7 – MEMOIRE : ORGANISATION

1. MEMOIRE LINEAIRE

Dans ce type de gestion, la totalité du programme et des données d'un processus doivent être chargés en mémoire pour pouvoir exécuter le processus. L'allocation d'un espace mémoire à un processus peut se faire suivant différentes stratégies :

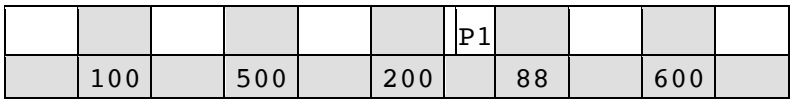
- La stratégie First Fit recherche le premier espace en mémoire de taille suffisante
- La stratégie Best Fit recherche le plus petit espace libre pouvant contenir le processus.
- La stratégie Worst Fit recherche le plus grand espace libre pouvant contenir le processus.

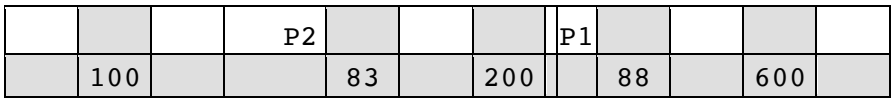
1.1.

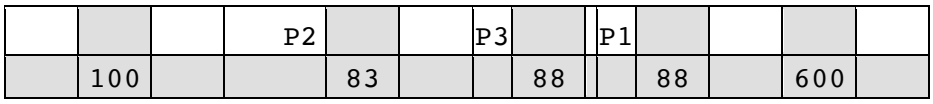
Soit une liste des blocs libres composée dans l'ordre de blocs de 100K, 500K, 200K, 300K et 600K. Comment des processus de tailles respectives 212K, 417K, 112K, et 426K, arrivés dans cet ordre, seraient-ils placés en mémoire

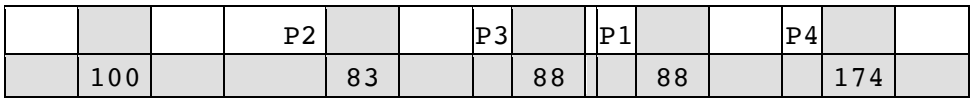
- avec une stratégie Best-Fit ?
- avec une stratégie First-Fit ?

On représente l'état de la mémoire après le placement de chacun des processus, avec la stratégie Best-Fit :

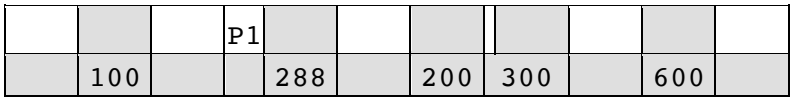


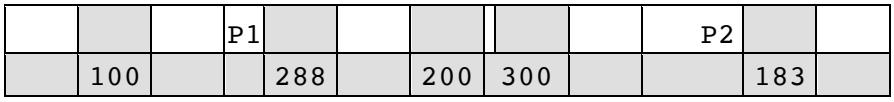


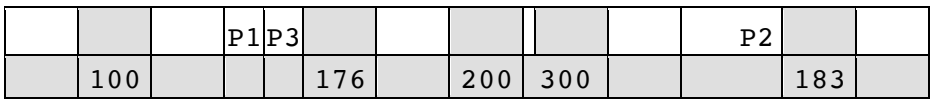




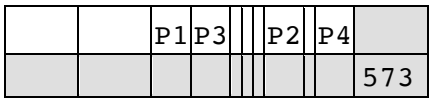
On reprend le problème avec la stratégie First-Fit :







Aucun trou en mémoire n'est suffisant pour contenir P4. Il faut donc procéder à un compactage :



1.2.

Quelle est la place réellement occupée par l'ensemble des processus de la question précédente si l'on alloue la mémoire par blocs de 10 K ? Comment s'appelle ce phénomène ?

Le premier processus occupe réellement 220 K, le deuxième 420, le troisième 120 et le quatrième 430. La place totale perdue est donc : 8 + 3 + 8 + 4 = 23 K.

Ce phénomène s'appelle fragmentation interne. L'allocation par blocs permet d'éviter d'avoir des trous trop petits en mémoire. Par contre, si les blocs sont trop grands, l'espace perdu devient important. Contrairement à la fragmentation externe, la fragmentation interne n'est pas récupérable.

2. SEGMENTATION

La segmentation divise l'ensemble des informations nécessaires à l'exécution d'une tâche en segments. Un segment, de longueur quelconque, représente une zone de mémoire pour la tâche. Il est associé à un aspect spécifique de la représentation de la tâche en mémoire (code, données, pile, ...). Un segment est chargé entièrement dans des zones de mémoire contiguës.

