TELECOM Nancy

UNIVERSITÉ DE LORRAINE

Examen du 18 octobre 2013 (1h)

RS : Réseaux et Systèmes

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction. Lisez entièrement le sujet avant de commencer calmement.

Documents interdits, à l'exception d'une feuille A4 recto-verso manuscrite.

CORRECTION

★ Exercice 1: Question de cours (6pts) (d'après Raymond Namyst).

▷ **Question 1:** Qu'est ce que l'attente active ?

Réponse

(1pt) Le processus attend qu'une condition devienne réalisée en évaluant en continu si elle l'est. Analogie de l'élève de maternelle qui demande l'attention de la maîtresse en l'appellant en continue : "Maîtresse, Maîtresse, Maîtresse *(ad lib)*".

Fin réponse

▷ **Question 2:** Qu'est ce qu'un i-node (ou i-nœud) ?

Réponse

(1pt) C'est un cluster du disque où sont stockées les meta-données du système de fichiers décrivant un fichier particulier. On y trouve le propriétaire du fichier, les droits d'accès, ainsi que la liste des clusters contenant les données du fichier (si le fichier est trop gros, il y a une indirection).

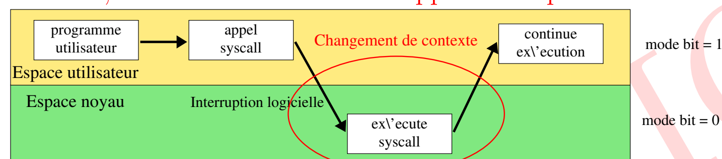
Fin réponse

▷ **Question 3:** Rappelez brièvement le principe de fonctionnement d'un appel système (sur un exemple concret de votre choix). Un petit schéma explicatif est bienvenu.

Réponse

(2pt, dont 1pt pour le schéma) Les processus applicatifs ne peuvent pas effectuer certains actions jugées critiques (telles que des accès direct aux périphériques, interactions avec les autres processus) directement. Les applications demandent au noyau de réaliser ces tâches à leur place en demandant l'exécution d'un appel système.

Par exemple pour écrire sur disque, l'application écrit les données souhaitées et autres paramètres à un emplacement prédéfini de la mémoire puis lance l'interruption `0x80`. Le matériel redonne alors le contrôle au système d'exploitation, qui vérifie la validité des paramètres avant de réaliser la tâche demandée. Une fois que la tâche est terminée, le contrôle est rendu à l'application qui avait fait le syscall.



Fin réponse

▷ **Question 4:** Quel est l'intérêt du mécanisme des appels systèmes dans un système d'exploitation multi-utilisateurs ? Pourquoi ne pas utiliser de simples appels de fonctions normaux à la place ?

Réponse

Cela permet de mettre en place de la préemption sur le CPU (on ne donne au processus applicatif que ce qu'on peut lui reprendre à coup sûr), ainsi que de la médiation sur les appels directs au matériel (on vérifie que les actions demandées sont compatibles avec la politique de sécurité en place). C'est la base de la sécurisation du système.

Fin réponse

▷ **Question 5:** Pourquoi est-on sûrs qu'aucune application ne peut contourner le mécanisme des appels systèmes par aucun moyen que ce soit une fois que le système d'exploitation a démarré ?

Réponse

Car le système d'exploitation met le matériel en mode non privilégié avant de donner le contrôle au code applicatif. Dans ce mode non-privilégié, le matériel ne fera aucune action qui pourrait compromettre la sécurité du système (accès direct à la mémoire ou au disques, etc). Le seul moyen de regagner le mode privilégié est de passer par un appel système.

Fin réponse

★ **Exercice 2: Relire des synchronisations (7pts)** (d'après Françoise Baude).

Voici ci-dessous une variante d'une synchronisation de type Producteurs / Consommateur.

▷ **Question 1:** Décrire **clairement** quelles sont les synchronisations mises en œuvres lorsqu'on exécute ce pseudo-code avec N producteurs et un unique consommateur. Donnez l'idée d'un scénario possible d'exécution, en indiquant bien quand certaines actions doivent se passer avant d'autres.

