Question 1 (4 pts) : Généralités

a) **[2 pts]** Considérez le moniteur *ProducteurConsommateur* suivant (cas d'un producteur et d'un consommateur) :

```
Moniteur ProducteurConsommateur
{       const N=100 ;
        int tampon[N] ;
        boolc nplein, nvide ; // variables de condition
        int compteur =0, ic=0, ip=0 ;
        void produire (int objet)
        {       if (compteur==N) wait(nplein) ;
                tampon[ip] = objet ;
                ip = (ip+1)%N ;
                compteur++ ;
                if (compteur==1) signal(nvide) ;
        }
        int consommer ( )
        {       int objet ;
                if (compteur ==0) wait(nvide) ;
                objet = tampon[ic] ;
                ic = (ic+1)%N ;
                compteur -- ;
                if(compteur==N-1) signal(nplein) ;
                return objet ;
        }
}
```

On veut remplacer les instructions de la fonction *produire* par :

```
if (compteur==N) wait(nplein) ;
if (compteur==0) signal(nvide) ;
tampon[ip] = objet ;
ip = (ip+1)%N ;
compteur++ ;
```

Pour chacune des deux interprétations de la fonction « signal » : «signal and wait» et «signal and continue», indiquez si ces modifications vont altérer le bon fonctionnement du producteur ou du consommateur. Justifiez votre réponse.

b) **[1 pt]** Sous Windows, est-ce qu'un processus fils peut hériter le descripteur (handle) d'un objet créé par la fonction GetProcessHeap ? Justifiez votre réponse.

c) **[1 pt]** Sur une clé USB de 1GiO, le système de fichiers de type FAT utilise des blocs de 4KiO. Chaque numéro de bloc est codé sur 32 bits (4 octets). Quelle est la taille, en octets, de la table FAT ? Quel est en pourcentage l'espace occupé par cette table ?

Question 2 (5.5 pts) : Moniteurs, variables de condition et interblocage

Pour permettre à un ensemble de threads d'un même processus de partager des données de type pile, on décide d'implémenter un moniteur *pile_t*. On vous donne le code à compléter du moniteur *pile_t* :

```
Moniteur pile_t
{    const int N = 2; // la taille de la pile
     int P[N]; // la pile
     .... // autres variables ou constantes
     void Empiler ( int o) { .... }; // doit bloquer si la pile est pleine
     int Depiler ( ) { ... .};// doit bloquer si la pile est vide
}
```

a) **[2 pts]** Complétez le moniteur *pile_t* (les déclarations et les fonctions *Empiler* et *Depiler*). Expliquez le rôle de chaque variable utilisée.

b) **[1.5 pt]** Pour permettre aux threads d'empiler dans une pile, l'un à la suite de l'autre, les éléments d'un tableau, on vous demande d'ajouter la fonction *EmpilerA* dans le moniteur. Sa signature est : « *void EmpilerA(int T[], int m) ;* », où *m* est la dimension du tableau *T*. La valeur de *m* est supposée supérieure ou égale à *1* et inférieure ou égale à la taille de la pile. Complétez et/ou modifiez le code donné en *a)* pour y inclure cette fonction. Il n'est pas demandé de gérer les problèmes de famine.

c) **[2 pts]** Trois *th1, th2 et th3* partagent *2* piles p1 et p2 de même taille 2. Chaque thread thi (i=1,2) boucle sur la séquence d'instructions : générer un nombre aléatoire en utilisant la fonction « int gen_alea() ; » et empiler le nombre généré dans la pile *pi*. Le thread *th3* boucle sur la séquence d'instructions : dépiler p1, dépiler p2 et empiler dans p2 le plus petit des deux nombres récupérés de p1 et p2. Est-ce que les threads peuvent se retrouver en situation d'interblocage ? Si, oui, donnez un scénario (un ordre d'appels aux fonctions *Empiler/Depiler* par les threads*)* qui mène vers un interblocage.