On note <s, d> une adresse segmentée composée d'un numéro de segment et d'un déplacement. On considère une tâche utilisant 3 segments, dont la table des segments à un instant donné est :

	B	L	p	m	u	xle
0		132	0			
1	7435	400	1	0	0	011
2		325	0			

où `B` est la base, `L` la longueur du segment, `p`, `m` , `u` et `xle` respectivement les bits de présence, de modification, d'utilisation et de droits d'accès au segment. On suppose que la tâche fait un accès en écriture à la donnée située à l'adresse <1, 260>.

2.1.

Quelle est l'adresse physique de cette donnée en mémoire ?

Le segment 1 est chargé en mémoire à l'adresse 7435. La donnée est donc à l'adresse

7435 + 260 = 7695.

2.2.

Donnez les modifications effectuées dans la table lors de cet accès, en précisant si elles sont effectuées par le système ou le matériel.

Le matériel positionne automatiquement à 1 le bit d'utilisation lors de chaque accès à un segment. Ici, il s'agit d'un accès en écriture donc le bit de modification est lui aussi positionné à 1 par le matériel.

2.3.

Que se passe-t-il lors d'un accès à l'instruction d'adresse <0, 125> ?

Le matériel teste le bit de présence. Comme celui-ci est à 0, il y a donc un défaut de segment. Le système doit alors allouer à la tâche un emplacement mémoire pour charger le segment (ce qui peut nécessiter l'évacuation d'un autre segment). Le segment est ensuite chargé en mémoire et le système met à jour la base B du segment. Il initialise les droits du segment en fonction des droits en mémoire secondaire. Le système positionne le bit de présence à 1, les bits d'utilisation et de modification à 0. Ces deux derniers bits seront mis à jour par le matériel lorsque l'accès à la donnée sera effectivement réalisé.

3. PAGINATION

Le principe de la pagination réside dans la division de la mémoire en zones de tailles fixes appelées "pages". L'espace de travail d'un processus est divisé en pages. Quand le processus est exécuté, seules les pages dont il a besoin sont chargées en mémoire centrale.

On dispose d'une machine monoprocesseur, ayant une mémoire centrale de 32K mots, une page faisant 512 mots (64 pages en mémoires centrales), un bloc (unité de stockage sur le disque) faisant 512 mots. On considère le programme suivant :

Adresse	Instruction	Signification
E	`Loadx N`	reg d'index X = N
	`Load S,X`	reg accumulateur A = S[X]
	`Add V`	A = A + V
	`Store T,X`	T[X] = A
	`Subx K`	X = X - K
	`Brxpz E + 1`	Si X >= 0 alors aller en E + 1
	`Stop`	

Où:

- X est un registre d'index,
- A est un registre accumulateur,
- E est l'adresse du début du programme : 512 * 1 + 508,
- S et T sont des tableaux de taille [0 .. N], S [0] étant à l'adresse 512 * 11 + 168 et T[0] se trouvant à l'adresse 512 * 12 +456,
- N est une constante égale à 799 se trouvant à l'adresse 512 * 6 + 500,
- V est une constante égale à 2 se trouvant à l'adresse 512 * 8 + 100,
- K est une constante égale à 2 se trouvant à l'adresse 512 * 8 + 10.,

Ce programme effectue la chose suivante :

```
x = N;
do {T[X] = S[X] + V; x -= K} while (x >= 0);
```

3.1.

Sachant que chaque instruction, constante ou variable simple occupe un mot mémoire, représenter l'espace d'adressage du processus (numéro de page / déplacement dans la page de chaque donnée). Quels droits doit-on affecter aux différentes pages du processus ?

Donnée	Page / Dépl.	Droits	Donnée	Page / Dépl.	Droits
E	1 / 508	LX	S[0]	11 / 168	L
E+3	1 / 511	LX	S[343]	11 / 511	L
E+4	2 / 0	LX	S[344]	12 / 0	LE
E+6	2 / 2	LX	S[799]	12 / 455	LE
N	6 / 500	L	T[0]	12 / 456	LE
K	8 / 10	L	T[55]	12 / 511	LE
V	8 / 100	L	T[56]	13 / 0	E
			T[567]	13 / 511	E
			T[568]	14 / 0	E
			T[799]	14 / 231	E

Les actions de gestion de la mémoire peuvent être représentées à l'aide des opérations suivantes :

Trap (p) : Défaut de page pour la page p.

Charg (p,c) : Chargement de la page p à la place c en mémoire centrale.

Dech (p,c) : Déchargement de la page p se trouvant à la place c en mémoire centrale.

Mode (p,c,d) : Modification de la table des pages, p : numéro de page du processus, c place en mémoire centrale, d droits d'accès de la page.

Le processus précédent dispose pour s'exécuter des pages de mémoire centrale 17, 21, 22, 23, 37 et 42.

