Intra - ift2245 Nom:_____ Matricule:_____

Examen Intra

IFT-2245

March 7, 2016

Directives

- Documentation autorisée: *une* page *manuscrite* recto.
- Pas de calculatrice, téléphone, ou autre appareil électronique autorisé.
- Répondre sur le questionnaire dans l'espace libre qui suit chaque question. Utiliser le verso des pages si nécessaire.
- Chaque question vaut 5 points pour un total maximum de 25 points.
- Les questions ne sont pas placées par ordre de difficulté.

0 Nom et prénom (1 point de bonus)

Écrire son nom et prénom et son code permanent en haut de chaque page.

1 Primitives de synchronisation

Les variables "full/empty" sont des outils de synchronisation inspirés des principes du *dataflow*. Une telle "variable" est un objet avec deux états:

- elle peut être *vide*
- ou *pleine*, auquel cas elle contient une valeur (par exemple de type `void*`).

Elle a deux opérations, une qui met une valeur et doit attendre que la variable soit *vide* avant de pouvoir le faire, et une autre qui prend la valeur et doit donc attendre que la variable soit *pleine* avant de pouvoir le faire.

Implémenter ces "variables" en utilisant les primitives de sémaphores standards: `void wait(semaphore *s)` et `void signal(semaphore *s)`.

```
struct fe_var {


};

void fe_put (struct fe_var *v, void *data)
{









}

void *fe_take (struct fe_var *v)
{









}
```

2 Fork & Threads

Soit le code suivant:

```
pid_t pid1 = fork();
if (pid1 == 0) {
  pid_t pid2 = fork();
  thread_create(...);
  thread_join(...)

  if (pid2 > 0)
    waitpid (pid2);
}
pid_t pid3 = fork();
```

En comptant le thread (et son processus) original:

1. Combien de processus sont créés?
2. Combien de threads sont créés?
3. Parmi ces processus et threads, combien de chaque peuvent être actifs en même temps?

3 Conditions de course

Soit une fonction qui renvoie des “numéros de ticket”, censés être uniques.

```
static int counter = 0;
int get_unique_ticket_number (void) {
  return counter++;
}
```

1. Donne 2 scénarios d’exécution différents où la fonction renvoie deux fois le même résultat.
2. Montre comment éviter ces problèmes avec des sémaphores.
3. Montre une autre solution qui utilise l’instruction *compare&swap* à la place.

4 Context switch

Placer les opérations suivantes dans l'ordre:

1. Le thread fait un appel système (*trap*) `read` pour lire du disque.

- Le thread est placé dans la queue *waiting*.
- Le SE sauve le contenu des registres dans le PCB.
- Le CPU poursuit l'exécution du code du thread.
- L'ordonnanceur à court terme choisi un autre thread.
- Le thread est retiré de la queue *ready*.
- Le CPU passe en mode *noyau*.
- Le SE charge le contenu des registres à partir du PCB.
- Le CPU saute à la routine de gestion des appels systèmes.
- Le CPU passe en mode *utilisateur*.

5 QCM

	oui	non
L'usage systématique de *moniteur* évite les conditions de course	☐	☐
Les processus de l'utilisateur `root` (ou `administrateur`) tournent en mode *noyau*	☐	☐
Un système multiprocesseur ne peut pas inhiber les interruptions	☐	☐
Avec la synchronisation optimiste, l'inversion de priorité est impossible	☐	☐
Les *spinlocks* n'ont de sens que dans un système monoprocesseur	☐	☐

Un thread parent et son thread enfant partagent:

- ☐ Les registres
- ☐ La pile
- ☐ Les descripteurs de fichiers
- ☐ Le tas
- ☐ Les variables globales

Un processus parent et son processus enfant partagent:

- ☐ Les registres
- ☐ La pile
- ☐ Les descripteurs de fichiers
- ☐ Le tas
- ☐ Les variables globales