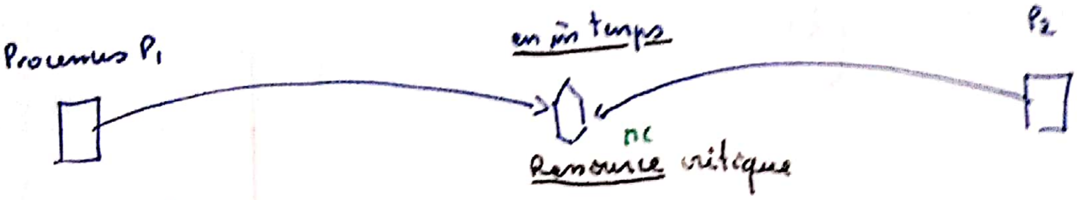
"Programmation Concourante" <L> — 17-09-2020

• Cas où des processus exécutent en m̂ temps un programme avec ressources commune.



Ici la ressource est en mémoire centrale et sur une même machine (Ressource et Processus locaux)

La ressource doit ∈ à l'espace d'adressage du Proc. P1 mais également à celui de P2.

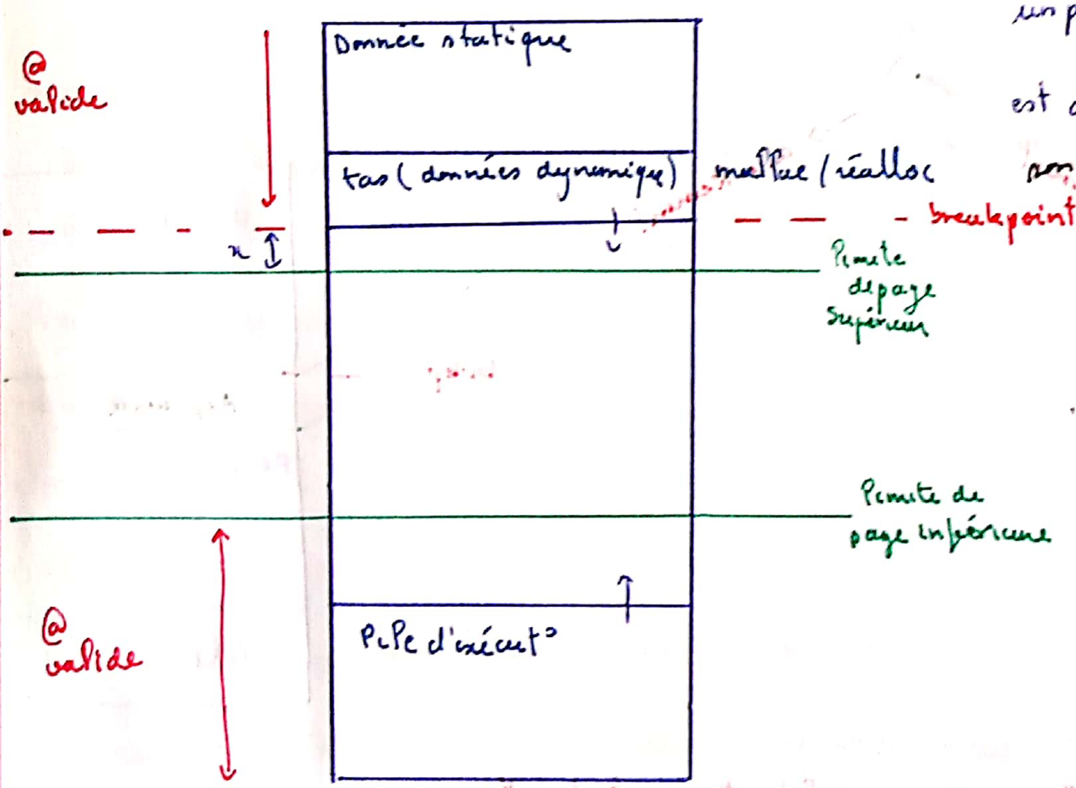
Ça en principe, on sait pas faire

I-> Segments de mémoire partagée

1-) Esp. d'adressage d'un processus.

divisé en 2/3 zones. espace de la mémoire dans laquelle un programme peut faire référence

est divisé en pages logiques qui sont transformées en pages Pq.



