. Démonstration de MicroWindows

La démonstration de Nano-X s'effectue par le script demo2.sh, qui correspond au lancement de micro-serveur puis des clients:

```
$ cat demo2.sh

# Nano-X applications, press <BREAK> key to exit
# bin/nano-X & bin/nanowm & bin/nxview bin/mwlogo.ppm & bin/nxclock & bin/nxmag
& sleep 10000
bin/nano-X & bin/nanowm & bin/nxterm & bin/nxeyes & bin/tux & sleep 10000
b
```

La fenêtre affichée a l'allure suivante:

Figure 7. Démonstration de Nano-X

En cas de compilation vers une autre architecture, il suffit de spécifier le nom parmi la liste citée au début du fichier `config` et bien sûr disposer de la chaîne de compilation croisée adéquate (exemple: `arm-linux-gcc`).

```
# build target platform
#
# Valid ARCH values are:
#
# LINUX-NATIVE
# LINUX-TCC
# LINUX-ARM
# LINUX-MIPS
# LINUX-POWERPC (BIGENDIAN=Y)
# LINUX-SPARC (BIGENDIAN=Y)
# LINUX-SH
# FREEBSD-X86
# SOLARIS (BIGENDIAN=Y)
# TRIMEDIA
# RTEMS
# DJGPP
# ELKS
#
# note: ELKS can't build client/server nano-X, nor widget lib
```

```
#
##################################################################
ARCH                        = LINUX-NATIVE
BIGENDIAN                   = N
ARMTOOLSPREFIX              = arm-linux-
MIPSTOOLSPREFIX             = mipsel-linux-
POWERPCTOOLSPREFIX          = powerpc-linux-
SHTOOLSPREFIX               = sh-linux-gnu
RTEMSTOOLSPREFIX            = i386-rtemself-
```

Pour information, les bibliothèque ont été testées avec succès sur un carte d'évaluation du processeur ATMEL AT91RM9200. Cette carte (AT91RM9200-DK) est équipée d'une sortie VGA pilotée par un contrôleur EPSON S1D13806 disposant d'un pilote frame-buffer.

