# Langage C et programmation système

DUT Informatique Villetaneuse

Module : Système d'Exploitation

christophe.cerin@iutv.univ-paris13.fr

# Variables d'environnement

- Votre problème : vous développez une application en C et vous souhaitez accéder à $PATH, $HOME, $SHELL… pour les lire, les modifier
- Rappel 1 : sous le prompt du shell, on accède aux variables d'environnement par `printenv, env`
- Rappel 2 : `export, setenv, unsetenv`

# Variables d'environnement

- Rappel 2 : elles sont exportées de l'environnement vers les activités lancées par le shell, par ex la valeur de $PATH peut avoir de l'importance dans un script qui appelle des exécutables
- Un programme en langage C peut accéder aux variables d'environnement via des paramètres de la fonction `main()` ou via des fonctions de la bibliothèque standard (`stdlib.h`).

# Variables d'environnement

```
#include<stdio.h>
/*
argc: nb d'argument sur ligne de commande
argv: tableau ptr sur les arguments lig de commande
arge: tableau ptr sur les var d'environnement
*/
main(int argc, char *argv[],char *arge[])
{
  int i;
  printf("Valeur de argc=%d\n",argc);
  printf("Valeur de argv: ");
  for(i=0;i<argc;i++) printf("%s ",argv[i]);
  printf("\n");
  printf("Valeur de arge");
  i=0;
  while(arge[i] != NULL) {
    printf("%s ",arge[i]);i++;
  }
  printf("\n");
}
```

# Variables d'environnement

```
Ordinateur-de-Christophe-Cerin:~ cerin$ a.out titi toto
Valeur de argc=3
Valeur de argv: a.out titi toto
Valeur de argeMANPATH=/sw/share/man:/usr/share/man:/usr/X11R6/man
TERM_PROGRAM=Apple_Terminal TERM=xterm-color
SHELL=/bin/bash PERL5LIB=/sw/lib/perl5
TERM_PROGRAM_VERSION=100.1.8
OLDPWD=/Users/cerin/Desktop/TRI XAPPLRESDIR=/sw/etc/app-defaults/
SGML_CATALOG_FILES=/sw/etc/sgml/catalog
USER=cerin __CF_USER_TEXT_ENCODING=0x1F5:0:1
PATH=.:/usr/local/bin:/sw/bin:/sw/sbin:/bin:/sbin:/usr/bin:
/usr/sbin:/usr/X11R6/bin
XML_CATALOG_FILES=/sw/etc/xml/catalog PWD=/Users/cerin
LD_PREBIND_ALLOW_OVERLAP=1 SHLVL=1
HOME=/Users/cerin LD_PREBIND=1
LD_SEG_ADDR_TABLE=/sw/var/lib/fink/prebound/seg_addr_table
LOGNAME=cerin INFOPATH=/sw/share/info:/sw/info:/usr/share/info
SECURITYSESSIONID=2129b0 _=./a.out
```

# Variables d'environnement

```
NAME
     getenv, putenv, setenv, unsetenv - environment variable functions

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdlib.h>

     char *
     getenv(const char *name);

     int
     setenv(const char *name, const char *value, int overwrite);

     int /* string: name=value => equivalent a setenv(name, value, 1); */
     putenv(const char *string);

     void
     unsetenv(const char *name);
```

# Variables d'environnement

```
#include<stdio.h>
#include<stdlib.h>

main(int argc, char *argv[], char *arge[])
{
        int             i;
        char           *path, *bar;

        path = getenv("PATH");
        if (path == NULL)
                printf("PATH non initialisee");
        else
                printf("AVANT: %s\n", path);

        bar = (char *) calloc(strlen(path) + 3, sizeof(char));
        if (bar == NULL) printf("Error");
        else {
                sprintf(bar, "%c%c%c%s", '~', '/', ':', path);
                setenv("PATH", bar, 1);
                path = getenv("PATH");
                if (path == NULL)
                        printf("PATH non initialisee");
                else
                        printf("APRES: %s\n", path);
        }
}
```

# Variables d'environnement

```
Ordinateur-de-Christophe-Cerin:~ cerin$ echo $PATH
.:/usr/local/bin:/sw/bin:/sw/sbin:/bin:/sbin:/usr/bin:/usr/sbin
Ordinateur-de-Christophe-Cerin:~ cerin$ a.out
AVANT:
.:/usr/local/bin:/sw/bin:/sw/sbin:/bin:/sbin:/usr/bin:/usr/sbin
APRES:
~/:.:/usr/local/bin:/sw/bin:/sw/sbin:/bin:/sbin:/usr/bin:/usr/sbi
n
Ordinateur-de-Christophe-Cerin:~ cerin$ echo $PATH
.:/usr/local/bin:/sw/bin:/sw/sbin:/bin:/sbin:/usr/bin:/usr/sbin
```

Il n'y a pas eu de modification globale ! La modification de PATH est locale au processus.

