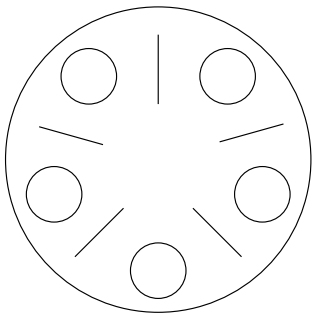
Le dîner des philosophes

On rappelle que le dîner des philosophes est un problème classique de synchronisation (cf TD 4) dont la formulation est :

> Cinq philosophes passent leur existence autour d'une table, à penser et à manger. Penser ne les fait pas interagir les uns avec les autres. Mais pour manger le contenu de son bol de riz, un philosophe doit se munir de deux baguettes, celle située à sa gauche et celle située à sa droite.

L'existence du philosophe *i* se résume donc à :

```
répéter
    penser ()
    P (baguette [i])
    P (baguette [(i+1) mod 5])
    manger ()
    V (baguette [i])
    V (baguette [(i+1) mod 5])
fin répéter
```



On veut simuler un tel système avec deux programmes :

- `table` *`nphilo`* : ce programme prend en paramètre le nombre de couverts à dresser (c'est-à-dire le nombre de philosophes, de bols ou de baguettes), initialise un segment de mémoire partagée nommé `/table` contenant, a minima, le nombre de couverts, le numéro du prochain philosophe attendu à la table (initialement nul, incrémenté à chaque fois qu'un philosophe s'attable), et autant de sémaphores que nécessaire. Une fois la table dressée, le programme peut se terminer. Le nombre de couverts ne doit être borné que par la mémoire disponible et non par une limite imposée par votre programme.
- `philo` *`nrepas`* : ce programme simule un philosophe avec la boucle ci-dessus. Il commence par obtenir son identité (son numéro) avec le numéro du prochain philosophe dans le segment de mémoire partagée, puis commence la boucle et s'arrête lorsqu'il a parcouru le nombre de tours indiqué. Pour simplifier le programme, les philosophes ne réfléchiront pas et leurs repas ne seront constitués que d'un simple affichage (ce sont des purs esprits) de la forme « *`p`* `: je mange` » où *`p`* est le numéro du philosophe (entre 0 et *`nphilo`*-1).

1. Implémentez une première version de ces programmes. Vous devez constater un interblocage au bout d'un certain nombre de repas. Donnez l'ordre de grandeur sur votre machine.
2. Implémentez une version sans interblocage de l'exercice précédent. Vous devrez sans doute modifier l'algorithme indiqué dans cet exercice. Vous utiliserez la compilation conditionnelle avec l'option `-DINTERB`.
3. On s'intéresse maintenant à la terminaison. Modifiez votre implémentation précédente (sans interblocage) pour que le programme `table` attende que le dernier philosophe ait quitté la table pour supprimer le segment de mémoire partagée et se terminer. Bien évidemment, vos programmes doivent utiliser exclusivement des sémaphores pour la synchronisation et ne doivent contenir aucune attente active ni temporisation. Vous utiliserez la compilation conditionnelle avec les options `-DINTERB` et `-DTERMINAISON`.

Voici quelques astuces à connaître :

- sur Linux, le segment de mémoire partagée est représenté par un fichier dans `/dev/shm`. Vous pouvez supprimer ce fichier avec la commande `rm`, notamment si vous changez le format des données dans le segment, le nombre de philosophes ou simplement si vous voulez réinitialiser la table.
- pour tester, vous pouvez lancer 5 philosophes en parallèle et en rafale avec (par exemple) :

  ```
  for i in 1 2 3 4 5 ; do ./philo 10 & done
  ```

  Ensuite :
  - `jobs` : donne la liste des *jobs* lancés et non encore terminés
  - `kill %1` : arrête le *job* numéro 1

— pour réaliser l'affichage dans le philosophe, vous pouvez utiliser `printf`. N'oubliez pas d'appeler `fflush(stdout)` après chaque affichage pour éviter que ceux-ci ne soient bufferisés lorsque la sortie est un fichier.