@valide sont celles où on peut écrire

. si on dépasse on fait une violat° de segment (théorique)

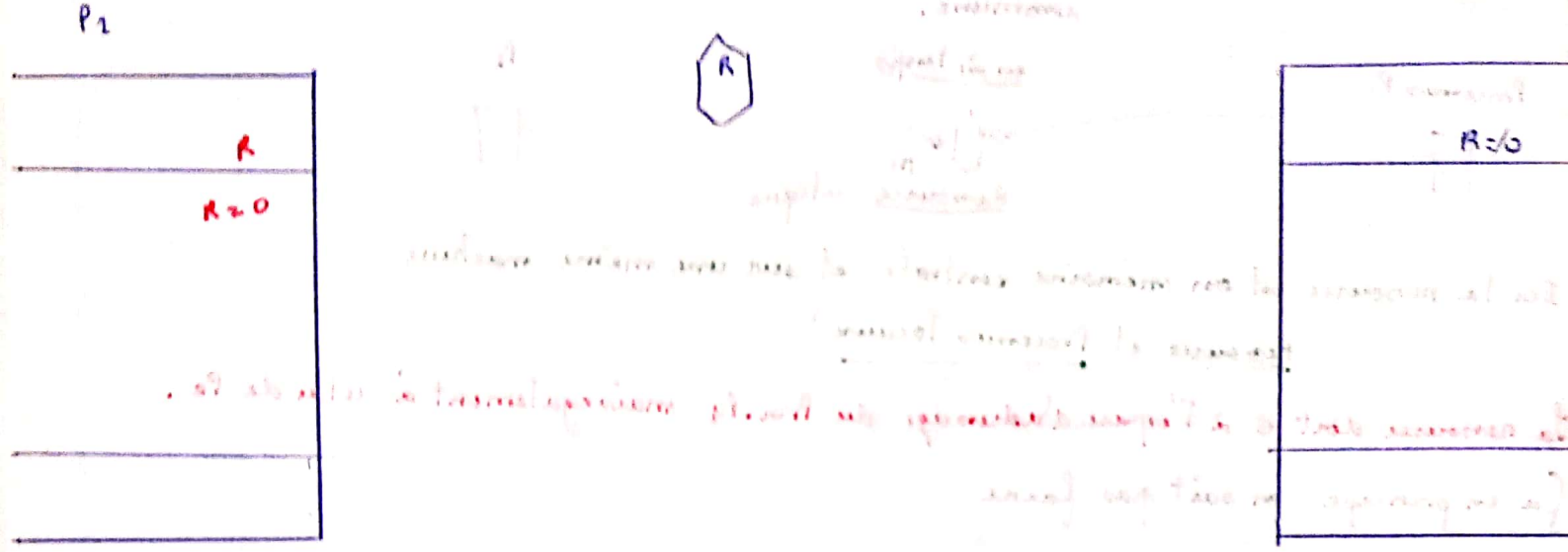
. mais en réalité, on peut écrire sur les adresses jusqu'à la limite de page supérieure.

