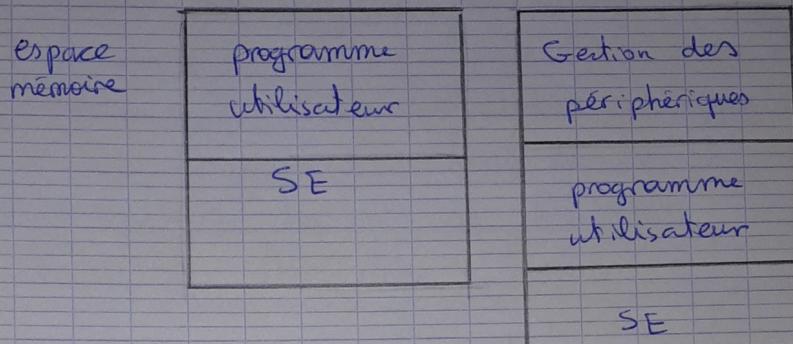


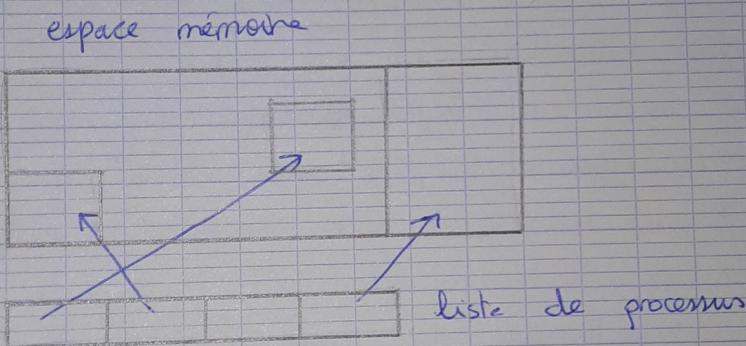
6

## Gestion de la mémoire

Cas de la mono programmation: (il y a 50 ans)

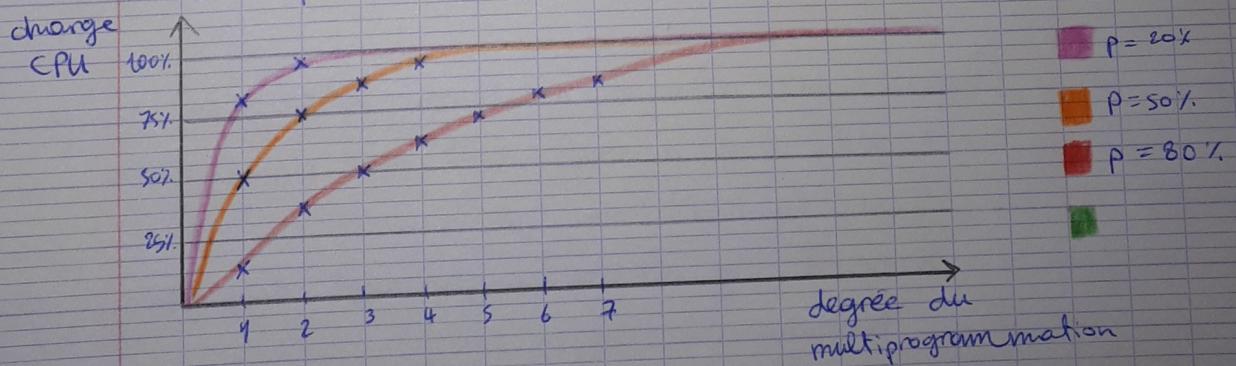


Cas de la multi programmation:



justification de la multiprogrammation:

soit  $p$  la proba pour un processus d'être en attente



Quand la mémoire est insuffisante pour maintenir tous les processus courants actifs :

- on sauvegarde des processus sur le disque
- on charge un processus de manière dynamique

2 approches complémentaires :

- le va et vient :
  - chaque processus est considéré dans son intégralité
  - exécution / sauvegarde sur disque
- la mémoire virtuelle
  - exécution des processus, même si ils ne sont que partiellement en mémoire.