Une explication claire, complète et concise est naturellement préférable à un texte long et confus.
Indication : Ce code ne présente aucune condition de compétition ni d'interblocage (sauf erreur).

```
1  Déclaration et initialisation de variables globales
2    tour: variable entière initialisée à 0
3    tampon: tableau d'objets de capacité N où sont stockés les objets produits
4    consomme: sémaphore initialisée à 0
5    mutex: sémaphore initialisée à 1
6
7  Procédure du processus Producteur numéro i {
8    variable locale : objet
9
10   objet = fonction_de_production() // produire un objet, sa nature importe peu ici
11   P(mutex)
12   tampon[tour] <- objet
13   tour <- tour + 1
14   V(mutex)
15   V(consomme)
16 }
17 Procédure du processus Consommateur { // ce processus est unique
18   Répéter N fois {
19     P(consomme)
20   }
21   Pour i prenant les valeurs entre 1 et N {
22     affichage(tampon[i])
23   }
24 }
```

Réponse

Chaque producteur produit un objet. Ensuite, il acquière le mutex afin de ranger cet objet dans la première case libre du tableau partagé (case dénotée par la variable `tour`). Il incrémente également la variable `tour` avant de libérer le mutex et signaler la fin de sa production.

Le consomateur attend que tous les producteurs aient terminé (il exécute N fois `P(consomme)`), puis il affiche toutes les données produites.

Fin réponse

▷ **Question 2:** Discutez les avantages et inconvénients par rapport au modèle classique de producteurs/consommateur.

Réponse

Le principal inconvénient est que le consommateur ne traite pas les objets produits au fur et à mesure, mais attend que le dernier objet soit produit avant de consommer.

Fin réponse

▷ **Question 3:** Que se passe-t-il si on inverse les lignes 10 et 11 du pseudo-code fourni ?

```
10   P(mutex)
11   objet = fonction_de_production() // produire un objet, sa nature importe peu ici
```

Réponse

La production d'objets est séquentialisée (un seul producteur actif à la fois). Cela enlève tout intérêt au schéma producteurs/consommateurs.

Fin réponse

▷ **Question 4:** Que se passe-t-il si on inverse (seulement) les lignes 13 et 14 du pseudo-code fourni ?

```
13   V(mutex)
14   tour <- tour + 1
```

CORRECTION

Réponse

Cela crée une condition de compétition : Il est possible qu'un producteur écrive dans `tampon[tour]` avant que le premier producteur a y avoir écrit n'ait le temps d'incrémenter `tour`.

Fin réponse

▷ **Question 5:** Que se passe-t-il si on inverse (seulement) les lignes 14 et 15 du pseudo-code fourni ?

```
14  V(consomme)
15  V(mutex)
```

Réponse

Inverser des lignes `V()` est sans effet sur le schéma de synchronisation. Ici, le consommateur sera libéré très légèrement avant le prochain producteur, mais cela n'a aucune influence notable.

Fin réponse

★ Exercice 3: Relire des tubes (7pts).

▷ **Question 1:** Dessinez les relations entre les différents processus (sous forme de patates) et tubes (sous forme de liens entre les patates) créés par le programme ci-contre.

▷ **Question 2:** Sachant que la macro `islower()` retourne vrai si le caractère passé en argument est en minuscule et faux si ce caractère est en majuscule, donnez les grandes lignes de ce qui se passe lorsqu'on exécute ce programme de la façon suivante :

```
$ gcc -Wall -o mystere mystere.c
$ echo BonJOUr | ./mystere SaLuT
```

▷ **Question 3:** Raffinez votre réponse en discutant les différents ordres d'affichage possibles avec la commande suivante :

```
$ gcc -Wall -o mystere mystere.c
$ echo abAB | ./mystere abAB
```

▷ **Question 4:** Que se passe-t-il si l'on supprime l'ensemble des lignes 24 à 27 ? Pourquoi ?

```
24  // close(A[1]);
25  // close(B[1]);
26  // wait(NULL);
27  // wait(NULL);
```