• On peut modifier la posit° du breakpoint

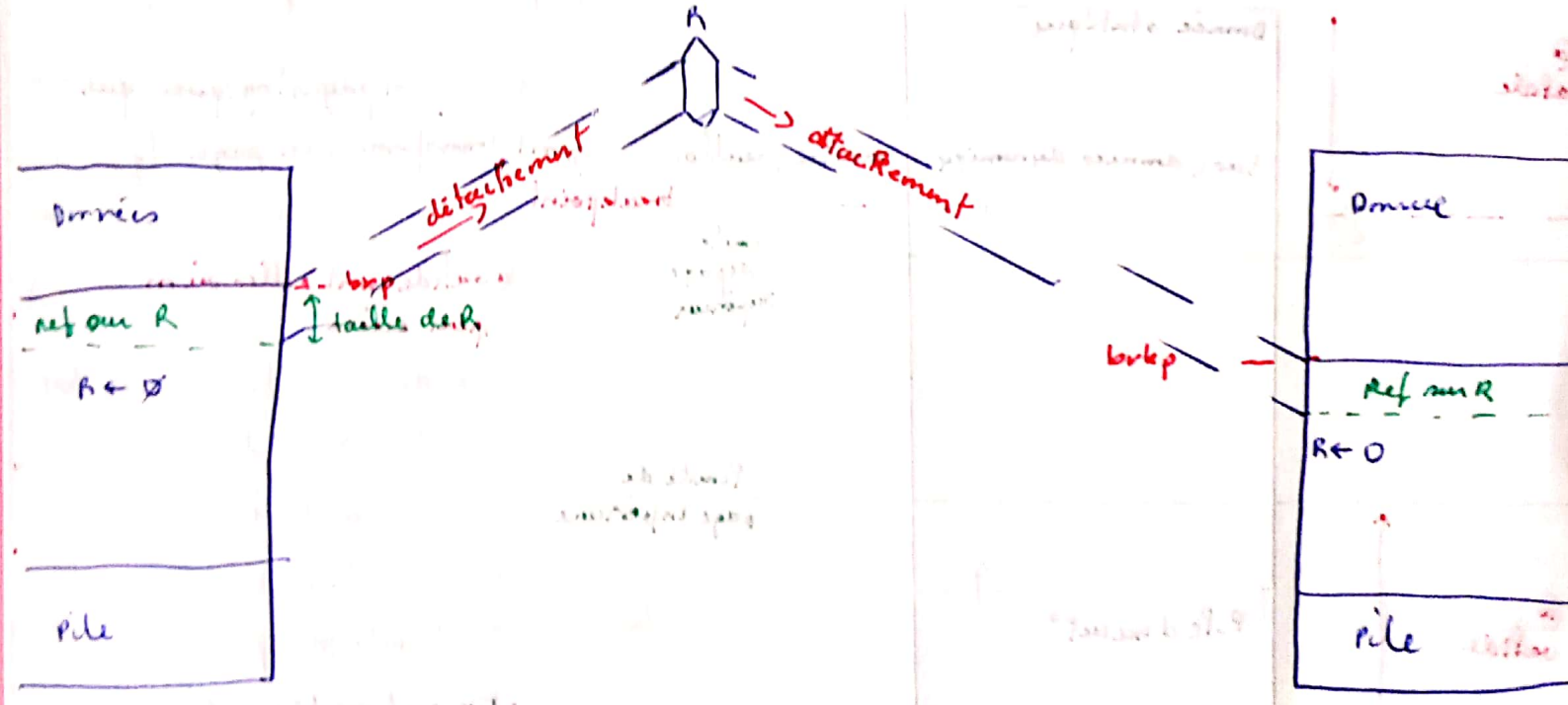
malloc le fait: ↑ limites des adresses utilisables et change le brkpt.

autre fonct° : brk, sbrk

Partagée de la mémoire entre 2 processus <=> à aux @valides des 2 processus



- Je réserve R dans P2 et je peux donner une valeur
- IP faut que R€ à l'espace d'adressage des 2 processus.



(1.) On abaisse le brkpt de la grandeur qu'il faut pour stocker R.

(2.) IP faut projeter un bloc de la Ressource R dans espace @ de P1 et P2

(3.) Si on manipule l'un, il est modifié dans l'autre.

IP faut déclarer la R en mémoire centrale (non fichier) -> Segment de mémoire partagée géré par la table des IPC.

2)

Tables des IPC.

Clé utilisateur (externe)	Clé ε interne	Files de messages
		Segment de mémoire partagé
		Sémaphore

si tu éteins le pc, c'est retenue, c'est pas effacé c'est comme un fichier.

- Primitive de création de s.m.p (attachement)

shmget : renvoie la clé si elle n'existe sys (interne)

Ref R: adresse de projection récupérée

*adr_R dans les données.

clé ← shmget(R

adr_R ← shmat(clé

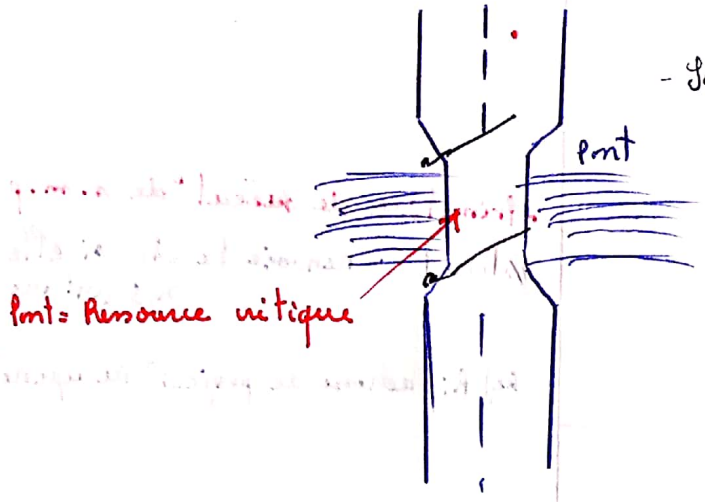
- Regarder segment mémoire partagée : ipcs -mm
- Clé user, c'est ce que je veux
- rm clé utilisateur clé ε
- effacer ipcrm -m clé ε
- Output Panier en synchrone (Cas général) avec &
- Ne garantine pas lecture et l'écriture en concurrence ; pas de synchronisat°
- Pas de garantie de la synchronisat° des MAJ.

les smp.

