Pierre DIEUMEGARD
prof de SVT
Lycée Pothier
F 45044 ORLEANS
courriel : pierre.dieumegard@ac-orleans-tours.fr

# Emploi de programmes exécutables comme pilotes pour les logiciels d'EXAO (Windows 32 bits et Linux)

On a souvent coutume d'opposer Windows, le système d'exploitation commercial de Microsoft, à Linux, le système d'exploitation gratuit.
En fait, un certain nombre de recettes sont utilisables dans les deux environnements, et l'on peut développer des logiciels pour l'un ou pour l'autre de ces systèmes.

## 1 Principe fondamental : utiliser des "pilotes d'appareils de mesure", spécifiques de l'appareil utilisé, pour faire des logiciels généralistes.

### *1.1 Très (trop ?) grande diversité du matériel*

Il y a une grande diversité dans le matériel de mesure connectable à un ordinateur : certains appareils sont des cartes à enficher à l'intérieur de l'ordinateur, d'autres sont des appareils à brancher sur une prise sérielle RS232, d'autres sur la prise parallèle, ou bien sur la prise "manettes de jeux", ou sur une prise USB, etc. Certains appareils ne remplissent qu'une fonction, par exemple des pHmètres, d'autres peuvent faire deux types de mesures, par exemple à la fois le pH et la température. Certains peuvent faire des mesures de divers types, mais une seule à la fois, comme les multimètres. Certains systèmes multifonctions (Jeulin ESAO, Orphy...) peuvent réaliser simultanément des mesures sur divers capteurs, et peuvent agir sur le système de mesure en actionnant des relais ou en modifiant la tension (en volts) d'un connecteur spécifique.

Lorsqu'un programmeur rédige un logiciel de mesure, il ne peut pas le concevoir d'emblée capable de fonctionner avec tous les appareils ou interfaces de mesure. Même si il a la liste de tous les appareils existant actuellement, il ne peut pas prévoir ceux qui seront inventés l'année prochaine. Il faut donc qu'il rédige un logiciel "adaptable", qui pourra s'adapter aux appareils à venir.
C'est le même problème que pour les logiciels de traitement de texte et les imprimantes : il est impossible de faire un logiciel qui tienne compte des particularités de toutes les imprimantes. C'est pourquoi ces logiciels fonctionnent avec des "pilotes d'imprimantes", ou "drivers", qui sont spécifiques du matériel en question.

Dans les explications qui suivent, il sera surtout question du boitier ADES, diffusé par la Maison des Enseignants de Provence. C'est un petit boitier à brancher sur la prise parallèle (prise "imprimante"), et qui a deux entrées analogiques sur 10 bits (normalement entre 0 et 5 V) et 4 sorties logiques sous forme de relais. Des adaptateurs lui permettent d'utiliser les capteurs de Calibration, fondamentalement conçus pour être utilisés avec les calculettes de la série TI82 (mesure d'oxygène, de pH, etc).
Il sera aussi question de l'interface Orphy-GTS, diffusée par Micrelec, et qui possède un grand nombre d'entrées et de sorties logiques et analogiques. Cet appareil doit être connecté sur une prise série.
A partir de ces deux appareils, on peut imaginer que les solutions envisagées sont universelles.

### *1.2 Plusieurs systèmes de pilotes sont possibles*

#### 1.2.1 Bibliothèques dynamiques (« DLL » de Windows, ou « so » de Linux)

Ce sont des petits fichiers contenant des instructions spécifiques, et qui sont appelables directement par le logiciel principal. Lorsque le fichier a été chargé par le logiciel principal (« lié au logiciel »), celui-ci peut utiliser les fonctions de la bibliothèque

comme si elles faisaient directement partie du logiciel ou du langage de programmation. Ceci permet donc des mesures très rapides (avec un intervalle de l'ordre de la microseconde).
On peut réaliser de telles bibliothèques dynamiques au moyen des langages compilés de type Pascal-(Delphi, Kylix, FreePascal), des divers types de C++, ou de certains langages Basic compilés (PureBasic, FreeBasic). On peut utiliser ces bibliothèques dynamiques par les mêmes langages, ainsi que par d'autres langages ou logiciels : OpenOffice (au moins sous Windows), Freemat, MSWLogo, Python...
Malheureusement, tous les langages de programmation ne permettent pas l'appel de ces bibliothèques dynamiques.

### 1.2.2 Programmes-pilotes faisant de petits fichiers

On peut réaliser de petits programmes qui font les mesures, et mettent la valeur mesurée dans un petit fichier. Le logiciel principal lit ce fichier, dont la valeur peut changer au cours du temps, et peut alors tracer le graphique de la valeur en fonction du temps, faire des calculs statistiques, etc.
Cette méthode est plus universelle que la méthode des bibliothèques dynamiques, puisqu'elle est possible même pour des langages de programmation qui ne permettent pas de faire de telles bibliothèques, mais elle est beaucoup plus lente, et il ne faut pas vouloir faire de mesures plus rapidement qu'une ou quelques fois par seconde.
Deux variantes en sont possibles :

- Un programme-pilote indépendant, que l'on lance d'abord, et qui fait ses mesures et ses fichiers à intervalles réguliers. Il reste en fonctionnement pendant toute la durée de la série de mesures.
- Un programme-pilote qui est lancé sur ordre du logiciel principal, qui règle donc la fréquence des mesures. Le programme-pilote fait sa mesure, en inscrit le résultat dans un fichier, puis se termine.

