HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Yann Thoma
Florian Vaussard

Laboratoire de Programmation Concurrente

semestre automne 2023

Modélisation par moniteur de Mesa

Temps à disposition : 6 périodes (trois séances de laboratoire)

1 Objectifs pédagogiques

Réaliser une modélisation d'un problème concurrent à l'aide de moniteurs de Mesa.

2 Énoncé du problème

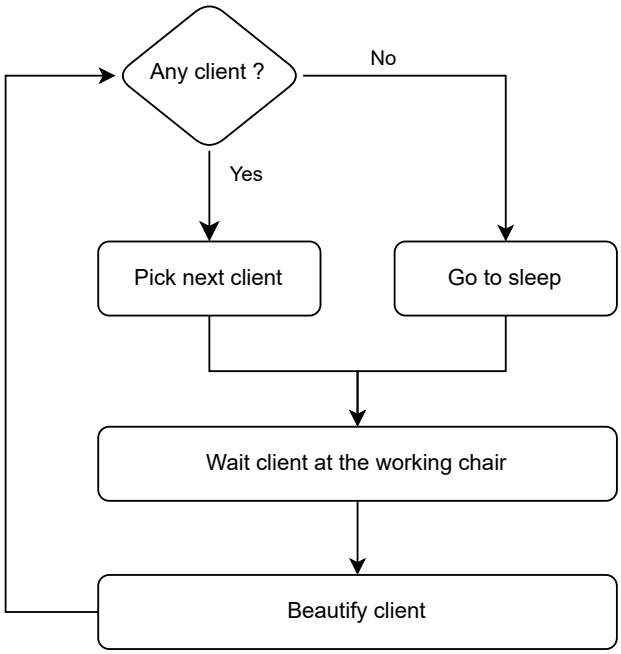
Dans ce laboratoire nous allons modéliser une boutique de barbier.

La boutique est composée d'un salon et d'une salle d'attente ayant un nombre défini de chaises (`NB_SIEGES`). Un certain nombre de clients (`NB_CLIENTS`) vont tenter de se faire couper les cheveux. Ces deux constantes se trouvent dans le fichier `main.cpp` et peuvent être modifiées au besoin.

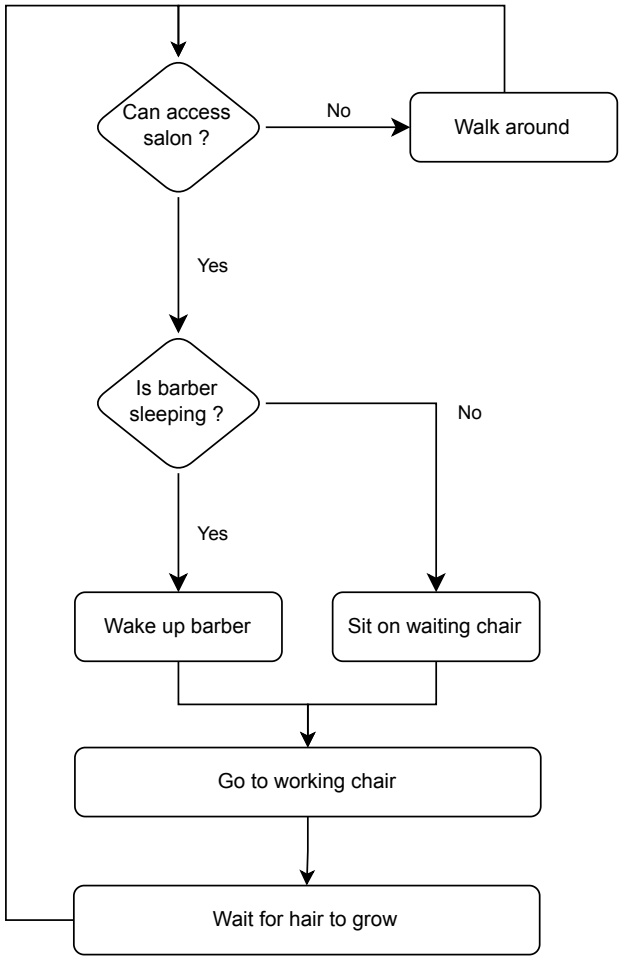
Concept global : le barbier possède un salon qui comporte plusieurs places assises pour que les clients puissent patienter dans le salon jusqu'à ce que leur tour arrive. Les clients qui souhaitent se faire couper les cheveux entrent dans la boutique lorsque celle-ci a suffisamment de place pour les accueillir. Ceux-ci se font coiffer et repartent le temps que leurs cheveux repoussent. Si un client ne peut pas accéder au salon, il va faire un tour et revenir plus tard pour retenter sa chance. Lorsque le barbier n'a plus de client, il part faire une sieste pour se reposer jusqu'à qu'un client vienne le réveiller.

