

## Rapport du test d'évaluation

## Exercice MCD

Le premier exercice que j'ai choisi de traiter est celui du modèle conceptuel de donnée car plus classique et moins long.

Le modèle ci-dessous permet de répondre aux différentes demandes de gestion de l'énoncé :

- La maintenance du parc est assurée grâce aux quatre premières tables « **Hôtel** », « **Chambre** », « **Classe** » et « **Catégories** ».
- Les disponibilités pour un *IdHotel* donné à une date (ou plage) précise peuvent être obtenues à l'aide des dates des « **Réservation** » et leur clé étrangère *NoChambre* (les chambres étant liées à leur hôtel avec *IdHotel*).
- L'enregistrement des réservations est directement faites dans la table « **Réservation** ».
- L'enregistrement de l'arrivée d'un client à l'hôtel est faite par mise à jour de l'attribut *DateArriveeClient* de sa réservation.
- La facture pour une réservation *IdReservation* qui touche à son terme peut être obtenue à l'aide des informations contenues dans quasiment toutes les tables. Pour calculer le prix, il nous faut le *tarif* de la chambre, le nombre de personnes présentes (*NbPersonne*) et l'ensemble des prestations utilisées.

Schématiquement *IdReservation* → *NoChambre* → *IdHotel* → *Classe*

*NoChambre* → *Categorie*

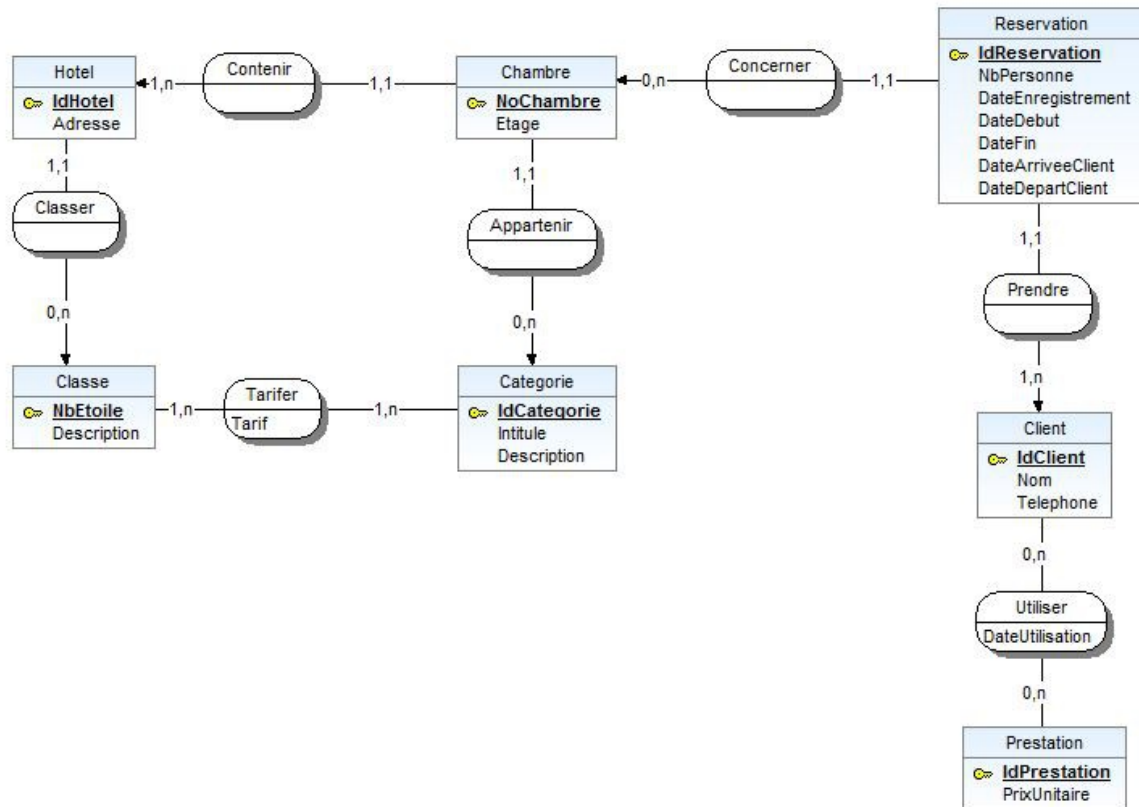
*Classe* + *Categorie* → *tarif*

*IdReservation* → *NbPersonne*

*IdReservation* → *IdClient* → *IdUtilisation* (dont la date est  
contenue dans l'intervalle du séjour)

- La liste des arrivants est obtenue en listant les clients de réservations dont l'intervalle [DateDebut - DateFin] contient celle du jour donné.
- L'état d'occupation est obtenu de la même manière que les disponibilités, en filtrant les chambres par leur clé étrangère *IdCategorie*.

## Modèle Conceptuel de Donnée :



## Exercice de programmation C#/.NET

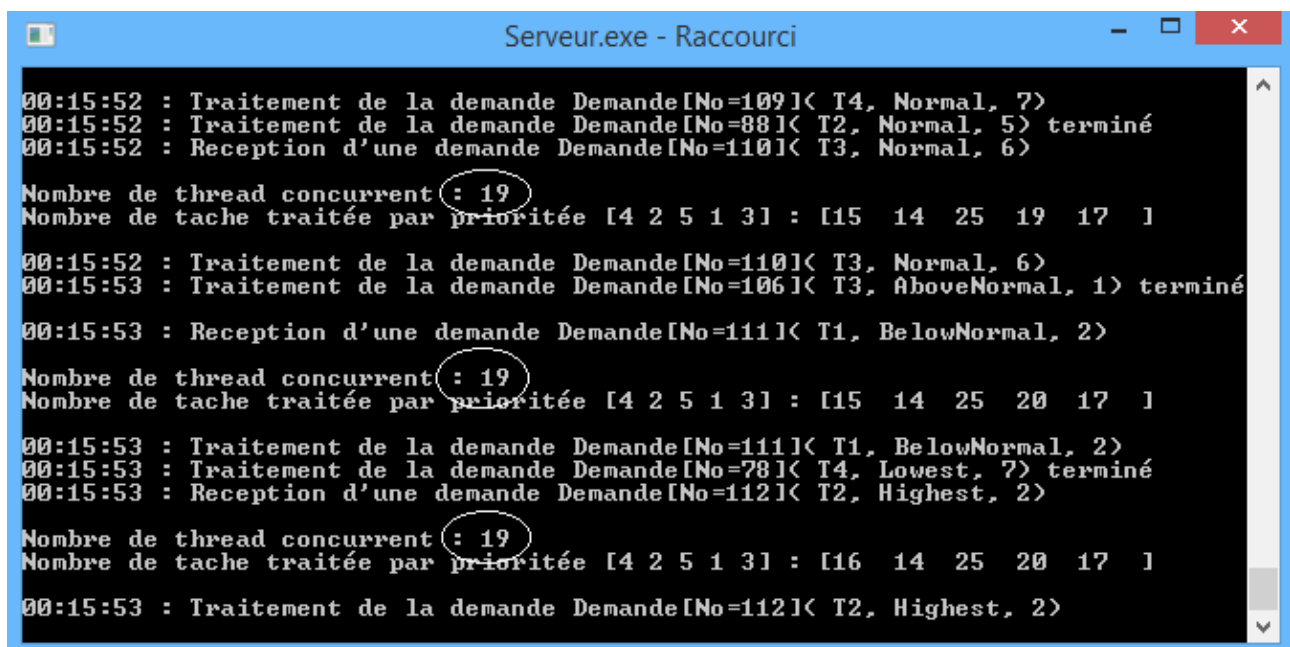
La solution peut être testée à l'aide de ses deux exécutables :

- TraitementParallele\Serveur\bin\Debug\Serveur.exe
- TraitementParallele\Client\bin\Debug\Client.exe

ou bien en lançant le script TraitementParallele\test.bat.