Question 3 (6.5 pts) : Gestion de la mémoire

On considère un système de pagination à trois niveaux dans lequel les adresses (virtuelles et physiques) sont codées sur 32 bits. La taille d'une page est de 2 KiO. La taille de chaque table de pages, peu importe son niveau, est égale à 512 octets. Chaque entrée d'une table de pages est composée de 4 octets. Le premier octet est réservé aux bits de contrôle (bit de présence, bit de référence, bit de modification, bits de protection, etc.). Les octets restants servent à localiser la page ou la table de pages de niveau suivant.

a) **[1 pt]** Quelle est la taille maximale, en nombre de pages, de l'espace virtuel d'un processus, supporté par un tel système ? Quel est le nombre maximal de tables de pages (de niveaux 1, 2 et 3) d'un processus ?

b) **[1.5 pt]** Donnez le format d'une adresse virtuelle. Expliquez au moyen d'un schéma simple comment vérifier si la page référencée par une adresse virtuelle est présente en mémoire physique.

c) **[2 pts]** Donnez le numéro de la page référencée par l'adresse virtuelle *0x000A0A00*. Supposez que la page référencée par cette adresse virtuelle est chargée en mémoire physique dans le cadre numéro *8* (la numérotation commence à partir de *0*). Donnez son adresse physique (en hexadécimal). Cette adresse physique est-elle dépendante du nombre de niveaux utilisés pour organiser les tables de pages ? Justifiez votre réponse.

d) **[2 pts]** Supposez que ce système réserve 6 cadres physiques libres à chaque processus créés. Aucun autre cadre n'est alloué au processus. Durant son exécution, un processus référence dans l'ordre les pages : 0 1 2 3 4 5 6 1 0 5 8 4 3 2 1. Donnez pour chacun des algorithmes de remplacement de pages suivants, l'évolution de l'état des cadres réservés au processus ainsi que le nombre de défauts de pages générés :
   - FIFO,
   - LRU.

Question 4 (4 pts) : Ordonnancement

On veut étudier l'ordonnancement de trois tâches temps réel périodiques synchrones T1, T2 et T3, ayant les caractéristiques suivantes :

Tâche	Date d'arrivée (Oi)	Durée d'exécution (Ci)	Période=Échéance (Pi=Di)
T1	0	2 (ER)	6
T2	0	2 (EE)	8
T3	0	5 (RRRRR)	12

Où R est une ressource partagée, en exclusion mutuelle, entre T1 et T3.

a) **[2 pts]** Ces tâches sont-elles ordonnançables selon RMA ? Est-ce qu'il y a un problème d'inversion de priorités ? Si oui, indiquez sa durée. Ces tâches sont-elles ordonnançables selon RMA, dans le cas où le protocole PIP est utilisé pour traiter les inversions de priorités ? Justifiez votre réponse.

b) **[2 pts]** Ces tâches sont-elles ordonnançables selon EDF ? Si deux tâches ont la même priorité dynamique, l'ordonnanceur va élire celle qui a la plus petite période. Justifiez votre réponse.

Joyeuses fêtes

Le corrigé

Question 1 :

a)

Pour «signal and continue», cette permutation n'affecte pas les comportements du producteur et du consommateur, car après l'exécution de la fonction signal, le producteur reste actif dans le moniteur et le consommateur ne pourra pas accéder au moniteur.

Pour «signal and wait», cette permutation peut affecter le comportement du producteur ou du consommateur, car après l'exécution de la fonction signal, le producteur est suspendu dans le moniteur, après avoir activé le consommateur.

Supposez que le consommateur rentre en premier, il se bloque suite à l'appel à wait(nvide) ;. Le producteur rentre ensuite dans le moniteur. Le compteur étant égal à 0, il réveille le consommateur avant de se suspendre. Le consommateur va consommer alors que la production n'est pas complétée puis décrémente le compteur qui devient égal à -1. Lorsque le consommateur quitte le moniteur, le producteur est activé. Il va produire puis incrémenter la variable compteur qui devient égal à 0. Si le consommateur rentre dans le moniteur, il va consommer la première production, etc.

b) Non car l'objet retourné est le tas (le heap) du processus. Ses adresses virtuelles font références à l'espace virtuel privé du processus père. Les processus fils ont des espaces virtuels différents de celui de leur père.

c) Taille = (1 GiO / 4 KiO) * 4 octets = 1 MiO.
Pourcentage : 100/Ki %= 0.09 %

Question 2 :

a)

```
Moniteur pile_t
{     const  int  N = 2;  // la  taille de la pile
      int  P[N];   // la pile
      int  s = 0;    // le sommet de la pile
     boolc  nplein,, nvide;  // pour bloquer si la pile est vide (pleine)
      void  Empiler ( int o)  {  while(s==N) wait(nplein);   P[s] = o; s++;  if(s==1) signal(nvide); };
      int  Depiler( )  { int t; while(s==0) wait(nvide); s--;    t=P[s] ; if(s==N-1) signal(nplein); return
t;};
}
```

b)

```
Moniteur pile_t
{     const  int  N = 2;  // la  taille de la pile
      int  P[N];   // la pile
      int  s = 0;    // le sommet de la pile
     boolc  nplein,, nvide;  // pour bloquer si la pile est vide ( n'a pas suffisamment de cases vides)
      void  Empiler ( int o)  {  while(s==N) wait(nplein);   P[s] = o; s++;  if(s==1) signal(nvide); };
      int  Depiler( )
```

```
    { int t; while(s==0) wait(nvide);  s--;    t=P[s] ; if(s==N-1) signal(nplein); return t;};
   void  EmpilerA ( int T[], int m)
    { int i;  while(s>N-m) wait(nplein);   for(i=0; i<m; s++,i++) {P[s] = T[i];
      if(s==m) signal(nvide);     };
}
```

c)