### 1.2.3 Programmes-pilotes renvoyant une chaîne de caractères

Fondamentalement, lorsqu'on lance ces programmes-pilotes, ils font une mesure, et affichent le résultat à l'écran, par une instruction du type « writeln(mamesure) » de Pascal, ou « print(mamesure) » de Basic.
De tels pilotes peuvent être réalisés à l'aide de divers langages compilés, y compris les vieux langages pour DOS de type TurboPascal.
Leur intérêt est que certains logiciels (par exemple Scilab) peuvent lancer de tels programmes extérieurs et récupérer la chaîne de caractères.

La suite de ce texte ne décrira que les pilotes de type « pilote exécutable », c'est à dire du dernier type décrit ci-dessus, ainsi éventuellement que de la deuxième variante du deuxième type . Les exemples seront d'abord décrits pour le système Microsoft-Windows, puis pour le système Linux.

# 2 Syntaxe proposée pour les pilotes exécutables :

Ce paragraphe correspond à une explication du "pourquoi et comment" a été fait le choix des caractéristiques des fonctions proposées. Elle est semblable à la syntaxe pour les pilotes de type « bibliothèque dynamique », et provient de la syntaxe des commandes des interfaces Orphy.
On lancera le pilote exécutable avec une ligne de commande, qui lui indiquera les opérations à effectuer. Le premier mot de la ligne de commande sera le nom de la fonction, le deuxième mot sera le paramètre de la fonction (et éventuellement le troisième mot sera un deuxième paramètre, etc). Le pilote exécutable renverra le résultat sous forme d'affichage à l'écran (par une instruction de type print de Basic, ou writeln de Pascal, ou printf de C++.

Par exemple, l'exécution de
monprog.exe ead 3
renverra la valeur lue à l'entrée analogique 3.

La numérotation des voies commencera à zéro, au moins pour ce qui est de la programmation, le paramètre correspondant sera donc un entier positif (mais désigné par un type « longint », pour employer le moins de types possible). Ceci n'empêche pas éventuellement que la première voie (numérotée 0 pour la programmation) puisse porter un nom tel que "Voltmètre 1".
La syntaxe proposée ici est voisine de celle commandant la centrale de mesure Orphy-GTS de Micrelec ; si l'opération qu'elles doivent faire est incorrecte (voie inexistante, par exemple) elles renvoient une valeur fixe (par exemple -777 pour ce qui est des fonctions numériques).

Chaque fonction est identifiée par son nom, qui est une chaîne de caractères. Puisqu'il y a une différence entre les majuscules et les minuscules ; il vaut mieux convenir que tous les noms de fonctions seront en minuscule.

Chaque variante de fonction de mesure sera associée à un nom particulier.
Un premier intérêt en est d'indiquer clairement à quoi correspond la variante. Par exemple, pour les interfaces ayant plusieurs calibres sélectionnables logiciellement (Leybold-Cassy...), chaque voie de mesure correspondra à un calibre, qui sera indiqué

dans ce nom, ou bien pour les interfaces ayant plusieurs voies, spécialisées chacune dans un type de mesure (Orphy-GTS, qui a des calibres différents selon les connecteurs, ou bien Pierron SMF10, qui a des entrées spécialisées pour le pH ou le magnétisme...), ce nom indiquera ces spécialités.
Un deuxième avantage, plus universel, est que ce nom permettra au programme appelant de savoir quelles sont les voies réellement appelables. Toutes les voies ayant un nom (non nul) seront appelables, et les voies qui n'existent pas réellement auront un nom de longueur nulle, ce qui permettra au logiciel appelant de savoir que ces voies n'existent pas.

Le plus simple est de faire commencer le nom de la fonction qui renvoie le nom de la voie par la lettre , comme "nom".

ea : nième entrée analogique (résultat directement renvoyé par le convertisseur analogique-numérique)
nea : nom de la nième entrée analogique
ead : nième entrée analogique (résultat converti en unité SI, le plus souvent en Volts)
nead : nom de la nième entrée analogique renvoyant le résultat sous forme de "double"
sa  envoi de valeur au convertisseur numérique analogique sur la nième sortie analogique ; si tout s'est bien passé, elle renvoie valeur.
nsa : nom de la nième sortie analogique fixant directement la valeur du CNA.
sad : Elle permet la fixation de la nième sortie analogique à valeur. Pour la plupart des interfaces, la valeur sera en volts, mais on peut imaginer des systèmes plus complexes, où une sortie puisse commander une température, une vitesse de rotation, une intensité lumineuse, ou d'autres grandeurs pouvant varier.
nsad :nom de la  fonction précédente
eb : nième entrée binaire (ou entrée logique). Le résultat "vrai" correspondra à 1, et le résultat "faux" correspondra à zéro.
neb : nom de la nième entrée binaire
sb : fixation de la nième sortie binaire à "vrai" si valeur vaut 1, et à "faux" si valeur vaut zéro ; si tout s'est bien passé, elle renvoie valeur
nsb : nom de la nième sortie binaire.
titre : renvoie le titre de la DLL, que l'on pourra utiliser dans des boites de dialogue.
detail : renvoie le nom détaillé de la DLL, avec par exemple le nom de l'auteur, la date de révision, etc.

