Examen INF552
Principes et programmation des systèmes d'exploitation
Majeure 2 – École Polytechnique

2006–2007

- L'examen dure 2 heures.
- Le sujet comporte 3 pages
- Tous documents autorisés.
- Le barème est donné à titre indicatif, il sert surtout à évaluer le poids respectif des sections.
- Il est impératif de commenter les programmes et de justifier vos réponses.

Le but de ce problème est d'implanter le mécanisme de synchronisation par variables de condition. Une variable de condition est un objet de synchronisation entre threads, sur lequel deux opérations sont possibles : attendre sur une condition, qui bloque le thread qui exécute cette opération, et signaler une condition, qui réveille tous les threads qui attendent sur la condition.

Le squelette de programme suivant définit le nom d'une structure et les prototypes des fonctions permettant de gérer les variables de conditions. Nous utiliserons cette structure et ces fonctions dans toute la suite de l'enoncé.

```
/* Le contenu d'une condition */
struct condition {
  /* À compléter */
};

/* La définition suivante permet d'utiliser [condition] plutot
   que [struct condition] */

typedef struct condition condition;

/* Générer une nouvelle variable de condition */

condition* condition_alloc(void);

/* Attendre sur une condition [cond]. Un lock doit être mis sur le
   sémaphore [sem]. Ce lock est libéré tant que le thread attend sur
   la condition, puis le lock est à nouveau pris par le thread quand il
   cesse d'attendre sur la condition. */

void condition_wait(condition *cond, sem_t sem);

/* Signaler une condition. Tous les threads attendant sur la
   condition sont libérés. */

void condition_signal(condition *cond);

/* Nettoyer une condition. La condition est désallouée. */

void condition_free(condition *cond);
```

Exercice 1 - Utilisation des variables de condition

Considérez le programme suivant. Deux threads exécutent respectivement les fonctions `writer_thread()` et `reader_thread()`, et partagent la variable `pointer`.

```
volatile int *pointer = NULL;

void writer_thread() {
  while (1) {
    if (pointer == NULL) {
      pointer = malloc(sizeof(int));
      *pointer = random();
    }
  }
}
void reader_thread() {
  while (1) {
    if (pointer != NULL) {
      printf("pointer = %d\n", *pointer);
      free(pointer);
      pointer = NULL;
    }
  }
}
```

Question 1.1

En exécutant ce programme, une erreur “Erreur de segmentation” apparaît au bout d’un certain temps. D’où provient cette erreur ?

Question 1.2

Résolvez ce problème en modifiant le programme par l’ajout d’un sémaphore. On supposera que le sémaphore est initialisé dans le programme principal.

Question 1.3

Peut-on résoudre ce problème sans utilisation d’un sémaphore ou de tout autre mécanisme d’exclusion mutuelle ? Même question si on exécute plusieurs paires écrivain (fonction `writer_thread()`) et lecteur (fonction `reader_thread()`) ?

Question 1.4

Ce programme fait il une utilisation efficace du processeur ?

Question 1.5

Nous proposons d’utiliser une variable de condition pour régler ce problème. La variable de condition doit servir, d’une part à garantir l’exclusion mutuelle entre les accès à `pointer` (rôle joué par le sémaphore utilisé dans les question précédentes), et d’autre-part à alerter l’autre thread d’une modification de la variable `pointer`. On supposera que la variable de condition est correctement initialisée dans le programme principal et que les fonctions `condition_wait()` et `condition_signal()` ont déjà été implantées. Modifiez le programme en conséquence.

Question 1.6

Ecrire le programme principal qui initialise les variables globales et lance les deux threads. On ne s’occupera pas de l’initialisation du contenu de la variable `cond` que l’on n’a pas encore définie.

Exercice 2 - Implantation des variables de condition

On désire implanter soi-même des variables de condition, en utilisant les signaux UNIX pour relancer les threads.

Question 2.1

Pour cela, on propose de placer dans la structure condition une liste des threads (`pthread_t`) en attente sur la condition, ainsi qu’un compteur, indiquant combien de fois la condition a été signalée. Définissez les types utilisés par le programme.

Question 2.2

On propose d’utiliser l’appel système `pause()` pour bloquer un thread qui attend sur une condition ; `pause()` est interrompu dès qu’un signal reçu provoque soit la terminaison du programme, soit l’exécution d’une fonction de traitement d’un signal. Pour empêcher l’interruption de `pause()` tant que la condition n’est pas signalée, on hésite entre ignorer tous les signaux et les bloquer : quelle différence y a t’il entre ces deux approches ?

Laquelle allez-vous choisir, sachant que la fonction `condition_wait()` ne doit pas modifier le comportement du programme qui l'exécute vis à vis des signaux rattrapés par une fonction de traitement (un handler) ?

Question 2.3

Après avoir modifié le compteur dans la condition, on désire interrompre l'attente des threads en leur envoyant un signal `SIGUSR1`, au moyen de la fonction `pthread_kill` :

```
int pthread_kill(pthread_t thread, int sig);
```

Cette fonction se comporte comme `kill()` mais en désignant un thread particulier.

Ecrivez la fonction `condition_signal`.

Question 2.4

Pour obtenir son identificateur (`pthread_t`), un thread peut utiliser la fonction `pthread_self()`. Ecrivez la fonction `condition_wait()` en conséquence. Attention : le signal `SIGUSR1` cause par défaut la terminaison du thread. Est-il vraiment nécessaire d'empêcher l'interruption de `pause()` par les signaux ?

Question 2.5

Pourquoi est-il nécessaire d'utiliser un sémaphore interne à la condition ? Dans `condition_wait`, l'ordre dans lequel on prend et libère les deux sémaphores (le sémaphore interne à la condition, et le sémaphore fourni en argument) est important. Pourquoi ?

Question 2.6

Que se passe t'il si l'application qui utilise nos variables de condition décide d'ignorer le signal `SIGUSR1` ? Comment y remédier ?

Exercice 3 - Applications

Question 3.1

Le mécanisme des conditions est particulièrement bien adapté à l'implantation des lectrues et écritures bloquantes sur un tube (`pipe()`). Expliquez pourquoi et commentez en décrivant le protocole mis en oeuvre en fonction des différents états du tube.

Question 3.2

Donnez 2 arguments pour comparer les conditions et les sémaphores : (1) du point de vue de la robustesse du programme, (2) du point de vue de l'efficacité. Développez chaque argument en 3 ou 4 phrases maximum plus un exemple pratique (pas forcément du code C).