# Modèle d’un système de Producteurs - consommateurs

TP4 de programmation concurrente

Soufiane El kharmoudi 16-04-2023

**Objectif pédagogique:**

TP Military convoy

Implémenter un système de producteurs – consommateurs concurrents et asynchrones en respectant les contraintes de développement.

Au début du développement de mon programme, j'ai rencontré des difficultés pour gérer correctement les problèmes de concurrence. Il était complexe de trouver le bon moment pour activer et débloquer les sémaphores afin d'assurer la synchronisation et la coordination entre les différents threads. De plus, il était nécessaire d'ajouter des mutex pour protéger les variables globales et les zones critiques du code, afin d'éviter les problèmes liés à l'accès concurrent et aux modifications inattendues des données.

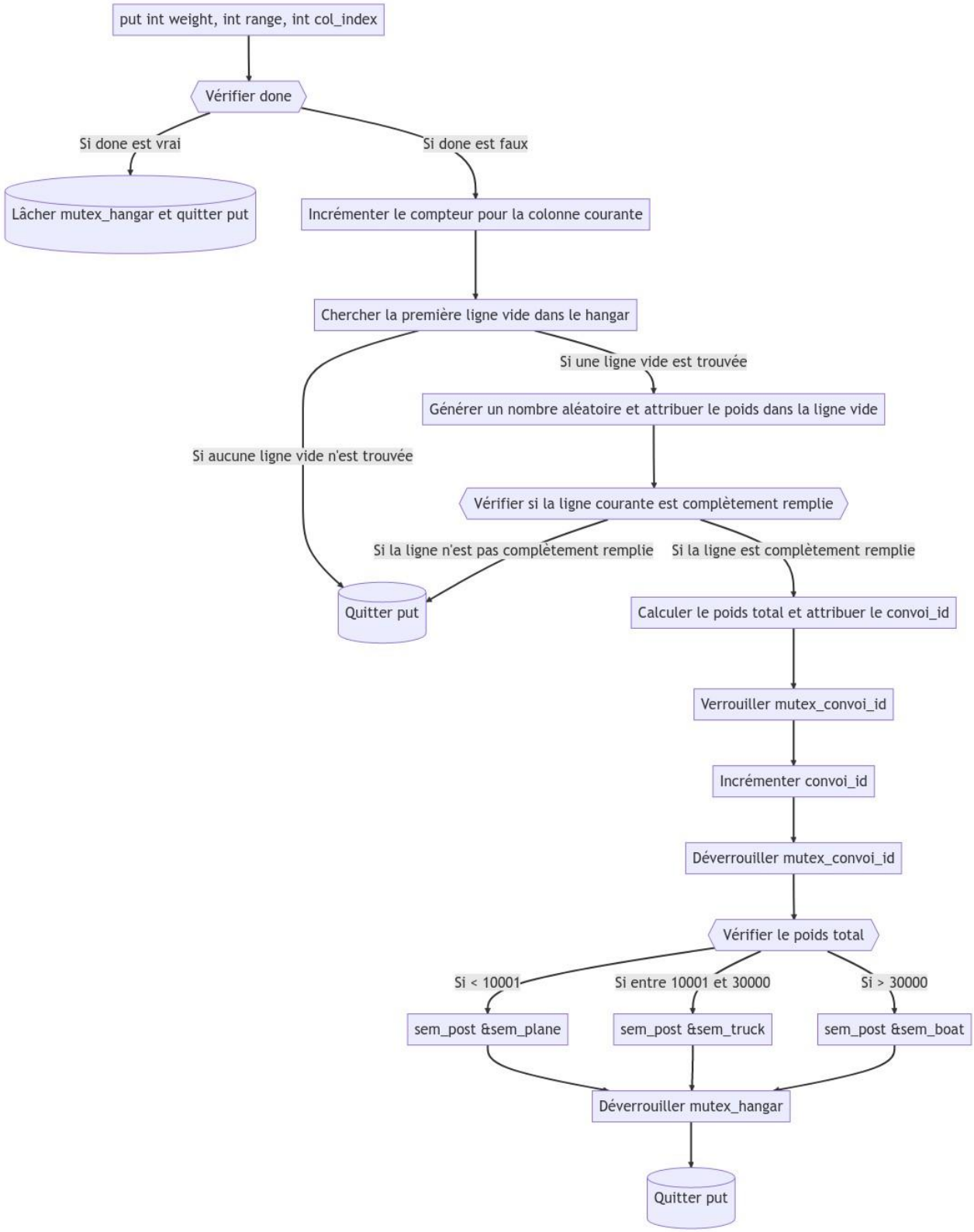
La mise en place des mécanismes de synchronisation, tels que les sémaphores et les mutex, a permis de garantir le bon fonctionnement du programme en évitant les problèmes de concurrence. Cela a nécessité une compréhension approfondie du fonctionnement des différents threads et des interactions entre eux, ainsi qu'une attention particulière portée à l'identification et la protection des zones critiques.

