TP № 3: Synchronisation

Samir Aknine, Antoine Gréa & **Rémy Grünblatt** 15/04/2020

Compétences à acquérir:

- Savoir synchroniser l'exécution des processus en C
- Connaître les cas de blocages
- Comprendre la notion d'atomicité d'exécution

Information: Il est extrement conseillé d'avoir recourir à une machine sous GNU/Linux (par exemple, Ubuntu). Il est possible d'utiliser `https://shell.univ-lyon1.fr` en secours, en ouvrant plusieurs terminaux (notamment dans le cas du processus zombie), mais cela ne devrait pas être la première option pour vous.

Important: Tous les TPs feront l'objet de rendus, et seront notés; les modalités de rendu seront précisées par email ou sur le chat discord de la classe. Il convient donc de bien noter les commandes exécutées, par exemple dans un fichier texte ou markdown, et leur résultats, afin de préparer le rendu du TP.

Exercice 1: Inter-blocage

Question 1: Sémaphores

Étudier, compiler le fichier « `semaphore.c` », et expliquer son fonctionnement. Que se passe-t-il si on inverse les valeurs initiales des sémaphores?

Information: On oubliera pas d'utiliser le manuel pour comprendre le fonctionnement des fonctions qui ne sont pas connues, comme « `semget` », « `semctl` » ou encore « `semop` ».

Question 2: Instinct Parental

Créer un programme où le processus père s'exécute toujours avant le processus fils pour afficher du texte, en utilisant un sémaphore. Tester cette propriété en rendant le père plus lent avec la fonction « `sleep(1)` ».

Exercice 2: Affichage collaboratif

Le but de cet exercice est de créer un programme avec **N** processus fils qui vont afficher chacun leur tour une ligne de fichiers respectif.

Ainsi, au cours du temps, on aura:

- Le processus 1 affiche la ligne 1 du fichier 1
- Le processus 2 affiche la ligne 1 du fichier 2
- ...
- Le processus N affiche la ligne 1 du fichier N
- Le processus 1 affiche la ligne 2 du fichier 1
- Le processus 2 affiche la liggne 2 du fichier 2
- ...

Information: Comme tous les programmes un petit peu complexes, il faut adopter une démarche incrémentale et tester au fur et à mesure que l'on code, et non attendre d'avoir écrit 100 lignes de code pour vérifier si ça compile.

Question 1: Afficher le contenu d'un fichier

Créer la fonction « `void print(int)` » qui prend en entrée un entier **i** et qui affiche le contenu du fichier « `data/fichier_i` » (où *i* est une variable dans le nom du fichier).

Question 2: Contrôle d'exécution

En utilisant un sémaphore, faites en sorte que votre programme crée six processus fils qui afficheront leur fichier respectif avant que le père ait affiché « Résultat : ».

Question 3: Chacun son tour

- Créer six sémaphores (en passant 6 en paramètres de la fonction « `semget` ») et faite attendre chaque processus sur leur sémaphore respectif. Ajouter un affichage avant d'attendre sur le sémaphore.
- Proposer un mécanisme qui permet d'exécuter vos processus dans l'ordre afin que chacun d'eux affiche une seule ligne de leur fichier
- Justifier et expliquer votre solution qui garantie l'ordre d'exécution, et construire une « preuve formelle ».