Exercice 2 – Modèles des producteurs/consommateurs

- On considère un buffer partagé par des activités parallèles de deux types :
  - Des producteurs qui déposent des messages dans ce buffer
  - Des consommateurs qui retirent des messages de ce buffer
- Le buffer comporte N cases et est géré circulairement

- Variante de base
  - Les dépôts se font dans l'ordre croissant des indices de cases, de manière circulaire
  - Les retraits se font dans l'ordre des dépôts, de manière circulaire aussi

- Proposer une solution utilisant des sémaphores pour que ces activités parallèles déposent et retirent leurs messages de manière cohérente

Exercice 2 – Exemple de dépôts/retraits

Demandes de dépôts/retraits

Producteur1

Producteur2

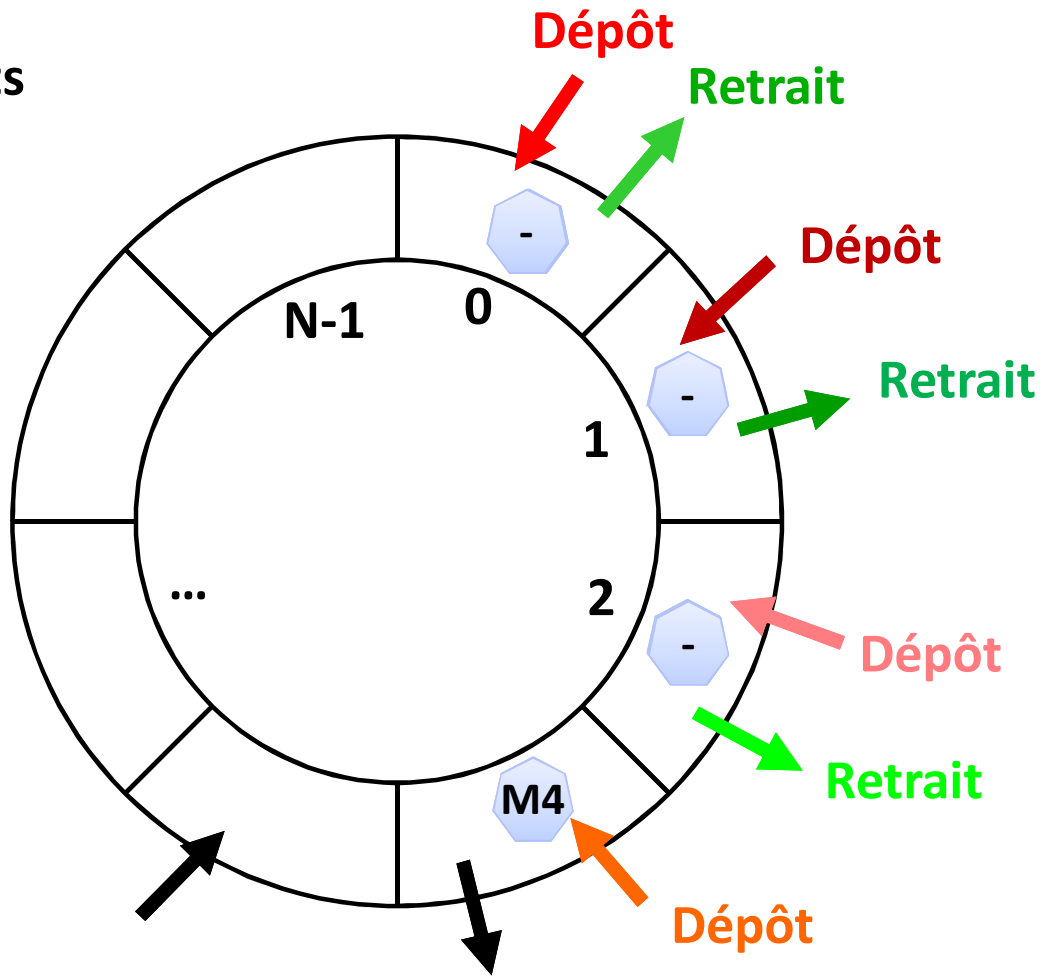
Consommateur1

Consommateur2

Consommateur3

Producteur3

Producteur4



Structure de donnés ?

