STI 3ème année – Programmation Système

TD 1: Norme POSIX

C. Toinard – J.-F. Lalande - J. Briffaut

POSIX est le nom d'une famille de standards définie depuis 1988 par l'IEEE et formellement désignée IEEE 1003. Ces standards ont émergé d'un projet de standardisation des API des logiciels destinés à fonctionner sur des variantes du système d'exploitation UNIX.

Le terme POSIX a été suggéré par Richard Stallman en réponse à la demande de l'IEEE d'un nom facilement mémorisable. C'est un acronyme de **Portable Operating System Interface**, dont le X exprime l'héritage UNIX de l'Interface de programmation.

Voici les différentes versions de la norme POSIX :

Norme	Symbole	Valeur
POSIX.1-1988	_POSIX_SOURCE	
POSIX.1-1990 (1003.1)	_POSIX_C_SOURCE	1
POSIX.1b-1993	_POSIX_C_SOURCE	199309L
POSIX.1c-1996	_POSIX_C_SOURCE	199506L
POSIX.1-2001	_POSIX_C_SOURCE	200112L
XPG3	_XOPEN_SOURCE	1
XPG4	_XOPEN_SOURCE	4
XPG4	_XOPEN_VERSION	4
SUS	_XOPEN_SOURCE	1
SUS	_XOPEN_SOURCE_EXTENDED	4
SUSv2	_XOPEN_SOURCE	500

Pour compiler de manière conforme à POSIX.1 :

```
gcc -D_POSIX_C_SOURCE=1 -c source.c
```

1 Constantes de compatibilité

Exercice 1 Créez un fichier source C permettant de tester le positionnement des différentes constantes définissant la compatibilité POSIX au moyen de deux façons différentes.

Première façon :

```
#ifdef _POSIX_SOURCE
printf("_POSIX_SOURCE=%d\n",_POSIX_SOURCE);
#endif
```

Deuxième façon :

```
if (_POSIX_C_SOURCE == 2008)
printf("Code de 2008 inclu !\n");
else
printf("Code par défaut\n");
```

Listing 1 – Solution de l'exercice 1

```
#include <stdio.h>

int main()
{
#ifdef _POSIX_SOURCE
printf("_POSIX_SOURCE=%d\n",_POSIX_SOURCE);
#endif
#ifdef _SVID_SOURCE
printf("_SVID_SOURCE=%d\n",_SVID_SOURCE);
#endif
#ifdef _GNU_SOURCE
printf("_GNU_SOURCE=%d\n",_GNU_SOURCE);
#endif
#ifdef _ISOC99_SOURCE
printf("_ISOC99_SOURCE=%d\n",_ISOC99_SOURCE);
#endif
#ifdef _POSIX_C_SOURCE
printf("_POSIX_C_SOURCE=%d\n",_POSIX_C_SOURCE);
#endif
#ifdef _XOPEN_SOURCE
printf("_XOPEN_SOURCE=%d\n",_XOPEN_SOURCE);
#endif
#ifdef _XOPEN_SOURCE_EXTENDED
printf("_XOPEN_SOURCE_EXTENDED=%d\n",_XOPEN_SOURCE_EXTENDED);
#endif
#ifdef _BSD_SOURCE
printf("_BSD_SOURCE=%d\n",_BSD_SOURCE);
#endif
#ifdef __STRICT_ANSI__
printf("__STRICT_ANSI__=%d\n",__STRICT_ANSI__);
#endif
#ifdef _LARGEFILE64_SOURCE
printf("_LARGEFILE64_SOURCE=%d\n",_LARGEFILE64_SOURCE);
#endif

if (_POSIX_C_SOURCE == 2008)
  printf("Code de 2008 inclu !\n");
else
  printf("Code par défaut\n");

#if _POSIX_C_SOURCE==200112
  printf("Code de 2008 inclu par un #if!\n");
#endif

}
```

Exercice 2 Comment pourriez-vous tester votre fichier précédent au moyen du compilateur ?

