Question 1 (9 pts) : Généralités

a) **[1 pt]** Donnez le nombre de processus créés par ce bout de code (justifiez votre réponse) :

```
int main () {  while (fork()==0)  execl ("/bin/ls", "ls", NULL);   return 0;}
```

b) **[1 pt]** Supposez un ensemble de processus concurrents qui partagent, en exclusion mutuelle, des ressources. Pour prévenir les interblocages, on propose d'utiliser une ressource supplémentaire qui doit être demandée, par chaque processus, avant toute autre ressource partagée et libérée en dernier. Cette solution permet-elle de prévenir les interblocages ? Justifiez votre réponse.

c) **[1 pt]** Lors du changement de contexte impliquant deux threads, faut-il invalider le contenu de la mémoire associative (TLB) du MMU, s'il s'agit de deux threads d'un même processus ? Justifiez votre réponse.

d) **[2 pts]** Dans un système d'exploitation qui gère les processus, on dispose d'une librairie qui permet de créer et de synchroniser des threads au niveau utilisateur. Le temps CPU alloué par le système à un processus est réparti (par un ordonnanceur au niveau utilisateur) entre les threads du processus. Les ordonnanceurs noyau et utilisateur sont circulaires. Supposez que vous créez un processus P qui, à son tour, crée trois threads utilisateur (th1, th2 et th3). Les états suivants sont-ils possibles ? Justifiez votre réponse.

État 1 : P (bloqué), th1 (prêt), th2 (élu), th3 (prêt).
État 2 : P (élu), th1 (prêt), th2 (élu), th3 (bloqué par un appel système).
État 3 : P (élu), th1 (prêt), th2 (élu), th3 (bloqué par un appel non système).
État 4 : P (prêt), th1 (élu), th2 (élu), th3 (bloqué par un appel non système).

e) **[2 pts]** Expliquez comment créer, sous Windows en utilisant l'API Win32, deux processus fils ayant comme sortie standard un même fichier ordinaire. Les résultats du second fils doivent apparaitre après ceux du premier fils (ne donnez pas de code, indiquez simplement les étapes à suivre).

f) **[2 pts]** Dans un système à pagination pure, 4 processus P1, P2, P3 et P4 partagent, selon le principe « Copy-On-Write », une page de données chargée dans le cadre 100 de la mémoire physique. Ces processus doivent-ils accéder à ce cadre via la même adresse virtuelle ? Expliquez, selon vous, comment appliquer le principe « Copy-On-Write », lorsque P2 tente d'accéder au cadre 100 en écriture. Justifiez votre réponse.

Question 2 (3 pts) : Moniteurs et variables de condition

Considérez un système de réservation de billets d'un spectacle. Ce système de réservation est composé de deux fonctions : *bool book()* et *void cancel()*. La fonction *book* permet de réserver une place pour le spectacle. La fonction *cancel* permet d'annuler une réservation pour le spectacle. On veut implémenter ces méthodes dans un moniteur « Spectacle_t » en utilisant si nécessaire des variables de condition. On vous donne le pseudo-code du moniteur Spectacle_t à compléter/modifier, si besoin est.

```
Moniteur Spectacle_t {
        int free=N ;
        bool book ( )
        {   if (free ==0) return false;
            free = free – 1;
           return true;
        }
      void cancel ( )
     {    if (free<N)  free = free + 1 ; }
}
```

La gestion de la réservation de billets pour le spectacle est prise en charge par un processus. Toutes les demandes des utilisateurs concernant ce spectacle sont dirigées vers ce processus. Ce processus crée un thread pour chaque demande reçue. Ce thread se charge de traiter la demande en faisant appel aux fonctions *book* ou/et *cancel du moniteur Spectacle_t*.

a) **[2 pts]** Complétez/modifiez le moniteur *Spectacle_t* afin de mettre en attente toute demande de réservation qui ne peut être satisfaite, jusqu'à ce qu'une place se libère (expliquez le rôle de chaque variable utilisée).

b) **[1 pt]** On veut limiter à 10 le nombre de threads bloqués à l'intérieur du moniteur par un appel à la fonction « *book »*. Si ce nombre est atteint, la fonction « book » ne bloque pas et sort du moniteur en retournant « false ». Complétez/modifiez le pseudo-code donné en a) pour prendre en compte cette nouvelle directive.