**Producer – put**

Dans ce code, nous avons une fonction **put** qui permet de placer un élément dans le hangar en fonction de son poids, de sa portée et de son index de colonne. Elle utilise un mutex pour protéger l'accès au hangar et vérifie si les conditions requises sont remplies pour les différents types de transport (avion, camion, bateau). La fonction **producer** est destinée aux threads producteurs (conducteur, militaire et matériel). Elle détermine le type de producteur en fonction du nom, définit les paramètres de poids, de portée et d'index de colonne, puis appelle la fonction **put** pour placer l'élément dans le hangar.

La fonction **put** vérifie d'abord si le compteur pour la colonne actuelle a atteint sa limite. Si ce n'est pas le cas, elle incrémente le compteur et cherche la première ligne vide du hangar pour la colonne en cours. Si une ligne vide est trouvée, elle génère un nombre aléatoire dans la plage spécifiée et ajoute le poids à la cellule correspondante du hangar. Ensuite, elle vérifie si la ligne est complètement remplie. Si c'est le cas, elle calcule le poids total et détermine le type de transport en fonction du poids total. Enfin, elle déverrouille le mutex pour le hangar.

La fonction **producer** attend que le sémaphore approprié soit disponible, puis appelle la fonction **put** avec le poids, la portée et l'index de colonne calculés. Si le nombre maximum de convois a été atteint, elle interrompt la boucle et libère les sémaphores appropriés.