Listing 2 – Solution de l'exercice 2

```
gcc -D_POSIX_C_SOURCE=2008 -o prog source.c
```

Exercice 3 A votre avis, les tests précédents sont-il réalisés à l'exécution ?

Les tests précédents (if ou #if) sont simplifiés à la compilation. Ils n'ont donc pas lieu à l'exécution.

Exercice 4 Au moyen de la fonction sysconf, vérifiez la valeurs d'autres constantes au moment de l'exécution.

Listing 3 – Solution de l'exercice 4

```
#include <unistd.h>
#include <stdio.h>
int main()
{
int j;
j = sysconf(_SC_VERSION);
printf("_SC_VERSION=%i\n",j);
}
```

2 Environnement

Les variables d'environnement sont des variables dynamiques utilisées par les différents processus d'un système d'exploitation

— L'environnement est accessible via : `extern char **environ;`

— Ou par la fonction POSIX :

```
#include <stdlib.h>
char *getenv(const char *nomDeVariable);
int putenv(const char *coupleNomValeur);
```

Exercice 5 Tapez "env" dans la console. Vous obtenez la liste de toutes les variables d'environnement du système.

Exercice 6 Ecrire un programme C équivalent à la commande env.

Listing 4 – Solution de l'exercice 6

```
#include <stdio.h>
#include <stdlib.h>
extern char **environ;
main() {
  char **ptr=environ;
  while (*ptr!=NULL) {printf("%s\n",*ptr); ptr++;}
}
```

Exercice 7 Ecrire un programme C qui prend en argument le nom d'une variable d'environnement et affiche sa valeur.

Listing 5 – Solution de l'exercice 7

```
#include <stdio.h>
#include <stdlib.h>
extern char **environ, *valeur;
int main(int argc, char ** argv) {
  int ind;
  char * valeur;
  if (argc>0)
  {
        valeur=getenv(argv[argc-1]);
        printf("%s\n",valeur);
  }
}
```

3 Appel système et gestion des erreurs

Exercice 8 Ecrivez un programme permettant d'écrire le message hello sur la sortie standard à l'aide d'un appel système.

Listing 6 – Solution de l'exercice 8

```
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
  (void)syscall(SYS_write, STDOUT_FILENO, "hello\n", 6);
}
```

Exercice 9 Modifiez ce programme afin qu'il génère une erreur lors de l'appel système.

Exercice 10 Traitez l'erreur retournée au moyen de perror (1 ligne).

Exercice 11 Sans utiliser perror, traitez l'erreur retournée au moyen de errno et de strerror. Il s'agit d'écrire un équivalent de la fonction perror qui écrit le texte explicatif suivi d'un message spécifique à l'application.

Listing 7 – Solution de l'exercice 11

```
#include <sys/syscall.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
int main()
{
  char *ptrBuff;
  int ret;
  ret=syscall(SYS_write, 5, "hello\n", 6);
  ptrBuff=strerror(errno);
  printf("1: errno=%d, ptrBuff=%d\n",errno, ptrBuff);
  ret=syscall(SYS_write, 6, "hello\n", 6);
  ptrBuff=strerror(errno);
  printf("2: errno=%d, ptrBuff=%d\n",errno, ptrBuff);
  if (ret==-1) perror("Erreur syscall");
  if (ret==-1)
  {
    printf("Mon perror:: %s: Erreur syscall, %d", ptrBuff, ptrBuff);
  }
}
```

4 Accès aux limites du système

Les limites du systèmes ne dépendent pas forcément que du système. Dans le cas des fichiers par exemple, une limite peut dépendre du fichier que l'on considère.