# 3 Exemples de pilotes exécutables

On peut réaliser ces programmes-pilotes à l'aide de tous les langages compilés, y compris les vieux langages sous DOS de type TurboPascal ou QuickBasic, en faisant un pilote par appareil de mesure.
Une autre possibilité est d'utiliser les langages modernes pouvant appeler des bibliothèques dynamiques (DLL de Windows). On fait alors un programme exécutable unique, qui appelle la bibliothèque dynamique qui lui est indiquée par le premier paramètre de la ligne de commande. Cela permet d'utiliser directement l'autre système de pilotes, à savoir les bibliothèques dynamiques.

## *3.1 Un exemple de programme pilote exécutable direct*

## *3.2 Un programme de pilote exécutable en Delphi, utilisant les bibliothèques dynamiques et renvoyant directement les valeurs*

```
program pdll32;
(*programme appelant une bibliothèque dynamique de mesure, et envoyant le résultat
à l'écran : à utiliser avec les logiciels interceptant les messages pour l'écran*)
(* conçu initialement pour Scilab*)
{$APPTYPE CONSOLE}
uses
  sysutils,wintypes;

  var l:thandle;
  var repchar:array[0..80] of char;
  var nead:function(n:word):pchar;
  var ead:function(n:word):double;
  var stdead:function(n:word):double;    stdcall;
  var stdnead:function(n:integer):pchar;    stdcall;
  var detail:function : pchar;
  var chemindll,nomfonction,chparam1,chparam2:string;
```

```
  var valparam1:double;
  var valparam2:double;
  var i:integer;

 begin
   chemindll:=paramstr(1);
  nomfonction:=paramstr(2);
  if paramcount>=3 then valparam1:=strtofloat(paramstr(3));
  if paramcount>=4 then valparam2:=strtofloat(paramstr(4));
  strpcopy(repchar,chemindll);
  @nead:=nil; @ead:=nil;@stdead:=nil;@stdnead:=nil;@detail:=nil;
   l:=loadlibrary(repchar);
   @nead:=getprocaddress(L,'nead');
   @ead:=getprocaddress(L,'ead');
   @stdead:=getprocaddress(L,'stdead');
   @stdnead:=getprocaddress(L,'stdnead');
   @detail:=getprocaddress(L,'detail');
   if nomfonction='ead' then writeln(ead(round(valparam1)):10:5);
   if nomfonction='nead' then writeln(nead(round(valparam1)));
   if nomfonction='stdead' then writeln(stdead(round(valparam1)):10:5);
   if nomfonction='stdnead' then writeln(stdnead(round(valparam1)));
   if nomfonction='detail' then writeln(detail);
  freelibrary(L);
end.
```

# 4 Récupération directe du résultat par le logiciel principal

## 4.1 Position du problème

Tous les langages de programmation, et normalement tous les logiciels ayant des fonctions de programmation, permettent de lancer un programme extérieur.
Certains logiciels (minoritaires) peuvent récupérer directement dans un tableau de chaînes de caractères tous les caractères qui normalement auraient dû être envoyés vers l'écran par le pilote.
Par exemple, supposons que le pilote doit écrire la chaîne de caractère «5.555» à l'écran, par l'instruction «writeln» de Pascal, ou «print» de Basic, ou «printf» de C++. Lorsque ce pilote est appelé par le logiciel, celui-ci récupère directement cette chaîne de caractère dans une variable. Il suffira donc ensuite de traiter cette chaîne de caractères par le logiciel, en la transformant en valeur numérique... C'est très pratique.

## 4.2 Utilisation par Scilab

Scilab est un logiciel de mathématiques, librement distribuable, et réalisé par l'INRIA.
En fouillant dans la documentation en anglais, il semble capable d'être lié à des bibliothèques dynamiques extérieures, mais cette documentation est très confuse.
En revanche, il peut lancer des logiciels extérieurs, et surtout récupérer facilement le résultat.
L'instruction la plus intéressante est unix_g.
`x=unix_g(commande)`
lance la commande «commande» du système d'exploitation. Cette commande peut être le nom du fichier à exécuter, suivi éventuellement d'instructions en ligne de commande.
Le point le plus intéressant est que si cette commande doit écrire à l'écran (du type writeln de Pascal), ce qui devrait être écrit à l'écran est envoyé vers la variable x.
Par exemple x= unix_g('dir') exécute la commande «dir» du DOS, qui affiche la liste de tous les fichiers du répertoire en cours ; x contient donc la liste de ces fichiers, que l'on pourra traiter ensuite ...

```
x=unix_g('pdll32.exe xadestar.dll ead 0')
```
fait la mesure de l'entrée analogique 0 dans la bibliothèque xadestar, et envoie le résultat de la mesure dans la variable x.