▷ **Question 5:** Que se passe-t-il si l'on commente les lignes 24 et 25 de ce programme en restaurant les lignes 26 et 27 ? Pourquoi ?

```
24  // close(A[1]);
25  // close(B[1]);
26  wait(NULL);
27  wait(NULL);
```

mystere.c

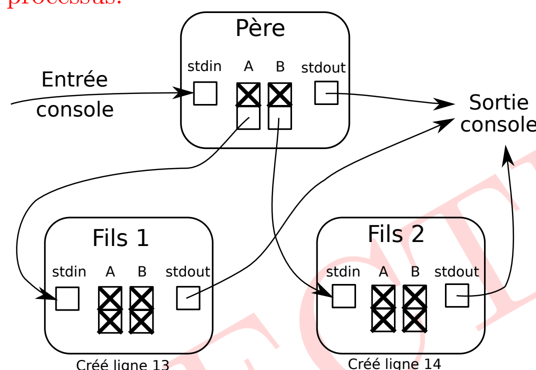
```
1   #include <stdio.h>
2   #include <ctype.h> // islower()
3   #include <unistd.h>
4   #include <sys/wait.h>
5
6   int main(int argc, char *argv[]){
7     int A[2], B[2];
8     char c;
9
10    pipe(A);
11    pipe(B);
12
13    if (fork()) {
14      if (fork()) {
15        close(A[0]);
16        close(B[0]);
17        while (read(0,&c,1) == 1) {
18          if (islower(c)) {
19            write(A[1], &c, 1);
20          } else {
21            write(B[1], &c, 1);
22          }
23        }
24        close(A[1]);
25        close(B[1]);
26        wait(NULL);
27        wait(NULL);
28      } else {
29        dup2(B[0],0);
30        close(A[0]);
31        close(A[1]);
32        close(B[0]);
33        close(B[1]);
34        while (read(0,&c,1) == 1)
35          printf("1: %c\n", c);
36      }
37    } else {
38        dup2(A[0],0);
39        close(A[0]);
40        close(A[1]);
41        close(B[0]);
42        close(B[1]);
43        while (read(0,&c,1) == 1)
44          printf("2: %c\n", c);
45    }
46    return 0;
47  }
```

Réponse

(page suivante)

CORRECTION CORRECTION

▷ **Question 1 :** Schéma des processus.



▷ **Question 2 :** Les grandes lignes du comportement.

Tous les caractères lus sur l'entrée standard du père sont envoyés aux enfants. Les minuscules sont envoyées au premier fils au travers du tube A, et les majuscules sont envoyées au second fils par le tube B. Chaque enfant affiche les caractères qu'il reçoit en les préfixant avec son numéro (1 ou 2).

Notons que c'est bien la chaîne BonJOUr qui est traitée de la sorte dans l'exemple, car elle est donnée au programme par son entrée standard. SaLuT, elle, est passée dans le `argv` du programme, qui est inutilisé dans le code.

Cela affiche par exemple :

```
2: B
1: o
1: n
2: J
2: O
2: U
1: r
```

▷ **Question 3 :** Les ordres possibles pour abAB.

La seule certitude, c'est que a est affiché avant b (puisque c'est dans le fils 1) et que A est affiché avant B (puisque c'est dans le fils 2). Il n'y a aucune contrainte sur l'ordre relatif entre majuscules et minuscules, qui sont traités dans des processus séparés. Cela donne les 6 ordres possibles suivants : abAB aAbB aABb AabB AaBb ABab.

▷ **Question 4 :** Si on supprime les lignes 24-27.

Constatons que les deux fils sont conçus pour s'arrêter dès qu'ils ont une erreur de lecture sur leurs tubes, c'est à dire dès qu'il n'y a plus d'écrivain.

Si on commente les lignes 24 à 27, le père termine sans fermer les tubes et sans attendre ses fils. On devrait donc avoir des zombis, où les fils attendent qu'il se passe quelque chose sur les tubes. Mais en fait, les ressources d'un processus sont nettoyées quand il termine.

Ici, le système ferme donc les tubes restés ouvert, et change les processus fils en orphelins (ils sont ratachés à init). Mais dès que les tubes sont fermés, les nouveaux orphelins terminent suite à une lecture renvoyant 0 caractère. Cela termine leur exécution. Ils ne deviennent pas zombis car init appelle `wait(NULL)` dès que l'un de ses fils d'adoption termine.

En conclusion, si on commente les lignes 24 à 27, le processus père ne fait pas le menage avant de terminer. Mais cela ne change pas grand chose car le système se charge de faire le ménage pour lui après sa fin.

▷ **Question 5 :** Si on supprime les lignes 24 et 25.

Dans ce cas-là, le père attend la fin de ses fils avant de terminer. Mais ses fils attendent le départ du dernier écrivain sur leur tube, qui est le père. C'est donc une situation d'interblocage où le père attend la fin des fils, qui attendent la fin du père.

Fin réponse

CORRECTION
CORRECTION