Exercice 12 Etudiez la fonction posix pathconf. Ecrivez une fonction ***void*** *pr_pathconf* *(**char** *path, **int** name, **char** * message)* qui prend en paramètre le nom de fichier à considérer, lnom de la limite (entier) et le message textuel á afficher pour cette limite. La fonction envoie sur la sortie standard le nom de la limite puis appelle la fonction pathconf afin de récupérer la valeur de la limite. La fonction envoie alors sur la sortie standard la valeur récupérée. Traitez les cas d'erreur avec errno de façon rudimentaire afin d'afficher le même message signalant l'erreur à la place du nom de la limite et de sa valeur.

Listing 8 – Solution de l'exercice 12

```
#include<unistd.h>

void pr_pathconf(char * mesg, char *
     path , int name) {
  long i;
  printf("%s",mesg);
  i=pathconf(path , name);
  if(i==-1) {
    printf("error\n");
  }
  else {
    printf("%d\n",i);
  }
}

int main() {
  char * chemin;
  chemin=(char*)malloc(256*sizeof(char)
       );
  printf("chemin?\n");
  //scanf("%s", chemin);
    fgets(chemin, 255, stdin);
    chemin[strlen(chemin)-1]='\0';
  pr_pathconf("nb max de liens associÕs
       : ", chemin ,_PC_LINK_MAX);
  free(chemin);
}
```

Exercice 13 Créez un programme main permettant de tester votre fonctions. Votre main ne testera que les limites POSIX.1.

5 Base de données utilisateur

Exercice 14 Ecrivez un programme qui prend en paramètre un nom d'utilisateur et affiche le contenu de l'enregistrement correspondant de la base de données des comptes.

Listing 9 – Solution de l'exercice 14

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

/*Fonction qui donne les infos d'un joueur passé en
     paramètre*/
static void info_user (char * name_user)
{
struct passwd *user;

/*getpwan permet de remplir la structure user*/
user = getpwnam (name_user);

if (user != NULL) {
  printf ("Nom : %s\n", user->pw_name);
  printf ("Password : %s\n", user->pw_passwd);
  printf ("UID : %i\n", user->pw_uid);
  printf ("GID : %i\n", user->pw_gid);
```

```
    printf ("Nom du groupe : %s\n", user->pw_gecos);
    printf ("Directory : %s\n", user->pw_dir);
    printf ("Shell : %s\n", user->pw_shell);
  }
  else
    printf ("Utilisateur inconnu\n");
}
```

Exercice 15 Proposez une méthode pour le tester pour l'ensemble des comptes existants sur votre système.

Listing 10 – Solution de l'exercice 15

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

/*Fonction qui donne les infos de tous les utilisateurs
     contenus dans /etc/passwd*/
static void info_all_users (void)
{
  FILE * p;
  int nb_users, i;
  char * name;

  /*On ajoute le nombre d'utilisateurs en début de fichier
       pour pouvoir faire une boucle sur les utilisateurs*/
  system ("wc -l /etc/passwd | cut -f1 -d : > users.txt");
  /*On ajoute tous les utilisateurs*/
  system ("cat /etc/passwd | cut -f1 -d : >> users.txt");
  p = fopen ("users.txt", "r");
  /*On extrait le nombre d'utilisateurs et leur nom*/
  fscanf (p, "%i %s", &nb_users, name);

  for (i = 0; i <= nb_users; i++) {
    fscanf (p, "%s", name);
    /*On utilise la fonction pour avoir les infos d'un seul
         utilisateur*/
    info_user (name);
    printf ("************\n");
  }
}
```

Exercice 16 Comment se fait-il que vous ne trouviez pas votre login dans les utilisateurs systèmes alors que votre programme est capable d'afficher les informations relatives à votre login ?

Le fichier /etc/passwd ne contient que les utilisateurs locaux (et principalement les login des utilisateurs systèmes). En fait, les machines accèdent aux login à distance sur un serveur ldap. Pour les récupérer, il faudrait scripter en utilisant des outils comme ldapsearch qui permettent d'interroger un serveur ldap.