# Modélisation par moniteur de Mesa

Auteurs: Arthur Junod et Edwin Haeffner

## Description des fonctionnalités du logiciel

Ce logiciel simule le fonctionnement d'un salon de coiffure à l'aide de plusieurs classes :

- Une classe principale modélisant le salon de coiffure
- Une classe modélisant le coiffeur
- Une classe modélisant les clients

Chaque entité (salon, coiffeur, clients) possède son propre thread pour simuler son comportement de manière indépendante.

Le salon est le thread principal. Le coiffeur et les clients tournent dans leurs propres threads en parallèle. Cette conception permet de reproduire le fonctionnement concurrent d'un véritable salon de coiffure.

L'objectif de ce programme est de simuler un salon de coiffure en utilisant les moniteurs MESA vus en cours afin de synchroniser les différents threads. Le but ultime est de modéliser le meilleur salon de coiffure de l'HEIG :^) !

## Choix d'implémentation

### client.cpp

La routine du client se trouve dans une boucle `while(true)` tant que le salon de coiffure est ouvert pour ainsi suivre rigoureusement le schéma donné dans le cahier des charges.

Si le salon est fermé, le client execute la commande `goHome`, sort de la boucle avec un `break` et ensuite le thread se termine.

Si le service se finit quand le client est en train d'attendre sur les chaises, il va patienter jusqu'a ce que le coiffeur lui coupe les cheveux, aller faire un tour afin que ses cheveux repousses et finalement rentrer à la maison. Cet ordre a été défini car s'il n'attendait pas que ses cheveux repoussent, il disparaissait du salon pour instantanément renter chez lui ce qui n'avait pas de sens au niveau de l'affichage.

### barber.cpp

Nous avons utilisé une boucle while dans le coiffeur afin de simuler la boucle de décision qui était décrite par l'organigramme dans le cahier des charges.

C'est de cette boucle que nous sortons si nous devons arrêter le service, donc arrêter le thread coiffeur, quand l'utilisateur décide d'arrêter le programme. Si on sort de la boucle, le thread se termine et le programme avec celui-ci.

### pcosalon.h

Dans ce fichier .h, les seuls changements que nous avons faits sont l'addition de variables dans la partie `protected:`

**5 variables de type `PcoConditionVariable`**

| Nom var | Description |
|---|---|
| `_zzzBarber` | La condition qui active et désactive l'attente lors de l'assoupissement du barbier (lorsqu'il n'y a pas de clients) |
| `_waitingForClient` | La condition d'attente lorsque le coiffeur appelle un client à la chaise |
| `_cutting` | Permet au client d'attendre que l'animation de coupe soit fini |
| `_waitingForCut` | Condition utilisée par le client pour attendre son tour lorsqu'il est sur une des chaises d'attente du salon |
| `_waitingToClose` | Utilisée pour attendre que tous les clients déjà dans le salon aient une belle coiffure avant de le fermer |

**2 variables de type `bool`**

| Nom var | Description |
|---|---|
| `_isSleeping` | Utilisée pour vérifier si le coiffeur dors, le cas échéant, le client le réveillera. |
| `_isOpen` | Indique si le salon est ouvert ou non, initialisée à `true` vu que le salon ouvre à l'ouverture du programme. |

**4 variable de comptage de type `unsigned int`**

| Nom var | Description |
|---|---|
| `_capacity` | La capacité maximale du salon, utilisée pour l'affichage des nombres de chaises disponibles dans le salon et de la logique autour de l'attente des clients |
| `_nbInSaloon` | Utilisée pour savoir combien de client sont actuellement en train d'attendre sur les chaises du salon, une fois qu'un client se dirige vers la chaise de travail, on décrémente cette variable pour laisser un nouveau client entrer dans le salon et s'asseoir sur une des chaises disponible. |
| `_ticket` | Variable pré-incrémentée à chaque fois qu'un client prend un ticket. Utilisée pour la gestion de l'ordre de passage des clients. |
| `_order` | Numéro de l'ordre de passage des clients, incrémentée par le coiffeur lorsqu'il appelle un nouveau client. |

Et finalement `_chairs`, 1 tableau de type `bool` pour la gestion des chaises. Si une chaise est libre, la valeur du tableau est `false`, sinon elle est `true`.