Attention : Comme mécanisme de synchronisation, vous devez vous limiter aux variables de condition.

Question 3 (4 pts) : Gestion de la mémoire

Considérez un système monoprocesseur avec une gestion de mémoire par pagination pure (pagination à la demande) et des tables de pages à un niveau. La mémoire physique est composée de 4 cadres. La taille de chaque cadre est de 4 KiO. L'adresse virtuelle est codée sur 16 bits. Supposez que 2 processus P1 et P2, composés respectivement de 7 et 5 pages, arrivent dans le système, l'un à la suite de l'autre. Le système charge dans l'ordre, les pages 0 et 1 de P1 dans les cadres 1 et 2, et la page 1 de P2 dans le cadre 3, avant de commencer l'exécution des processus P1 et P2 (pré-pagination).

a) **[2 pts]** Donnez l'adresse physique de l'adresse virtuelle : 0001 0011 0111 1000, pour chacun des cas suivants :
   - P1 référence cette adresse virtuelle et
   - P2 référence cette adresse virtuelle.

b) **[2 pts]** Supposez maintenant que lorsqu'un défaut de page se produit et qu'un retrait de page est nécessaire, le système effectue un remplacement global en utilisant LRU. Représentez l'évolution de l'état de la mémoire physique, à partir de l'état courant, dans le cas où le processeur reçoit, dans l'ordre suivant, les accès aux pages des processus P1 et P2 :

   (0, P1) (1, P1) (1, P2) (5, P1) (4, P2) (5, P1) (6, P1) (1, P1) (2, P1)

   où (0, P1), par exemple, référence la page 0 du processus P1.

   Donnez aussi le nombre de défauts de pages provoqués par chaque processus.

Question 4 (4 pts) : Ordonnancement

a) **[2 pts]** Considérez les 5 processus suivants :

Processus	Date d'arrivée	Priorité	Temps d'exécution
P1	0	4	5 (2) 3
P2	1	4	6
P3	2	3	2 (3) 2
P4	4	1	2
P5	5	3	2

Où x (y) z signifie x unités de temps CPU, y unités de temps d'E/S et z unités de temps CPU.

Supposez que :

- le système est monoprocesseur
- le système dispose d'un seul périphérique d'E/S partagé entre les processus selon le principe « premier arrivé, premier servi »,
- le temps de commutation est égal à 0, et
- la priorité 1 est la plus basse.

Donnez le diagramme de Gantt montrant l'ordonnancement des processus dans le cas d'un ordonnancement préemptif à files multiples et priorités fixes. L'ordonnancement des processus de même priorité est circulaire avec un quantum de 3.

b) **[2 pts]** Considérez les processus suivants qui partagent en exclusion mutuelle la ressource R :

Processus	Date d'arrivée	Temps d'exécution	Échéance = Période
P1	0	3 : ERE	6
P2	0	1 : E	8
P3	0	5 : ERRRE	12

Donnez le diagramme de Gantt montrant l'ordonnancement des processus dans le cas d'un ordonnancement RMA combiné avec PIP (protocole d'héritage de priorités pour traiter les inversions de priorité). Sont-ils ordonnançables ? L'intervalle d'étude à considérer est [0,13].

Question 1 : Généralités

a) **Un seul processus fils est créé car la condition de la boucle « while » est évaluée à vraie pour le fils uniquement. Le processus fils exécute le code « ls » qui ne crée aucun processus. Le processus père se termine juste après la création de son fils.**

b) **Oui car un seul processus va parvenir à acquérir la ressource supplémentaire. Les autres vont attendre cette ressource. Il n'y a aucun risque d'interblocage car ceux qui sont en attente de la ressource supplémentaire ne détiennent aucune autre ressource (pas de détention et attente).**

c) **Non, car ils partagent tous le même espace d'adressage.**

d) **État 1 : Oui, si th2 fait un appel système qui bloque. L'ordonnanceur noyau va bloquer le processus (pour l'ordonnanceur au niveau utilisateur th2 est toujours élu).**
État 2 : non, car si th3 est bloqué par un appel système tout le processus est bloqué.
État 3 : oui, car si th3 est bloqué par un appel non système, le processus P ne bloque pas. L'ordonnanceur au niveau utilisateur a élu th2 lorsqu'il est appelé suite au passage à l'état bloqué de th3..
État 4 : non, car deux threads utilisateur ne peuvent pas être élus en même temps.