Oui, pour par exemple, le scénario suivant :
th1 p1.Empiler(gen_alea()) ;
th2 p2.Empiler(gen_alea()) ; p2.Empiler(gen_alea()) ;
th3 t1=p1.Depiler() ; t2=p2.Depiler() ;
th2 p2.Empiler(gen_alea()) ; p2.Empiler(gen_alea()) ; → il bloque car pile p2 est pleine
th3 p2.Empiler(min(t1,t2)) ; → il bloque car pile p2 est pleine.
th1 p1.Empiler(gen_alea()) ; p1.Empiler(gen_alea()) ; → il bloque car pile p1 est pleine
th1, th2 et th3 bloqueront indéfiniment.

Question 3

a) **$2^{32} / 2^{11} = 2^{21} = 2$ Mi pages. Chaque table a 512 / 4 = 128 entrées**
Le nombre maximal de tables de pages : $1 + 2^7 + 2^7 * 2^7 = 1 + 128 + 128 * 128$

b) **7 bits (entrée dans la table du 1er niveau) + 7 bits (entrée dans la table du 2nd niveau) + 7 bits (entrée dans la table du 3ième niveau) + 11 bits (pour le déplacement dans une page). Les 7 bits de poids fort donnent l'entrée dans la table du 1er niveau qui contient l'adresse de la table du second niveau. Les 7 bits suivants avec l'adresse de la table de second niveau nous donne l'entrée dans la table à considérer pour récupérer l'adresse de la table de 3ième niveau. Les 7 bits suivants avec l'adresse de la table de troisième niveau nous donne l'entrée dans la table à considérer pour récupérer les informations concernant la page référencée par l'adresse virtuelle, le bit de présence va nous indiquer si la page est mémoire ou non.**

c) **0x000A0A00 : 0000 0000 0000 1010 0000 1010 0000 0000. Le numéro de page est donné par les 21 bits premier bits de poids fort : 0000 0000 0000 1010 0000 1. Soit $1 + 2^6 + 2^8 = 65 + 256 = 321$.**

d) **FIFO**

	0	1	2	3	4	5	6	1	0	5	8	4	3	2	1
c1	0					0	6	6	6	6	6	6	6	6	6
c2		1				1	1	1	0	0	0	0	0	0	0
c3			2			2	2	2	2	2	8	8	8	8	8
c4				3		3	3	3	3	3	3	3	3	2	2
c5					4	4	4	4	4	4	4	4	4	4	1
c6						5	5	5	5	5	5	5	5	5	5

11 défauts de pages

LRU

	0	1	2	3	4	5	6	1	0	5	8	4	3	2	1
c1	**0**					0	**6**	6	6	6	6	6	**3**	3	3
c2		**1**				1	1	1	1	1	1	1	1	**2**	2
c3			**2**			2	2	2	**0**	0	0	0	0	0	**1**
c4				**3**		3	3	3	3	3	**8**	8	8	8	8
c5					**4**	4	4	4	4	4	4	4	4	4	4
c6						**5**	5	5	5	5	5	5	5	5	5

12 défauts de pages

Question 4

a)

Les tâches sont synchrones. L'intervalle d'étude est donc [0,24] (de bas en haut T3, T2 et T1).

E	R					E						R	E	R				E					R
		E	E					E	E							E	E						
				R	R		R			R	R				R				R	R	R	R	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Non ordonnançable car T1 va manquer son échéance à la date 12. Oui, il y a inversion de priorité entre T1 et T3 de durée 12-7 =5 .

E	R					E				R		E	R					E					R
		E	E								E			E		E	E						
				R	R		R	R	R						R				R	R	R	R	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Ordonnançable, la durée de l'inversion de priorité est 10-7 = 3.

b)

E	R					E				R			E	R				E					R
		E	E								E	E				E	E						
				R	R		R	R	R						R				R	R	R	R	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Ordonnançable car toutes les tâches vont respecter leurs échéances.