HEIG-VD — PCO

Laboratoire 5 – Rapport

Loïc Herman

14 décembre 2023

1 Description des fonctionnalités du logiciel

L'objectif de ce laboratoire est de simuler un salon de coiffure avec une file d'attente tout en traitant les clients dans l'ordre d'arrivée.

Le système doit en outre prendre en charge la situation de fin de journée, dans laquelle les derniers clients seront coiffés et les autres clients devront rentrer chez eux.

2 Choix d'implémentation

Toute la gestion de la synchronisation a été effectuée dans le salon, les clients et le coiffeur effectuent simplement des appels à leurs interfaces respectives du salon.

Le salon gère la synchronisation des clients à l'aide d'un moniteur de mesa et d'une structure de donnée classique pour la gestion de l'ordre d'arrivée.

2.1 Gestion des threads

L'implémentation des threads du coiffeur et des clients répond dans l'ensemble aux diagrammes d'état présentés dans la consigne. Une petite exception a été faite pour le coiffeur, il appellera dans tous les cas le client suivant, même dans la situation où le coiffeur a été réveillé par un nouveau client.

On notera aussi que pour prendre en charge le cas où la fin de journée arrive quand le coiffeur est au lit, le coiffeur sera forcément réveillé et une vérification du nombre de clients restants sera effectuée avant de terminer le thread du coiffeur.

2.2 Ordre d'arrivée

Quand un client arrive dans le salon et une fois le mutex obtenu, il est en premier placé en queue d'une `std::queue`, permettant ainsi au moniteur d'avoir une référence sur l'ordre d'arrivée.

Si un client est en tête de la queue, c'est le client qui devrait être en train de se faire embellir.

2.3 Synchronisation

Le salon gère la synchronisation à l'aide de 4 variables de condition qui servent à faire attendre un client ou le coiffeur, mais pas les deux simultanément afin d'éviter des blocages. Ainsi, une variable de condition est soit attendue par le coiffeur et notifiée par le client, ou, au contraire, attendue par le client et notifiée par le coiffeur. Un diagramme de cette synchronisation avec les attentes effectuées par les clients et le coiffeur est disponible en figure 1.

2.3.1 Variable `wakeUp`

Quand il n'y a plus de clients en attente, le coiffeur ira se coucher et attendra immédiatement sur la variable `wakeUp` qui sera notifiée lorsqu'un client entre dans le salon ou que le salon se ferme pour que le coiffeur puisse fermer boutique.

2.3.2 Variable `chairEmpty`

Quand un client se dirige vers la chaise de travail du coiffeur et s'il n'est pas au sommet de la file d'attente, il va effectuer un *wait* sur la variable `chairEmpty`. Cela permet d'éviter qu'un client se dirige immédiatement sur la chaise de travail une fois qu'il s'est assis sur une des chaises de la salle d'attente.

Le coiffeur s'occupe de notifier cette variable quand il appelle le prochain client, il notifiera tous les threads en attente, car seul celui qui se trouve en tête de la file d'attente sortira de la boucle `while` qui englobe l'appel au `wait`.

2.3.3 Variable `clientReady`

Quand le coiffeur appelle un nouveau client, il attendra que ce dernier soit prêt pour la coup au moyen de la variable de condition `clientReady`.

Une fois que le client à terminé son animation et est confortablement installé sur la chaise de travail, il notifiera donc le coiffeur au moyen de cette variable pour qu'il puisse commencer la coupe.

Cette variable de condition est associée à un booléen `clientSeated` pour les rares cas où le client serait assis et notifie la variable de condition avant que le thread du coiffeur ait effectué son `wait`. Ainsi, le thread du barbier vérifiera d'abord si personne n'est assis avant d'effectuer le `wait` résultant à un blocage.

2.3.4 Variable `haircut`

Quand le client est confortablement installé sur la chaise, il attend la fin de sa coupe au moyen de la variable de condition `haircut`, qui sera notifiée par le coiffeur une fois la coupe terminée.

2.4 Gestion des animations

Une des erreurs qui est survenue au début de la conception du système était de déclencher les animations du barbier depuis un thread client, ce qui avait pour effet de provoquer un blocage des deux threads. Nous avions originalement essayé d'éviter d'ajouter la variable de condition `wakeUp` mais elle est importante car le thread du coiffeur doit être placé en attente tant qu'il est au lit pour éviter de sauter des étapes de la machine d'état.

3 Tests effectués

Initialement, nous avons testé le bon déroulement du salon avec les paramètres par défaut. Nous avons donc vérifié qu'un client unique soit assis sur une chaise simultanément. L'ordre d'arrivée des clients a aussi été contrôlé pour s'assurer que le premier client arrivé soit le premier servi.

Nous avons testé le cas où un client viendrait à réveiller le coiffeur lorsqu'il n'y a personne dans le salon et s'assurer que la situation se déroule normalement en modifiant le nombre total de clients pour éviter d'avoir une rotation constante.

En changeant les valeurs par défaut, nous avons également pu tester le cas où il n'y a aucun client qui vient dans le salon pour s'assurer que le barbier qui est au lit puisse correctement fermer le salon quand nous terminons le programme. Nous avons également testé la terminaison dans la situation où il n'y a qu'un seul client dans le salon qui est en train de se faire coiffer, et quand il y a plusieurs clients en attente pour s'assurer que tous les clients en attente reçoivent leur coupe et que le salon soit fermé une fois qu'il ne reste plus que le coiffeur.

