INFO-F-201 – Systèmes d'exploitation

Précisions diverses

Joël Goossens (N8.107) Olivier Markowitch (N8.106)
Arnaud Leponce (N8.213) Yannick Molinghen (N8.210)
Alexis Reynouard (N8.215)

Exercice 1. *Dans le système Unix, est-ce que tout processus a un père ?*

Exercice 2. *Que se passe-t-il lorsqu'un processus devient orphelin (mort de son père) ?*

Exercice 3. *Est-ce qu'un processus peut exécuter plusieurs programmes, et est-ce qu'un programme peut être exécuté par plusieurs processus à la fois ? Expliquez.*

Exercice 4. *Vous souhaitez accélérer le traitement d'une application en parallélisant son contenu. Vaut-il mieux utiliser des threads ou des processus ? Pourquoi ?*

Exercice 5. *Quelle est la différence entre un ordonnanceur préemptif et un ordonnanceur non préemptif ?*

Exercice 6. *Quels sont les événements qui peuvent provoquer l'interruption de l'exécution d'un processus ?*

Exercice 7. *Expliquez la notion de pseudo parallélisme qu'implique un ordonnanceur préemptif dans un système monoprocesseur.*

Exercice 8. *Dans un système d'ordonnancement circulaire (round robin = tourniquet), quel est l'effet d'un quantum de temps trop long ? Quel est l'effet d'un quantum de temps trop court ?*

Exercice 9. *Comment expliquez-vous qu'un processus qui attend une entrée sur stdin par l'utilisateur n'occupe pas 100% d'un processeur (ce qui serait de l'attente active) ? Décrivez le mécanisme qui permet cela.*

Exercice 10. *Définissez un interblocage (deadlock) en quelques mots.*

Exercice 11. *Quelle est la différence entre un thread et un processus au niveau de leur création et*

de leur terminaison.

Exercice 12. *Considérons un système mono-processeur comprenant les 5 ressources informatiques additionnelles $R_1, R_2, \ldots, R_5$ à accès exclusif (non partageables). Dans les pseudo-codes ci-dessous la fonction `lock(R)` permet d'allouer la ressource au processus appelant si celle-ci est libre. Sinon, le processus appelant est bloqué. La fonction `unlock(R)`, appelée par le détenteur de la ressource, libère cette ressource.*

Considérons les 4 processus P_1, P_2, P_3 et P_4 qui exécutent respectivement les codes suivants :

```
// P1
while (1){
    lock(R1);
    lock(R2);
    lock(R3);
    lock(R4);
    // Utiliser R1, R2, R3, R4
    unlock(R1);
    unlock(R2);
    unlock(R3);
    unlock(R4);
}
```

```
// P2
while (1){
    lock(R5);
    lock(R2);
    lock(R3);
    lock(R4);
    // Utiliser R2, R3, R4, R5
    unlock(R5);
    unlock(R2);
    unlock(R3);
    unlock(R4);
}
```

```
// P3
while (1){
    lock(R4);
    lock(R5);
    // Utiliser R4, R5
    unlock(R4);
    unlock(R5);
}
```

```
// P4
while (1){
    lock(R3);
    lock(R4);
    lock(R5);
    // Utiliser R3, R4, R5
    unlock(R3);
    unlock(R4);
    unlock(R5);
}
```

1. *Donnez une séquence d'entrelacements (par exemple P_1 lock(R4), P_2 lock(R2), …) des instructions des 4 processus qui mènent à un interblocage.*
2. *Pourrions-nous éviter les interblocages en modifiant l'ordre des demandes (les locks) des ressources ? Justifiez votre réponse.*