- De quoi a-t-on besoin ?
- Du buffer partagé
  - On suppose exister un type Message

  Message buffer[N];
- D'indices partagés pour savoir
  - Dans quelle case déposer

  int indDepot;
  - Où initialement ? Comment progresse-t-on ensuite ?

  indDepot = 0; On incrémente de 1 modulo N
  - De quelle case retirer

  int indRetrait;

  Où initialement ? Comment progresse-t-on ensuite ?

  indRetrait = 0; On incrémente de 1 module N

Exprimer la synchronisation

- Pourquoi un producteur ne peut déposer ?
- Comment le représenter en termes de sémaphores ?

Nombre de tickets = nombre de cases vides

Semaphore semCaseVide;

- Initialisation ?

init(semCaseVide, N);

- Pourquoi un consommateur ne peut retirer ?
- Comment le représenter en termes de sémaphores ?

Nombre de tickets = nombre de cases pleines

Semaphore semCasePleine;

- Initialisation ?

init(semCasePleine, 0);

Gérer les dépôts / retraits du buffer

```
void depot (Message m) {
    // Déposer dans la case d'indice indDepot du buffer
    Buffer[indDepot] = m;
    // Progresser en gérant le buffer circulairement
    indDepot = (indDepot + 1) % N;
}
void retrait (Message *m) {
    // Retirer de la case d'indice indRetrait du buffer
    *m = Buffer[indRetrait];
    // Progresser en gérant le buffer circulairement
    indRetrait = (indRetrait + 1) % N;
}
```

Section critique ➔ E.M

Section critique ➔ E.M

Producteur

```
Producteur (Message m) {
    while (1) {
     // Préparer le message pour le dépôt
      . . .
     // Puis-je avoir une case vide où déposer ?
     P(semCaseVide)
    // Une case est vide ➔ y déposer mon message
    depot(m)
    // En déposant, j'ai créé une case pleine qu'attend peut-être un consommateur
    V(semCasePleine)
    . . .
    }
}
```

Consommateur

```
Consommateur (Message *m) {
    while (1) {
      . . .
     // Puis-je avoir une case pleine à consommer ?
     P(semCasePleine)
    // Une case est pleine ➔ y retirer mon message
    retrait(m)
    // En retirant, j'ai créé une case vide qu'attend peut-être un producteur
    V(semCaseVide)
    // Utiliser le message consommé
    . . . .
    }
}
```

Cela vous rappelle-t-il quelque chose?

Cela est-il suffisant ?

Exclusion mutuelle sur les indices partagés !

- Protéger l'indice de dépôt pour les conflits entre producteurs

Sémaphore mutexP;

- Initialisation ?

init(mutexP, 1);

- Protéger l'indice de retrait pour les conflits entre consommateurs

Sémaphore mutexC;

- Initialisation ?

init(mutexC, 1);

Le producteur devient

```
Producteur (Message m) {
    while (1) {
     // Puis-je avoir une case vide où déposer ?
     P(semCaseVide)
    // Une case est vide ➔ y déposer mon message
    // S 'assurer d'être seul à utiliser l'indice de dépôt
    P(mutexP);
    depot(m)
    // Relâcher l'accès en E.M. sur l'indice de dépôt
    V(mutexP);
    // En déposant, j'ai créé une case pleine qu'attend peut-être un consommateur
    V(semCasePleine);
    }
}
```

Le consommateur devient

```
Consommateur (Message *m) {
    while (1) {
     // Puis-je avoir une case pleine à consommer ?
     P(semCasePleine)
    // Une case est pleine ➔ y retirer mon message
    // S 'assurer d'être seul à utiliser l'indice de retrait
     P(mutexC);
     retrait(m)
    // Relâcher l'accès en E.M. sur l'indice de retrait
     V(mutexC);
    // En retirant, j'ai créé une case vide qu'attend peut-être un producteur
    V(semCaseVide)
    }
}
```