Kevin Auberson, Alexandre Shyshmarov

Introduction

Le présent rapport détaille le développement d'un logiciel de simulation d'un salon de coiffure utilisant des moniteurs Mesa pour la synchronisation des threads. L'objectif principal consiste à approfondir le concept de moniteur et de synchronisation des threads afin de mettre en oeuvre la logique de fonctionnement d'un salon de coiffure.

Description des fonctionnalités du logiciel

Le logiciel simule un salon de coiffure composé d'un barbier et de clients. Chaque entité est modélisée comme un thread distinct. Les principales fonctionnalités implémentées sont les suivantes :

Barber

La classe Barber représente le barbier du salon. Son fonctionnement est concentré dans la fonction run(), qui implémente la machine d'état de la donnée dans une boucle while. La première condition étant la plus importante, elle vérifie si le salon est en service.

Le barbier attend que les clients s'installent sur la chaise de coiffure, puis les coiffe.

De plus à la fin du travail, nous avons rajouté l'animation de sommeil pour le barbier, car à la fin de chaque journée de travail, il est important de se reposer !

Client

La classe Client représente les clients du salon. Comme pour le barbier, toute la logique est dans le run() qui est une implémentation de la machine d'état du client. Nous avons décidé d'utiliser une boucle while qui à la même première condition que le barbier. Les clients tentent d'accéder au salon, attendent leur tour, se font coiffer, puis repartent en attendant que leurs cheveux repoussent.

PcoSalon

La classe PcoSalon utilise 4 moniteurs Mesa pour gérer la synchronisation et la coordination entre le barbier et les clients. Elle comprend plusieurs méthodes pour gérer l'accès au salon, les animations, les notifications entre le barbier et les clients, et la terminaison du service.

Choix d'implémentation

getNbClient et isInService

Ce sont des getters que nous utilisons afin de gérer le salon. Les deux méthodes ont la même conception. Nous créons une variable temporaire pour le retour, à laquelle nous assignons la valeur de la ressource partagée (clientInSalon.size() ou inService).

endService

Cette méthode met fin au service du salon en assignant la valeur false au boolean inService qui permet de vérifier l'état du salon.

accessSalon

La fonction accessSalon est essentielle pour gérer l'entrée des clients dans le salon de coiffure simulé. Cette fonction implémente la logique de l'accès des clients au salon en tenant compte de la capacité maximale du salon et en orchestrant l'arrivée des clients et leur attente jusqu'à ce qu'ils puissent être coiffés par le barbier. En première partie, nous avons le refus du client si le salon est plein, cette condition dépend de la taille de la queue.
Si la file n'est pas pleine alors on push le client dans la file (clientInSalon). De cette façon en plus d'être plus simple, nous permet de nous affranchir du system de ticket proposé en cours que nous avons essayé d'implémenter cependant sans succès. Après ça nous préparons la variable chairPos qui s'occupe de déterminer la chaise d'attente du client.

(nextChair) est incrémenté pour attribuer la prochaine position de chaise disponible. Afin d'éviter que nextChair ne dépasse la capacité maximale de chaises d'attente (MAX_WAITING), l'opération % (modulo) est utilisée.

Ensuite, nous effectuons l'animation et la notification au barbier, si ce dernier dort.
La dernière partie du code (le if et le while) s'occupe de gérer l'attente de client. Le if est responsable de l'animation du client qui va sur une des chaises d'attente.
Le while est l'attente du thread. On effectue l'attente grâce à un moniteur de Mesa. Le while nous permet de sélectionner le bon thread. Plus loin dans le code, on effectue un notifyAll() ce qui réveille tous les threads de clientWait, seul le thread avec un clientId correspondant au front de la queue pourra sortir du while.

goForHairCut

Cette méthode représente le moment où le client rejoint la chaise de coiffure pour se faire coiffer. Le client attend que le barbier soit prêt à commencer la coupe de cheveux. Pendant ce temps, l'animation associée à cette action est déclenchée. Lorsque le barbier est prêt, la méthode barberCutting.notifyOne() est appelée pour signaler au barbier que le client est prêt.