## 4.3 Small Basic

SmallBasic est un langage Basic gratuit, réalisé par Nicholas Christopoulos, et qui existe pour divers systèmes d'exploitation, en particulier Windows et Linux, mais aussi PalmOS et d'autres.
L'instruction x=run(commande) envoie le résultat affichable de la commande dans la variable x.
Par exemple, `x=run("command.com /C dir *.exe")` provoque l'envoi dans la variable x d'un ensemble de chaînes de caractères, résultat de la commande `"dir *.exe "`.
On peut donc utiliser cette instruction pour lancer un petit programme de mesure, et récupérer le résultat, comme pour Scilab.

```
x=run("c:/mes documents/scilab/pdll32 xadestar.dll nead 1")
print x
```

Voici donc un petit programme qui affiche le nom de l'appareil (fonction «détail»), puis le nom et la valeur de l'entrée analogique 1 :

```
rem essai de programme pour Small Basic, utilisant les pilotes exécutables

nomexe="c:/mes documents/scilab/pdll32.exe"
nomdll="xdllvide.dll"
x=run(nomexe+" "+nomdll+" detail")
print "voici le nom de l'appareil et de son pilote"
print x
print "voici le nom de l'entrée analogique 1:"
x=run(nomexe+" "+nomdll+ " nead 1")
print x
print "voici la valeur lue à l'entrée analogique 1:"
x=run (nomexe+" "+nomdll+" ead 1")
print val(x)
```

## 4.4 Yabasic

Yabasic est un petit langage Basic sans prétentions, d'origine allemande, et fonctionnant avec divers systèmes d'exploitation : Windows, Linux,... et même la console de jeux PS2, que je n'ai pas essayée...
Il est téléchargeable en http://www.yabasic.de/yabasic.htm
L'instruction

```
a$=system$("dir *.exe /w ")
```

envoie dans la variable a$ l'ensemble de chaînes de caractères correspondant à la commande dir *.exe /w.
En remplaçant cette commande par l'appel à un programme de mesure, comme pour Scilab, on peut capturer le résultat de ce programme.

```
rem petit programme pour Yabasic, permettant d'afficher le nom détaillé et l'entrée
analogique 0
rem print system$("command.com /C dir *.exe /w")
nomexe$="e:pdll32.exe"
nomdll$="e:xdllvide.dll"

print "voici le nom détaillé de la dll :"
print system$("command.com  /C "+nomexe$+" "+nomdll$+"  detail")
print "voici le nom de l'entrée analogique 0"
print system$("command.com /C "+nomexe$+" "+nomdll$+" nead 0")
print system$("command.com /C e:scidll32.exe e:xdllvide.dll ead 0")
```

## 4.5 PHP

Normalement, PHP est un langage de scripts pour les serveurs Web.
C'est le même principe, grâce à l'instruction `system($commande)`, qui effectue la commande correspondant au contenu de la chaîne de caractère $commande.

# 5 Ecriture dans un fichier, pour des logiciels ne pouvant pas récupérer le résultat de la commande

## 5.1 Les pilotes précédents peuvent écrire leur résultat dans un fichier de texte

On peut faire des programmes spéciaux, mais il est possible aussi de forcer les programmes précédents, ceux qui affichent leur résultat à l'écran, à écrire le résultat dans un fichier en mettant en fin de ligne de commande >fichier.txt.
exemple :

```
monpilote.exe nead 1 >fichier.txt
```

Ensuite, il suffit de lire le contenu du fichier fichier.txt.

## 5.2 Utilisation par Matlab

Le lancement du programme se fait en commençant la ligne par le point d'exclamation :

```
!monpilote.exe ead 1 >fichier.txt
```

Ensuite, il faut lire le fichier «fichier.txt» par Matlab.
Si celui-ci contient une valeur numérique, par exemple renvoyée par ead, il n'y a pas de problème : a=load('fichier.txt') envoie cette valeur dans la variable a.

C'est un peu plus délicat pour les chaînes de caractères, par exemples celles renvoyées par «detail» ou par «nead». Matlab est un logiciel conçu pour traiter les nombre, et beaucoup moins pour traiter les chaînes de caractères.
Ceci se fait par l'instruction textread, qui a un grand nombre de possibilités, assez confuses dans la documentation.
On peut proposer :

```
machaine=textread('fichier.txt','%c','whitespace','\t')'
```

Quelques explications :
l'instruction %c indique que l'on lit une chaîne de caractères.
«whitespace» et «\t» indique que le caractère «espace» est la tabulation. Ceci peut paraître idiot, car il n'y a pas de tabulation dans le fichier, et que les espaces sont des vrais espaces entre les mots. Oui, mais si on ne met pas cette instruction, les espaces disparaissent dans la chaîne finale.
Enfin, le caractère «'» final permet d'avoir une chaîne en ligne, alors que lorsqu'on l'omet, la chaîne est affichée en colonne. Ce n'est pas simple !

