va trouver toutes les informations systèmes de l'OS, l'interface */sys* n'affiche que les informations relatives aux périphériques réellement détectés. Cette interface permet depuis le shell de piloter ou en tous les cas de paramétrer les périphériques qui proposent une interface */sys* (Figure 4). Tout d'abord nous chargeons le driver I2C sur la cible :

```
insmod pcf8574.ko
```

Listing 6. Affichage sur le LCD

```
void SetupScreen(void)
{
   int i;
   thescreen.X = 320;
   thescreen.Y = 240;
   thescreen.backId = GrLoadImageFromFile("back.gif", 0);
   thescreen.RelayId_OFF = GrLoadImageFromFile("relay_off.gif", 0);
   thescreen.RelayId_ON  = GrLoadImageFromFile("relay_on.gif", 0);
   thescreen.LedId_OFF = GrLoadImageFromFile("Led_red_off.gif", 0);
   thescreen.LedId_ON  = GrLoadImageFromFile("Led_red_on.gif", 0);
   GrDrawImageToFit(MainWindow, gc, 0, 0, 320, 240, thescreen.backId);

   for (i = 0; i < 3; i++)
   {
      Relay[i].X = 15+(i*100);
      Relay[i].Y = 40;
      Relay[i].ImageId = thescreen.RelayId_OFF;
      Led[i].X = 40+(i*100);
      Led[i].Y = 170;
      Led[i].ImageId = thescreen.LedId_OFF;
      GrDrawImageToFit(MainWindow, gc, Relay[i].X, Relay[i].Y, 80, 100,
            Relay[i].ImageId);
      GrDrawImageToFit(MainWindow, gc, Led[i].X, Led[i].Y, 20, 20,
            Led[i].ImageId);
   }
}
```

Listing 7. Basculement d'images

```
void ToggleRelay(int index)
{
   static char Value = 0;

   if (Relay[index].ImageId == thescreen.RelayId_ON)
   {
      Relay[index].ImageId = thescreen.RelayId_OFF;
      Value = Value & ~(1<<index);
      I2CWrite(I2C_RELAY, Value);
   }
   else
   {
      Relay[index].ImageId = thescreen.RelayId_ON;
      Value = Value | 1<<index;
      I2CWrite(I2C_RELAY, Value);
   }
   GrDrawImageToFit(MainWindow, gc, Relay[index].X, Relay[index].Y,
         80, 100, Relay[index].ImageId);
}
```

Pour connaître les périphériques détectés, nous affichons les informations postées par les drivers à l'aide de la commande `dmesg`. Dans notre cas, nous verrons qu'un périphérique est détecté à l'adresse 0x21.

Pour tester la communication avec le périphérique à l'adresse 0x21, nous pouvons utiliser l'interface */sys* sans avoir besoin de réaliser une application de test :

```
mount -t sysfs none /sys
```

Un `echo` adéquat va permettre d'activer le troisième relais de l'I0expander (`0x04 = 1 << 3`).

Création de l'application

Il est temps à présent de coder l'application. Commençons par créer les fichiers *demo.c* et *demo.h*, ainsi que le fichier Makefile associé (Listing 1).

Le *Makefile* est ici réduit à une très simple expression. Vous remarquerez que l'application est compilé en mode `debug` (`-g`) avec l'affichage des warning étendu.

Il existe 2 modes de fonctionnement avec le serveur graphique Nano-X : un mode client / serveur séparés pour lequel le serveur est lancé avant l'application graphique en tant que processus séparé, et un mode *lié* pour lequel l'application et le serveur ne font qu'un (mode `LINK_APP_TO_SER-VER`).

C'est ce mode de fonctionnement que nous avons choisi pour des raisons de facilité (un seul processus à lancer et plus de rapidité à l'exécution).

Pour cela nous devons linker notre application avec toutes les librairies Nano-X, et nano-X doit être reconstruit en mode `LINK_APP_TO_SERVER` si cela n'est pas le cas.

Ensuite, nous créons un fichier *demo.c* minimum avec l'ouverture des différents périphériques (Listing 2).

Nous utilisons essentiellement 2 types de périphériques : le bus I2C pour piloter des entrées / sorties, et l'écran tactile du LCD. Nous utiliserons donc 2 variables globales relatives au 2 file descriptors des périphériques.

Concernant le bus I2C nous choisirons par la suite d'accéder au bus par son interface */dev/i2c/0* et les fonctions `read` et `write`. Nous pourrions aussi utiliser l'interface */sys* et effectuer des opérations de lecture et d'écriture au travers de cette interface…

À présent, nous anticipons un peu en plaçant dans le fichier *demo.h* toutes les définitions dont nous aurons besoin (Listing 3).