goToSleep

La méthode goToSleep() permet au barbier de dormir s'il n'y a pas de client à coiffer. Elle utilise un mécanisme de condition (barberSleep.wait(&_mutex)) pour permettre au barbier de s'endormir et d'être réveillé lorsqu'un client entre dans le salon.

waitClientAtChair & pickNextClient

Ces méthodes sont utilisées par le barbier pour sélectionner le prochain client à coiffer, attendre que le client soit prêt sur la chaise de coiffure. Elles utilisent des verrous et des variables de condition pour synchroniser les actions du barbier avec celles des clients et attendre la fin des animations clients.

beautifyClient

Cette méthode permet au barber d'effectuer l'animation pour couper les cheveux du client. Elle notifie ensuite les clients en attente qu'il a terminé pour que le prochain puisse commencer à venir sur la chaise de travail. Puis libère, le client de la file d'attente.

Tests effectués

Test du temps

Le programme a été en fonction pendant plus 1h30 afin de vérifier sur une plus grande plage d'exécution si des problèmes surviendraient. Ce test a été validé avec l'apparition d'aucun problème est un fonctionnement continuel.

Cas limite

Zéro siège d'attente

Dans ce scénario, le salon de coiffure est configuré sans aucune chaise d'attente (NB_SIEGES = 0). Le test a été réalisé pour évaluer la réaction du système lorsque les clients arrivent, mais ne peuvent pas attendre, car il n'y a aucune chaise disponible.

Résultat du test :

- Un seul client se fait couper les cheveux et les autres clients sont refusés d'entrer dans le salon si aucun siège d'attente n'est disponible.
- Ils font un tour et reviennent ultérieurement pour savoir si le siège de travail est libre.

Grand nombre de client avec peu de chaise

Ce test simule un scénario où un grand nombre de clients tentent d'entrer dans le salon alors qu'il y a peu de chaises d'attente disponibles (MAX_WAITING est considérablement inférieur au nombre de clients).

Résultat du test :

- Les clients excédant la capacité d'attente sont refusés à l'entrée du salon.
- Ils font un tour et retentent leur chance ultérieurement.
- Les clients admis attendent leur tour pour se faire couper les cheveux.

Petit nombre de client avec beaucoup de chaise

Ce scénario suppose un petit nombre de clients cherchant à entrer dans le salon, alors que le salon dispose de nombreuses chaises d'attente (MAX_WAITING est beaucoup plus grand que le nombre de clients).

Résultat du test :

- Les clients sont immédiatement admis dans le salon sans attendre, car il y a suffisamment de chaises d'attente disponibles.
- Les clients admis attendent leur tour pour se faire couper les cheveux.

Grand nombre de client avec beaucoup de chaise

Dans ce test, un grand nombre de clients essaient d'entrer dans le salon de coiffure qui possède un grand nombre de chaises d'attente disponibles (MAX_WAITING est beaucoup plus grand que le nombre de clients).

Résultat du test :

- Les clients sont admis dans le salon car il y a un grand nombre de chaises d'attente.
- Il n'y a pas de refus d'entrée de clients.
- Les clients admis attendent leur tour pour se faire couper les cheveux.

Fin de service

Ce test simule un scénario où la fin de journée s'effectue correctement avec la fin d'admission de client dans le salon ainsi que la terminaison des derniers clients encore dans le salon.

Résultat du test :

- Le système permet à tous les clients présents dans le salon d'être coiffés avant de fermer.
- Aucun nouveau client n'est admis après que le barbier a décidé de ne plus accepter de clients.
- Une fois tous les clients servis et partis, le barbier s'endort.

Conclusion

Le logiciel de simulation du salon de coiffure a été développé avec succès en utilisant des moniteurs Mesa pour la synchronisation des threads. Les tests effectués ont démontré une réponse adéquate du système dans des situations diverses.