Examen de Systèmes d'Exploitation

(1h45 - Documents de cours autorisés)

7 mars 2019

Exercice 1 : Ordonnancement

4 tâches $T_1 \ldots T_4$ s'exécutent sur un système dont l'ordonnanceur utilise un mécanisme de priorité et un tourniquet au sein d'une priorité donnée. La priorité d'une tâche est calculée en cumulant le temps pendant lequel la tâche s'est exécutée ; elle est exprimée en unité de temps et n'est pas nécessairement entière. La tâche la plus prioritaire est celle dont la valeur numérique de la priorité est la plus faible.
À l'instant 0, les quatre tâches ont une priorité initialisée à 0 et sont stockées dans une liste dans l'ordre de sortie $T_1 \ldots T_4$ (la tâche T_1 a donc la priorité la plus élevée). L'unité de temps utilisée est la durée du quantum. Le basculement ne se fera que si la tâche en cours se bloque ou si son quantum est terminé. Le temps de basculement est négligeable.
Ces différentes tâches manipulent des ressources communes protégées par deux verrous V_1 et V_2. Le tableau 1 montre l'utilisation que chaque tâche fait des verrous ($L(V_i)$ signifie que la tâche chercher à verrouiller le verrou V_i, $U(V_i)$ signifie qu'elle le déverrouille).
Dans le tableau 1, pour chaque tâche, la première ligne donne la date (en unité de temps et dans le temps tel que le « voit » la tâche), la deuxième donne l'action réalisée.

Question 1 : *Montrer l'ordonnancement dans le temps de ces quatre tâches en supposant que chacune d'elles nécessite une durée d'exécution totale de quatre unités de temps.*

T_1	0.5	2.5		T_2	0.5	2.5		T_3	0.5	2		T_4	0.7	1	
	$L(V_1)$	$U(V_1)$			$L(V_2)$	$U(V_2)$			$L(V_1)$	$U(V_1)$			$L(V_2)$	$U(V_2)$	

TABLE 1 – Utilisation des ressources pour chaque tâche

Exercice 2 : Mise en œuvre de fonctions d'une pile de protocole

On désire implanter les échanges des données entre 2 couches de niveau différent d'une pile de protocole.

1. La couche N+1 fournit les données à émettre à la couche N. Une fonction « de_Couche_N+1 » (`void de_Couche_N+1()`) de la couche N permet récupérer ces données via un buffer, lorsque celui-ci n'est pas vide.
2. La couche N reçoit des données de la couche N-1. Après traitement ces données sont transmises à la couche N+1. La fonction « vers_Couche_N+1 » (`void vers_Couche_N+1()`) exécutée par la couche N insère les données dans un buffer si celui-ci n'est pas plein.

Question 1 : *Qu'est-ce qu'une tâche ? Quelles sont les différences entre un processus lourd et un processus léger ?*

Les deux fonctions « de_Couche_N+1 » et « vers_Couche_N+1 » sont exécutées dans des processus légers.

Question 2 : *Ecrire, en langage C, les instructions permettant d'exécuter les fonctions « de_Couche_N+1 » et « vers_Couche_N+1 » dans deux threads différents.*

La communication entre la couche N et N+1 est assurée ici à travers deux buffers : un buffer d'émission, noté `BufEmission`, et un buffer de réception, noté `BufReception`. La fonction « de_Couche_N+1 » lit les données présentent dans le buffer de réception s'il y en a. Et la fonction « vers_Couche_N+1 » écrit les données dans le buffer d'émission s'il n'est pas plein. Pour stocker les données (le type `Data` n'a pas d'importance : ce sont des `int`, `struct`, ..., ce que vous voulez), les buffers sont constitués d'un tableau de données de taille finie, définie par une constante `TAILLE_MAX` et le nombre de données qu'il contient. Ainsi, `nb_data == 0` signifie que le

buffer est vide et `nb_data == TAILLE_MAX` signifie que le buffer est plein. La structure de données `Buffer` est donc la suivante :

```
typedef struct {
        Data tab[TAILLE_MAX];
        int nb_data;
} Buffer;
```

Question 3 : *Proposez une implantation des fonctions « de_Couche_N+1 » et « vers_Couche_N+1 » en tenant compte du partage des buffers d'émission et de réception entre les couches de différents niveaux. Quelles modifications doit-on apporter à la structure de données* `Buffer` *?*

Pour la couche N+1, la lecture dans le buffer d'émission (les données sont écrites par la couche N via la fonction « vers_Couche_N+1 ») est effectué par la fonction de_Couche_N. Le buffer d'émission peut être rempli par plusieurs couches N.

Question 4 : *Comment gèreriez-vous cette contrainte ?*

Question 5 : *Puisque les processus légers semblent offrir de meilleures performances que les processus Unix traditionnels, pourquoi les systèmes de type Unix ne sont-ils pas ré-écrits de sorte à ne plus utiliser que les processus légers ?*

Exercice 3 : Virtualisation

Question 1 : *Qu'est-ce qu'un hyperviseur ?*

Question 2 : *Quel est l'intérêt de la paravirtualisation ?*

Question 3 : *Pourquoi, sous Android, la machine virtuelle Dalvik remplace la JVM ?*

Question 4 : *Quels sont les avantages à utiliser les conteneurs Docker ? A-t-on réellement un système virtualisé ?*

Question 5 : *Ecrire le contenu d'un fichier Dockerfile qui :*

- *exécute le système minimal alpine ;*
- *exécute la commande apk pour installer gcc ;*
- *et compile le fichier toto.c.*

Quelle commande doit-on exécuter pour lancer un terminal dans le système alpine du conteneur ?
Comment faire pour que le fichier compilé (toto.c → a.out) soit exécuté au lancement du conteneur ?

Exercice 4 : Micro-noyaux

Question 1 : *Quels sont les avantages des micro-noyaux ? A quoi servent les IPC ?*

Question 2 : *Qu'est-ce que le TCB ? A quoi sert le buffer dans le TCB des threads du micro-noyau L4 ?*

Question 3 : *Un micro-noyau comme L4 peut être utilisé comme hyperviseur d'une machine virtuelle exécutant un système Linux. Que se passe-t-il lorsque le système Linux exécute la commande* `fork` *? Qui contrôle le processus créé ?*