Sémaphores

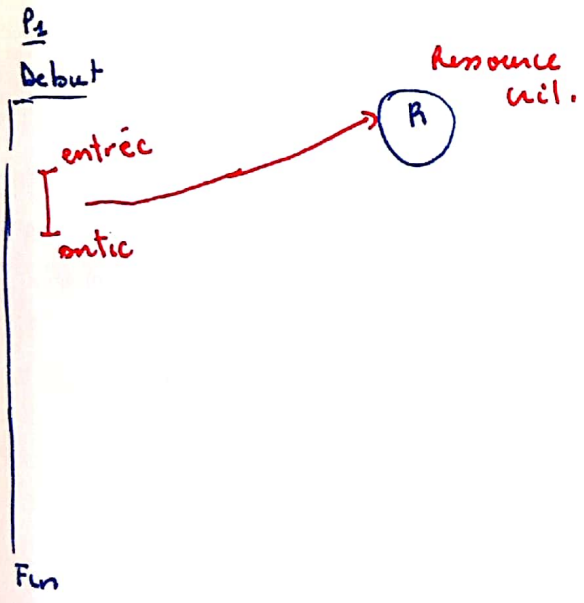
Permet de régler le problème de l'exclusion mutuelle (garantir l'exclusivité de l'utilisat° d'une ressource à un moment donné.

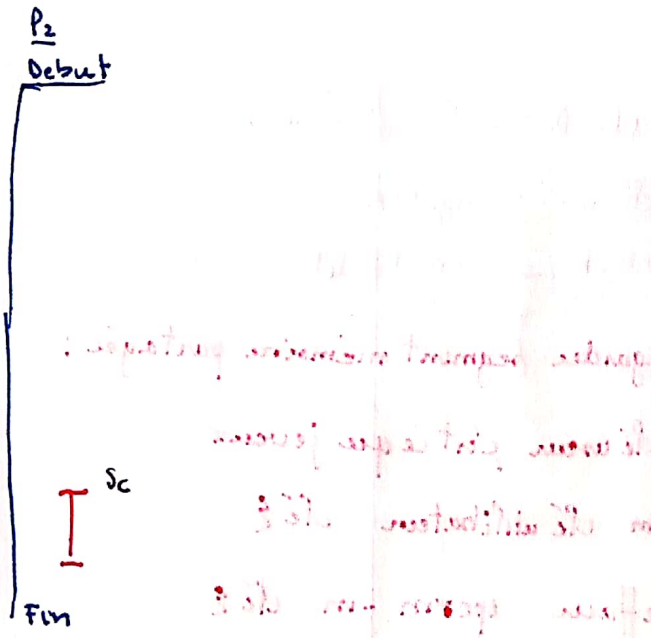
Analogie: Pont de voiture. (1 seule voie) (Le pont ne peut supporter que le poids d'une seule voiture,



- Solut°: dès qu'une voiture est sur le pont, il faut l'exclusivité du pont.
. On peut mettre des barrières
Ls Un mécanisme pour empêcher les autres voitures d'utiliser le pont.

Code: 1e seule ressource pour deux processus.





I utilisat° de R
(section critique) partie de code qui manipule la Rc.

but : Eviter que section critique de P_1 et P_2 s'exécute en même temps.
. Entrée en Sc d'un des deux Proc quand l'autre est déjà en Sc.
(Interdire)
. Pourquoi? On veut avoir des cas d'erreurs reproductibles

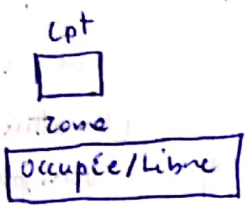
père

sc [cpt++

fils

sc [cpt++] Ça marche pas car, ce n'est pas une opération atomique.

Solution 1



Libre au départ

P1

/* entrée en SC */

si c

Tq zone == Occupée FAIRE

rien

FTq

/* SC */ zone ← Occupée

cpt++

/* sortie de la SC */

zone ← Libre

P2

/* idem */

Tq zone == Occupé

rien

FTq

/* SC */

zone ← Occupée;

cpt++

/* sortie SC */

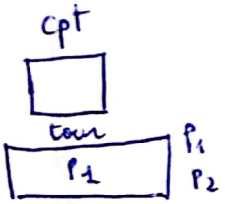
zone Libre;

Problème : tous les deux voient la zone à libre et continue la suite du programme. et font cpt++. On a déplacé le problème de cpt → zone

Zone est manipulé en même temps par les deux procs.