(Note : certains systèmes permettent une modification permanente ; man setenv svp)

# Exécuter un fichier depuis C

- Famille de primitives : `execl, execlp, execle, exect, execv, execvp`
- Ces fonctions remplacent le processus courant par le fichier à exécuter
- Premier paramètre : le chemin d'accès au fichier ; Paramètres suivants dépendant de la primitive ;

# execl

**Using execl()**

The following example executes the *ls* command, specifying the pathname of the executable (**/bin/ls**) and using arguments supplied directly to the command to produce single-column output.

```
#include <unistd.h>
int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

# execle

**Using execle()**

The following example is similar to Using execl(). In addition, it specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>
int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

# execv

**Using execv()**

The following example passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>
int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execv ("/bin/ls", cmd);
```

# execve

**Using execve()**

The following example passes arguments to the *ls* command in the *cmd* array, and specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>
int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home",
                "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

# On peut utiliser « system »

```
/* fichier e.c */
#include<stdlib.h>
main(int argc, char *argv[], char *arge[])
{
        system("/bin/ls -1 > toto");
}

Ordinateur-de-Christophe-Cerin:~ cerin$ gcc e.c
Ordinateur-de-Christophe-Cerin:~ cerin$ a.out
Ordinateur-de-Christophe-Cerin:~ cerin$ more toto
164expertsGT03.doc
3d_animals_009.jpg
3d_animals_053.jpg
APADE_dossier.doc
APADE_dossier.odt
APADE_dossier.rtf
```

# popen permet de récupérer plus directement des résultats

```
#include<stdio.h>
#include<stdlib.h>
main(int argc, char *argv[], char *arge[])
{
  char buf[256];
  FILE *pp;

  // Keep some information about me (localhost)
  if ((pp = popen ("uname -n", "r")) == NULL)
    {
      perror ("popen error");
      exit (1);
    }
  fgets (buf, sizeof (buf), pp);
  buf[strlen (buf) - 1] = '\0';
  pclose (pp);

  printf("Param: %s\n",buf);
}

Ordinateur-de-Christophe-Cerin:~ cerin$ uname -n
Ordinateur-de-Christophe-Cerin.local
Ordinateur-de-Christophe-Cerin:~ cerin$ a.out
Param: Ordinateur-de-Christophe-Cerin.local

Ordinateur-de-Christophe-Cerin:~ cerin$
```

# Les appels système

- Définition : exécution d'un code qui manipule les structures de données internes au SE sur demande de l'utilisateur
- Rappel : le SE a son propre code (sauvegardé dans /usr/src) et donc des structures de données propres
- Rappel : les droits !!!

# Les appels système

- Pour l'utilisateur cela se présente comme des fonctions (par ex celles des pages précédentes)
- C'est l'exécution des appels système qui n'est pas identique à l'exécution des fonctions
- Un « code système » accède partout et peut tout modifier dans le SE ; il peut exécuter des opérations spéciales pour accéder au matériel
- Il faut généralement être root pour exécuter un appel système et simple user pour une fonction (C'est pas une règle générale)

# Les appels système

- Il y a vérification des arguments des appels système et du droit de l'appelant pour l'action demandée
- Tentative d'uniformisation des appels système pour masquer les différences dans les structures de données d'un noyau à un autre (POSIX)

# Les appels système

- Faire un man syscalls pour lister les > 150 appels disponibles
- Exemple d'appel système : on veut implémenter des écritures sur disques synchrones c.à.d que l'instruction qui suit l'écriture ne s'exécute que lorsque le fichier est écrit.

# Les appels système

*** open, write, close : tout se joue dans les options à l'ouverture du fichier.**
*** Remarquez O_SYNC pour les écritures synchrones et S_IRWXU pour définir des droits sur le fichier.**
*** Ici un simple user peut exécuter ce code**
*** ATTENTION si écriture via NFS**

```
if ((MYfpOUT = open (ofName, O_CREAT| O_WRONLY |
                             O_SYNC | O_TRUNC, S_IRWXU)) == -1){
            perror ("io: openning input file");
            exit (1);
}

if (write (MYfpOUT, final, sizeof (Item) * sum_proc[mypid]) !=
          sizeof(Item)*sum_proc[mypid]) {
              perror ("io: writing input file");
              exit(1);
}

close (MYfpOUT);
```

# Les appels système : exemples

- Gestion des processus : fork, wait, getpid, signal, kill, exit…
- Il s'agit de faire de la communication locale
- Au programme de ce module
- Communication distante : socket, listen, bind, send, recv…
- Au programme de la deuxième année du DUT (système / réseau)