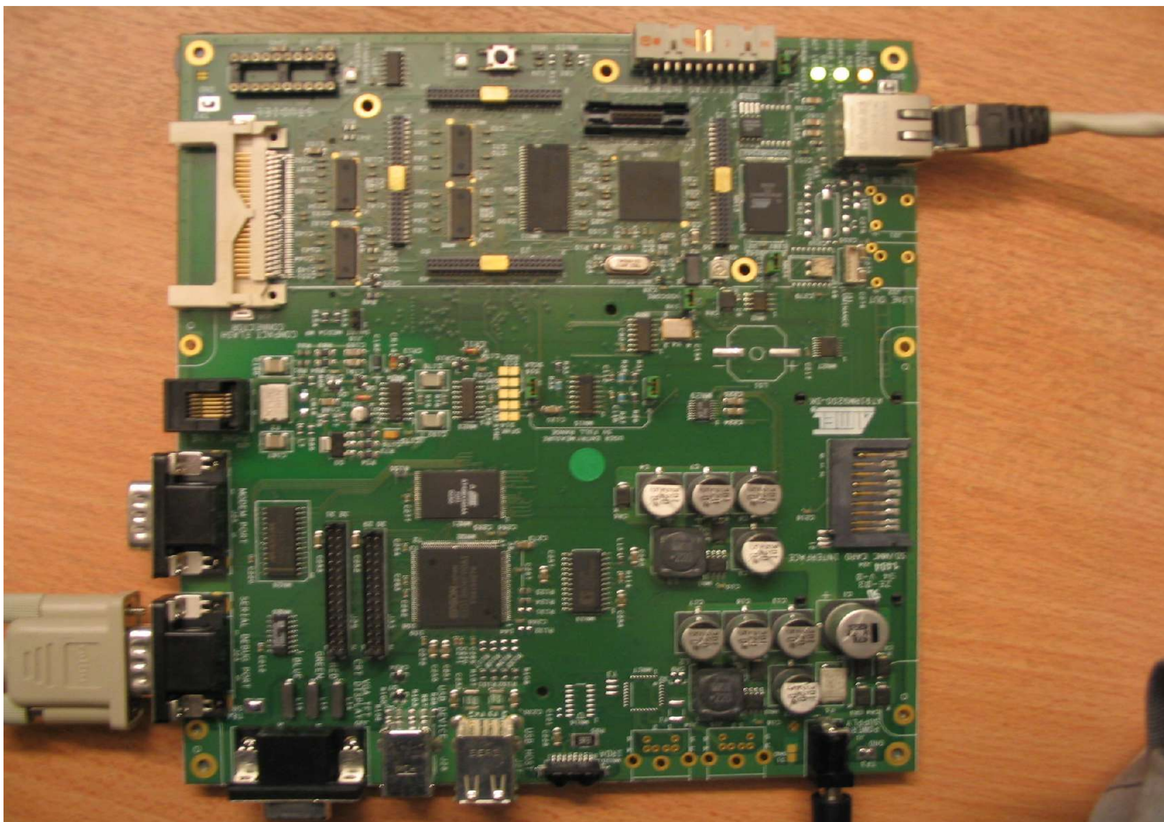

Figure 8. Carte AT91RM9200-DK

Une bibliothèque d'affichage LCD: LCDproc

Même si les systèmes embarqués deviennent de plus en plus puissants, les applications industrielles n'ont pas toujours besoin d'un affichage complexe. Bon nombre de systèmes se contentent d'un afficheur à cristaux liquides type LCD. Pour ce faire, le projet *LCDproc* (http://lcdproc.omnipotent.net) fournit une bibliothèque puissante, simple, et utilisable sur plusieurs afficheurs du commerce. En plus de cela, la bibliothèque fournit un pilote de test utilisant l'interface *curses* ce qui permet de mettre au point l'application sans disposer d'afficheur réel.

Le principe de *LCDproc* est basé sur une technique de client/serveur. Le serveur est chargé

d'effectuer l'affichage sur l'écran LCD et reçoit pour cela les requêtes des clients à travers un *socket* TCP sur un le port 13666. Par défaut le serveur et les clients sont exécutés sur une même machine (soit *localhost*) mais rien n'empêcherait d'effectuer l'affichage à travers Internet.

La compilation de *LCDproc* ne présente pas de difficultés particulières. Après extraction de l'archive, le mieux est de compiler le paquetage en incluant tous les pilotes d'écran disponibles.

```
$ ./configure --enable-drivers=all
$ make
```

Pour tester le système, il faut tout d'abord sélectionner un pilote dans le fichier de configuration du serveur soit `LCDd.conf`. Dans notre cas nous utiliserons le pilote de test *curses*.

```
[server]
# Server section with all kinds of settings for the LCDd server

#Driver=none
Driver=curses
#Driver=HD44780
```

Il faut ensuite exécuter le programme serveur dans un terminal comme suit. L'option `-s` indique que l'affichage des messages d'erreurs utilisera le démon *syslog*.

```
$ ./server/LCDd -c LCDd.conf -s
```

Le fenêtre prend alors l'allure suivante.

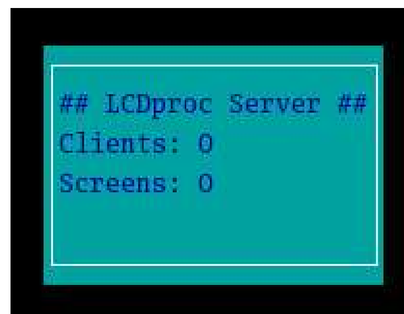

Figure 9. Démarrage du serveur LCDproc

On peut alors utiliser le client `lcdproc` fourni pour effectuer un test d'affichage. Ce client affiche l'état du système en temps réel.

```
$ ./clients/lcdproc/lcdproc
```

La fenêtre d'affichage prend alors l'allure suivante.

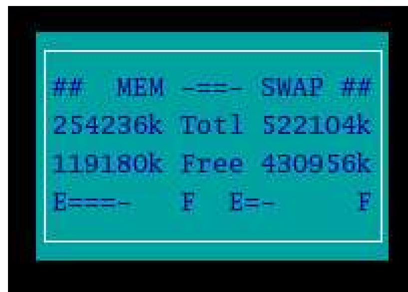

Figure 10. Serveur LCDproc et client lcdproc

Il est bien entendu possible de piloter directement le serveur depuis un simple client `telnet` puisque nous utilisons le protocole TCP. La description des différents objets disponibles est présente dans le fichier `docs/netstuff.txt` de la distribution *LCDproc*.

Dans notre exemple nous effectuons une session *telnet* dans laquelle:

1. Nous initialisons le protocole *LCDproc*.
2. Nous créons un écran d'affichage.
3. Nous créons trois objets *title*, *string* et *scroller* auxquels nous affectons des valeurs.

Le texte en gras représente les commandes tapées par l'utilisateur, le reste du texte représente les réponses du serveur.

```
$ telnet localhost 13666
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
connect LCDproc 0.4.5 protocol 0.3 lcd wid 20 hgt 4 cellwid 5 cellhgt 8
screen_add s
success
listen s
widget_add s t title
success
widget_set s t "Test LCDproc"
success
widget_add s str string
success
widget_set s str 1 4 Texte
success
widget_add s scroll scroller
success
widget_set s scroll 2 3 18 3 h 1 "Une ligne defilante"
success
```

La fenêtre du serveur *LCDproc* présente alors l'allure suivante.

