TP1 : fork, exec et signaux

RS : Réseaux et Systèmes

Deuxième année

Rappel : Il est indispensable d'ajouter les drapeaux `-Wall -Wformat` pour profiter des vérifications de base à la compilation. En effet, `gcc` accepte par défaut du code manifestement faux sans le moindre avertissement, ce qui mène à des erreurs difficiles à trouver.

★ Exercice 1: Appel système `fork`.

▷ **Question 1:** Écrire un programme qui crée 10 processus fils. Chacun d'entre eux devra afficher dix fois d'affilé son numéro d'ordre entre 0 et 9. Vérifiez que votre programme affiche 100 caractères.

Réponse

```
#include <stdio.h>

int main() {
  int i, j;

  /* boucle pour créer les 10 processus fils */
  for (i=0; i < 10; i++) {
    switch (fork()) {
      case -1: fprintf(stderr, "Erreur dans %d\n", getpid());
               perror("fork");
               exit(1);

      case 0:  /* On est dans un fils */
               for (j = 0; j < 10; j++) {
                        printf("%d",i);
                 fflush(stdout);
               }
               /* Ne pas oublier de sortir sinon on cree fact(10) processus */
               exit(0);

      default: /* On est dans le pere; ne rien faire */ ;
    }
  }

  /* 2e boucle, attendre les 10 fils */
  for (i=0; i < 10; i++) {
      wait(NULL);
  }
  exit(0);
}
```

Pour compter les caractères, faire utiliser wc. Utiliser une variable ne fonctionne pas, car elle n'est pas partagée avec les fils.

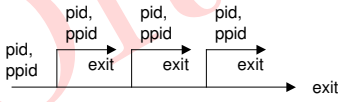
On peut remarquer :

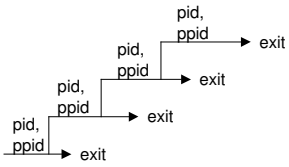
- Que les processus ne sont pas forcément exécutés dans l'ordre
- Que l'exécution des processus est parfois entrelacée
- Que le processus père termine avant les processus fils : l'invite du shell est affichée au milieu de la sortie des fils

Fin réponse

▷ **Question 2:** Reprise de la Question 4 du TD1

On considère les deux structures de filiation (chaîne et arbre) représentées ci-après. Écrire un programme qui réalise une chaîne de `n` processus, où `n` est passé en paramètre de l'exécution de la commande (par exemple, n = 3 sur la figure ci-dessus). Faire imprimer le numéro de chaque processus et celui de son père. Même question avec la structure en arbre.





Dans le cas de la filiation en arbre, il est tout à fait normal que l'invite du shell apparaisse au milieu des affichages des fils. C'est parce que le shell l'affiche dès que le processus qu'il a lancé lui-même termine. Et dans cette filiation, le père termine avant le fils. On peut continuer à utiliser le shell comme si de rien n'était, ou tout effacer d'un Ctrl-L.

De même, il est normal que le PID indiqué pour le père soit parfois 1 dans la filiation en arbre. Cela se produit quand le père a quitté avant que son fils n'affiche son ppid. Dans ce cas, le fils est déjà rataché au processus init, de pid 1.

Réponse

```
#include <stdlib.h> /* exit, atoi */
#include <stdio.h>  /* printf */
#include <unistd.h> /* getpid, fork */
#include <wait.h>   /* wait */

void arbre(int n) {
  int i;
  printf("Lance %d processus en arbre\n\n",n);
  printf("proc %d fils de %d (shell)\n",getpid(),getppid());
  for (i=0; i<n; i++) {
    if (fork() == 0) { /* fils */
      printf("proc %d fils de %d\n",getpid(),getppid());
    } else {
      // wait(NULL); pas de wait, ou les pères quitent pas assez vite
      exit(0);
    }
  }
}

void chaine(int n) {
  int i;
  printf("Lance %d processus en chaine\n\n",n);
  printf("proc %d fils de %d (shell)\n",getpid(),getppid());

  for (i=0; i<n; i++) {
    if (fork() == 0) { /* fils */
      printf("proc %d fils de %d\n",getpid(),getppid());
      exit(0);
    }
  }

  for (i=0; i<n; i++)
    wait(NULL);
}

int main(int argc, char *argv[ ]) {
  if (argc<2) {
    printf("Usage:\n%s 1 <n> pour lancer <n> processus en arbre\n%s 2 <n> pour lancer <n> processus en chaine\n",
           argv[0], argv[0]);
    exit(1);
  }
  switch(atoi(argv[1])) {
  case 1: arbre(atoi(argv[2]));  break;
  case 2: chaine(atoi(argv[2])); break;
  }
  return 0;
}
```

Fin réponse

★ Exercice 2: Appel système exec.