3.2.

En utilisant un algorithme premier chargé/premier déchargé et en supposant qu'au départ, aucune page n'est chargée, décrire les actions de gestion de mémoire sous la forme d'une suite composée des opérations précédentes. Donner la table des pages finale.

```
E :     TRAP(1), CHARG(1, 17), MOD(1, 17, XL)          /* Debut du programme */
        TRAP(6), CHARG(6, 21), MOD(6, 21, L)           /* Reference à N */
E+1     TRAP(12), CHARG(12, 22), MOD(12, 22, LE)       /* Référence à S[799] */
E+2     TRAP(8), CHARG(8, 23), MOD(8, 23, L)           /* Reference à V */
E+3     TRAP(14), CHARG(14, 37), MOD(14, 37, E)        /* Référence à T[799] */
E+4     TRAP(2), CHARG(2, 42), MOD(2, 42, XL)          /* Suite du programme */
```

Pas de défaut de page jusqu'à X = 567.

Lorsqu'un nouveau défaut survient, il n'y a plus de case disponible en MC. Il faut donc en réquisitionner une, la première chargée. C'est la case 17 qui contient la page 1. Il n'est pas nécessaire d'effectuer un déchargement car cette page n'a pas été modifiée.

```
E+3     TRAP(13), MOD(1, NC, A), CHARG(13, 17), MOD(13, 17, E)
                /* Référence à T[567] */
```

Lorsqu'on revient en E+1, la page de programme n'est plus chargée :

E+1 TRAP(1), MOD(6, NC, A), CHARG(1, 21), MOD(1, 21, XL)

Pas de défaut de page jusqu'à X = 343.

E+1 TRAP(11), MOD(12, NC, A), CHARG(11, 22), MOD(11, 22, LE)

/* Référence à S[343] */

Remarque : on n' a pas déchargé la page 12 avec Decharg car elle est en LE mais pas encore modifiée. Le système sait qu'elle n'a pas été modifiée avec le bit M de modification à 0.

Pas de défaut de page jusqu'à X = 55.

E+3 TRAP(12), MOD(8, NC, A), CHARG(12, 23), MOD(12, 23, LE)

/* Référence à T[55] */

E+4 TRAP(8), DECHARG(14, 37), MOD(14, NC, A), CHARG(8, 37), MOD(8,37, L)

Remarque : le décharg est nécessaire car la page 14 a été modifiée.

Ensuite, il n'y a plus de défaut de page. En fin d'exécution, on décharge les pages modifiées :.

E+6 DECHARG(12, 23), DECHARG(13, 17)

NB : on ne représente que les TRAP qui correspondent aux défauts de page.

4. SEGMENTATION PAGINEE

On considère une mémoire *segmentée paginée*. La taille des pages est de 512 mots.

Le processus P possède 3 segments : le segment 0 pour le code, le segment 1 pour la pile et le segment 2 pour les données. Le segment 0 a 1500 mots, le segment 1 en a 2000 et le segment 2 en a 3000. On suppose que la table des segments et les tables de pages sont déjà chargées en mémoire.

4.1.

Quels sont les droits associés à chaque segment ? Quel est l'avantage de découper ainsi l'espace d'adressage du processus ? `

Code : Lecture, Execution

Pile : Lecture Ecriture

Données : Lecture Ecriture

Avantage :

1) Protection de zone de donnée "sensiblee" (ainsi il est impossible de modifier le code)

2) Possibilité facile de partage pour les segments à accès en lecture. Par exemple on peut partagé sans problème le segment de code

Seules les pages suivantes ont été chargées (on donne le doublet <numéro de segment, numéro de page>) :

<0, 0> à l'adresse physique 2560

<1, 2> à l'adresse physique 4096

<2, 3> à l'adresse physique 1024.

4.2.

Décrire brièvement ce qui se passe sur une tentative de lecture en <0, 1678>, <1, 567> et <2, 1600> (Ces adresses sont au format segmenté).

(0,1678) => Dépassement de la capacité du segment, donc erreur le système reprend la main.

(1,567) => correspond à l'adresse virtuelle (1, 1, 55), la page (1,1) n'étant pas chargée il y aura un défaut de page.

(2, 1600) => correspond à l'adresse virtuelle (2,3,64) la page (2,3) étant déjà chargée en 1024 , l'accès se fera sans problème (à l'adresse physique : 1024+54 = 1088)

4.3.

Connaît-on l'adresse physique de la variable d'adresse virtuelle (1, 1060>. Si oui, quelle est-elle ? Sinon, justifiez.

(1,1060) => correspond à l'adresse virtuelle (1,2,36) la page (1,2) est chargé en 4096 l'adresse physique correspondante est donc 4096+36 = 4132