Exo1 :

Lecture ← 0 ; Ecriture ← 1 — RC = 0 ; RC = 1 ← lancer le transfert — RE = 0 Occupé ; RE = 1 Libre

① Les ≠ Registres des controleurs et DMA

(1 pt)

- RE_Disk, RC_Disk, RD_Disk
- RE_Scanner, RC_Scanner, RD_Scanner
- RAD_DMA, RCpt_DMA, RE/C_DMA ← pour le sens du transfert

② Les ≠ pgmes systèmes

N1, N2

SVC (cause, id_périph, @, N ..) (0,5 pt)

Debut
  <S.G.Cxt>
  switch cause of
    ⋮
    Lecture : Etat_processus ← bloqué ;
      // Appel au pilote du scanner
      Entrée_Asynchone_scanner.Init (@, N1) ;
      // Appel au scheduler
      Lpsw (scheduler) ;
    Ecriture : Etat_processus ← bloqué ;
      Sortie_Asynchrone_DMA_Disque.Init (@, N2) ;
      Lpsw (scheduler) ;
  fin ;
  <R.G.Cxt>
fin.

Entrée_Asynchrone_scanner.Init(@, N1) (1pt)

Debut
  si RE_scanner = 1 Alors // Lancer le transfert
    RE_scanner = 0 ;
    RC_scanner = 1 ;
  sinon Enfiler (FEIS, SemEIS) ;
  fsi ;
fin

```
Sortie_Asynchrone_DMA_Disque.Init(@, N2). (1,5pt).
Debut
  si RE_Disk = 1 Alors // Initialiser le DMA.
                       RAD_DMA ← @.
                       RCpt DMA ← N2
                       RdC_DMA ← 1 ; // 1 ≡ ecriture
                       // Initialiser le controleur.
                       RE_Disk ← 0 ;
                       RC_Disk ← 1 ;
                  sinon // mettre en attente la demande E/S
                       enfiler (FE/S, Den E/S.
  fsi.
fin.
```

Routine d'it Disque( ) (1pt).

```
Debut
  <S.G.Ctl>
  si erreur Alors Traiter_Erreur();
            sinon Etat_processus ← prêt;
                  si ¬vide (FE/S)
                     Alors // Lancer une autre dem E/S.
                     Sortie_Asynchrone_DMA_Disk.Init(dem.@, dem.N2)
                  fsi
  fsi
  <R.G.Ctl>
fin.
```

Routine d'it scanner (); (2pts).

```
Debut
    <$.G. Cats.
    si erreur Alors Traiter_Errem();
              sinon // Recuper le car.
                    R ← RB_scanner ; // R est un registre du processeur
                    Mov a0, R ;
                    N-- ;
                    si N ≠ 0 Alors a0++
                                   // Lancer le transfert d'1 autre car
                                   RE_Scanner ← 0 ;
                                   RC_scanner ← 0 ;
                             sinon Etat_processus ← Prêt
                                   si ¬vide (FEIS)
                                      Alors Defiler (dem, FEIS)
                                            Entree_Asynchrone_scanner.Init
                                                (dem.a0, dem.d1)
                                   fsi
                    fsi
    fsi
fin.
```

Exercice 2

1. Diagramme d'exécution :

 = 1ms

Periph												3	3	3	3	3								1	1		1	1	1	1							
CPU	1	1	1	2	2	2	3	3	3	3	3	1	2	2	2	2	3	3	3	3	3	3	1	1	2	2	1	2	2	2	2	2	1			

2. Etat des files d'attente :

	A l'instant 15ms	A l'instant 25ms	A l'instant 30ms
File Prêt	→ P1	∅	∅
File Bloqué	P3	P1	P1

3. Les priorités des processus :

	A l'instant 15ms	A l'instant 25ms	A l'instant 30ms
P1	6	10	0
P2	6	0	2
P3	10		

(1) le nombre maximum d'entrées dans la table des pages.

16 bits pour coder le n° de la page

⇒ on peut avoir au maximum 2^{16} pages.