J'ai choisi une communication *TCP/IP* entre les deux applications. Ainsi, le lancement de Serveur.exe demande un numéro de port pour l'ouverture du socket d'écoute des demandes. Celui-ci écoutera alors sans fin les demandes de service.

Client.exe quand à lui, est un générateur de client. Après avoir demandé une IP et un port de communication, il créera sans fin des threads « client », ainsi que leur demande générée aléatoirement. La limite de threads concurrents est maintenue grâce à un ThreadPool de capacité maximale de 20. Ceci est vérifiable sur la console du serveur à la ligne « Nombre de thread concurrent » qui ne dépasse jamais 20 malgré les nouvelles demandes (une période d'émission des demandes de 100 ms est nécessaire pour obtenir ce résultat) :



```
Serveur.exe - Raccourci
00:15:52 : Traitement de la demande Demande[No=109]< T4, Normal, 7>
00:15:52 : Traitement de la demande Demande[No=88]< T2, Normal, 5> terminé
00:15:52 : Reception d'une demande Demande[No=110]< T3, Normal, 6>
Nombre de thread concurrent : 19
Nombre de tache traitée par priorité [4 2 5 1 3] : [15 14 25 19 17 ]
00:15:52 : Traitement de la demande Demande[No=110]< T3, Normal, 6>
00:15:53 : Traitement de la demande Demande[No=106]< T3, AboveNormal, 1> terminé
00:15:53 : Reception d'une demande Demande[No=111]< T1, BelowNormal, 2>
Nombre de thread concurrent : 19
Nombre de tache traitée par priorité [4 2 5 1 3] : [15 14 25 20 17 ]
00:15:53 : Traitement de la demande Demande[No=111]< T1, BelowNormal, 2>
00:15:53 : Traitement de la demande Demande[No=78]< T4, Lowest, 7> terminé
00:15:53 : Reception d'une demande Demande[No=112]< T2, Highest, 2>
Nombre de thread concurrent : 19
Nombre de tache traitée par priorité [4 2 5 1 3] : [16 14 25 20 17 ]
00:15:53 : Traitement de la demande Demande[No=112]< T2, Highest, 2>
```

Le client est configurable grâce au fichier Client.exe.config. On peut y définir la période d'émission des demandes en millisecondes, l'ip et le port par défaut. On peut de même configurer le port du serveur avec Serveur.exe.config.

