Concurrence – Master 1
TD 9 : Moniteurs

<u>Exercice</u> 1 : (**Coiffeur dormeur**) Dans une ville tranquille, un coiffeur possède un petit salon ayant une porte d'entrée, une porte de sortie, un fauteuil de coiffure et quelques chaises. Les clients arrivent par la porte d'entrée et sortent par la porte de sortie après avoir eu leur coupe de cheveux. Comme le salon est petit, uniquement le client sur le fauteuil de coiffure peut être servi à un moment donné par le coiffeur. Le coiffeur passe sa vie entre dormir et servir ses clients. Quand il n'a aucun client, le coiffeur dort dans le fauteuil. Quand un client arrive et trouve le coiffeur endormi, il le réveille, s'assoit dans le fauteuil et s'endort pendant que le coiffeur lui coupe les cheveux. Si un client arrive et le coiffeur est occupé, le client s'endort dans une chaise libre. Après avoir fini une coupe, le coiffeur ouvre la porte de sortie et attend que le le client coiffé quitte le salon. Ensuite, s'il y a des clients en attente, le coiffeur réveille un et attend qu'il occupe le fauteuil de coiffure. S'il n'y a aucun client, le coiffeur s'endort.

Modéliser ce problème en utilisant un moniteur pour la synchronisation entre le coiffeur et ses clients. Les procédures du moniteur doivent être :

- `req_coif` : appelée par le client qui demande une coupe ; le client sortira de cette procédure après avoir été servi.
- `coif_suivante` : appelée par le coiffeur ; le coiffeur en sortira quand un client est disponible.
- `coif_finie` : appelée par le coiffeur ; le coiffeur en sortira quand le client coiffé quitte le salon.

Pour arriver à écrire une solution et à la prouver, essayez de suivre les étapes suivantes :

1. Identifier les états des clients et du coiffeur.
2. Pour chaque état, utiliser une variable entière qui indique le nombre de fois que ces états ont été atteints par (tous) les clients et par le coiffeur. Écrire la sémantique de chaque variable et puis les relations entre ces variables.
3. En utilisant les relations écrites, en déduire un invariant pour le moniteur.
4. Écrire le moniteur ayant comme variables abstraites les variables ci-dessus et pas de variables condition. Pour suspendre un processus, utiliser l'attente active (instruction **await**) sur les conditions écrites ci-dessus.
5. Introduire des variables condition qui correspondent aux conditions d'attente du point précédent.
6. Écrire le moniteur avec des variables condition et des variables abstraite nécessaires (pas forcement le compteurs utilisés précédemment).

Correction 1 :

1. Coiffeur : disponible, coupe, fin coupe.
   Client : en attente, sorti.
2. On aura donc 5 variables entières :
   - `bdispo` : nombre de fois que le coiffeur est disponible ;
   - `boccup` : nombre de fois que le coiffeur effectue une coupe ;
   - `bfin` : nombre de fois que le coiffeur fini une coupe ;
   - `catt` : nombre de client ayant occupé le fauteuil du coiffeur ;
   - `csorti` : nombre de clients sortis.

   Les relations entre ces variables sont :
   - exécution séquentielle des états du coiffeur :
   $$C_1 \ : \ \texttt{bdispo} \geq \texttt{boccup} \geq \texttt{bfin}$$
   - exécution séquentielle des états des clients :
   $$C_2 \ : \ \texttt{catt} \geq \texttt{csorti}$$
   - synchronisation sur le début de la coupe :
   $$C_3 \ : \ \texttt{bocc} \leq \texttt{catt} \leq \texttt{bdispo}$$
   - synchronisation sur la fin de la coupe :
   $$C_4 \ : \ \texttt{csorti} \leq \texttt{bfin}$$
3. L'invariant sera : $CD : C_1 \wedge C_2 \wedge C_3 \wedge C_4$.
4. Le moniteur sans variables condition mais avec de l'attente active (**await**) :

```
monitor Salon_Coif { /* Invariant CD */
  int bdispo = 0, boccup = 0; bfin = 0;
  int catt = 0, csort = 0;
  procedure req_coif() { /* appele par le client */
    await (catt < bdispo); catt++;
    await (csorti < bfin); csorti++;
  }
  procedure coif_suivante () { /* appele par le coiffeur */
    bdispo++;
    await (boccup < catt); boccup++;
  }
  procedure coif_finie () { /* appele par le coiffeur */
    bfin++;
    await (csorti == bfin);
  }
}
```

   Rappel : cette solution est une fausse car avoir de l'attente active dans un moniteur bloque l'entrée d'autres processus dans le moniteur. Elle nous sert uniquement pour clarifier les conditions de blocage des processus dans les procédures du moniteur.

5. La remarque est qu'utiliser des compteurs n'est pas très bon car leurs valeurs peuvent croître sans limite. A leur place, on peut utiliser des variables qui expriment la différence entre les compteurs :
   - `coif = bdispo - catt`
   - `faut = catt - boccup`
   - `sort = bfin - csorti`

   et ces variables satisfont l'invariant :

$$CD\ :\ 0 \leq \texttt{coif} \leq 1\ \wedge\ 0 \leq \texttt{faut} \leq 1\ \wedge\ 0 \leq \texttt{sort} \leq 1$$

   Avec cet échange de variables, les conditions des **await** deviennent :

```
/* await(catt < bdispo); */ do :: (coif == 0) -> wait(coif > 0)
/* await(sorti < bfin);  */ do :: (sort == 0) -> wait(sort > 0)
/* await(boccup < catt); */ do :: (faut == 0) -> wait(faut > 0)
/* await(csorti==bfin);  */ do :: (sort > 0)  -> wait(sort==0)
```

   On a donc besoin de quatre variables condition, une par condition ci-dessus.

6. Le moniteur final sera :

```
monitor Salon_Coif {
  int coif = 0, faut = 0, sort = 0;
  cond coif_dispo; /* signale quand coif > 0 */
  cond faut_occ;   /* signale quand faut > 0 */
  cond porte_ouv;  /* signale quand sort > 0 */
  cond cl_sorti;   /* signale quand sort == 0 */

  procedure req_coif() { /* appele par le client */
    do :: coif == 0 -> wait(coif_dispo) :: else -> break od;
    coif_dispo--;
    faut++; signal(faut_occ);
    do :: sort==0 -> wait(porte_ouv) :: else -> break od;
    sort++; signal(cl_sorti)
  }
  procedure coif_suivante () { /* appele par le coiffeur */
    coif++; signal(coif_dispo);
    do :: faut == 0 -> wait(faut_occ) :: else -> break od;
    faut--
  }
  procedure coif_finie () { /* appele par le coiffeur */
    sort++; signal(porte_ouv);
    do :: sort>0 -> wait(cl_sorti) :: else -> break od
  }
}
```