TP 1 : Processus Unix
Systèmes d'Exploitation – SR1

11-09-2023

Processus Unix : interface de programmation POSIX

Conseils pour compiler les programmes C

Pour limiter le nombre d'erreurs à l'exécution, vous pouvez vous inspirer de l'exemple suivant :

```
gcc -std=c11 -pedantic -Wall -Werror monprog.c -o monproggcc
```

dans lequel "`-std=c11 -pedantic`" demande à `gcc` de vérifier que le programme respecte rigoureusement la norme ISO/CEI 9899 :2011 (anciennement ANSI), "`-Wall`" demande à `gcc` d'être plus scrupuleux dans ses vérifications et d'afficher des messages d'avertissement (*warnings*) dès qu'il a un doute et "`-Werror`" demande à `gcc` de considérer les *warnings* comme des erreurs et donc de ne pas produire le fichier exécutable s'il y a des *warnings*.

Pour éviter de devoir taper toutes ces options à chaque compilation, vous pouvez utiliser la commande `make` avec le fichier de définition `Makefile` dont vous trouverez un exemple sur Moodle. Après avoir recopié le fichier `Makefile` dans le répertoire de vos fichiers `C`, il suffit de taper `make monprog` pour compiler le fichier `monprog.c`

Exercice 1

1. écrire une fonction qui affiche toutes les informations du processus courant (cf. Cours 01).
2. écrire un programme permettant de tester cette fonction.
3. écrire une nouvelle version de ce programme de telle sorte que :
   → il crée un processus fils ;
   → il affiche les informations (en utilisant la fonction précédente) concernant le père et le fils ;
   → le père et le fils affichent un message juste avant de se terminer en indiquant leur code de retour ;
   → le père attend, avant de se terminer, la terminaison du fils et affiche le code de retour du fils. Cela change-t'il quelque chose ?

Exercice 2

écrire un programme qui lancer l'exécution des commandes suivante :
`ls -al`
`date`
en respectant la séquentialité des commandes :
→ un processus fils réalisera la commande `ls -al` ;
→ le processus père effectuera l'appel à la commande `date` ;
→ dans premier temps le processus père n'attend pas la fin du fils pour lancer sa commande ;
→ modifiez le programme pour que le processus père attende la fin du fils pour lancer sa commande.

Exercice 3 ☆

1. écrire un programme permettant de lancer l'exécution des commandes qui lui sont passées en paramètres. Pour chaque commande, le programme doit :
   → créer un processus fils qui doit lancer l'exécution de la commande grâce à la fonction `execvp` ;
   → attendre que ce fils soit terminé ;
   → passer à la commande suivante.
   Des messages doivent être affichés à l'écran afin de suivre l'exécution du programme (voir exemple ci-dessous). Chaque message doit être précédé de l'identificateur du processus produisant cet affichage. Voici un exemple d'exécution dans lequel `lancer` est le nom de la version exécutable du programme demandé :

```
> lancer "sleep 5" pwd date
[1207] J'ai délégué sleep 5 à 1208. J'attends sa fin...
[1208] Je lance sleep 5 :
[1207] 1208 terminé.
[1207] J'ai délégué pwd à 1209. J'attends sa fin...
[1209] Je lance pwd :
/users/linfg/linfg0
[1207] 1209 terminé.
[1207] J'ai délégué date à 1210. J'attends sa fin...
[1210] Je lance date :
Thu Jan  7 19:43:35 MEST 2010
[1207] 1210 terminé.
[1207] J'ai fini.
>
```

La syntaxe de la fonction `execvp` (famille `exec`) est la suivante :

```
#include <unistd.h>
int execvp(const char *nom_fichier, char *const argv[])execvp
```

où argv doit être un tableau de pointeurs de caractères semblable à celui qui est utilisé en paramètre de la fonction `main`, mais avec `NULL` comme dernier élément.

Afin de construire un tel tableau correspondant à une commande stockée sous la forme d'une chaîne de caractères, vous pouvez utiliser la fonction suivante (code source sur Moodle) :

```
#define NBMOTSMAX 20
/* Construction d'un tableau de pointeurs vers le début des mots d'une chaîne
 * de caractères en vue de l'utilisation de la fonction execvp.
 * Retourne le nombre de mots trouvés.
 */
int Decoupe(char Chaine[], char *pMots[])
{
  char *p;
  int NbMots=0;

  p=Chaine; /* On commence par le début */
  /* Tant que la fin de la chaîne n'est pas atteinte et qu'on ne déborde pas */
```

```
  while ((*p)!='\0' && NbMots<NBMOTSMAX)
  {
    while ((*p)==' ' && (*p)!='\0') p++; /* Recherche du début du mot */
    if ((*p)=='\0') break; /* Fin de chaine atteinte */
    pMots[NbMots++]=p;     /* Rangement de l'adresse du 1er caractère du mot */
    while ((*p)!=' ' && (*p)!='\0') p++; /* Recherche de la fin du mot */
    if ((*p)=='\0') break; /* Fin de chaîne atteinte */
    *p='\0';               /* Marquage de la fin du mot */
    p++;                   /* Passage au caractère suivant */
  }
  pMots[NbMots]=NULL;      /* Dernière adresse */
  return NbMots;
}
```

Par exemple, si la commande est stockée dans le tableau de caractères `Chaine` sous la forme d'une chaîne de caractère (donc terminée par le caractère `'\0'`), alors il suffit de déclarer un tableau de pointeurs de caractères :
`char *pMots[NBMOTSMAX+1];`
d'appeler la fonction :
`Decoupe(Chaine,pMots);`
et de lancer l'exécution avec :
`execvp(pMots[0],pMots);`

2. écrire une nouvelle version du programme précédent dans laquelle les processus fils sont tous créés (le père n'attend pas la terminaison d'un fils pour créer le suivant) et, ensuite, le père attend la terminaison de chacun de ses fils.