▷ **Question 1:** Reprise de la Question 8 du TD1 Écrire un programme `doit` qui exécute une commande Unix que l'on lui passe en paramètre. *Exemple :* `doit ls -lt /`

Réponse

Réponse utilisant argv en place :

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

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

   if (argc < 2) {
      fprintf(stderr,"Necessite au moins un argument\n");
      exit(1);
   }

   execvp(argv[1],&argv[1]);
   return 0;
}
```

Fin réponse

▷ **Question 2:** Écrire un programme `multido` qui exécute au plus cinq fois (dans des processus séparés) une commande Unix que l'on lui passe en paramètre. Si l'une des exécutions provoque une erreur, il ne faut pas réaliser les exécutions suivantes.
Pour tester votre travail, vous pouvez utiliser la commande `./multido mkdir toto` car `mkdir toto` renvoit un code d'erreur si le répertoire à créer existe déjà.

Réponse

```
#include <stdio.h>
#include <errno.h>
#include <wait.h>
```

```
int main(int argc,char *argv[]) {
    int i;
    int status;

    if (argc < 2) {
        printf("Necessite au moins un argument\n");
        exit(1);
    }
    for (i=0; i<5; i++) {
        switch (fork()) {
            case -1:
                perror("pb de fork");
                exit(1);
            case 0:
                execvp(argv[1],&argv[1]);

            default:
                //if (waitpid(-1, &status, 0) > 0) {
                if (wait(&status) > 0) {
                    printf("status: %d %d %d\n", status, WIFEXITED(status), WEXITSTATUS(status));
                    if (WIFEXITED(status) != 0 && WEXITSTATUS(status) != 0) {
                        printf("Child returned an error (code:%d). Balling out\n",
                                WEXITSTATUS(status));
                        exit(1);
                    }
                }
        }
    }
    return 0;
}
```

Exemple de programme qui plante (pour tester) :
— La première fois : `cd titi` ou bien `false` qui renvoie toujours faux
— La deuxième fois : `mkdir toto`, ou bien `rm truc` si on crèe le fichier truc avant de la lançer.

Fin réponse

★ **Exercice 3: Signaux.**

Rappel : nous utilisons l'interface POSIX. Évitez donc les tutos Internet qui utilisent la fonction `signal()`.

▷ **Question 1:** Écrire un programme ne se terminant qu'au cinquième Ctrl-C.

Réponse

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

void Recuperation(int sig) {
  static compt = 0;

  if (++compt == 5) {
    printf("J'en ai assez!!\n");
    exit(0);
  } else {
    printf("OK je recupere le signal SIGINT\n");
  }
}

int main() {
  struct sigaction nvt, old;
  memset(&nvt,0,sizeof(nvt)); /* memset necessaire pour flags=0 */

  nvt.sa_handler = Recuperation;

  sigaction(SIGINT,  &nvt, &old);

  printf("Taper 5 ^C pour arreter le programme\n");
  while(1);
}
```

Fin réponse

▷ **Question 2:** Écrire un programme créant deux fils, en envoyant le signal SIGUSR1 à son fils cadet. À la réception de ce signal, le fils cadet devra envoyer le signal SIGUSR2 au fils ainé (qui provoque sa terminaison) avant de s'arrêter.
Indication : il est très difficile d'interchanger les rôles des deux fils.

