USTHB 2017/2018

**Faculté d'Electronique et Informatique**

**Département Informatique**

**CORRIGE DE L'EMD (SYSTEME)**

**Exercice 1 (5pts)**

1. Le nombre d’adresses que peut contenir un bloc d’index est :

Nb = Taille\_Bloc/Taille\_Adresse = 256 ko/1 ko = 256.

1. Les structures de données pour :

* F1 : le nombre de bloc = 2304 ko/256 ko = 9 blocs

|  |
| --- |
| Attribut |
| @b1  …..  @b9  null |
| null |
| null |
| null |

* F2 : le nombre de blocs = 2560/256 = 10 blocs

|  |
| --- |
| Attribut |
| @b1  …..  @b9  @b10 |
| null |
| null |
| null |

* F3 : le nombre de blocs = 20480 / 256 = 80 blocs

|  |
| --- |
| Attribut |
| @b1  …..  @b9  @ b10 |
| @ SI |
| null |

|  |
| --- |
| @ b11  ……  @ b80 |

|  |
| --- |
| null |

* F4 : le nombre de blocs = 68096 / 256 = 266 blocs

|  |
| --- |
| Attribut |
| @b1  …..  @b9  @ b10 |
| @ SI |
| null |

|  |
| --- |
| @ b11  ……  @ b80  ……..  @ b266 |

|  |
| --- |
| null |

1. Le nombre d'accès pour lire

F1 est 9\* ( 0+1) = 9 accès DD

F4 est 10 ( 0+1) + 256 (1+1) = 522 accès DD

1. Ajout d'un bloc à la fin de chaque fichier:

F1 : allouer un bloc de donnée b et le mettre dans inode. Adr[10],

B = allouerBlocDD() ; inode.adr[10] = B ;

F2 : allocation d'un bloc d'index SI et d'un bloc de données.

B = allouerBlocDD() ; SI = allouerBlocDD() ; inode.adr[11] = SI ; SI[1] = B ; SI[2] = null ;

F3 : il faut allouer un bloc de données.

B = allouerBlocDD() ; SI = inode.adr[11] ; SI[81] = B ; SI[82] = null ;

F4 : allocation d'un bloc d'index DI et d'un bloc d'index index1 et d'un bloc de données.

B = allouerBlocDD() ; DI = allouerBlocDD() ; Index1 = = allouerBlocDD() ;

inode.adr[12] = DI ; DI[1] = Index1 ; DI[2] = null ;

Index1[1] = B ; Index1[2] = null;

1. L’algorithme

Debut

Soit taille le nombre de blocs du fichier

B = allouer\_BlocDD() ;

Si B= -1

Alors écrire ( ‘’ DD saturé’’) ;

Sinon

Nb = taille ; taille ++

Si Nb < 10

Alors; inode.adr[taille +1] = B ; inode.adr[taille+2] = null ;

Sinon si Nb = 10

Alors SI = allouer\_BlocDD() ;

Si SI= -1

Alors écrire ( ‘’ DD saturé’’) ;

Sinon inode.adr[11] = SI ;

SI[1] = B : SI[2] = null ;

Fsi

Sinon si Nb < 266

Alors Nb =Nb -10 ;

SI = inode.adr[11] ;

SI[Nb] = B ;SI (NB+1] = null ;

Sinon si Nb = 266

Alors

DI = = allouer\_BlocDD() ;

Index = = allouer\_BlocDD() ;

Si ( DI = -1 ou Index = -1 )

Alors écrire ( ‘’disque saturé’’) ;

Sinon

Inode.adr[12] = DI ;

DI[1] = Index ; DI[2] = null ;

Index[1] = B ; Index[2] = nul ;

Fsi

Sinon si Nb < 10+ 256\*256

………………………………

………………………..

Pareillement

1. Extension d'un fichier

Un fichier peut ne pas être étendu dans deux cas:

1) Insuffisance d'espace disque : disque saturé

2) La taille du fichier est au maximum. Taille maximale = 10 + 256 + 2562 + 2563 blocs= 10 + 256 + 65536 + 16777216 = 16843018 blocs = 4311812608 Ko = 4.311812608 To

**Exercice 3**

1. Ecrire une solution qui permet aux avions d'atterrir en se synchronisant à l’aide **des sémaphores** au niveau des pistes **PS1** et **PS2.**

|  |  |
| --- | --- |
| **Var** S1, S2 : Sémaphore ;  Init(S1, 1); // Permet l'accès en EM à la piste PS1 entre les avions de voyageurs  Init(S2, 1); // Permet l'accès en EM à la piste PS2 entre les avions de marchandises | |
| **Processus Avion\_Voyageurs(I, T)**  **Début**  **P(S1) ;**  <Atterrir sur PS1>  ;  **V(S1) ;**  **Fin.** | **Processus Avion\_Marchandises(I, T)**  **Début**  **P(S2) ;**  <Atterrir sur PS2>  ;  **V(S2) ;**  **Fin.** |

1. Sachant que tout avion fournit sa réserve en carburant **R**, modifier la solution pour favoriser l’avion qui possède **la plus faible réserve**.

