**Corrigé indicatif de série2 : Gestion de la mémoire**

**Exercice 1 :**

**A\ +250,+200, +100, +150, -250, +300, -100, -200, +50, +100, -150.**

1. **First Fit :** on cherche le premier espace qui convient.

|  |
| --- |
| **600** |
| **300** |
| **100** |

|  |
| --- |
| **+250**  **350** |
| **300** |
| **100** |

|  |
| --- |
| **+250**  **350** |
| **+200**  **100** |
| **100** |

|  |
| --- |
| **+250**  **350** |
| **+200**  **100** |
| **+100** |

**Etat initial +250 +200 +100**

|  |
| --- |
| **600** |
| **+200**  **100** |
| **+100** |

|  |
| --- |
| **+250**  **350** |
| **+200**  **100** |
| **+100** |

|  |
| --- |
| **+150** |
| **450** |
| **+200**  **100** |
| **+100** |

|  |
| --- |
| **+150** |
| **450** |
| **+200**  **100** |
| **+100** |

**+150 en attente -250 +150 +300 en attente**

|  |
| --- |
| **+150** |
| **450** |
| **300** |
| **100** |

|  |
| --- |
| **+150** |
| **450** |
| **+300** |
| **100** |

|  |
| --- |
| **+150** |
| **450** |
| **+300** |
| **+50** |
| **50** |

|  |
| --- |
| **600** |
| **+300** |
| **+50** |
| **50** |

|  |
| --- |
| **+100** |
| **500** |
| **+300** |
| **+50** |
| **50** |

**-100 et -200 +300 +50 et +100 en attente -150**

**+100**

**2)Et 3) même étapes à suivre.**

**B\ vous pouvez développer ces algorithme avec plus de détailles.**

**First Fit :**

**Algorithme First\_Fit(Tpgm:entier)**

Type

Liste=structure

Taille :entire;

Suiv: ↑liste ;

Fin liste ;

Var tete,p,q : ↑liste ;

Tr : booléen ;

Debut

Tr=faux ;

P=tete ;

Tanque ((p !=null) et (non(tr)) faire

Si (Tpgm <=p.taille) alors

Tr=vrai ;

Finsi ;

P=p.suiv ;

Fin tq ;

Si (tr) alors /\* on a trouvé une première zone libre et suffisante\*/

Si p.taille=Tpgm alors

Supprimer (p) ;

Sinon

p.taille=p.taille- Tpgm ;

finsi ;

Sinon

Ecrire(« pas de zone suffisante ; pgm en attente) ;

Fin.

**Best Fit :**

**Algorithme Best\_Fit(Tpgm:entier)**

Type

Liste=structure

Taille :entire;

Suiv: ↑liste ;

Fin liste ;

Var tete,p,q : ↑liste ;

Min,K :entier ;

Debut

k=0 ;

P=tete ;

Tanque ((p !=null) et (non(tr)) faire

Si (k=0) alors

Si (Tpgm <=p.taille) alors

K=1 ;

min=p.taille-Tpgm ;

q=p ;

Finsi

sinon

si (p.taille>Tpgm) et (p.taille-Tpgm<min) alors

min= p.taille-Tpgm ;

Finsi ;

P=p.suiv ;

Fintq ;

Si (k=0) alors

Ecrire(« pas d’espace suffisant, attente ») ;

Sinon

Si min=0 alors

Supprimer (p) ;

Sinon

q.taille=q.taille- Tpgm ;

finsi ;

Fin.

Worst fit la même solution en changeant < à> et min par max.

**Evaluation des trois méthodes d’allocation.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | First fit | Best fit | Worst Fit |
| Avantages | Simple, rapide | Réduire la fragmentation interne | Bonne gestion d’espace mémoire |
| Inconvénients | Fragmentation interne | complexe | Complexe |

**Exercice 2 :**

1. L’algorithme commence par placer P1, P2 et P3 en mémoire, ce qui ne laisse plus que 50Ko de libre. On s’occupe donc de placer P4, P5 et P6 après que P1 et P3 aient été discordés de la mémoire après 2 ms.

— A 2 ms : 600 P2 :150 450

— Placement de P4 : P4 :300 300 P2 :150 450

— Placement de P5 : P4 :300 P5 :200 100 P2 :150 450 2

— Placement de P6 : P4 :300 P5 :200 100 P2 :150 P6 :400 50

Taux de fragmentation à 3 ms = (100 + 50)/1200 = 0.13

1. Déroulez l’algorithme d’allocation Best-fit (plus petit bloc libre) en précisant à chaque instant les allocations et désallocations qui sont effectuées, et donnez le taux de fragmentation au temps 3 ms (1 point)

Correction : L’algorithme commence par placer P1, P2 et P3 en mémoire, ce qui ne laisse plus que 50Ko de libre. On s’occupe donc de placer P4, P5 et P6 après que P1 et P3 aient été discardés de la mémoire après 2 ms.