Figure 11. Serveur LCDproc et client telnet

Une bibliothèque de gestion d'écran LCD: omm_api

L'origine de cette bibliothèque est un projet de lecteur/enregistreur MP3 réalisé avec Eric BENARD il y a quelques années. Ce projet est décrit en détails dans l'ouvrage Linux embarqué cité en bibliographie. Le but de la bibliothèque *omm_api* est de gérer un écran LCD non seulement au niveau de l'affichage graphique mais aussi des « événements » reçus, ces événements correspondant à des codes ASCII. De ce fait, l'application finale est constituée d'un automate pour lequel les différents événements modifient l'état.

Le pilotage de l'application se fait au travers de tubes nommés ou bien pour le test via l'entrée standard (stdin). Pour les besoins de cet article nous avons rapidement réalisé une adaptation de la bibliothèque afin de s'appuyer également sur une version très légèrement modifiée de *LCDproc*, ce qui n'était pas le cas auparavant, et cela permet de disposer de tous les afficheurs LCD gérés par *LCDproc*. La distribution télé-chargeable sur le site de l'auteur (voir bibliographie) fournit deux exemples:

1. Un programme de démonstration très simple: `apidemo`
2. Un sélecteur de fichier: `fileselector`

Ces programmes sont disponibles pour les environnements matériels suivants:

- X11
- Curses
- Frame-buffer en utilisant la *SVGALIB* (http://www.svgalib.org), qui est une bibliothèque bas-niveau d'accès au frame-buffer.
- LCDproc

Pour compiler la bibliothèque et les exemples, il suffit d'extraire l'archive puis d'exécuter les commandes suivantes:

```
$ make
$ cd fileselector
$ make
```

Pour tester la démonstration il suffit de se placer dans le répertoire des source d'*omm_api* et de taper pour la version X11:

```
$ ./apidemox
```

Les autres versions sont respectivement `apidemoc` (pour *curses*) `apidemof` (pour le frame-buffer) `apidemol` (pour *LCDproc*). La démonstration affiche un titre centré, une chaîne de caractères qui défile horizontalement ainsi que le caractère envoyé par l'utilisateur soit depuis le clavier soit au travers de la FIFO de communication `/tmp/myfifo`. Le caractère `'q'` provoque la sortie de l'application. L'exécution du programme affiche la fenêtre suivante.

Figure 12. Démonstration omm_api

Le sélecteur de fichier *fileselector* suit la même logique et existe donc en version X11, curses, frame-buffer et *LCDproc*. Ce sélecteur de fichiers indépendant est piloté à partir de codes ASCII comme décrit dans le code source `fileselector.c`.

```
/* States handling definition */
static omm_event_handler fileselector_event_handlers[] = {
  'd', OMM_STATE_INIT, lcd_dump_screen,        /* Dump screen to log file */
  'b', OMM_STATE_INIT, lcd_bg_screen,
  'r', OMM_STATE_INIT, lcd_fg_screen,
  12, OMM_STATE_INIT, filesel_redraw,          /* ^L redraw */
  'A'-0x40, OMM_STATE_INIT, filesel_home,      /* ^A home */
  'E'-0x40, OMM_STATE_INIT, filesel_end,       /* ^E end */
  'U'-0x40, OMM_STATE_INIT, filesel_pgup,      /* ^U PgUp */
  'V'-0x40, OMM_STATE_INIT, filesel_pgdown,    /* ^V PgDn */
  'j', OMM_STATE_INIT, filesel_down,           /* j Down */
  'k', OMM_STATE_INIT, filesel_up,             /* k Up */
  'h', OMM_STATE_INIT, filesel_right,          /* r Right */
  'l', OMM_STATE_INIT, filesel_left,           /* l Left */
  'a', OMM_STATE_INIT, filesel_parent_dir,     /* a Parent directory  */
  'n', OMM_STATE_INIT, filesel_enter_dir,      /* n Child directory */
  '\n', OMM_STATE_INIT, filesel_select,        /* LF Select file(s) */
  '\r', OMM_STATE_INIT, filesel_select,        /* CR Select file(s) */
  'D', OMM_STATE_INIT, filesel_delete,         /* Remove a file */
  't', OMM_STATE_INIT, filesel_mark_file,      /* Mark a file */
  'u', OMM_STATE_INIT, filesel_mark_file,      /* Unmark a file */
  ' ', OMM_STATE_INIT, filesel_toggle_file,    /* toggle-mark file */
  'T', OMM_STATE_INIT, filesel_mark_all_files, /* Mark all files */
  'U', OMM_STATE_INIT, filesel_unmark_all_files,/* Unmark all files */
  27, OMM_STATE_INIT, filesel_quit,
  0, 0, NULL
};
```