## 5.3 StarOffice-Openoffice

Bien sûr, OpenOffice pour Windows permet d'utiliser les DLL. Mais il peut aussi lancer des fichiers extérieurs.
En frappant le sous-programme ci-dessous dans StarBasic, puis en associant un bouton à ce sous-programme, on déclenche l'affichage d'une fenêtre montrant la valeur de l'entrée analogique 1.

```
sub afficheead1
Dim iNumber As Integer
Dim aFile As String
dim sline as string
dim temps0 as long
rem dim commande$ as string
lignecommande$=dirdll$+"\pdll32.exe xdllvide.dll ead 1 >fichier.txt"
aFile = "fichier.txt"
iNumber=Freefile rem ces trois lignes sont pour détruire le fichier fichier.txt, s'il
existe
open aFile for output as #inumber
close #inumber
temps0=getsystemticks()
shell(lignecommande$,false)
while (filelen(afile)<2)and(getsystemticks()-temps0<1000)
      wend rem cette boucle attend que fdll32.exe ait fait son travail, fdll32.txt
iNumber = Freefile
Open aFile For input As #iNumber
Line Input #iNumber, sLine
close #inumber
```

```
msgbox(val(sline))
end sub
```

## *5.4 vieux langages DOS*

### 5.4.1 GW-Basic et Ubasic

GW-Basic (alias Basica) a du mal à fonctionner sur les ordinateurs modernes, mais on peut trouver sur Internet une version compatible avec les ordinateurs du XXIe siècle.. Ubasic, de Yuji KIDA, gagne à être connu pour les nostalgiques : il a la même présentation que le vieux, très vieux GW-Basic, mais a une puissance mathématique époustouflante, et beaucoup plus de puissance générale.
Le miniprogramme suivant affiche simplement le résultat de «detail» du pilote monpilote.exe :

```
10   doscmd "monpilote.exe detail >fichier.txt"
20   open "fichier.txt" for input as #1
30   input #1,Resultat$
35   close #1
40   print Resultat$
```

### 5.4.2 QBasic

En un peu plus complexe, par exemple, voici un petit programme en Qbasic, qui commande l'activation et l'extinction, à intervalles réguliers, de la sortie analogique 0 de l'appareil correspondant au pilote monpilote.exe.

```
INPUT "Quelle est la durée de l'impulsion ?", dimp
INPUT "quelle est la durée du cycle ?", dcycl
tdepart = TIMER
eteint = 1
DO
position = (TIMER - tdepart) MOD dcycl
IF (eteint = 0) AND (position > dimp) THEN
        SHELL ("monpilote.exe sb 0 0 >fichier.txt")
        eteint = 1
        OPEN "fichier.txt" FOR INPUT AS #1
        INPUT #1, reponse$
        PRINT reponse$
        CLOSE #1
        END IF
IF (eteint = 1) AND (position <= dimp) THEN
        SHELL ("monpilote.exe sb 0 1 >fichier.txt")
        eteint = 0
        OPEN "fichier.txt" FOR INPUT AS #1
        INPUT #1, reponse$
        PRINT reponse$
        CLOSE #1
        END IF
LOOP UNTIL (TIMER - tdepart > 20)
```

### 5.4.3 TurboPascal 7

```
program testedll;
uses dos,crt;
    {$M $4000,0,0 }   { 16K stack, no heap }
const eteint:boolean=false;
var fich:text;
    chainelue:string;
    tdepart:real;
    durimp,durcycle,position:longint;

function tsecondes:real;
var h,m,s,c:word;
begin
gettime(h,m,s,c);
tsecondes:=h*3600+m*60+s+c/100;
end;
```

```
begin
  writeln('quelle est la durée d''une impulsion ?');
  readln(durimp);
  writeln('quelle est la durée d''un cycle ?');
  readln(durcycle);
  tdepart:=tsecondes;
    assign(fich,'fichier.txt');
repeat
  position:=round((tsecondes-tdepart)) mod durcycle;
  if (eteint and (position<durimp))
  then begin
       exec('monpilote.exe stdsb 0 1 >fichier.txt');
       reset(fich);
       read(fich,chainelue);
       writeln(chainelue);
       close(fich);
       eteint:=false;
       end;
  if (not eteint)and(position>=durimp)
     then begin
          exec('pdll32.exe','xdllrobotcom2.dll stdsb 0 0 >fichier.txt');
          reset(fich);
          read(fich,chainelue);
          writeln(chainelue);
          close(fich);
          eteint:=true;
          end;
until keypressed;
end.
```

## 5.5 Python

A l'intérieur du module os (il faut donc faire «from os import *»), il y a les commandes :
- startfile, qui lance le logiciel qu'on lui passe en paramètre.
- system, qui lance la commande-système qu'on lui passe en paramètre.

# 6 Et pour Linux ?

Le principe des pilotes est très voisin.

