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 processus en utilisant des conditions (dans le module **`multiprocessing`**). Voir la documentation à votre disposition sous Moodle.

Exercice 1 – Modèle des producteurs/consommateurs – Version sans synchronisation

- Récupérez le code nommé *m1_prodCons_base.py* fourni sous Moodle.
- Exécutez l'application pour différentes valeurs de paramètres.

Exemples d'exécution :

```
python3 m1_prodCons_base.py 3 3 1
python3 m1_prodCons_base.py 2 2 2
python3 m1_prodCons_base.py 4 4 4
```

et constatez les problèmes liés aux variables non partagées.

- Modifiez ce programme pour remplacer les constantes **`nb_times_prod`** et **`nb_times_cons`** (nombre de dépôts réalisés par un producteur et nombre de retraits réalisés par un consommateur) par deux paramètres supplémentaires donnés lors de l'exécution.
- Modifiez ce programme pour utiliser des variables partagées entre les processus. Constatez les conflits d'accès sur les variables partagées

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 – 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. Comme dans l'exemple du court de tennis, construisez un monitor dans une classe python.

- 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). Pour les tests, utilisez un nombre de producteurs pair.

[Code à déposer sous Moodle]

Exercice 4 – 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). On ne forcera plus l'alternance des dépôts.

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

Modifiez votre code (**en conservant** la version précédente) pour implanter une version où à **chaque** dépôt, un producteur dépose **deux** messages de manière (insécable et) **consécutive** dans le buffer (donc, la fonction deposer() devra être adaptée). Les retraits des messages seront effectués de la même manière que dans la version de base, i.e. 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 processus pendant quelques microsecondes ou nanosecondes à l'aide des primitives :

time.sleep(secondes)

Voir le manuel en ligne pour leur utilisation :

https://docs.python.org/fr/3/library/time.html#time.sleep

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

https://docs.python.org/3/library/random.html

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 !