## pcosalon.cpp

**Gestion de l'ordre**

Si le salon est ouvert ou bien qu'il y a de la place, nous lançons la logique liée à l'attente des clients dans le salon.

`accessSalon()` utilise un système de tickets.

Chaque client prend son ticket quand il accède au salon et choisit sur quelle chaise il va s'asseoir parmi celles qui sont libres. Le client ensuite attends gentiment sur sa chaise grâce à un moniteur MESA, puis le coiffeur, une fois arrivé à la chaise de travail, libère tous les clients qui attendent sur leurs chaises avec un `notifyAll()`. Mais, le moniteur se trouve dans une boucle while qui ne laisse passer que le ticket le moins récent, ce qui permet de garantir une queue FIFO. Le client va ensuite libérer sa chaise et aller se faire couper les cheveux pendant que les autres clients qui ont un numéro de ticket plus élevé vont continuer à attendre.

Le coiffeur, en appelant tous les clients, va incrémenter une variable `_order` et on compare le numéro de ticket à cette variable. Si le ticket du client est d'un numéro plus élevé que celui de la variable `_order`, alors il va continuer d'attendre. On peut faire le lien avec l'attente de son tour à la poste, on entend le bip, on regarde le numéro sur l'écran et on regarde s'il est égal à notre ticket ! C'est pratiquement le même système ici.


Gestion de file d'attente à la poste

**goToSleep()**

Vu que le coiffeur va dormir s'il n'y a plus de client, on vérifie dans cette fonction si le salon est ouvert ou non, et s'il est fermé, on indique au moniteur `_waitingToClose` que la fermeture est possible et qu'on peut enfin mettre fin au thread du coiffeur.

Nous l'avons implémenté de cette façon car `goToSleep()` est appelée systématiquement lorsqu'il n'y a plus de client. Cela semblait donc logique de vérifier la condition de fermeture du thread coiffeur à cet endroit.

**Commentaires généraux**

Dans ce labo, il fallait bien faire attention à incrémenter le nombre de clients avant de faire l'animation. Et de manière générale, il fallait être attentif à bien faire son code pour que l'appel aux animations ne cause de problème dû aux relâchements du mutex....

Au début du projet, nous avions mis l'attente dans le `goForHaircut()` mais apparemment d'après ce que nous avons entendu d'un assistant (vous :^)), la logique devrait être mise dans `accessSalon()`. Donc c'est ce que nous avons fait.

## Test

Nous pouvons tester notre programme visuellement grâce aux animations et "textuellement" grâce aux messages que nous avons inséré dans notre code (`cout` ou `interface->consoleAppendTextClient()`/`interfcae->consoleAppendTextBarber()`).

| Test | Résultat |
|---|---|
| Vérification que les clients choisissent tous une chaise différente | PASS |
| Vérification que la file d'attente pour aller se faire couper les cheveux soit FIFO | PASS |
| Vérification que lorsque l'on ferme le salon tous les clients à l'intérieur finisse de se faire servir | PASS |
| Verification que lorsqu'il n'y a aucun clients, le coiffeur dorme bien (à faire avec une seule place dans le salon et un client) | PASS |
| Vérification que le barbier arrive bien avant les clients à la chaise de travail pour les appeler | PASS |
| Vérification que les clients ne puisse pas rentrer si le salon est plein | PASS |
| Vérification que les clients aille faire un tour avant de rentrer chez eux quand le salon est fermé (afin qu'ils ne disparaissent pas) | PASS |

Au final, on voit visuellement que le salon fonctionne bien, les clients se suivent de façon fluide et dans un ordre FIFO. Il y a très peu de pauses, voir aucune, ce qui signifie que le programme est bien géré et que les threads ne se bloquent pas entre eux. Donc pour nous l'objectif est atteint.

## Conclusion

Ce laboratoire nous a permis de mettre en pratique les moniteurs MESA vus en cours. Nous avons pu bien voir à quel point l'utilisation des sémaphores sont simplifiées avec l'utilisation de ces moniteurs !