## 6.1 le problème d'utilisation des ports en Linux

Linux est un système d'exploitation conçu pour être multi-tâches et multi-utilisateurs. Normalement, il y a un super-utilisateur (administrateur du réseau), nommé «root», qui peut commander complètement l'ordinateur, et les autres utilisateurs, qui n'ont que des droits réduits.
Les utilisateurs normaux n'ont pas le droit d'accéder directement aux ports de l'ordinateur, ce qui est dans l'absolu une bonne mesure de sécurité, car l'accès direct à l'électronique permet de faire beaucoup de bêtises. Malheureusement, pour beaucoup d'appareils de mesure, en particulier pour les cartes insérées dans les connecteurs d'extension, il est indispensable de pouvoir écrire et lire des valeurs sur les ports.
Une solution est bien sûr d'utiliser l'ordinateur en tant que super-utilisateur «root». Mais dans ce cas, il n'y a aucune protection contre les erreurs !

Le problème est donc d'autoriser l'utilisateur normal à accéder à certains ports, ceux utiles pour la mesure.

La solution que je propose est la suivante :
- écrire et compiler les pilotes en tant que «root», qui sera donc le légitime propriétaire et utilisateur du pilote.
- ensuite, toujours en tant que «root», modifier les droits d'accès de ce pilote, en déclarant que tous les utilisateurs pourront s'en servir, avec les mêmes droits que «root».

Deux méthodes sont possibles pour donner cette autorisation :
- la méthode «ligne de commande» (Merci à Georges Khaznadar): si le pilote est une bibliothèque dynamique nommée liblxmachine.so, il faut entrer la commande :
chmod 4755 liblxmachine.so
Si le pilote est un exécutable nommé monpilote, il faut modifier ses propriétés par chmod 4755 monpilote

Le nombre mystérieux 4755 signifie que toute personne ayant droit d'exécution a les mêmes droits que le propriétaire initial, et que tout utilisateur aura le droit d'exécution.

- la méthode «cliquage de souris» avec les versions modernes de Linux, comme par exemple Mandrake 8.
En mode graphique, lorsqu'on clique avec le bouton droit de la souris sur l'icône représentant le fichier liblxmachine.so, on peut modifier les propriétés de ce fichier, et donc donner les droits précédents.

Ensuite, après ces opérations faites en tant que «root», les utilisateurs normaux pourront utiliser la bibliothèque liblxmachine.so ou le pilote exécutable monpilote qui accèderont aux ports de mesure.

## *6.2 Pilotes exécutables qui envoient le résultat vers l'écran*

Le principe est le même que pour Windows.