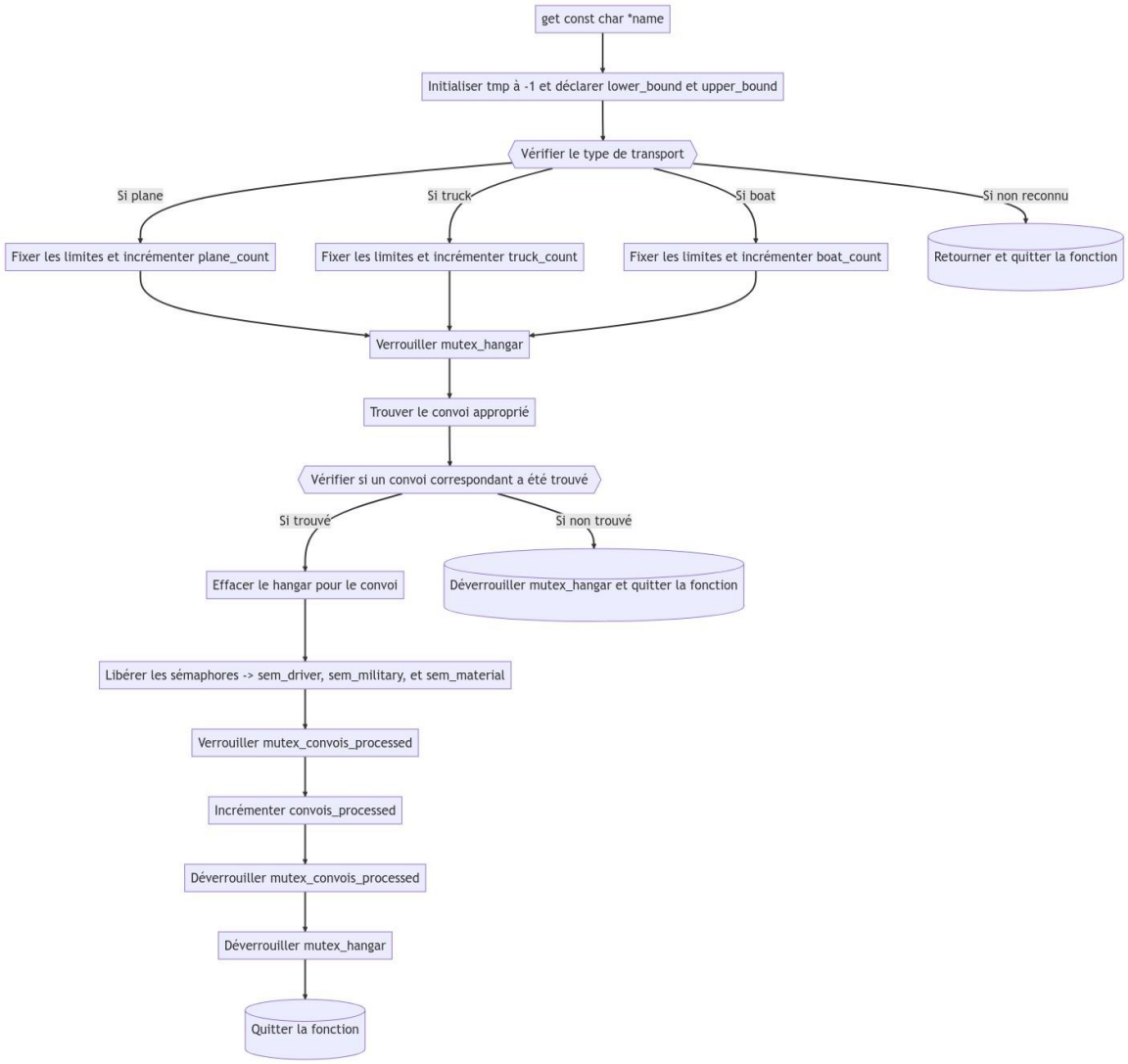

put int weight, int range, int col_index
Vérifier done
Si done est vrai
Si done est faux
Lâcher mutex_hangar et quitter put
Incrémenter le compteur pour la colonne courante
Chercher la première ligne vide dans le hangar
Si une ligne vide est trouvée
Générer un nombre aléatoire et attribuer le poids dans la ligne vide
Si aucune ligne vide n'est trouvée
Vérifier si la ligne courante est complètement remplie
Si la ligne n'est pas complètement remplie
Si la ligne est complètement remplie
Quitter put
Calculer le poids total et attribuer le convoi_id
Verrouiller mutex_convoi_id
Incrémenter convoi_id
Déverrouiller mutex_convoi_id
Vérifier le poids total
Si < 10001
Si entre 10001 et 30000
Si > 30000
sem_post &sem_plane
sem_post &sem_truck
sem_post &sem_boat
Déverrouiller mutex_hangar
Quitter put

**Consumer - Get**

Dans ce code, nous avons une fonction **get** qui récupère et traite un convoi en fonction de son type de transport (avion, camion ou bateau). Elle détermine les limites inférieure et supérieure pour le type de transport et cherche le convoi approprié dans le hangar en vérifiant si son poids se situe dans les limites déterminées. La fonction **consumer** est destinée aux threads consommateurs (avion, camion et bateau). Elle détermine le type de consommateur en fonction du nom, puis appelle la fonction **get** pour récupérer et traiter le convoi.

La fonction **get** utilise un mutex pour protéger l'accès au hangar et recherche le convoi approprié en fonction du poids et du numéro de convoi. Si un convoi correspondant est trouvé, elle efface les données du hangar pour ce convoi, libère les sémaphores et augmente le nombre de convois traités.

La fonction **consumer** attend que le sémaphore approprié soit disponible, puis appelle la fonction **get** avec le nom du consommateur. Si le nombre maximum de convois a été atteint, elle interrompt la boucle et libère les sémaphores appropriés.




Voici un exemple d'affichage de l'exécution du programme:

```
soufiane@sek:~/Bureau/PCO/pcu_tp4_militaire$ make clean
rm -f obj/*.o main
soufiane@sek:~/Bureau/PCO/pcu_tp4_militaire$ make
gcc -Wall -Wextra -pedantic -std=c11 -lpthread -g -Iheader -c main.c -o obj/main.o
gcc -Wall -Wextra -pedantic -std=c11 -lpthread -g -Iheader -c producer.c -o obj/producer.o
gcc -Wall -Wextra -pedantic -std=c11 -lpthread -g -Iheader -c consumer.c -o obj/consumer.o
gcc -Wall -Wextra -pedantic -std=c11 -lpthread -g -Iheader -c shared_resources.c -o obj/shared_resources.o
gcc -Wall -Wextra -pedantic -std=c11 -lpthread -g -Iheader obj/main.o obj/producer.o obj/consumer.o obj/shared_resources.o -o main
soufiane@sek:~/Bureau/PCO/pcu_tp4_militaire$ make run
./main
Convoys executed: 200
Planes dispatched: 16
Trucks dispatched: 70
Boats dispatched: 114
Total Planes + Trucks + Boats dispatched: 200
soufiane@sek:~/Bureau/PCO/pcu_tp4_militaire$ make run
./main
Convoys executed: 200
Planes dispatched: 16
Trucks dispatched: 81
Boats dispatched: 103
Total Planes + Trucks + Boats dispatched: 200
soufiane@sek:~/Bureau/PCO/pcu_tp4_militaire$
```