Les comportements peuvent être représentés avec les machines d'états suivantes :

Machine d'état du barbier



Machine d'état du client



3 Cahier des charges

Nous souhaitons réaliser un logiciel simulant les interactions entre le barbier et ses clients. Le laboratoire portera sur la gestion des accès au salon ainsi que des interactions entre les acteurs.

Vous devez :

1. Mettre en place les routines des clients (`run()` dans `client.cpp`) et du barbier (`run()` dans `barber.cpp`) selon les machines d'états leur correspondant. Les méthodes à appeler sont celles de la classe PcoSalon explicitée dans le fichier `pcosalon.h`. Les clients auront accès aux méthodes du salon à travers l'interface `SalonClientInterface` (saloncliententerface.h) et le barbier aura accès aux méthodes de l'interface `SalonBarberInterface` (salonbarberinterface.h). Les méthodes de ces interfaces vous permettront de mettre en place leurs comportements respectifs.
2. Implémenter les méthodes du salon (dans `pcosalon.cpp`) au moyen d'un moniteur de Mesa afin de gérer la concurrence et les animations (voir ci-dessous). Vous pouvez rajouter les variables privées nécessaires dans la déclaration de la classe (`pcosalon.h`). La classe contient déjà un mutex nommé `_mutex`, faites donc attention à utiliser celui-ci pour votre moniteur.
3. Une fois que votre programme commence à fonctionner, assurez vous que les clients soient servis dans leur ordre d'arrivée. Si ce n'est pas le cas, modifiez votre code en conséquence.
4. Implémenter la terminaison du programme en utilisant les méthodes `endService()` et `isInService()` dans le fichier `pcosalon.cpp`. Pour terminer le programme, il suffit qu'une entrée utilisateur soit déclenchée dans le terminal. Les clients présents dans le salon doivent être coiffés par le barbier, et lorsque tous les clients sont partis et que le salon est vide le

barbier a terminé sa journée et le programme se termine. Il n'y a pas d'autres contraintes strictes concernant la fin du programme.

3.1 Animation

Des méthodes privées de la classe PcoSalon servent à l'animation de l'interface graphique. Elles sont toutes nommées `animation...()`. Il vous est demandé de les appeler au bon moment depuis les fonctions de votre moniteur Mesa afin d'animer correctement le salon. Ces méthodes sont codées de manière à ce que l'animation se fasse hors du moniteur (`_mutex.unlock()`/`_mutex.lock()` autour de l'animation) afin que d'autres threads puissent rentrer dans le moniteur durant l'animation. Certaines animations sont bloquantes tandis que d'autres ne le sont pas. Vous retrouverez des détails sur ce comportement dans les entêtes des fonctions concernées. S'il n'y a pas de précision concernant l'animation, celle-ci est bloquante, dans le cas contraire, l'entête de la fonction vous fournira l'information.

3.2 Compilation

Contrairement aux précédents laboratoires, celui-ci utilise *cmake* au lieu de *qmake* pour la génération des makefiles. Ceci sera transparent pour vous, la seule différence étant qu'il faut ouvrir le "projet" `CMakeLists.txt` au lieu d'un fichier `.pro` si vous utilisez QtCreator. Pour les adeptes de la ligne de commande, vous connaissez certainement *cmake*, et si ce n'était pas le cas, il doit bien y avoir un professeur ou un assistant à disposition pour vous aider.

4 Travail à rendre

— Utilisez le script de rendu fourni pour générer l'archive pour Cyberlearn

— La description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans votre rapport.

— Inspirez-vous du barème de correction pour connaître là où il faut mettre vos efforts.

5 Barème de correction

Conception, conformité au cahier des charges et simplicité	40%
Exécution et fonctionnement	20%
Rapport	30%
Documentation du code (commentaires, entêtes...) et codage	10%