UNIVERSITÉ BORDEAUX 1 Sciences Technologies

ANNÉE 2011-2012
Étape : Master Sciences Technologies (semestre 1)
UE : J1INAW11 (Systèmes d'Exploitation)
Modalités : Épreuve sans document

Date : 18 novembre 2011
Heure : 10h15
Durée : 1h30

---

1 Questions de cours (échauffement)

Question 1 Rappelez précisément le rôle du circuit `MMU` dans un processeur. À quel endroit de la machine se trouvent les tables des pages des processus ? Expliquez par quelle technique les systèmes d'exploitation peuvent diminuer la taille totale occupée par ces tables (aidez-vous d'un petit schéma).

Question 2 Rappelez le principe de *segmentation* de la mémoire utilisé par certains systèmes d'exploitation. Donnez-en les principaux avantages et inconvénients.

2 Gestion de processus

L'algorithme d'ordonnancement interactif implanté dans les noyaux Linux 2.4.x utilise un système de quanta de temps (i.e. crédits) que les processus sont autorisés à utiliser pendant une durée que l'on appelle une « *époque* ». Le nombre de quanta attribués à chaque processus lors du démarrage d'une nouvelle époque dépend de sa priorité et du nombre de quanta qu'il n'a pas consommés lors des époques précédentes.

Question 1 Décrivez l'algorithme principal, c'est-à-dire le code exécuté dans le noyau à chaque fois que le temporisateur (horloge) expire. Comment un processus peut-il encore avoir des crédits à la fin d'une époque ?

Question 2 En prenant un exemple simple constitué d'un processus P_1 ayant droit à 8 quanta de temps au début d'une époque, et de P_2 ayant droit à 2 quanta, tracer un petit chronogramme (sur la durée d'une époque) illustrant à quels moments surviendront les changements de contexte.

Question 3 Imaginez une modification de l'algorithme permettant un meilleur entrelacement de l'exécution des processus dans des cas tels que celui observé à la question précédente.

3 Moniteurs de Hoare

On dispose d'une simulation de trafic automobile à une intersection (entre deux voies) munie de feux tricolores. Cette simulation s'appuie sur deux programmes UNIX : `carrefour` et `voiture`. Un seul processus exécute le programme `carrefour` ; il a pour tâche de gérer le régime des feux et il est lancé en tout début de simulation. Les autres processus de la simulation (peu importe le nombre) exécutent le programme `voiture` de manière concurrente. Tous les processus ont accès à une zone commune de mémoire partagée, qui contient les variables suivantes :

```
boolean peux_passer = TRUE;
int sens_actif = 1; /* 1 ou 2 */
int voitures_engagées = 0;  /* nombre de voitures en train de traverser */
```

Voici le code du programme `carrefour` :

```
while(1) {
  sleep(PERIODE_REGIME);
  peux_passer = FALSE;
  while(voitures_engagées > 0) /* rien */ ;
  sens_actif = 3 - sens_actif; /* inversion du sens */
  peux_passer = TRUE;
}
```

Et voici le code du programme `voiture` :

```
int sens = tirage_aleatoire_du_sens();
while(!peux_passer || sens != sens_actif) /* rien */ ;
voitures_engagées++;
sleep(TEMPS_TRAVERSEE);
voitures_engagées--;
```

En résumé, une voiture qui veut traverser dans le sens i **s'arrête** si `sens_actif` $\neq i$ (feu rouge) ou si `sens_actif` $= i$ mais `peux_passer` $=$ `FALSE` (feu orange) ; **passe** sinon (feu vert).

Question 1 Cette simulation va-t-elle solliciter intensivement le ou les processeur(s) de la machine sur laquelle elle va s'exécuter ? Expliquez pourquoi.

Question 2 Montrez que la programmation de la simulation telle que décrite ci-dessus risque d'aboutir à des accidents, c'est-à-dire à des situations où plusieurs voitures s'engagent simultanément dans des sens différents.

Question 3 Corrigez donc le protocole en introduisant des moniteurs de Hoare (et peut-être d'autres variables) partagés et en modifiant les programmes. Profitez de l'occasion pour éviter l'utilisation des boucles d'attente active. N'oubliez pas de préciser les valeurs initiales.

Voici pour rappel les primitives que vous pouvez utiliser :

```
typedef ... mutex_t ;
typedef ... cond_t ;
void mutex_lock(mutex_t *m);
void mutex_unlock(mutex_t *m);
void cond_wait(cond_t *c, mutex_t *m);
void cond_signal(cond_t *c);
void cond_bcast(cond_t *c);
```