Consolidation de Janvier 2020 (1h)

RS : Réseaux et Systèmes

Deuxième année

★ Questions de cours.(5pts)

▷ **Question 1:** Qu'est-ce qu'on appel système ?

▷ **Question 2:** Expliquez précisément ce que fait l'appel système `exec`.

▷ **Question 3:** Qu'est-ce qu'un interblocage ?

▷ **Question 4:** Comment prévenir les interblocages ? Donnez et expliquez trois méthodes.

★ Exercice 1: Utilisation de `fork`, `exec`, `wait`, `waitpid`, `pipe`, `dup`, `dup2` (6pts)

Donner le code C (sans utiliser la fonction `system`) correspondant à ce que fait un *shell* lorsqu'on tape les lignes de commandes suivantes. Afin d'obtenir un code lisible et court, vous êtes dispensés des tests d'erreur des appels systèmes.

▷ **Question 1:** `prog1 || prog2`

(Il faut donc : lancer un programme *prog1*, attendre sa fin, récupérer son code de retour, et si ce code de retour est différent de 0, lancer un programme *prog2* et attendre sa fin)

▷ **Question 2:** `prog1 | prog2`

(Il faut donc : lancer deux programmes *prog1* et *prog2*, en liant la sortie standard de *prog1* à l'entrée de *prog2*)

Quelques fonctions utiles

```c
pid_t fork(void);
int execlp(const char *file, const char *arg, ...);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
WIFEXITED(status) -- returns true if the child terminated normally
WEXITSTATUS(status) -- returns the exit status of the child
int pipe(int pipefd[2]);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

★ Exercice 2: Clonage de processus (4pts)

▷ **Question 1:** Combien de lignes «hello !» imprime chacun des programmes suivants ?

Programme 1 :

```c
int main() {
    fork();
    fork();
    fork();
    printf("hello!\n");
    exit(0);
}
```

Programme 2 :

```c
int main() {
  int i;

  for (i=0; i<2; i++)
    fork();
  printf("hello!\n");
  exit(0);
}
```

Programme 3 :

```c
void doit() {
  fork();
  fork();
  printf("hello!\n");
}
int main() {
  doit();
  printf("hello!\n");
  exit(0);
}
```

Programme 4 :

```c
int main() {
  if (fork())
    fork();
  printf("hello!\n");
  exit(0);
}
```

Tourner la page

★ **Exercice 3: Savoir reconnaître les schémas de synchronisation classiques et utiliser les sémaphores (5 pts)** (inspiré d'un exercice de Cédric Bastoul)

Un étudiant qui se spécialise en anthropologie et accessoirement en informatique s'est embarqué dans un projet de recherche pour voir s'il était possible d'enseigner les interblocages aux babouins d'Afrique. Il repère un profond canyon et y jette une corde au travers, de sorte qu'un babouin puisse le traverser à bouts de bras.

Chaque babouin exécute l'algorithme suivant :

- RÉPÉTER à l'infini
  - Se présenter devant la corde
  - Attraper la corde
  - Traverser le canyon
  - Continuer à pieds
- FIN RÉPÉTER

L'installation ayant du succès, les babouins se rendent vite compte que si deux d'entre eux commencent à traverser dans les deux sens opposés, ils se retrouvent bloqués au milieu du canyon.

▷ **Question 1:** Proposez une solution (modifiez l'algorithme ci-dessus) à base de sémaphore garantissant que seul un babouin à la fois pourra accéder à la corde et traverser.

▷ **Question 2:** Votre solution se rapproche d'un schéma de synchronisation classique. Lequel ?

Cette solution est loin d'être idéale. Les babouins se rendent rapidement compte que plusieurs babouins pourraient traverser simultanément, à condition qu'ils traversent tous dans le même sens, ce qui serait bien plus efficace.

▷ **Question 3:** Modifiez l'algorithme ci-dessus. Il est conseillé d'écrire deux algorithmes : celui pour les babouins qui traversent du Nord vers le Sud, et celui pour ceux qui font l'inverse.

▷ **Question 4:** Votre solution se rapproche d'un schéma de synchronisation classique. Lequel ?

▷ **Question 5:** Que pouvez-vous dire d'un problème de famine éventuel avec votre solution ? (Il n'est pas demandé de proposer une solution sans famine, juste d'expliquer dans quel scénario une famine pourrait peut-être se produire)