3IS - Système d'exploitation linux
-
Programmation système

2010 – David Picard
Contributions de : Arnaud Revel, Mickaël Maillard

picard@ensea.fr

Environnement

- Les programmes peuvent être exécutés dans des contextes très différents
  - Application terminal
  - Application graphique/interactive
  - service/tâche de fond
- Il est souvent nécéssaire d'avoir accès à des informations sur différents paramètres
  - Du système d'exploitation
  - De l'utilisateur lançant le programme
  - De la session dans laquelle le programme est lancée
- Ces informations sont accessibles via **l'environnement**

Variables d'environnement

- Les variables d'environnement sont des chaînes de caractères de la forme **NOM=VALEUR**
  - ex : **HOSTNAME=eris**
- Un certain nombre de variables sont initialisées lors de la création du processus
  - ex : **$PATH**
- L'environnement est hérité du père après `fork()`

extern char ** environ

- Les chaînes sont stockées dans une variable externe environ, initialisée au démarrage du programme
- Le dernier élément pointé du tableau est NULL

Exemple de programme

```
#include <stdio.h>

extern char ** environ;

int main(int argc, char ** argv)
{

   int I = 0;

   for(i = 0 ; environ[i] != NULL ; i++)
   {
      printf("%d : %s\n", i, environ[i]);
   }

   return 0;
}
```

Accéder à une variable

- On a souvent besoin de s'intéresser à seulement quelques variables d'environnement bien choisies
- La bibliothèque C fournit des fonctions pour ceci
  - `getenv()`, `setenv()`, `putenv()`
- Pas d'espace dans le nom de la variable
- Sensibilité à la casse (HOME différent de Home)

Exemple de `getenv()`

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

int main(int argc, char **argv)
{
    int i;
    char* variable;

    for(i = 1 ; i < argc ; i++)
    {
        variable = getenv(argv[i]);
        if(variable == NULL)
            printf("%s non définie\n", argv[i]);
        else
            printf("%s : %s \n", argv[i], variable);

    return 0;
}
```

```
int putenv(const char * chaine);
```

Pour ajouter ou modifier une variable (chaine de la forme “VARIABLE=VALEUR”)

```
int setenv(const char * nom, const char * valeur , int ecraser);
```

Pour ajouter ou modifer une variable (ecraser sert a gérer les droits de remplacement d'une variable)

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

Pour supprimer une variable d'environnement

Hérédité

- L'environnement est transmis d'un processus père à son fils par duplication
- Toute modification après le fork() ne propage pas les changements au père/fils

Variables populaires

- HOME : répertoire de l'utilisateur lançant le programme
- SHELL : interpreteur de commande par défaut de l'utilisateur
- TERM : type de terminal utilisé
- PATH : chemin dans lesquels se trouvent les programmes (séparés par ':')
- USER ou LOGNAME : nom de l'utilisateur

Options de programme

- Les arguments passés à un programme sont récupérés en arguments de la fonction `main()`
- `int argc` permet de savoir combien d'arguments ont été passés
- `char ** argv` permet d'avoir un tableau de chaine de caractères contenant les arguments
- `argv[0]` contient généralement le nom du programme

Options et arguments

- Les programmes unix permettent souvent de passer des options en plus des arguments
  - Ex : -f, -v, -r
- Certaines options prennent elles-mêmes un argument
  - Ex : l'option -f de tar nécéssite le nom du fichier à archiver
- La lib C permet d'accéder facilement aux options de la norme Posix

getopt()

```
/* parcours la list des arguments et renvoie le caractère correspondant à
l'option courante.
Renvoie -1 si toutes les options on été parcourrues.
Renvoie ? Si le caractère est non reconnu comme option valide
*/
int getopt(int argc, char ** argv, const char * options);

/* compteur d'indice dans le tableau argv */
int optind;

/* affiche un message d'erreur sur stderr si différent de zséro, lorsque qu'un
option non reconnue est parsée
*/
int opterr;

/* contient le caractère passé en option en cas d'option non reconnue
*/
int optopt;

/* contient l'argument de l'option si besoin */
char * optarg;
```

```c
#include <stdio.h> <unistd.h>

int main(int argc, char ** argv)
{
    char* list_options = "ac:";
    int option;
    opterr = 0; // pas de message en cas d'erreur

    while( (option = getopt(argc, argv, list_options)) != -1)
    {
        switch(option)
        {
            case 'a' : printf("option a\n");
                break;
            case 'c' : printf("option c avec paramètre %s\n", optarg);
                break;
            case '?' : printf("option non reconnue : %c\n", optopt);
                break;
        }
    }

    while (optind != argc)
        printf("argument restant : %s\n", argv[optind++]);
    return 0;
}
```

Fonctions simplifiées pour exécuter un sous-programme

- `FILE *popen(const char *command, const char *type);`
  - Construit un pipe en lecture ou écriture (argument `type`)
  - fork
  - Exécute /bin/sh en lui passant `command` comme argument
  - Le `FILE*` passé en retour permet d'écrire (resp. de lire) sur l'entrée (resp. la sortie) standard du programme invoqué
- `int pclose(FILE *stream);`
  - Sert à fermer un pipe ouvert avec `popen()`

I/O non bloquantes

- Intéressant quand on ne veut pas mettre le processus en attende d'I/O
- Fonction `fcntl(int fd, int command, …)`
  - Effectue une opération sur le descripteur de fichier fd
  - Troisième argument optionnel
- Insertion de l'attribut `O_NONBLOCK` au descripteur de fichier avec la commande `F_SETFL`

```c
#include <fcntl.h> <stdio.h>
#include <stdlib.h> <unistd.h>

int main(void) {

    int tube[2];
    char c;

    if(pipe(tube)!=0) exit(1);

    if(fork() == 0) {
        close(tube[0]);
        while(1) {
            write(tube[1], &c, 1);
            usleep(700000);
        }
    }
```

```c
     else {
            close(tube[1]);
            fcntl(tube[0],
F_SETFL, O_NONBLOCK);
            while(1) {
                if(read(tube[0],
&c, 1) == 1)
                    printf("read
Ok\n");
                else
                    printf("read
raté\n");
                usleep(100000);
            }
        }
        return 0;
}
```