3. Allocateur de ressources

N ressources équivalentes à usage exclusif *(ex: page mémoire)*
Code d'un processus :

- k <- nb de ressources nécessaires
- demander k ressources
- ...
- libérer k ressources

> ⚠ **Attention**
>
> Pas 2 demandes consécutives sans libération entre.

Stratégie priorité aux petits demandeurs

Originalité : lors de libérer(k), on peut débloquer 0, 1 ou plusieurs activités.

Interface	Condition d'acceptation	Variables d'état	Prédicats d'acceptation
demander(k)	au moins k ressources libres	nbDispo : nat	nbDispo ≥ k
libérer(k)	---		true

> ✎ **Invariants**
>
> **inv** 0 ≤ nbDispo ≤ k

> 📋 **Variable Condition**
>
> Accès[N]

Codage

demander(k)

```
si ¬(nbDispo >= k) alors
        att[k]++
        Accès[k].wait
        att[k]--
finSi
nbDispo <- nbDispo - k
réveiller_suivant(k)  (*réveil en chaine*)
```

libérer(k)

```
nbDispo <- nbDispo + k
réveiller_suivant(nbDispo - k)
```

réveiller_suivant(départ)

```
i <- départ
tantQue Accès[i].empty ^ i <= nbDispo faire
        i++
finTantQue
si i <= nbDispo alors
        Accès[i].signal
finSi
```

Améliorations

demander : réveiller_suivant -> début à k
libérer : réveiller_suivant -> début à n

Variantes de stratégie :

- Petits demandeurs -> famine des gros demandeurs
- Priorité aux gros demandeurs -> faible parallélisme
- Best-fit : débloquer le plus possible
  - descendre jusqu'à trouver une activité (ou personne)
  - maximise utilisation des ressources en réduisant

<- retour

#TD/SC