— A 2 ms : 600 P2 :150 450

— Placement de P4 : 600 P2 :150 P4 :300 150

— Placement de P5 : P5 :200 400 P2 :150 P4 :300 150

— Placement de P6 : P5 :200 P6 :400 P2 :150 P4 :300 150

Taux de fragmentation à 3 ms = (150)/1200 = 0.13

1. Déroulez l’algorithme d’allocation Worst-fit (plus grand bloc libre) en précisant à chaque instant les allocations et désallocations qui sont effectuées, et donnez le taux de fragmentation au temps 3 ms (1 point)

**Correction** : L’algorithme commence par placer P1, P2 et P3 en mémoire, ce qui ne laisse plus que 50Ko de libre. On s’occupe donc de placer P4, P5 et P6 après que P1 et P3 aient été discardés de la mémoire après 2 ms. chacun des algorithmes suivants, et précisez le taux de fragmentation au temps 3 ms :

— A 2 ms : 600 P2 :150 450

— Placement de P4 : P4 :300 300 P2 :150 450

— Placement de P5 : P4 :300 300 P2 :150 P5 :200 250

— On est forcé d’attendre la fin de P4 (durée 3 ms) pour placé P6

— A 5 ms : 600 P2 :150 P5 :200 250

— Placement de P6 : P6 :400 200 P2 :150 P5 :200 250

Taux de fragmentation à 3 ms = (300 + 250)/1200 = 0.46

**Exercice 3:**

**A)**

1- les adresses virtuelles : **100, 210, 355, 120,420, 110, 200, 550, 139, 201, 395, 404, 505.**

Les références : diviser les adresses sur la taille de la page (100).

1,2,3,1,4,1,2,5,1,2,3,4,5.

**FIFO**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 4 | 4 |
|  | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 |
|  |  | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
|  |  |  | 4 | 4 | 4 | 4 | 3 | 3 | 3 |

NbDp=10

**LRU**

1,2,3,1,4,1,2,5,1,2,3,4,5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  | 3 | 3 | 5 | 5 | 4 | 4 |
|  |  |  | 4 | 4 | 3 | 3 | 3 |

NbDp=8

**OPTIMAL**

1,2,3,1,4,1,2,5,1,2,3,4,5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 4 |
|  | 2 | 2 | 2 | 2 | 2 |
|  |  | 3 | 3 | 3 | 3 |
|  |  |  | 4 | 5 | 5 |

NbDp=6

B)

1)+2)

-Plusieurs algorithmes ont été proposés :

Méthodes des compteurs ;

Méthode de la pile ;

Méthode des masques

-Vous pouvez implémenter autres méthodes puisque la méthode des masques a été présentée dans le cours.

**Algorithme LRU avec la méthode des masques :**

C’est une méthode d’implémentation de l’algorithme de remplacement LRU selon la méthode des masques.

1. A chaque page est associée un octet.

2. La valeur initiale du masque est un octet (00000000). Le bit de poids fort est mis à 1 à chaque utilisation de cette page.

3. A chaque période (N ms), on fait un décalage à droite de l’octet associé à chaque page.

4. la page victime c’est la page qui a la plus petite valeur de masque au moment du lancement de l’algorithme.

**Algorithme** LRU ;

**Var trouv :booléen ;**

Min : **entier ;**

**Début**

Min← octet-min(nombre-page-charge-mc) ;

Trouv← test-unique(min) ;

**Si** (trouv) **Alors {la page est unique}**

Num-page-victime← Selection(min,list-page-charg) ;

**Sinon {la page n’est pas unique}**

Num-page-victime ← Selection(min,list-page-charg, FIFO) ;

**Finsi ;**

3)



**Exercice 4:**

I a)

1) 224 = 16Mo

2) 9 bits

3) 24 – 9 = 15 bits

4) 20 bits

5) 20 – 9 = 11 bits

6) 215 = 32Ko

I b) Oui. Le programme a besoin de 2000 octets de données et de 1024 octets de code, donc un total de 3024 octets en mémoire. Le nombre de pages occupées : [3024 / 512] = 6 pages. Dans la dernière page, il reste 512 – 464 = 48 octets libres, ce qui cause la fragmentation interne.

II) Le format PT1(8bits) PT2(8bits) PT3(8bits) Offset(8bits)

Le nombre de tables : 1 + 256 + (256\*256).

Le nombre d’entrées : 256

**Exercice 5 :**

8212 = 2\*4096 + 20

1. le segment = S1

2. le numéro de page dans le segment = 3

3. le déplacement dans la page = 20

4. cadre = 0

5. le déplacement dans la case = 20

6. L’adresse physique est donc 20 en décimal. L’adresse physique est exprimée sur 16 bits (64 kio = 2 16), dont 4 bits pour le numéro de cadre et 12 bits pour le déplacement dans la case (4 kio = 2 12 ) , ce qui nous donne en binaire 0000 0000 0001 0100.