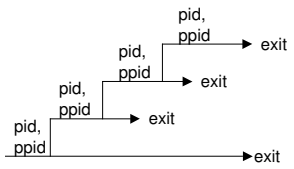
Réponse

```
#include <signal.h>

int aine, cadet;

void cadet_sigusr1() {
  printf("Reception du signal SIGUSR1 dans %d\n", getpid());
  kill(aine, SIGUSR2);
  exit(0);
}
```

```
void aine_sigusr2() {
  printf("Reception du signal SIGUSR2 dans %d\n", getpid());
  exit(0);
}

int main(int argc, char **argv) {
  /* Lancer le fils aine */
  if ((aine=fork()) == 0) {
    /* fils */
    struct sigaction nvt, old;
    memset(&nvt,0,sizeof(nvt));
    nvt.sa_handler = aine_sigusr2;
    sigaction(SIGUSR2,  &nvt, &old);

    pause();
  }

  /* Lancer le fils cadet */
  if ((cadet=fork()) == 0) {
    struct sigaction nvt, old;
    memset(&nvt,0,sizeof(nvt));
    nvt.sa_handler = cadet_sigusr1;
    sigaction(SIGUSR1,  &nvt, &old);

    pause();
  }

  /* On est dans le pere */
  sleep(1); /* pour etre sur que les signaux sont bien detournes */
  printf("Aine=%d  Cadet=%d\n", aine, cadet);
  kill(cadet, SIGUSR1);
  sleep(2); /* pour attendre les affichages */
  return 0;
}
```

Fin réponse

▷ **Question 3:** Seconde reprise de la Question 4 du TD1.
Modifiez votre filiation en arbre afin que le premier père attende le dernier fils, mais que les pères intermédiaires quittent au plus tôt.
Indication : cette question est placée dans un exercice nommé «Signaux».



Réponse

```
#include <stdlib.h> /* exit, atoi */
#include <stdio.h>  /* printf */
#include <unistd.h> /* getpid, fork */
#include <wait.h>   /* wait */

void gestionnaire_sigusr1() {
  printf("Reception du signal SIGUSR1 dans %d\n", getpid());
}

void arbre(int n) {
  int i;
  printf("Lance %d processus en arbre\n\n",n);
  printf("proc %d fils de %d (shell)\n",getpid(),getppid());
  int pid_pere = getpid();
  for (i=0; i<n; i++) {
    if (fork() == 0) { /* fils */
      printf("proc %d fils de %d\n",getpid(),getppid());
      if (i==n-1) {
        kill(pid_pere,SIGUSR1);
        printf("Le dernier fils part\n");
      }
    } else {
      if (i==0) {// premier père
        pause();
        printf("Le premier père termine\n");
      }
      exit(0);
    }
  }
}

int main(int argc, char *argv[ ]) {
  if (argc<2) {
    printf("Usage:\n%s <n> pour lancer <n> processus en arbre\n", argv[0]);
    exit(1);
  }
  struct sigaction nvt, old;
  memset(&nvt,0,sizeof(nvt));
  nvt.sa_handler = gestionnaire_sigusr1;
  sigaction(SIGUSR1,  &nvt, &old);
  arbre(atoi(argv[1]));
  return 0;
}
```

Fin réponse

★ Exercice 4: Réimplementer if.

▷ **Question 1:** Écrire une version simplifiée du programme `if` que vous nommerez `si`.

Testez votre travail avec la commande suivante : `./si test -e toto alors echo present sinon echo absent`

Réponse

si.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char **argv) {
  int alors_pos = 0;
  int sinon_pos = 0;
  int i = 0;
  int status;

  while(argv[++i]!=NULL)
    if (!strcmp(argv[i],"alors"))
      alors_pos = i;
    else if(!strcmp(argv[i],"sinon"))
      sinon_pos = i;

  argv[alors_pos] = NULL;

  if (sinon_pos)
    argv[sinon_pos] = NULL;

  if(!(alors_pos)) {
    printf("Syntaxe : ./si expr alors expr [sinon expr]\n");
    return -1;
  }

  if(fork()==0) {
    execvp(argv[1],&argv[1]);
    perror("execvp cassé !");
    return -1;
  }

  wait(&status);

  if(WIFEXITED(status)) {
    if(!WEXITSTATUS(status)) {
      if(!sinon_pos)
        sinon_pos=argc;

      execvp(argv[alors_pos+1],&argv[alors_pos+1]);

      perror("execvp cassé !");
      return -1;

    } else if(sinon_pos) {

      execvp(argv[sinon_pos+1],&argv[sinon_pos+1]);

      perror("execvp cassé !");
      return -1;
    }
    // test "si" faux, pas de "sinon"
    return 0;

  } else {
    // test "si" pas sorti par exit
    return -1;
  }
}
```

Fin réponse