

TP 2

Concurrence et synchronisation – INFO3 S5

Moniteurs

Objectifs du TP : Reprendre les éléments théoriques vus en cours sur les moniteurs, puis en TD, et les appliquer à la conception de solutions en langage Python aux problèmes posés par la concurrence de processus s'exécutant en parallèle

1 L'alternance

On souhaite que les 2 opérations `Ping()` et `Pong()` respectent les contraintes suivantes :

- Exclusion mutuelle
- On débute par `Ping()`
- `Ping()` et `Pong()` doivent alterner strictement (`Ping-Pong-Ping-Pong-Ping-Pong-....`)

- 1.1 Lisez le programme source Python contenu dans le fichier `1_alternance.py`, qui reprend en Python l'exercice 2 du TD 2. Exécutez-le. Que constatez-vous ?
- 1.2 A partir de la solution en *pseudo-code* de l'exercice 2 du TD 2, implémentez en Python la solution à base de *moniteur* permettant de reproduire ce comportement d'alternance. Pour cela vous devrez créer le fichier `moniteur_1_alternance.py` à partir d'une copie de `moniteur_0_exemple.py`, puis y définir le code du moniteur (variables d'état, mutex, conditions, points d'entrée).
NB : on utilisera les mutex/verrous `threading.Lock` (`Lock()`, `acquire()`, `release()`), et les conditions `threading.Condition` (`Condition(mutex)`, `wait()`, `notify()`) du module `threading`
NB : Le langage de programmation Python ne dispose que de pseudo-moniteurs où l'exclusion mutuelle (EM) doit être gérée manuellement avec un sémaphore binaire (mutex) dans chaque *point d'entrée*.
- 1.3 Testez votre solution sur différentes simulations, en changeant les paramètres de lancement des threads `ThreadingGenerator([(thread_ping, NbProcess), (thread_pong, NbProcess)])`, de `NbProcess=1` puis `2`, et la variable entière `NbCoups`.
- 1.4 Dessinez le *chronogramme* de l'évolution des 2 processus en traçant l'ordre des opérations `wait()` et `notify()` sur les variables condition à l'aide d'affichage `tprint()` avant et après ces opérations.
- 1.5 Dans cette solution, `Ping` doit nécessairement débiter l'alternance. Proposez une modification permettant au premier processus quel qu'il soit (`Ping` ou `Pong`) de débiter l'alternance.

2 Le problème des Producteurs Consommateurs

Le `tampon=Tampon_fifo()` est une ressource commune aux processus, qui est munie d'un ensemble d'opérations de manipulation `deposer()`, `retirer()`, etc ... définies dans le fichier `Tampon_fifo.py`.

On souhaite que les opérations `deposer()` et/ou `retirer()` sur le tampon respectent les contraintes suivantes :

- C1 : Exclusion mutuelle des opérations `deposer()` et/ou `retirer()` → mutex
- C2 : Attente des consommateurs si le tampon est vide (`estVide()`) → condition
- C3 : Attente des producteurs si le tampon est plein (`estPlein()`) → condition

- 2.1 Lisez le programme source Python contenu dans le fichier `2_producteur_consommateur.py`, qui reprend en Python l'exercice 2 du TD 2. Exécutez-le. Que constatez-vous ?
- 2.2 Complétez le code Python afin d'implémenter votre solution par moniteur, en reprenant les solutions en *pseudo-code* vue dans l'exercice 2 du TD 2, afin de respecter les 2 contraintes C1 et C2. Pour cela vous devrez créer le

fichier `moniteur_2_producteur_consommateur.py` à partir d'une copie de `moniteur_0_exemple.py`, puis y définir le code du moniteur (variables d'état, mutex, conditions, points d'entrée).

- 2.3 Testez votre solution sur différentes configurations en modifiant les variables `ThreadingGenerator([(thread_producteur, NbProcess)...])`, avec `NbProcess=1` puis `2`. Vérifiez que les *assertions logiques* contrôlant les opérations `wait()` et `notify()` sont correctes, que les configurations indésirables sont bien évitées, et qu'il n'y a pas d'interblocage.
- 2.4 Dessinez le *chronogramme* de l'évolution de chaque processus en repérant l'ordre des opérations `wait()` et `notify()` sur les variables condition.
- 2.5 Répéter les questions 3.2 et 3.3 pour résoudre les 3 contraintes C1 et C2 et C3

3 Une usine d'assemblage d'aéroplanes

Les au moins 7 processus `Carlingue` `Aile` `Moteur` `Roue` `Carlingue1Ailes2` `Carlingue1Ailes2Roues3` `Avion` de l'usine d'assemblage d'aéroplanes fonctionnent de manière autonome et en parallèle.

Cependant différentes contraintes de synchronisation doivent être respectées

- La chaîne de montage `Carlingue1Ailes2` ne peut fonctionner que si au moins 2 ailes et 1 carlingue ont été produites par `Carlingue` et `Aile`
- La chaîne de montage `Carlingue1Ailes2Roues3` peut fonctionner que si au moins 3 roues et un assemblage `Carlingue1Ailes2` ont été produits par `Roue` et `Carlingue1Ailes2`
- La chaîne de montage `Aeroplan` peut fonctionner que si au moins 2 moteurs et un assemblage `Carlingue1Ailes2Roues3` ont été produits par `Moteur` et `Carlingue1Ailes2Roues3`

- 3.1 Lisez le programme source Python contenu dans le fichier `3_aeroplanes.py`, qui reprend en Python l'exercice 3 du TD 2. Exécutez-le. Que constatez-vous ?.
- 3.2 Implémentez en Python la solution vue en TD utilisant des tampons contrôlés par les moniteurs Producteurs Consommateurs vus à la question 2. Testez-la sur différentes situations, en vérifiant le bon ordonnancement des opérations d'assemblage. Pour cela vous pouvez modifier les variables : `nbAvionsPerProcess` et `tailleMax`.
- 3.3 Dessinez le *chronogramme* de l'évolution de chaque processus en repérant l'ordre des opérations `wait()` et `notify()` sur les variables condition. Vérifiez que l'ordre est conforme à vos prévisions.

4 Un Sémaphore de comptage défini par moniteur

Un sémaphore de comptage est défini par les éléments suivants :

- Un compteur entier
- Une liste d'attente fifo
- 2 opérations atomiques `P()` et `V()` (en exclusion mutuelle)

- 4.1 Créez le fichier `moniteur_4_semaphore.py` à partir d'une copie de `moniteur_0_exemple.py`, puis y définir le code du moniteur (variables d'état, mutex, conditions, points d'entrée) implémentant un sémaphore de comptage avec un moniteur traité dans l'exercice 4 du TD 2..
- 4.2 Reprenez un des problèmes du TP1 (exclusion mutuelle ou alternance) et y remplaçant la solution sémaphore python (classe `Semaphore`) par l'utilisation de l'instance de moniteur que vous avez proposée en 4.1. Vous testerez vos solutions avec différentes valeurs initiales de sémaphore, afin de vérifier le bon fonctionnement de votre version de sémaphore implémentée par moniteur.
- 4.3 Puis vous testerez votre solution sur les 2 exemples proposés (EM et alternance) avec différentes valeurs initiales de sémaphore (ex :0, 1, 2).