⇒ **2^{16} entrées dans la table des pages.**

(2) Nombre de pages occupées par les tableaux A, B, etc.

Un entier est sur 1 octets

la taille de la page est 256 octets

la taille de chaque tableau est 1024 octets.

⇒ le nombre de pages de chaque tableau

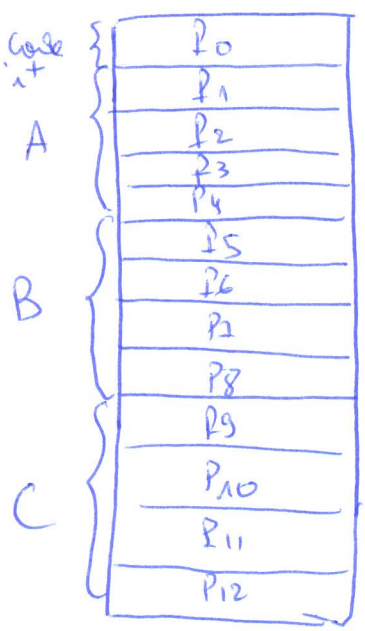
$= \frac{1024}{256} =$ **4 pages**

Donc, le nombre de pages pour les tableaux A, B, C

est **12 pages**.

(3) La chaine de references

pgme



0 1 5 9 | 0 2 6 10 | 0 3 7 11 | 0 4 8 12

256 fois | 256 fois | 256 fois | 256 fois

Rq° si on considère le code en assemblem

C[I] = A[i] + B[i].

=

```
Mov R1 , A[I]
Mov R2 , B[I]
ADD R1 , R2
Mov C[i] , R1
```

⇒ la chaine = 0 1 0 5 0 9 | 0 2 0 6 0 10 | 0 3 0 7 0 11 | 0 4 0 8 0 12 (2)

256 fois | 256 fois | 256 fois | 256 fois

Exo 3 :

- Taille de la MC = 1 Mo = 2^{20} octets.
- Taille de la page = 256 octets = 2^8 octets
- L'espace d'adressage logique est de 16 Mo = 2^{24} octets

(1)(a) Le nombre de bits d'une adresse logique. (0,25pt)

@ logique = (p, d).

l'espace d'adressage logique est de 16 Mo = 2^{24}.

⇒ **24 bits pour coder une @ logique**

(A) le nombre de bits du déplacement. (25pt)

taille de la page = 256 octets = 2^8 octets

⇒ **8 bits pour coder le déplacement**

(*) le nombre de bits du numéro de la page. (25pt)

= le nombre bits de l'@ logique − nombre de bits du déplacem

= 24 − 8 = **16 bits pour coder le numéro de la page**

(*) le nombre de bits d'une @ physique. (25pt)

taille de MC = 1 Mo = 2^{20} octets

⇒ **20 bits pour coder une @ physique.**

* le nombre de bits du numéro de frame. (1pt)

$$\frac{\text{Taille de la MC}}{\text{Taille page}} = \frac{2^{20}}{2^8} = 2^{12}$$

⇒ **12 bits pour coder le n° de frame**

④ Application de LRU avec 4 frames ←

	0	1	5	9	0	2	6	10	0	3	7	11	0	4	8	12
F1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2		1	1	1	1	2	2	2	2	3	3	3	3	4	4	4
F3			5	5	5	5	6	6	6	6	7	7	7	7	8	8
F4				9	9	9	9	10	10	10	10	11	11	11	11	12
DP	x	x	x	x	-	x	x	x	-	x	x	x	-	x	x	x
//	//	//	Nb_DP = 13/16 = ? faux													

⑤ LRU avec 3 frames.

	0	1	5	9	0	2	6	10	0	3	7	11	0	4	8	12
F1	0	0	0	9	9	9	6	6	6	3	3	3	0	0	0	12
F2		1	1	1	0	0	0	10	10	10	7	7	7	4	4	4
F3			5	5	5	2	2	2	0	0	0	11	11	11	8	8
DP	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Nb_DP = 16 = 100 % faux.																