Nous avons testé le cas où le salon n'aurait aucune chaise dans la salle d'attente et nous avons constaté que le salon tourne sans trop de problèmes mais d'une façon pas très optimale car le coiffeur va automatiquement retourner se coucher avant de se faire immédiatement réveiller par un nouveau client. Cette situation n'étant pas nécessairement spécifiée, nous pensons que ce comportement est adéquat.

Nous avons ensuite répété les tests précédents en ajoutant un délai aléatoire lors des événements multi-threadés avec `PcoManager::getInstance()->setMaxSleepDuration(1e5);` pour s'assurer que nous étions sans blocages dû à des attentes sur des événements envoyés avant le début de l'attente. Par exemple, on ne voudrait pas que le coiffeur attende que le client s'installe alors qu'il est déjà installé.

Un dernier test pour la bonne mesure a été d'augmenter le nombre de chaises d'attente tel qu'il soit plus élevé que le nombre de clients, nous avons donc pu observer que le positionnement en salle d'attente est correctement cyclique et que les clients sont traités dans leur ordre d'arrivée.

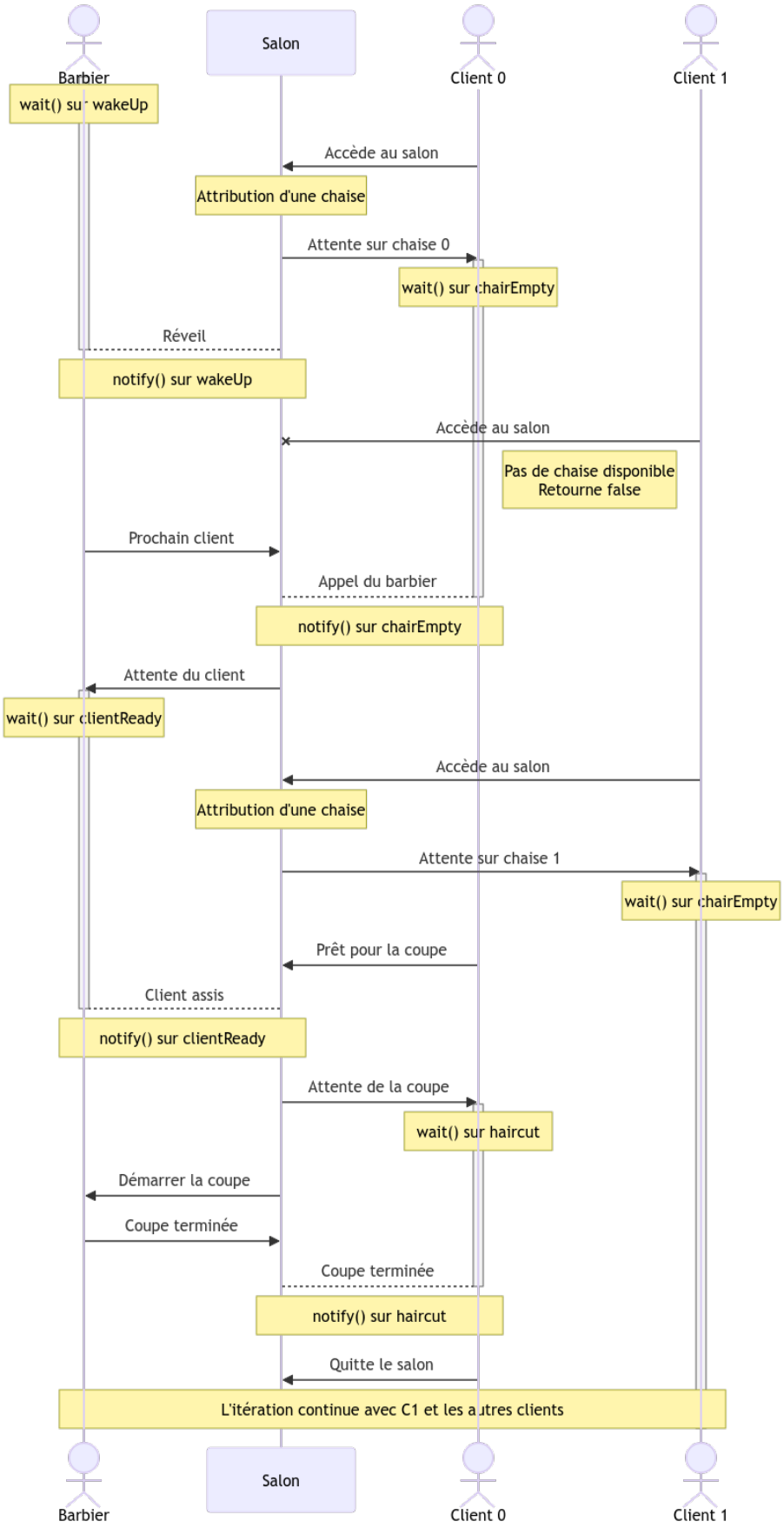

FIGURE 1 – Plan de synchronisation des threads.
LÉGENDE – Les zones activées des acteurs représentent les instants où le thread sera en attente via un appel au moniteur. Quelques situations sont décrites mais le diagramme n'est pas nécessairement exhaustif.