Le sélecteur permet de naviguer dans une arborescence de fichiers d'un type donné par leur suffixe et d'effectuer les opérations classiques de parcours des répertoires, sélection simple ou multiple, effacement, etc. Lorsqu'une liste de fichiers est sélectionnée, l'action sur la touche *Entrée* provoque l'affichage de la liste des fichiers sélectionnés sur `stderr`. L'arborescence test fournie dans l'archive

permet de valider le sélecteur. La session suivante indique la recherche de fichiers `'.c'` (option `-S`) dans le répertoire courant (option `-d`).

```
$ cd fileselector/test
$ ../fileselectorx -s -d . -S c
```

Le sélecteur apparaît sous la forme suivante.

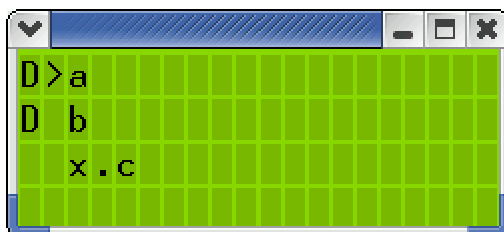

Figure 13. Sélecteur de fichiers

En actionnant la touche `'n'` pour entrer dans le répertoire `'a'` puis en sélectionnant les deux fichiers par la barre d'espace on obtient l'écran suivant.

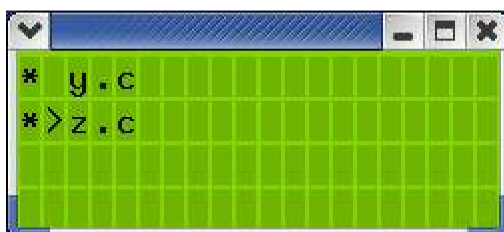

Figure 14. Sélection de fichiers

Finalement, en appuyant sur *Entrée* on obtient la liste suivante et l'on quitte le sélecteur.

```
/home/pierre/.../omm_api/fileselector/test/a/y.c
/home/pierre/../omm_api/fileselector/test/a/z.c
```

D'autres bibliothèques graphiques embarquées

La liste des projets citée dans cet article n'est bien entendu pas exhaustive et il existe un grand nombre de projets dédiés aux IHM embarquées (voir le lien vers la page *LinuxDevices.com* dans la bibliographie). On peut cependant citer quelques projets intéressants ou prometteurs.

SVGALIB (http://www.svgalib.org)

Cette bibliothèque a déjà été citée dans le paragraphe précédent. Elle fournit des fonctions bas niveau d'accès au frame-buffer (tracé de lignes, figures géométrique, affichage de texte, etc.). Elle n'est plus trop maintenu à ce jour mais elle est simple d'emploi et offre un support VESA acceptable.

DirectFB (http://www.directfb.org)

Beaucoup plus récente, cette bibliothèque fournit un accès au frame-buffer en bas niveau mais avec des pilotes accélérés.

WxWindows (http://www.wxwindows.org)

Cette bibliothèque en C++ permet de programmer des applications graphiques de manière portable sur plusieurs systèmes d'exploitation (Linux, eCOS, Windows, Windows CE, etc.) et ce en s'appuyant sur diverses couches graphiques (X11, GTK+, MicroWindows, etc.).

Bibliographie

- L'ouvrage *Linux embarqué* paru aux édition Eyrolles en octobre 2002 dont la présentation est accessible depuis http://www.ficheux.com.
- La liste des principaux projets d'interfaces graphiques embarquées sur http://www.linuxdevices.com/articles/AT9202043619.html
- Le site du projet Xfree86 sur http://www.xfree86.org
- Le site du standard VESA sur http://www.vesa.org
- Le site de télé-chargement de la bibliothèque Qt-Embedded sur http://www.trolltech.com/download/qt/embedded.html.
- Le site du projet GTK+ sur http://www.gtk.org
- Le site du projet MicroWindows/Nano-X sur http://www.microwindows.org
- Le site du projet LCDproc sur http://lcdproc.omnipotent.net
- Le patch à appliquer à GTK+-2.4.4 afin de compiler la version frame-buffer sur http://www.ficheux.com/articles/lmf/embedded_ihm/gtk_2.4.4_linux-fb.patch
- La bibliothèque *omm_api* sur http://www.ficheux.com/articles/lmf/embedded_ihm/omm_api.tgz
- Le patch à appliquer à *LCDproc-0.4.5* pour le test avec *omm_api* sur http://www.ficheux.com/articles/lmf/embedded_ihm/lcdproc-0.4.5_omm_api.patch