e) **Il suffit de : 1) créer et ouvrir le fichier en appelant la fonction CreateFile, 2) affecter le handle retourné par cette fonction au champ de la structure StartupInformation qui définit la sortie standard, 3) créer l'un à la suite de l'autre les deux processus en appelant la fonction CreateProcess avec comme cinquième paramètre « true », avant dernier argument la structure StartupInformation, et dernier paramètre une structure où récupérer notamment le handle du processus créé, 4) ajouter entre la création des deux fils une attente de la fin du premier fils (WaitForSingleObject). 5) fermer le handle du fichier créé et 6) attendre la fin du second fils (WaitForSingleObject).**

f) **Non pas forcément, chaque processus accède au cadre via son espace d'adressage privé. L'adresse virtuelle du cadre 100 pour P1 peut être différente de celles des autres. Pour appliquer le principe « Copy-On-Write », lorsque P2 veut accéder en écriture au cadre 100, le cadre est dupliqué en mémoire physique pour l'allouer à P2 en remplacement du cadre 100. La table de pages de P2 est mise à jour pour remplacer 100 par le numéro de cadre contenant la copie créée pour P2.**

Question 2 : Moniteurs et variables de condition

a) Moniteur Spectacle_t {

```
int free=N ;
boolc bw ; // pour attendre qu'une place se libère
int nbw=0 ; // nombre de threads en attente
bool book ( )
{   if (free ==0) { nbw++ ;wait(bw); }
    free = free – 1;
    return true;
 }
void cancel ( )
{    if (free<N)  {
          if (nbw>0) { nbw-- ; signal(bw) ; }
          else free = free + 1;
          }
  }
}.
```

b) Moniteur Spectacle_t {

```
int free=N ; boolc bw ;  int nbw=0  ;
bool book ( )
{   if(nbw >= 10) { return false ; }
   if (free ==0) { nbw++ ; wait(bw); }
    free = free – 1;
    return true;
 }
void cancel ( )
{    if (free<N)  {
          if (nbw>0) { nbw-- ; signal(bw) ; }
          else free = free + 1;
          }
  }
}
```

Question 3 : Gestion de la mémoire

a) La taille d'une page est 4kiO. Le déplacement est donc codé sur 12 bits.

L'adresse virtuelle est sur 16 bits : 4 bits pour le numéro de page et 12 bits pour le déplacement dans la page.

P1 : 00 01 00 11 01 11 10 00 => il s'agit de la page 1 de P1. Cette page est chargée dans le cadre 2 => L'adresse physique est : 00 10 00 11 01 11 10 00

P2 : 00 01 00 11 01 11 10 00 => il s'agit de la page 1 de P2. Cette page est chargée dans le cadre 3 => L'adresse physique est : 00 11 00 11 01 11 10 00

b) LRU : 5 défauts de pages au total : 4 par P1 et 1 par P2

État courant	(0, P1)	(1, P1)	(1, P2)	(5, P1)	(4, P2)	(5, P1)	(6, P1)	(1,P1)	(2, P1)
				(5,P1)	**(5,P1)**	**(5,P1)**	**(5,P1)**	**(5,P1)**	**(5,P1)**
(0,P1)	**(0,P1)**	**(0,P1)**	**(0,P1)**	**(0,P1)**	**(4,P2)**	**(4,P2)**	**(4,P2)**	**(4,P2)**	**(2,P1)**
(1,P1)	**(1,P1)**	**(1,P1)**	**(1,P1)**	**(1,P1)**	**(1,P1)**	**(1,P1)**	**(6,P1)**	**(6,P1)**	**(6,P1)**
(1,P2)	**(1,P2)**	**(1,P2)**	**(1,P2)**	**(1,P2)**	**(1,P2)**	**(1,P2)**	**(1,P2)**	**(1,P1)**	**(1,P1)**

Question 4 : Ordonnancement

a)

P1	P1	P1	P2	P2	P2	P1	P1	P2	P2	P2	P1	P1	P1	P3	P3	P5	P5	P4	P3	P3	P4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

b)

P1	E	R	E				E			R	E		E	R
P2				E								E		
P3					E	R		R	R					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	P1 P2 P3						P1		P2				P1 P3	

Non ordonnançables car on a une échéance manquée pour P3 à la date 12 (traitement non terminé à la fin de sa première période).