



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

Travaux pratiques – n°1

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 – Modèle des producteurs/consommateurs – Version sans synchronisation

- Récupérez et compilez le code nommé `m1_prodCons_base.c` fourni sous Moodle.

Pour avoir la totalité des messages affichés par cette application, définissez les variables qui les conditionnent (voir dans le code, les directives au préprocesseur : `#ifdef VARIABLE . . . #endif`) grâce à l'option `-D` de compilation :

```
gcc m1_prodCons_base.c -o prodConso -lpthread -DVARIA
```

- Exécutez l'application pour différentes valeurs de paramètres.

Exemples d'exécution :

```
%prodConso 3 3 1
```

```
%prodConso 2 2 2
```

```
%prodConso 4 4 4
```

et constatez les problèmes liés aux conflits d'accès sur les variables partagées.

- Modifiez ce programme pour remplacer les constantes `NB_FOIS_PROD` et `NB_FOIS_CONSO` (nombre de dépôts réalisés par un producteur et nombre de retraits réalisés par un consommateur) par deux paramètres supplémentaires donnés lors de l'exécution.

Exercice 2 – Modèle des producteurs/consommateurs – Version de base

À partir du code fourni, **ajoutez la synchronisation** de type « moniteur » – utilisant les verrous d'exclusion mutuelle et les conditions Posix – nécessaire pour implanter la version de base des producteurs-consommateurs (vue en cours et TD) dans laquelle les retraits se font dans l'ordre des dépôts.

- Exécutez l'application pour différentes configurations afin de vous assurer de la validité des résultats obtenus.

[Code à déposer sous Moodle]

Exercice 3 – Modèle des producteurs/consommateurs – Dépôts alternés

Modifiez votre code (**en conservant** la version précédente) pour implanter la version où les producteurs déposent de manière alternée leurs messages dans le buffer (voir V2 du TD).

[Code à déposer sous Moodle]

Exercice 4 – Pour continuer... Modèle des producteurs/consommateurs – Retraits à la demande

Modifiez votre code (**en conservant** la version précédente) pour implanter la version où les consommateurs demandent à retirer un message d'un certain type et où leurs retraits doivent être effectués dans l'ordre des dépôts (voir V3 du TD).

Exercice 5 – Pour continuer... Modèle des producteurs/consommateurs – Dépôts doubles

Modifiez votre code (**en conservant** la version précédente) pour implanter la version où les producteurs veulent déposer **deux** messages de manière **consécutif** dans le buffer et où leurs retraits doivent être effectués dans l'ordre des dépôts (contrôle terminal juin 2017).

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 partagées. En d'autres termes : toute exécution d'une application parallèle doit donner un résultat cohérent **sans** temporisation !