### 6.2.1 Ecriture de pilote exécutable en Kylix, appelant une bibliothèque dynamique

Plutôt que de faire un pilote exécutable par matériel de mesure, il est possible d'écrire un seul pilote exécutable, qui lui-même appellera la bibliothèque de l'appareil de mesure.

```
program pdll32;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  wintypes;

var l:thandle;
  var repchar:array[0..80] of char;
  var ea:function(n:word):integer;
  var nea:function(n:word):pchar;
  var stdea:function(n:word):integer;stdcall;
  var stdnea:function(n:word):pchar ;stdcall;
  var nead:function(n:word):pchar;
  var ead:function(n:word):double;
  var stdead:function(n:word):double;    stdcall;
  var stdnead:function(n:word):pchar;    stdcall;
  var sa:function(n:word;valeur:word):integer;
  var nsa:function(n:word):pchar;
  var rsa:function(n:word):integer;
  var stdsa:function(n:word;valeur:word):integer;stdcall;
  var stdnsa:function(n:word):pchar;stdcall;
  var stdrsa:function(n:integer):integer;stdcall;
  var sad:function(n:word;valeur:double):double;
  var nsad:function(n:word):pchar;
  var rsad:function(n:word):double;
  var stdsad:function(n:word;valeur:double):double;stdcall;
  var stdnsad:function(n:word):pchar;stdcall;
  var stdrsad:function(n:word):double;stdcall;
  var eb:function(n:word):integer;
  var neb:function(n:word):pchar;
  var stdeb:function(n:word):integer;stdcall;
  var stdneb:function(n:word):pchar;stdcall;
  var sb:function(n:word;valeur:word):integer;
  var nsb:function(n:word):pchar;
  var rsb:function(n:word):integer;
  var stdsb:function(n:word;valeur:word):integer;stdcall;
  var stdnsb:function(n:word):pchar;stdcall;
  var stdrsb:function(n:word):integer;stdcall;
  var detail,titre:function : pchar;
  var stddetail,stdtitre:function:pchar;stdcall;
  var chemindll,nomfonction,chparam1,chparam2:string;
  var valparam1:double;
  var valparam2:double;
  var i:integer;
 begin
  // Insérer le code utilisateur ici
  //for i:=0 to paramcount do writeln('paramètre ',i,':',paramstr(i));
  valparam1:=11;valparam2:=22;
```

```
chemindll:=paramstr(1);
nomfonction:=paramstr(2);
if paramcount>=3 then valparam1:=strtofloat(paramstr(3));
if paramcount>=4 then valparam2:=strtofloat(paramstr(4));
strpcopy(repchar,chemindll);
@ea:=nil;@nea:=nil;  @nead:=nil; @ead:=nil;
@sa:=nil;@nsa:=nil;@rsa:=nil;@sad:=nil;@nsad:=nil;@rsad:=nil;
@eb:=nil;@neb:=nil;@sb:=nil;@nsb:=nil;@rsb:=nil;
@detail:=nil;@titre:=nil;@stddetail:=nil;@stdtitre:=nil;
@stdea:=nil;@stdnea:=nil; @stdead:=nil;@stdnead:=nil;
@stdsa:=nil;@stdnsa:=nil;@stdrsa:=nil;
@stdsad:=nil;@stdnsad:=nil;@stdrsad:=nil;
@stdeb:=nil;@stdneb:=nil;@stdsb:=nil;@stdnsb:=nil;@stdrsb:=nil;
 l:=loadlibrary(repchar);
 @ea:=getprocaddress(L,'ea');
 @nea:=getprocaddress(L,'nea');
 @nead:=getprocaddress(L,'nead');
 @ead:=getprocaddress(L,'ead');
 @sa:=getprocaddress(L,'sa');
 @nsa:=getprocaddress(L,'nsa');
 @rsa:=getprocaddress(L,'rsa');
 @sad:=getprocaddress(L,'sad');
 @nsad:=getprocaddress(L,'nsad');
 @rsad:=getprocaddress(L,'rsad');
 @eb:=getprocaddress(L,'eb');
 @neb:=getprocaddress(L,'neb');
 @sb:=getprocaddress(L,'sb');
 @nsb:=getprocaddress(L,'nsb');
 @rsb:=getprocaddress(L,'rsb');
 @detail:=getprocaddress(L,'detail');
 @titre:=getprocaddress(L,'titre');
 @stddetail:=getprocaddress(L,'stddetail');
 @stdtitre:=getprocaddress(L,'stdtitre');
 @stdea:=getprocaddress(L,'stdea');
 @stdnea:=getprocaddress(L,'stdnea');
 @stdead:=getprocaddress(L,'stdead');
 @stdnead:=getprocaddress(L,'stdnead');
 @stdsa:=getprocaddress(L,'stdsa');
 @stdnsa:=getprocaddress(L,'stdnsa');
 @stdrsa:=getprocaddress(L,'stdrsa');
 @stdsad:=getprocaddress(L,'stdsad');
 @stdnsad:=getprocaddress(L,'stdnsad');
 @stdrsad:=getprocaddress(L,'stdrsad');
 @stdeb:=getprocaddress(L,'stdeb');
 @stdneb:=getprocaddress(L,'stdneb');
 @stdsb:=getprocaddress(L,'stdsb');
 @stdnsb:=getprocaddress(L,'stdnsb');
 @stdrsb:=getprocaddress(L,'stdrsb');
 if nomfonction='ea' then writeln(ea(round(valparam1)));
 if nomfonction='nea' then writeln(nea(round(valparam1)));
 if nomfonction='ead' then writeln(ead(round(valparam1)):10:5);
 if nomfonction='nead' then writeln(nead(round(valparam1)));
 if nomfonction='sa' then writeln(sa(round(valparam1),round(valparam2)));
 if nomfonction='nsa' then writeln(nsa(round(valparam1)));
 if nomfonction='rsa' then writeln(rsa(round(valparam1)));
 if nomfonction='sad' then writeln(sad(round(valparam1),valparam2):10:5);
 if nomfonction='nsad' then writeln(nsad(round(valparam1)));
 if nomfonction='rsad' then writeln(rsad(round(valparam1)):10:5);
 if nomfonction='eb' then writeln(eb(round(valparam1)));
 if nomfonction='neb' then writeln(neb(round(valparam1)));
 if nomfonction='sb' then writeln(sb(round(valparam1),round(valparam2)));
 if nomfonction='nsb' then writeln(nsb(round(valparam1)));
 if nomfonction='rsb' then writeln(rsb(round(valparam1)));
 if nomfonction='titre' then writeln(titre);
 if nomfonction='detail' then writeln(detail);
 if nomfonction='stdea' then writeln(stdea(round(valparam1)));
 if nomfonction='stdnea' then writeln(stdnea(round(valparam1)));
 if nomfonction='stdead' then writeln(stdead(round(valparam1)):10:5);
 if nomfonction='stdnead' then writeln(stdnead(round(valparam1)));
 if nomfonction='stdsa' then writeln(stdsa(round(valparam1),round(valparam2)));
```

```
   if nomfonction='stdnsa' then writeln(stdnsa(round(valparam1)));
   if nomfonction='stdrsa' then writeln(stdrsa(round(valparam1)));
   if nomfonction='stdsad' then writeln(stdsad(round(valparam1),valparam2):10:5);
   if nomfonction='stdnsad' then writeln(stdnsad(round(valparam1)));
   if nomfonction='stdrsad' then writeln(stdrsad(round(valparam1)):10:5);
   if nomfonction='stdeb' then writeln(stdeb(round(valparam1)));
   if nomfonction='stdneb' then writeln(stdneb(round(valparam1)));
   if nomfonction='stdsb' then writeln(stdsb(round(valparam1),round(valparam2)));
   if nomfonction='stdnsb' then writeln(stdnsb(round(valparam1)));
   if nomfonction='stdrsb' then writeln(stdrsb(round(valparam1)));
   if nomfonction='stdtitre' then writeln(stdtitre);
   if nomfonction='stddetail' then writeln(stddetail);
   freelibrary(L);
end.
```

