

TP 1

Concurrence et synchronisation en Python – INFO3 S5

Sémaphores

Objectifs du TP : Reprendre les éléments théoriques vus en cours sur les sémaphores, 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'exclusion mutuelle

Lisez le programme source Python contenu dans le fichier `probleme_0_EM.py`, qui reprend en Python l'exemple de compétition sur une ressource critique vue en Cours et dans l'exercice 1 du TD 1.

Vous devez y retrouver :

- L'utilisation du module `threading` (`import threading`) pour accéder aux classes `Thread` et `Semaphore`,
- L'utilisation de `tprint` (`ThreadingGenerator.tprint`) pour un affichage avec estampille temporelle,
- Une variable globale commune `RC` (ressource critique) non protégée,
- Deux threads prenant en charge les fonctions : `thread_P1()` et `thread_P2()`,
- Le programme principal qui crée les 2 threads (`ThreadingGenerator()`), puis les lance (`startTest()`).
NB : vous pouvez consulter le fichier `threading_generator_tester.py` si vous voulez comprendre plus en détails la classe `ThreadingGenerator`.

- 1.1 Lancez tel quel l'exécution du programme `probleme_0_EM.py`. Que constatez-vous ? Expliquez pourquoi il est nécessaire de protéger la variable `RC`. Repérez dans le code les *sections critiques* qui doivent être protégées par un mécanisme d'*exclusion mutuelle*.
- 1.2 En repartant du *réseau de Pétri* dessiné en TD (TD 1 exercice 1) pour résoudre ce problème, éditez-le dans le simulateur de réseau de Pétri (jpns ou pipe2) disponible sur la page madoc du cours, puis exécutez-le en mode *pas à pas* (attention avec jpns à utiliser l'option «settings-parallel-manual» ou «settings-parallel-random», essayez les 2 possibilités). Observez le comportement du réseau de Pétri avec différentes conditions initiales du sémaphore (0, puis 1, 2). Qu'en déduisez-vous ?
- 1.3 Complétez le code Python afin de résoudre le problème de concurrence à l'aide d'un sémaphore. Puis vous testerez votre solution avec trois valeurs initiales de sémaphore distinctes : 0, 1, 2.
NB : Un sémaphore `s` est créé et initialisé à 3 par « `Semaphore(3)` », et manipulable par les opérations `s.acquire()` pour `P(S)` et `s.release()` pour `V(S)`.
- 1.4 A partir des traces d'exécution, dessinez le chronogramme de votre solution. Pour cela, vous ajouterez 2 lignes affichant un texte avant et après chaque opération sur les sémaphores. Par exemple `tprintf("Avant P(S)")` (ou `tprint("%s:Avant P(S)" % (nom,))` ou `tprint("{}:Avant P(S)".format(nom))`), noté `P(S)*` dans les chronogrammes du cours ; et `tprintf("Avant P(S)")` (ou ...), noté `*P(S)`. Vous dessinerez un chronogramme pour chaque valeur initiale du sémaphore : 0, 1, 2.

2 L'étreinte fatale

Reprenez les solutions *réseau de Pétri* et *pseudo code* vues en TD (TD 1, exercice 3).

- 2.1 Lisez le code Python contenu dans le programme `1_alternance.py`. Exécutez le programme Python. Que constatez-vous ? Et que constatez-vous en simulant le *réseau de Pétri* correspondant sous jpns ?
- 2.2 Editez et simulez sous jpns le *réseau de Pétri* permettant de corriger ce problème. Vérifiez que votre solution est correcte.
- 2.3 Implémentez votre solution à l'aide sémaphores dans le programme python. Puis testez-la.
- 2.4 Dessinez le chronogramme avant puis après de votre solution.

3 L'alternance

Reprenez les solutions *réseau de Pétri* et *pseudo code* vues en TD (TD 1, exercice 2).

- 3.1 Lisez le code Python contenu dans le programme `1_alternance.py`. Exécutez le programme Python. Que constatez-vous ? Et que constatez-vous en simulant le *réseau de Pétri* correspondant sous jpns ?
- 3.2 Editez et simulez sous jpns le *réseau de Pétri* permettant d'implémenter l'exclusion mutuelle et l'alternance stricte des 2 processus (ping-pong-ping...). Vérifiez que votre solution est correcte.
- 3.3 Implémentez votre solution à l'aide sémaphores dans le programme python.
- 3.4 Puis testez-la. Dessinez le chronogramme sans et avec synchronisation.

4 Le problème des Producteurs Consommateurs

Reprenez les solutions *réseau de Pétri* et *pseudo code* vues en TD (TD 1, exercice 4).

- 4.1 Lisez le code Python contenu dans le fichier `2_producteur_consommateur.py` et le module `tampon_fifo.py`. Exécutez ce programme Python. Que constatez-vous ? Et que constatez-vous en simulant le *réseau de Pétri* correspondant sous jpns ?
- 4.2 Editez et simulez sous jpns le *réseau de Pétri* permettant d'implémenter la contrainte C1 : l'exclusion mutuelle sur les opérations `deposer()` et `retirer()` du tampon. Vérifiez que votre solution est correcte (interblocages, famine, ...).
- 4.3 Implémentez votre solution à l'aide sémaphores dans le programme python. Puis testez-la.
- 4.4 Répéter les questions 4.2 et 4.3 pour résoudre les 2 contraintes C1 (exclusion mutuelle) et C2 (tampon non vide pour retirer), puis les 3 : C1 et C2 et C3 (tampon non plein pour déposer)

5 La chaîne d'assemblage d'un avion

Reprenez les solutions *réseau de Pétri* et *pseudo code* vues en TD (TD 1, exercice 5).

- 5.1 Lisez le code Python contenu dans le fichier `3_aeroplanes.py`. Exécutez ce programme Python. Que constatez-vous ? Et que constatez-vous en simulant le *réseau de Pétri* correspondant sous jpns ?
- 5.2 Editez et simulez le *réseau de Pétri* résolvant ce problème à l'aide de sémaphores. Vérifier qu'aucune configuration interdite n'est atteinte, et qu'aucun interblocage ne se produit.
- 5.3 Implémentez votre solution à base de sémaphores dans le programme Python, puis testez-la.