Examen du 10 novembre 2006

RS : Réseaux et Systèmes

Deuxième année

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction. Barème approximatif. Il est conseillé de lire l'intégralité du sujet avant de commencer.

Tous documents interdits

Questions de cours (7pt)

▷ **Question 1.** Définissez les deux fonctions principales des systèmes d'exploitation.

▷ **Question 2. (1pt)** Représentez graphiquement l'espace mémoire d'un processus, et détaillez le rôle des différentes segments.

▷ **Question 3.** À quoi servent respectivement les appels système `dup2` et `pipe` ?

▷ **Question 4.** À quoi servent respectivement les appels système `fork` et `exec` ?

▷ **Question 5.** Donnez deux signaux existant sous Unix. Dans quelles conditions sont-ils émis ? Quel est leur effet par défaut sur le processus les recevant ?

▷ **Question 6.** Qu'est ce qu'une section critique ?

▷ **Question 7.** Comment prévenir les interblocages ? Donnez et expliquez trois méthodes.

▷ **Question 8.** Qu'est ce qu'une famine en informatique ? Dans quelles conditions cela se produit-il ?

▷ **Question 9.** Qu'est ce qu'une lecture tronquée ? Dans quelles conditions cela se produit-il ?

▷ **Question 10.** Que font les commandes `jobs`, `fg` et `bg` ?

▷ **Question 11.** Que font `bash` et `gcc` ? Quels sont les points communs et les différences entre leurs approches ?

▷ **Question 12.** Définissez symboles forts et faibles et donnez des exemples.

▷ **Question 13.** Discutez les avantages et inconvénients des processus légers par rapport aux processus lourds.

Exercice 1 : Communications et synchronisations (2pt)

Parmi les paradigmes de synchronisation de processus cités ci-dessous, choisissiez deux solutions implémentables uniquement avec les tubes de communication inter-processes (donc, sans sémaphore, moniteur, variable de condition, mémoire partagée ou autre). Expliquez l'implémentation envisagée (5 lignes maximum par paradigme retenu).

1. Exclusion mutuelle
2. Cohorte
3. Passage de témoins et envoi de signaux
4. Producteurs/consommateurs
5. Lecteurs/rédacteurs

Exercice 2 : Traitement des signaux (2pt)

▷ **Question 1.** Que fait le programme ci-contre ?

▷ **Question 2.** Modifiez ce programme de façon à ce qu'il survive à un <CTRL-C> (et à un seul).

```
1  #include <signal.h>
2  void gestionnaire(int sig) {
3    printf("ctrl-z pressé !\n");
4    exit(0);
5  }
6  int main() {
7    struct sigaction old,new;
8    memset(&new,0,sizeof(new));
9    new.sa_handler=gestionnaire;
10   sigaction(SIGTSTP, &new, &old);
11   pause();
12 }
```

Exercice 3 : Clonage de processus (2pt)

Combien de lignes «plop !» imprime chacun des programmes suivants ?

Question 1

```
1  void toto() {
2      fork();
3      printf("plop!\n");
4      fork();
5      fork();
6  }
7  int main() {
8      toto();
9      printf("plop!\n");
10     exit(0);
11 }
```

Question 2

```
1  int main() {
2      int i=0;
3
4      while (i<3) {
5          fork();
6          i++;
7      }
8      printf("plop!\n");
9      exit(0);
10 }
```

Question 3

```
1  int main() {
2      if (fork() && fork())
3         printf("plop!\n");
4      printf("plop!\n");
5      exit(0);
6  }
```

Exercice 4 : Relecture de code (3pt)

Pour chacun des extraits de code suivants, listez les (nombreuses) anomalies (erreur de programmation, erreur d'utilisation ou bien code non optimal). Pour chaque anomalie trouvée, donnez sa localisation et une explication. Note : chaque listing est supposé être un programme complet, rien n'est défini par ailleurs.

Question 1

```
1  void do_mail() {
2    int p[2];
3    pipe(p);
4
5    if (fork()) {
6      dup2(p[1],1);
7      write(p[1],"coucou", 7);
8      close(p[0]); close(p[1]);
9    } else {
10     dup2(p[0], 0);
11     close(p[0]); close(p[1]);
12     execlp("mail","jo@foo.com");
13   }
14   wait(NULL);
15   printf("Message parti\n");
16 }
17 main() {
18   while (1) {
19     do_mail();
20     sleep(5);
21 } }
```

Question 2

```
1  void *hello( void *id ) {
2    printf("%d: hello world \n", (char *) id);
3    pthread_exit(0);
4  }
5
6  int main (int argc, char *argv[ ]) {
7    int th[3];
8    int i;
9
10   for (i=0;i<3;i++) {
11     printf("Crée thread %d\n",i);
12     pthread_create(&th[i], NULL, hello, (void *)&i);
13   }
14
15   return 0;
16 }
```

Problème : Robotique dans les étoiles (4pt)

Dans le cadre des efforts actuels pour apporter plus de paix et de démocratie aux peuples opprimés, la confédération intergalactique souhaite renforcer l'utilisation de droïdes de combat. L'objectif est de les utiliser dans les missions dangereuses pour éviter les pertes humaines, mal acceptées par les électeurs.

Une escouade classique est composée d'un robot lourd d'assaut (nommé *freedom*) et de deux robots plus légers (nommés *democracy*). La mission principale de ces escouades consiste à arraisonner les vaisseaux rebelles. Le scénario prévu est que le robot *freedom* neutralise les moyens de défense du vaisseau, tandis que les robots *democracy* capture les membres de l'équipage. Étant donné le faible équipement militaire des robots *democracy*, il est important que ces derniers n'entrent pas avant d'avoir reçu la certitude que *freedom* a accompli sa tâche.

▷ **Question 1.** De quel schéma de synchronisation ce problème se rapproche-t-il ?

▷ **Question 2.** Implémentez les processus `freedom` et `democracy` grâce à des sémaphores.

En 2542, la situation s'aggrave pour la confédération : les rebelles ont doté leurs vaisseaux d'un deuxième niveau de sécurité composé d'un détecteur et d'un canon. *freedom* ne peut désactiver que le premier niveau à lui seul, mais pas le canon du second niveau tant que le détecteur fonctionne. De plus, l'action des deux robots *democracy* est nécessaire pour désactiver le détecteur du second niveau.

Le scénario de capture d'un vaisseau rebelle est donc le suivant : *freedom* entre dans le vaisseau et détruit le générateur du champ de force constituant le premier niveau de sécurité. Les robots *democracy* peuvent alors s'introduire dans la cabine. Chacun des deux doit alors couper un fil donné pour désactiver le détecteur de second niveau. Une fois ceci fait, `freedom` peut s'approcher du canon du second niveau et le détruire, assurant ainsi une pleine liberté de mouvement aux robots *democracy* pour capturer l'équipage.

▷ **Question 3.** Discutez les schémas de synchronisation mis en œuvre.

▷ **Question 4.** Implémentez les nouvelles versions de `freedom` et `democracy` grâce à des sémaphores.