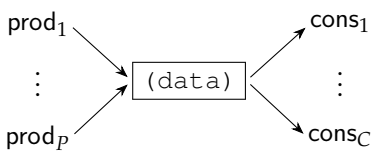
TP 2 – Mutex et sémaphores

Exercice 1 – mutex

1. Écrivez un programme composé de 4 threads (en plus du thread principal) incrémentant chacun un million de fois une variable globale `compteur` de type `long int`. Une fois les threads terminés, affichez la valeur du compteur. Expliquez pourquoi vous n'obtenez pas 4 millions.
2. Pour avoir un résultat conforme aux attentes, on propose d'utiliser une synchronisation à base de *mutex*. Implémentez cette solution.
3. Comparez les performances de la solution correcte avec votre premier programme.

Exercice 2 – tampon borné

On veut écrire un programme composé de P threads *producteurs* (qui fournissent des données à traiter) et de C threads *consommateurs* (qui traitent des données). Les deux types de threads communiquent via un tampon fini de taille fixée B. Chaque donnée est munie d'un identificateur unique (un nombre entier positif). Le fonctionnement est le suivant :



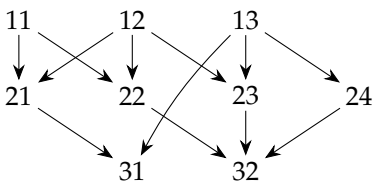
— chaque thread producteur produit un nombre D fixé de données ; pour chaque donnée produite, le thread lui attribue un numéro `n` unique globalement (obtenu par incrémentations successives d'une variable globale), la place dans le tampon et affiche un message de la forme « `P n (v)` » où `n` est le numéro de la donnée et `v` sa valeur ;
— chaque thread consommateur obtient une donnée en la prenant dans le tampon et affiche un message de la forme « `C n (v)` ».

Ces affichages vous permettront de tester la correction de votre programme.

1. Écrivez le programme qui prend en paramètres les nombres D, P, C et B. Les données seront générées pseudo-aléatoirement avec les fonctions de bibliothèque `rand` et `srand`. Vous pouvez exceptionnellement utiliser des variables globales.
2. Pour que le système puisse s'arrêter correctement lorsqu'il n'y a plus de données et que les consommateurs puissent détecter qu'il n'est plus nécessaire d'attendre, on suggère de produire des données *fictives* (par exemple la valeur -1). Lorsqu'un thread consommateur reçoit une telle donnée, il doit s'arrêter. Vous n'oublierez pas de tester votre programme avec des valeurs de P et de C différentes.

Exercice 3 – sémaphores et dépendances

Un graphe de dépendances est un graphe orienté dont les sommets sont des « tâches » et les arcs représentent des contraintes d'ordre entre les tâches. La seule contrainte sur le graphe est qu'il soit sans cycle. Nous savons que l'on peut utiliser un sémaphore pour signaler un événement quelconque d'un thread à un autre.



Vous trouverez ci-contre un tel graphe, où l'on voit par exemple que les tâches 11 et 12 doivent être effectuées pour que les tâches 21 et 22 puissent démarrer (et s'exécuter en parallèle). La tâche 32 ne pourra s'exécuter qu'après la fin des tâches 22, 23 et 24.

Écrivez un programme exécutant les tâches de ce graphe en respectant les dépendances. On décide d'allouer un thread à chaque tâche : le code de chaque tâche est un appel à une fonction `tache()` unique, avec en argument le numéro de la tâche (cette fonction se contentera d'afficher le numéro de la tâche.)