Système d'exploitation MIAGE 2 : Vie et mort des processus

Licence MIAGE — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 6 — 2008-09

Cette feuille de travaux dirigés est fortement inspirée du cours de J.-B. Yunès à Paris-VII.

1 Manipulation de clonage

Exercice 1 — Effet de l'instruction fork.

Soit l'instruction :

```
for (i=0; i<n; i++) fork();
```

1. Dessinez l'arbre des processus créés lorsque $n = 1$, $n = 2$ et $n = 3$.
2. Combien de processus sont créés dans le cas général ? Justifiez.

Exercice 2 — Utilisation de fork.

Écrire pour chaque question une fonction de prototype

```
void creer_fils(int n,void (*g)(void));
```

1. permettant récursivement de créer n processus fils
2. permettant itérativement de créer n processus fils
3. permettant récursivement de créer n processus en lignée (fils, petit fils, etc.)
4. permettant itérativement de créer n processus en lignée (fils, petit fils, etc.)

exécutant tous la fonction g passée en paramètre (pour faire simple, on suppose que le père exécute lui aussi g). Par exemple, les fonctions que vous allez écrire peuvent compléter le code suivant :

```
#include <sys/types.h>
#include <unistd.h>

void infini(void){
  while(1) ;
}

void creer_fils(int n,void (*g)(void));

int main(int argc, char *argv[]){
  creer_fils(argc-1,&infini) ;
  while(1) ;
  return 0 ;
}
```

Exercice 3 — Arbre de parenté entre processus.

Considérons le programme :

```
fork(); if (fork()) fork();
fork(); if (fork()) fork();
```

En supposant qu'aucun appel n'échoue, combien de processus sont créés à l'exécution du programme ci-dessus ? Dessinez l'arbre de création des processus.

Quelques particularités des relations père-fils chez les processus.

Exercice 4 — Orphelin et zombie.

1. Qu'est-ce qu'un processus orphelin ?
2. Écrire un programme conduisant à la création d'un processus orphelin.
3. Qu'est qu'un processus zombie ?
4. Écrire un programme dont l'exécution conduit à la création d'un processus zombie ?
5. Comment fait-on disparaître un tel processus ?

2 Manipulation de mutation

Exercice 5 — Synthèse : clonage — mutation.

Considérons les programmes suivants :

```
/* Programme 1 */
int main() {
  main();
}
```

```
/* Programme 2 */
int main() {
  while (1) fork();
}
```

```
/* Programme 3 */
int main(int argc,char *argv[]){
  execvp(argv[0],argv);
}
```

```
/* Programme 4 */
int main(int argc,char *argv[]){
  execvp(argv[1],argv);
}
```

Décrivez le comportement de chacun de ces programmes.

Exercice 6 — Compréhension d'un code.

Soit le programme test.c suivant :

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

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

  if (argc==1) exit(EXIT_SUCCESS);
  n = atoi(argv[argc-1]); /* atoi  - convert a string to an integer */
  for (i=0; i<n; i++) {
    switch(fork()) {
    case 0:
      argv[--argc] = NULL;
      execvp(argv[0],argv);
      exit(EXIT_FAILURE);
    case -1:
      exit(EXIT_FAILURE);
    }
  }
  ret = EXIT_SUCCESS;
  for (i=0; i<n; i++) {
```

```
    if (wait(&status)!=(pid_t)-1) {
      if (!(WIFEXITED(status) && WEXITSTATUS(status)==EXIT_SUCCESS))
        ret = EXIT_FAILURE;
    }
  }
  exit(ret);
}
```

Après compilation de ce source, on suppose obtenir un exécutable `test`.

La macro `WIFEXITED(status)` retourne vrai quand le processus se termine par un appel à `exit`. Si c'est le cas, la macro `WEXITSTATUS(status)` retourne le code de sortie du fils.

Questions.

1. Combien de processus sont créés lors d'une exécution par test sans plus de paramètre ? par `test 3` ? par `test 2 4` ? par `test 2 1 5` ? pour chaque exécution dessinez l'arbre généalogique des processus.
2. Dans quel ordre les processus se terminent-ils ?
3. Si l'on supprime la seconde boucle for() et son contenu. Quelle différence cela fait-il ?
4. Si l'on remplace la seconde boucle for() et son contenu par un simple appel à wait(NULL) ?

Exercice 7 — Exemple de récursion sur la fonction principale.

Soit le programme suivant :

```
#define N une_certaine_valeur
static int n=N;
int main(int argc,char *argv[]) {
  int i;
  for (i=0; i<n; i++) if (fork()==0) { n--; main(argc,argv[]); }
  while (wait(NULL)!=-1);
  exit(EXIT_SUCCESS);
}
```

Questions.

1. Combien de processus sont-ils créés lors d'une exécution ?
2. Dans quel ordre se terminent les processus créés.
3. Que se passe-t-il si l'on remplace l'appel à main(argc,argv[]) par un appel à execvp(argv[0],argv) ?