Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Travaux pratiques – n°3
Threads Posix et synchronisation de type « moniteur »

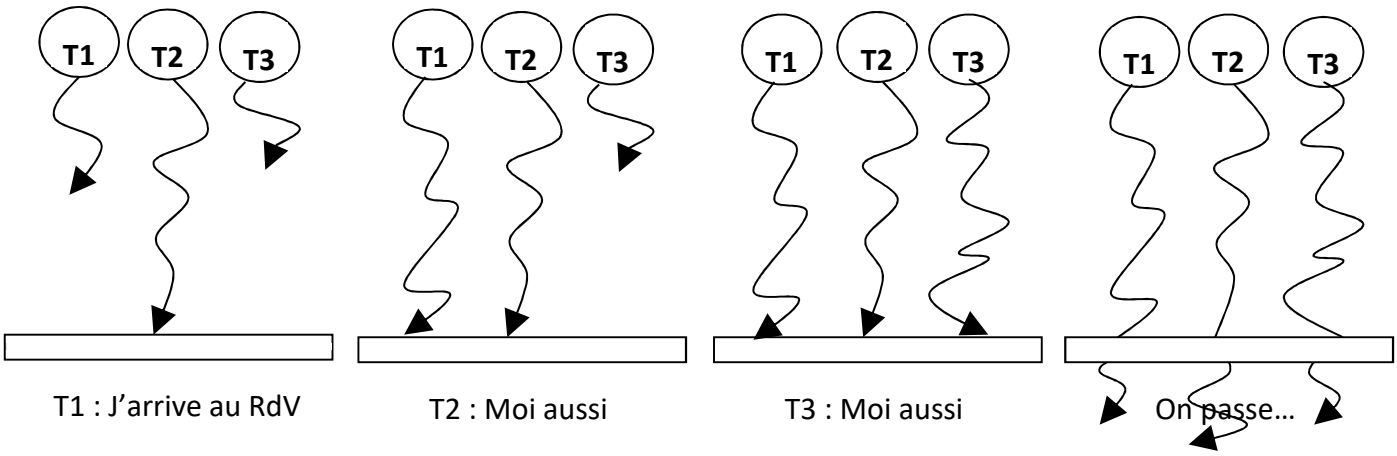
Documentation

Le concept de « moniteur » peut être utilisé pour synchroniser des threads Posix en utilisant des conditions Posix (type pthread_cond_t). Voir la documentation à votre disposition sous Moodle.

Exercice 1 – Rendez-vous à N

On désire réaliser un rendez-vous entre N threads. Un thread arrivant au point de rendez-vous se met en attente s'il existe au moins un autre thread qui n'y est pas arrivé. Tous les threads bloqués sur cette « barrière » peuvent la franchir lorsque le dernier y est arrivé.

La figure ci-dessous illustre ce comportement.



Chaque thread a le comportement suivant :

```
Début
      Je fais un certain traitement ;
      J'arrive au point de rendez-vous
            et j'attends que tous les autres y soient aussi… ;
      …Avant de pouvoir continuer mon traitement ;
Fin
```

Écrire une application dans laquelle N threads établissent un rendez-vous (N peut être le paramètre de l'application).

[Code à déposer sous Moodle]

Exercice 2 – Terrains de tennis (Décembre 2013)

On considère des activités parallèles s'exécutant en parallèle et représentant des joueurs de tennis et des arbitres qui veulent accéder à un terrain de tennis.

Une activité Joueur acquiert ou libère le terrain grâce aux opérations :

```
void demanderTerrain();
void libererTerrain(…);
```

Une activité Arbitre acquiert ou libère le terrain grâce aux opérations :

```
void debuterArbitrage();
void finirArbitrage(…);
```

Un joueur ne choisit pas son adversaire (les joueurs jouent dans l'ordre d'arrivée).

On ne se préoccupe ni du temps de jeu ni de la terminaison du match.

On suppose qu'un joueur ayant réussi à accéder au terrain finit par le quitter et qu'un arbitre l'ayant obtenu finit aussi par le quitter.

Q1. Version 1. Les joueurs s'entraînent et n'ont pas besoin d'arbitre (seuls des activités Joueur veulent accéder au terrain). Sur l'unique terrain, on doit assurer qu'il n'y a en même temps que **deux joueurs** exactement.

Q2. Version 2. Les joueurs jouent des matches arbitrés (des activités Joueur et Arbitre veulent accéder au terrain). Sur l'unique terrain, on doit assurer qu'il n'y a en même temps qu'**un seul arbitre et deux joueurs** exactement.

Q3. Version 3. On souhaite généraliser la variante 2 à l'utilisation parallèle de NB_TERRAINS de tennis.

Rappel : Si les affichages sont trop rapides, il est possible de temporiser l'exécution d'un thread pendant quelques microsecondes ou nanosecondes à l'aide des primitives :

int usleep (useconds_t usec) ;

int nanosleep(const struct timespec *req, struct timespec *rem) ;

Voir le manuel en ligne pour leur utilisation (man 3 usleep ou man 2 nanosleep).

On peut utiliser une valeur générée aléatoirement (voir les fonctions srand et rand) pour varier les délais d'attente d'un thread à un autre.

Mais, **attention**, la temporisation n'est pas là pour résoudre les problèmes d'accès concurrents à des variables partagée. En d'autres termes : toute exécution d'une application parallèle doit donner un résultat cohérent **sans** temporisation !