Le va et vient :

- chargement complet des processus
- sauvegarde complète sur disque
- au chargement nécessaire le (re)localiser
- apparitions de "trous" en mémoire
  - ↳ technique du compactage (très coûteux)  
SOLUTION : réarranger l'espace mémoire)
- quand la taille du processus est fixe, le SE alloue exactement ( $\sim$ ) la taille nécessaire

Quand un processus a besoin d'  $\rightarrow$  sa taille :

- soit il y a suffisamment de place libre à côté (contigü) du processus
- si il n'y a pas suffisamment de place attente / terminaison
- si il est contigü à un autre processus  $\rightarrow$  déplacer tout le processus

## Gestion :

## Tableau de bits

La mémoire est répartie en unités d'allocation (UA) chaque UA possède une taille, à chaque UA est associé un bit dans le tableau :

- 0 si l'UA est vide
  - 1 sinon

$P_1$ (SUA)	$P_2$ (UUA)	$P_3$ (SUA)
		
11111	0001111	00000011111100
$\underbrace{\quad}_{3UA}$	$\underbrace{\quad}_{SUA}$	$\underbrace{\quad}_{2UA}$

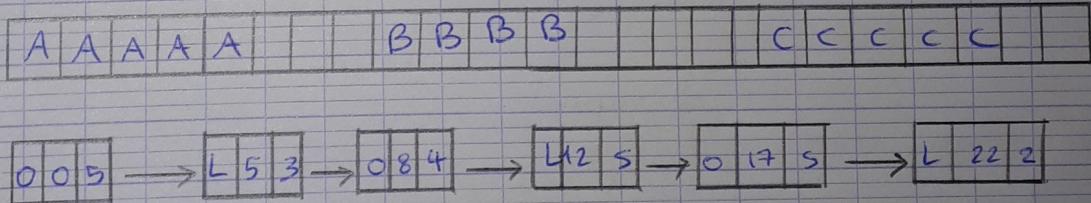
mém 24 UA

## Gestion par liste chaînée :

→ liste chaînée des segments mémoire

→ chaque entrée indique :

- l'état du segment (libre ou occupé)
  - ~~Hebdo~~ l'adresse de début du segment
  - la taille du segment

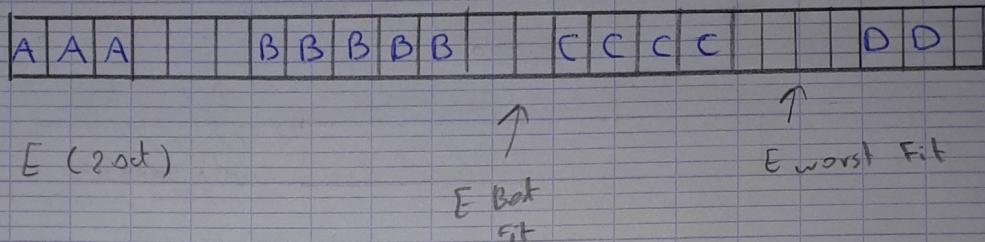


Q? - où placer le processus à charger?

- dans quel espace libre ?
  - peut-on optimiser la taille des espaces libres ?
  - peut-on optimiser la recherche d'un espace libre ?

## Algorithmes:

- First Fit
- next Fit
- Best Fit
- Worst Fit

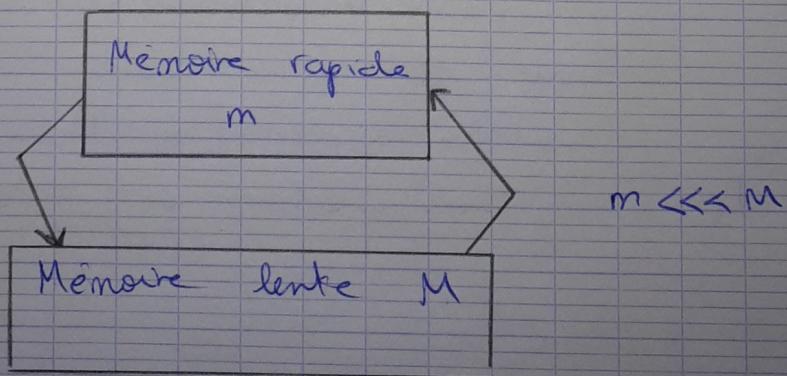


## La mémoire virtuelle:

- la taille d'un programme + données + pile peut être supérieur à la capacité mémoire physique
- le système d'exploitation conserve en mémoire des parties d'un programme en cours d'exécution, le reste est stocké sur disque