Solution 2

On remplace zone par tour qui contient le processus qui travaille, initialisé avec un des Proc2



P1

Tq /* entrée en SC */

Tq tour ≠ P1 FAIRE

rien

FTq

/** SC **/

cpt++;

/** Sortie **/

tour ← P2

FTq

P2

Tq /* entrée en SC */

Tq tour ≠ P2 FAIRE

rien

FTq

/** SC **/

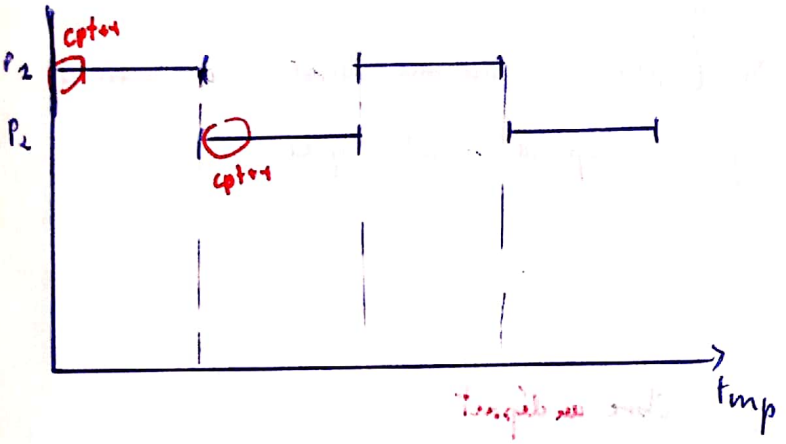
cpt++

/* Sortie SC */

tour ← P1

FTq

Temps partagé P_1 P_2 (1 CPU)

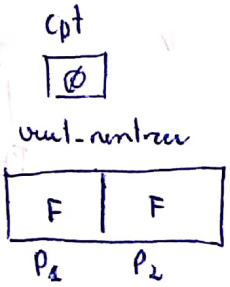


. Il faut cpt++ ; il donne la main, mais son temps n'est pas fini.

- On donne du temps cpu à un processus qui n'utilise pas la ressource critique.

La contrainte de continuité n'est pas respecté (On attribue du temps à un processus qui ne veut pas la Rc.)

. Pour résoudre ce problème: On ajoute Pc désir de rentrer en Sc.




```
Ta
| /* entrée en Sc */
|   veut_entrer[P1] ← Vrai
|   Tq veut_entrer[P2] FAIRE RIEN FTQ
|
|   /* Sc */
|     cpt++ ;
|   /** sortie Sc */
|   veut_entrer[P1] ← FAUX
FTq
```

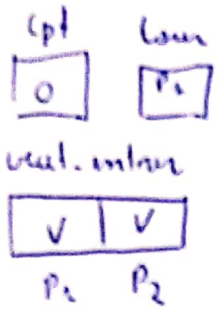
```
Tq
| /* entrée */
|   veut_entrer[P2] ← Vrai
|   Tq veut_entrer[P1] FAIRE
|   FTQ ... rien
|     /* Sc */
|       cpt++
|       /** sortie Sc */
|
|   veut_entrer[P2] ← FAUX
FTq
```

Problème Interblocage.

Solution 4: Tour + veut-entrer

l'algorithme de Dekker ; Pour rentrer en sc

(4)



P2

```
Tq /* entrée en Sc */
   veut_entrer[P2] ← VRAI
   Tq veut_entrer[P1] FAE
      SI
         tour == P1 ET tour
         veut_entrer[P2] ← FAUX
         Tq tour == P1 FAE nom FTq
         veut_entrer[P2] ← VRAI
      FSI
   FTq
   cpt++
      tour ← P1
FTq
```

P2

```
Tq  Tq /* entrée */
       veut_entrer[P2] ← Vrai
       Tq veut_entrer[P1] FAE
          Si tour == P1 alors
           veut_entrer[P2] ← FAUX
           Tq tour == P1 FAE nom FTq
           veut_entrée [P2] ← VRAI
          FSI
    FTq
    cpt++
       tour ← P1
```