**a/ Structures de données**

* S : un tableau N de sémaphores privé ( init à 0), un sémaphore pour chaque avion. On est obligé d’avoir un tableau vu qu’il y a la priorité entre eux. N étant le nombre maximal d’avions.
* **Piste1occupée et piste2occupée : var booléennes initialisées à faux**
* FM,FV: files d’enregistrement chacune contenant l’identité de l’avion et la quantité du carburant. Elles sont vides initialement.
* MutexV, MutexM : Sémaphores binaires (init à 1)  pour protéger la file f et les variables booléennes **pisteVoyOccupé et pisteMarchOccupé**

**b/ l’algorithme**

Processus Avion\_Voyageurs(I, T, C)

**Début**

P(MutexV);

Si piste1occupée = vrai alors inf.id = i ; inf.carb = c ; emfiler(fV,inf) ;v(MutexV) p(S[i]);

Sinon piste1Occupée = vrai ; v(MutexV) ;

fsi

< Attérir piste 1>

P(MutexV ;

Piste1Occupée = faux ;

Si non file vide (fV)

Alors trierCroissant(fV) ;

x = tete(fV) ->inf;

j = x.id;

V(s[j]) défiler(fV);

piste1Occupée = vrai ;

  fsi

**Fin**

**3.** Modifier la solution 1 pour permettre à un avion de voyageurs, en attente, d’exploiter la piste **PS2**. A la libération de la piste **PS2** de marchandises, celle-ci peut être attribuée à un avion de voyageurs si :

* aucun avion de marchandises n’est en attente.
* le nombre d’avions en attente sur **PS1** dépasse un certain seuil **S\_Max**.

**a/ Structures de données :**

S1, S2, MutexV, MutexM : Sémaphore ;

attM : entier initialisé à 0 exprimant le nombres d’avions de marchandise en attente

attV : entier initialisé à 0 exprimant le nombres d’avions de voyageurs en attente

orienter : booléen initialisé à faux ; détermine si on doit orienter les avions de voyageurs vers la piste 2 ou non.

Init(S1, 0); // Permet de bloquer les avions de voyageurs quand leur piste est occupée

Init(S2, 0); // Permet bloquer les avions marchandises quand leur piste est occupée

Init(Mutex, 1); permet de protéger l’accès aux variables orienter, attV et attM.

**b/ algorithmes**

**Acces\_Marchandise()**

**Début**

**P(Mutex);**

**Si** piste2occupée = vrai

Alors attM++ ; V(Mutex) ; P(S1);

Sinon // la piste 2 est livre

Piste2occupée = vrai; v(mutex) ;

Fsi

Fsi

<Attrir piste2> ;

appel procédure liberer\_Piste2()

fin

procédure liberer\_Piste2()

// On libère d’abord un avion de marchandise, mais si n’existe aucun avion de marchandise en attente, on réveille l’avion de voyageur en tête de file si le nombre d’attente dépasse S\_max.

debut

piste2occupée = faux;

si attM ≠ 0

alors piste2occupée = vrai ; attM-- ; V(S2) ;

sinon si ( attV > S\_Max )

alors attV-- ; piste2occupée = vrai ; V(S1) ; orienter = vrai

fsi

fsi

**Acces\_Voyageur()**

**// Le processus avion accède ou se bloque normalement mais dès qu’il est réveillé, on doit  déterminer vers quelle piste l’orienter.**

**Début**

**P(Mutex);**

**Si** piste1occupée = vrai

Alors attV++ ; v(Mutex) ; p(S1);

Sinon // la piste est livre

piste1occupée = vrai; v(Mutex) ;

fsi

appel procédure Sélectionner\_Piste()

fin

procédure Sélectionner\_Piste()

début

P(Mutex)

Si (orienter = faux)

appel procédure Liberer\_Piste1() ;

Sinon orienter = faux ; V(Mutex) ;

< Atterrir Piste2> ;

appel procédure Liberer\_Piste2() ;

fsi ;

fin

procédure liberer\_Piste1()

debut

piste1occupée = faux;

si attV ≠ 0

alors piste1occupée = vrai ; attV-- ; V(S1) ;

fsi

fin

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **Condition d'entrée à la SC** | **Exclusion Mutuelle** | **Blocage Mutuel** | **Progression** | **Attente Bornée** |
| Jetoni <= Jetonj | **Jetoni >Jetonj** | Non assurée car  si pj en sc  Pi arrive met Jetoni =Jetonj +1  Donc Jetoni > JetonJ  Pi rentre aussi en SC | On ne traite pas vu que EM non assurée | On ne traite pas vu que EM non assurée | On ne traite pas vu que EM non assurée |
| Jetoni > Jetonj | **Jetoni  <=Jetonj** | Non assurée car  Si pi et pj arrive en même temps au départ ils ont leurs jetons à 0  On peut avoir le scénario où  Jetoni  = Jetonj = 1 ;  Donc pi et pj rentrent tout les deux | On ne traite pas vu que EM non assurée | On ne traite pas vu que EM non assurée | On ne traite pas vu que EM non assurée |
| Jetoni >= Jetonj | **Jetoni  < Jetonj** | Assurée car  1. pj en SC ; et pi désire rentrer Jetoni = Jetonj +1  Jetoni >jetonj donc cond non vérifiée d’où pi ne peut pas rentrer  2. Sc libre et pi et pj désirent rentrer  On peut avoir trois cas .  a/ Jetoni<Jetonj : pi rentre seul  b/ Jeton i> Jetonj : pj rentre seul  c/ Jeton i = Jetonj : aucun ne rentre | BM Non évité d’âpres 2c  Il existe un scénario ou les deux bloquent et aucun ne rentre en SC | Non assurée car  Si pj est loin de la SC, Jetonj = 0  Jetoni = 0+1=1> Jetonj  Donc la condition d’entrée n’est pas vérifiée donc pi reste en attente | On ne peu pas étudier vu que le BM n’est pas évité |