⇒ pagination

## Pagination:



Le défaut de page: est un accès à une page mémoire non présente en mémoire rapide.

dans ce cas:

- le système d'exploitation choisit une page en mémoire rapide
- écrit, si nécessaire son contenu en mémoire lente
- charge en mémoire rapide à l'emplacement libéré la page qui faisait défaut

### Algorithme de Belady:

- à chaque page correspond le numéro de la ligne du programme ayant besoin de cette page
- quand un défaut de page se produit on remplace la page qui porte le plus grand numéro (celle utilisée le plus tard)
- non implémentable
- utilité: fournir un échantillon de comparaison

Soit une mémoire rapide permettant de contenir les pages mémoires et la séquence d'utilisation des pages.

$$6 = 1, 4, 3, 1, 2, 4, \overset{(1)}{5}, \cancel{4, 3, 7, 5, 6, 2, 3, 1, 4, 3, 5, 7, 6, 2, 3, 4, 3}$$

111111	5	s	s	s	s	s	s	s	s	s	s	s	7	7	7	8	...
444444	4	4	4	2	2	6	6	6	6	6	6	6	6	6	6	2	2
333333	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
222222	2	2	2	2	2	2	2	2	2	1	4	4	4	4	4	4	4

## Stratégies (implémentables) :

- FiFo (premier chargé premier évincé)
- LiFo (dernier \_\_\_\_\_)
- LRU (moins récemment utilisée)
- Algo de seconde chance
- FWF (vider quand c'est plein)

même exemple :

$6 = 1, 4, 3, 1, 2, 4, 5, 4, 3, 7, 5, 6, 2, 3, 1, 4, 3, 5, 7, 6, 2, 8, 4, 3$

Fifo: 17 défaut de pages

1, 1, 1, 1, 1, 1, 1	5	5	5	5	5	5	5	1	1	1	1	1	1	6
4	4	4	4	4	4	4	4	7	7	7	7	7	7	4
3	3	3	3	3	3	3	3	3	6	6	6	6	6	5
2	2	2	2	2	2	2	2	3	3	3	3	3	7	7

LiFo: 14 défauts de pages

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	4	4	4	4	4	4	4	,	,	4	4	4	4	,
3	3	3	3	3	3	3	3	,	,	3	3	3	3	,
2	2	2	2	7	7	7	3	3	3	3	3	2	8	,

LRU: 19 défauts de pages

1	1	1	1	1	1	1	3	3	3	2	2	2	5	5	5	5	8	8	8
4	4	4	4	4	4	4	6	6	6	6	4	4	4	4	6	6	6	3	
3	3	3	3	5	,	5	5	5	5	1	1	1	7	7	7	7	4	4	
2	2	2	2	7	7	7	3	3	3	3	3	3	3	2	2	2	2		

FWF 21 df

1	1	1	1	1	5	5	5	,	6	6	6	6	4	4	4	4	6	6	6	3
4	4	4	4	4	4	4	4	,	2	2	2	3	3	3	3	2	2	2	2	
3	3	3	3	3	3	3	3	,	3	3	3	3	3	3	3	8	8	8	8	
2	2	2	2	7	,	7	7	1	1	1	1	7	7	7	7	4	4	4	4	

### Algo de la seconde chance

→ liste chaînée des pages

→ pour chaque page on a un bit  $R$

→ on parcourt la liste des pages, en consultant le bit  $R$

→ si  $R == 0 \Rightarrow$  on remplace cette page

si  $R == 1 \Rightarrow$  on place  $R à 0$  et on consulte la page suivante

→ quand une page est utilisée on place  $R à 1$