## *6.3 Appel du pilote exécutable par des logiciels qui récupèrent directement le résultat envoyé à l'écran.*

### 6.3.1 Scilab fonctionne sans problème, avec le même protocole que pour Windows.

```
chdir('/home/utilisateur/travail');
x=unix_g('./monpilote ead 1');
disp(x);
```

Les commandes précédentes mettent comme répertoire courant «/home/utilisateur/travail», puis lancent le pilote «monpilote» en demandant la mesure sur l'entrée analogique réelle 1 («ead(1)»). Le résultat de cet appel est envoyé vers l'écran par le pilote, mais Scilab récupère cet envoi dans la variable x, qui est finalement affichée dans la fenêtre de Scilab..

### 6.3.2 SmallBasic et Yabasic font de même.

Pour SmallBasic, les instructions suivantes envoient dans la variable x$ la valeur de l'entrée analogique double 1, puis affichent la valeur :

```
x$=run("/home/utilisateur/travail/monpilote ead 1 ")
print(val(x$))
```

Pour Yabasic, si on suppose qu'on est dans le répertoire où sont les fichiers (pilote exécutable et bibliothèque), on peut imprimer le nom de l'entrée analogique 1 :

```
print system$("./monpilote nead 1 ")
```

### 6.3.3 Sous Linux, Python possède le module «commands».

Il faut d'abord charger ce module par l'instruction `import commands`, en plus du module «os» chargeable par `import os`.
La commande la plus importante est : `getoutput` qui exécute une commande et récupère son résultat.

```
os.chdir('./home/utilisateur/')
data=commands.getoutput('./monpilote nea 1')
```

Dans la variable data se trouve le résultat de la commande, c'est à dire le nom de l'entrée analogique 1.

### 6.3.4 StarOffice et OpenOffice

StarOffice, et son successeur OpenOffice existent aussi bien pour Windows que pour Linux.
Malheureusement, le système d'appel des bibliothèques à liaison dynamique semble ne fonctionner que pour Windows.
Pour Linux, il semble obligatoire d'utiliser les fonction d'appel de logiciels extérieur, ce qui est beaucoup plus lent.
Par exemple, le sous-programme en StarBasic ci-dessous affiche dans une boite de dialogue le résultat de la mesure sur l'entrée analogique 1 :

```
sub afficheead1
Dim iNumber As Integer
Dim aFile As String
dim sline as string
dim temps0 as long
rem dim commande$ as string
lignecommande$=dirdll$+"\monpilote ead 1"
```

```
aFile = "fdll32.txt"
iNumber=Freefile
open aFile for output as #inumber
close #inumber
temps0=getsystemticks()
shell(lignecommande$,false)
while (filelen(afile)<2)and(getsystemticks()-temps0<1000)
wend
iNumber = Freefile
Open aFile For input As #iNumber
Line Input #iNumber, sLine
close #inumber
msgbox(val(sline))
end sub
```

(ce programme est valable pour StarOffice 5.2. Avec OpenOffice et les versions récentes de StarOffice, la syntaxe du langage est différente)

# 7 Bref résumé des compatibilités Windows/Linux pour ce qui est des bibliothèques dynamiques et des pilotes d'appareils de mesure :

- Pour écrire les bibliothèques : les langages Pascal (Delphi, Kylix, FreePascal...), Basic (PureBasic) et C++ (C++Builder, g++...) sont utilisables, aussi bien sous Windows que sous Linux. .

- Pour utiliser directement les bibliothèques :
Sous Windows et Linux, les mêmes langages Pascal , Basic et C++ sont utilisables (Delphi, Kylix, PureBasic, C++Builder, g++...)
Sous Windows, d'autres langages permettent d'utiliser les DLL : MSW-Logo, StarOffice-StarBasic, Lotus SmartSuite (Lotus Script)...
Avec quelques particularités, sous Windows, Python peut aussi lire les DLL grâce au système PythonForDelphi.

- Certains logiciels permettent de récupérer directement dans des chaînes de caractères (ou des tableaux de chaînes) les messages envoyés normalement à l'écran par les programmes qu'ils ont lancés. Dans ce cas, on peut utiliser un «pilote exécutable» sous la forme d'un programme exécutable qui reçoit des précisions en ligne de commande, et qui renvoie le résultat des opérations sous la forme d'une chaîne de caractères à l'écran.
Ces programmes exécutables peuvent être des programmes autonomes, mais aussi et surtout, des programmes écrits en Pascal ou C, et qui eux-mêmes appellent une bibliothèque dynamique dont le nom est passé en paramètre, avant les autres paramètres décrivant les voies à mesurer ou les valeurs à fixer. Ceci permet, indirectement, d'utiliser les bibliothèques dynamiques pour les logiciels ne pouvant pas les appeler directement.
Sous Windows et Linux, c'est le cas de Scilab, de SmallBasic et YaBasic.
Sous Linux, c'est le cas de Python.

(il semblerait que PHP ait aussi de telles caractéristiques, grâce à l'instruction system($commande)... à creuser...)