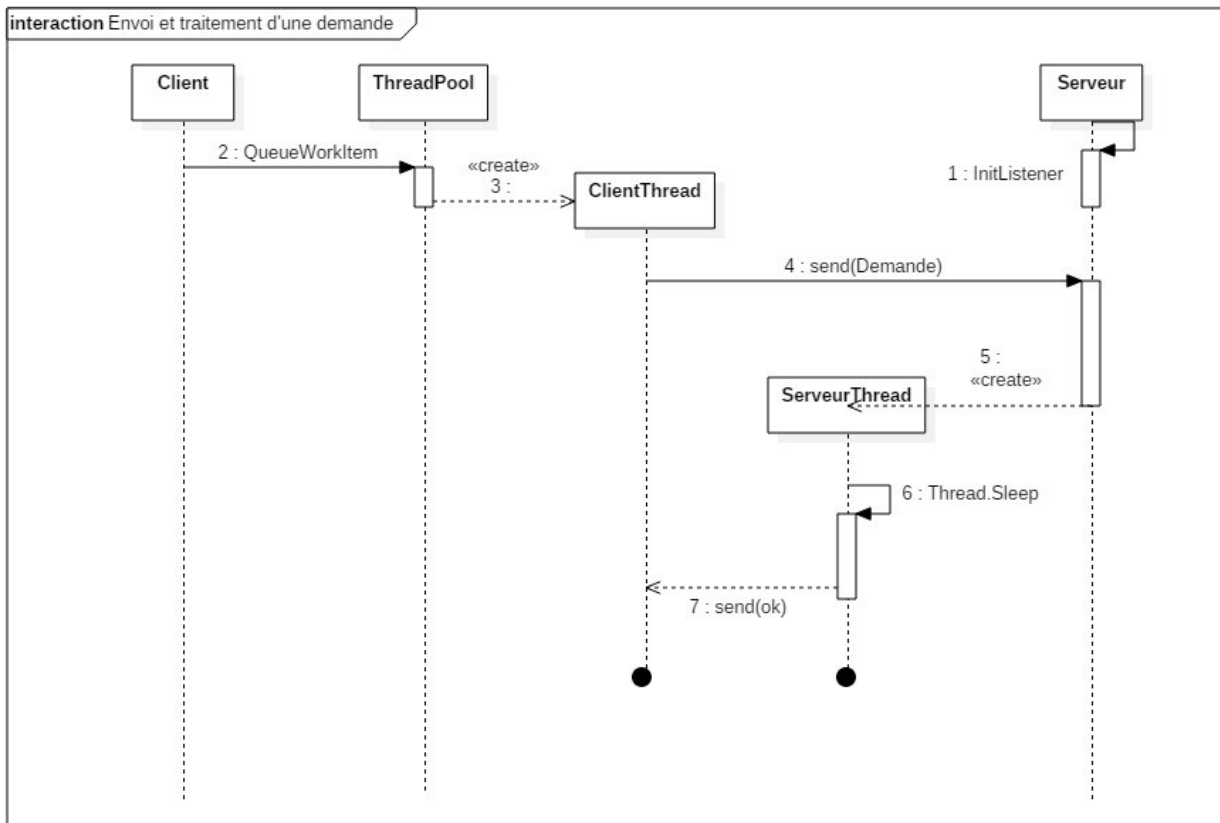
On peut aussi apercevoir sur cette même capture d'écran l'ordonnancement des traitements à la ligne « Nombre de tâches traitées par priorité ». Mais celle-ci reflète assez peu l'objectif attendu ( $25+16 = 41$  tâches de haute priorité accomplies contre  $14+20 = 34$  tâches moins prioritaires). Cela est dû au caractère aléatoire de la génération des tâches. En effet statistiquement, il y a autant de tâches de chaque priorité qui sont créées, donnant ainsi de faibles écarts entre chaque « score ». L'utilisation d'un `System.Windows.Threading.Dispatcher` résoudrait sûrement ce problème.

Enfin je décrirai la structure d'une demande comme elle est affichée à la dernière ligne : « Demande[No=112](T2, Highest, 2) ».

Son numéro permet de l'identifier entre le client et le serveur. C'est aussi son rang dans l'ordre de création des demandes. *T2* est son type de traitement, information nécessaire en cas de vrai traitement de la demande par le serveur. Ici elle est inutilisée. Highest est sa priorité, celle-ci peut prendre sa valeur dans l'énumération `Thread.Priority` = { AboveNormal , BelowNormal , Highest , Lowest , Normal }. La dernière donnée indique le temps de pause en seconde simulant la durée du traitement de la demande. Une interface C# de représentation d'une demande permettrait le découplage des deux applications.

J'ajouterai que le nombre de threads concurrents et le « tableau compteur » de tâches accomplies sont des ressources critiques que les threads fils du serveur se partagent à l'aide d'un Mutex. En cas de besoin (multiple sections critiques et attentes mutuelles), celle-ci peut être traitée plus finement à l'aide d'un Monitor.

Ci-dessous un diagramme de séquence montrant un cycle de traitement d'une demande :



Ce cycle est répété indéfiniment jusqu'au signal Ctrl+C sur l'une ou l'autre des applications.