TP6 d'administration Unix

F. Butelle M. Mayero

2015

Conseil : rédigez un compte rendu de TP et consultez le polycopié au besoin.

1 Gestion de processus en C

1.1 Exemple du cours

1. Taper le programme suivant d'exemple d'utilisation de fork :

```c
/*  prog_fork.c
 *  Exemple utilisation primitive fork() sous UNIX
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    int pid; // PID du processus fils
    int statut=0;

    pid = fork();
    switch (pid) {
    case -1:
        printf("Erreur: echec du fork()\n");
        exit(1);
        break;
    case 0:  // PROCESSUS FILS
        printf("Processus fils : pid = %d\n", getpid() );
        sleep(10); // attends 10 s avant de mourrir
        exit(3);   // fin du processus fils: ici code d'erreur!
        break;
    default: // PROCESSUS PERE
        printf("Ici le pere (pid=%d), pid du fils=%d\n",
               getpid(), pid);
        wait(&statut); // attente de la fin du fils
        printf("Fin du pere. Le fils a renvoye %d\n",
               WEXITSTATUS(statut));
    }
    return(EXIT_SUCCESS);
}
```

2. Compilez-le, testez-le, utilisez `ps auf` pour voir l'arborescence des processus. Commande pour compiler un programme C propre :
`gcc -Wall -Werror -o prog_fork prog_fork.c`

1.2 Exemple d'utilisation de fork, wait, pipe en C

En utilisant google, ou comment se former à des choses que l'on ne connaît pas...

1. Ecrire un programme C qui crée un fils qui envoie un message “Hello” à son père par un tube créé par l'appel système `pipe`.
2. Ecrire un programme C qui calcule x^n de façon récursive en créant un fils qui va calculer $x^{n/2}$ et le reste par le père. En clair vous devrez avoir une fonction calcul(x,n) qui créera le fils et pourra s'appeler elle-même comme suit :

   Algorithme de calcul(x,n) :

```
Si n = 0 alors retourne 1
Si n = 1 alors retourne x
creation tube (voir pipe)
p ← partie entiere de n/2
Si fork = 0
  alors // fils
     fermer le tube en lecture (voir close)
     a ← calcul(x,p)
     sleep(2)  // pour voir la table des processus evoluer
     ecrire a dans le tube // voir write
     exit(0)   // ferme aussi le tube en ecriture
  sinon
     fermer le tube en ecriture
     lire b la valeur contenue dans le tube
     // voir read: bloque tant qu'il n'y a rien a lire
     attendre la fin du fils
fin
retourner(b*calcul(x,n-p))
```

3. Ecrire le même programme mais en créant deux processus fils qui chacun calculent $x^{n/2}$, le père calculant le produit et ainsi de suite récursivement.

2 make et Makefile

L'objectif est de montrer l'intérêt de la gestion de projet comportant plusieurs fichiers à compiler/traiter pour obtenir un exécutable/résultat final.

La commande make lit le contenu du fichier Makefile du répertoire courant. Ce fichier respecte la syntaxe suivante :

Listing 1 – Makefile

```
#Commentaire
Variable1=valeur

objectif1 :  source1 source2 ...
	action1 $(Variable)
	action2
	action3...
...
```

Si un des fichiers sources a une date de dernière modification ultérieure à celle de l'objectif, `make objectif` lancera les actions prévues. Cela est souvent utilisé pour la compilation séparée de fichiers source pour un grand projet comportant plusieurs milliers de lignes de code.

Nous allons l'illustrer avec des exemples très simples. Soit le fichier print.c :

Listing 2 – print.c

```
#include <stdio.h>
void print() {
        printf("Bonjour\n");
}
```

Soit le fichier main.c :

Listing 3 – main.c

```
/* prototype de la fonction */
void print();

int main() {
        print();
        return 0;
}
```

La compilation séparée d'un fichier se fait par `gcc -Wall -c fichier.c` cela produit un fichier `fichier.o`. L'exécutable final est produit par «l'édition de liens» que fait automatiquement gcc quand on ne lui donne pas l'option -c : `gcc -Wall -o monexec print.o main.o`

Ecrire le fichier Makefile permettant de générer l'exécutable final avec recompilation séparée des fichiers sources.