SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PC3R - TME1: Producteurs / Consommateurs

Equipe Enseignante PC2R

03/02/2020

Objectif: Comparer des techniques proposées par différents langages pour empêcher les compétitions: Mutex en *C*, synchronisations en *Java* et `Arc` en *Rust*.

Rendu: Le rendu final pour ce TME doit être une unique archive composée de trois répertoires, contenant chacun les sources pour un langage différent:

- les sources pour C peuvent être, au choix, un unique fichier `.c`, ou un ensemble de fichiers accompagnés d'un `Makefile`
- les sources pour Java peuvent être, au choix, un ensemble de fichiers `.java`, ou un `.jar` contenant, entre autres, les `.java`.
- les sources pour Rust doivent être un projet `Cargo`.

Evaluation: Le rendu est évalué sur:

- sa correction et sa solidité vis-à-vis des caractéristiques concurrentes: absence de compétition, d'attente active, d'interbloquage.
- la simplicité des mécanismes utilisés,
- la lisibilité du code.

1 Description du Système

Le threads du système sont divisés ainsi:

- un thread principal, qui crée le tapis, les autres threads et attend leur terminaison,
- n threads *producteurs* qui fabriquent des paquets et les enfilents sur le tapis,
- m threads *consommateurs* qui consomment les paquets.

Les données du systèmes sont organisées ainsi:

- les *paquets* sont des structures du langage (`struct`, `class`) qui encapsulent une unique chaîne de caractères.
- le *tapis* est une structure du langage qui contient une file de paquet, et une capacité. Il ne peut y avoir un nombre de paquet dans la file du tapis strictement plus grand que sa capacité.
- le *tapis* dispose d'une procédure / méthode permettant à un thread d'*enfiler* un paquet. Si le tapis n'est pas plein, le paquet est mis en bout de file, sinon, le thread attend que le tapis soit non-plein avant de réessayer.
- le *tapis* dispose d'une procédure / méthode permettant à un thread de *défiler* un paquet. Si le tapis n'est pas vide, le paquet en tête de file est retiré et renvoyé au thread comme valeur de retour de la procédure / méthode, sinon, le thread attend que le tapis soit non-vide avant de réessayer.

Producteurs Les threads producteurs sont initialisés avec un nom de produit (différents pour chaque producteur), une cible de production entière. Leur comportement est donné par:

- chaque producteur tourne en boucle tant qu'il a produit moins de paquet que sa cible de production. A chaque tour de boucle:
  - il crée un nouveau paquet avec comme contenu le nom du produit associé concaténé à un entier comptant le nombre de produits déjà créé (par exemple `"Pomme 3"`)
  - il enfile le paquet dans le tapis en utilisant la procédure / méthode décrite plus haut.

Consommateurs Les threads consommateurs sont initialisés avec un identifiant entier et un référence à un compteur. Leur comportement est donné par:

- chaque consommateur tourne en boucle tant que le compteur est supérieur à 0. A chaque tour de boucle:
  - il défile un paquet du tapis.
  - il imprime sur la sortie standard une chaîne correspondant à son numéro de consommateur concaténée au contenu du produit (par exemple `"C1 mange Pomme 3"`).
  - il décrémente le compteur.

Thread principal Le thread principal initialise le tapis, le compteur à une valeur égale à la cible de production des producteurs multipliée par le nombre de producteurs, et lance les producteurs et les consommateurs. Il attend ensuite que le compteur arrive à zéro, puis termine.

2 Implémentation

En `C` Le tapis est une `struct` classique implémentant une FIFO. L'absence de compétition est assurée par les API POSIX (Mutex, Conditions).

En `Java` Le tapis est un objet contenant un `Array` de paquets. L'absence de compétition est assurée par une utilisation judicieuse de `synchronized`.

En `Rust` Le tapis est une `struct` contenant un `VecDeque` de paquets. L'absence de compétition est assurée par l'utilisation d'`ARC` (*Atomically Reference Counted smart pointers*) vers des `Mutex` encapsulant les valeurs partagées (le tapis et le compteur).