Question 1 (5 pts) : Généralités

1.1 [3 pts] Donnez l'arborescence des processus créés par le code suivant (supposez que les appels système ne retournent pas d'erreur). **Donnez également** tous les affichages possibles.

```
int main() {
  if (fork() == 0) {
      if (fork() == 0) { write(1,"3",1);}
      else {  wait(NULL); write(1,"4",1); }
      exit(0);
  } else {
      if (fork() == 0) { write(1,"1",1); exit(0); }
      }
  while(wait(NULL)>0);
  write(1,"2",1);
  return 0;
}
```

1.2 [2 pts] Une application temps réel est composée de *3* tâches périodiques indépendantes : *T1(c1,p1), T2(c2,p2) et T3(c3,p3)*, où *ci* et *pi* sont respectivement le pire temps d'exécution et la période de la tâche *Ti*, pour *i* de *1* à *3*. Supposez que *c2 = 2c1, c3 = 4c1, p2=2p1* et *p3=2p1*.

Donnez en fonction de *c1*, les plus petites valeurs de *p1, p2* et *p3* pour que les tâches soient ordonnançables ?

Question 2 (6 pts) : Moniteurs et interblocage

On veut utiliser un moniteur et des variables de condition pour limiter à *MaxCalls* le nombre d'appels en cours d'une même fonction *F* supposée non récursive. Cette fonction est appelée à répétition par les threads d'un même processus comme indiqué ci-dessous. *LimiteCalls* est un moniteur composé de deux fonctions *getA* et *endA*. La fonction *getA* est appelée avant chaque appel à la fonction *F*. Elle met en attente l'appelant, si le nombre d'appels en cours de *F* a atteint *MaxCalls*. La fonction *endA* est appelée au retour de la fonction *F*. Elle permet d'activer un thread en attente d'une autorisation d'appel à *F*.

LimiteCalls LF(2) ;		
T0 { *while(true) {* *LF.getA();* *F(0);* *LF.endA();* *}* *}*	*T1 {* *while(true) {* *LF.getA() ;* *F(1);* *LF.endA();* *}* *}*	*T2 {* *while(true){* *LF.getA();* *F(2);* *LF.endA();* *}* *}*

```
Moniteur LimiteCalls (const int N) ;
{       int MaxCalls = N ;
        …
        void getA()  { … }
        void endA ( ) {  …  }
}
```

2.1 [3 pts] Complétez le moniteur *LimiteCalls* précédent qui permet de limiter à *N* le nombre d'appels en cours à une fonction.

2.2 [3 pts] Supposez maintenant que le nombre de threads créés par le processus est *M, N>2* et que la fonction *F* est récursive avec une profondeur maximale de récursivité égale à *2* (c-à-d le premier appel à *F* peut faire appel à *F* qui, à son tour, peut faire appel à *F*). Chaque appel à *F* est encadré par les appels aux fonctions *getA* et *endA*.

Est-ce que les threads peuvent se retrouver interbloqués ?

Si vous répondez oui, donnez :

- un scénario complet qui mène vers un interblocage ainsi que
- la valeur minimale de *N*, en fonction de *M,* pour que les threads ne se retrouvent jamais interbloqués.

Si vous répondez non, prouvez qu'il n'est pas possible d'atteindre une situation d'interblocage.

Question 3 (6 pts) : Gestion de la mémoire

Considérez un système de pagination pure à *2* niveaux dans lequel les adresses virtuelles sont codées sur *32* bits et la taille d'une page est *4KiO*. L'entrée de chaque table des pages, peu importe le niveau, est de *8* octets.

3.1. [1 pt] Quelle est la taille maximale en nombre de pages de l'espace d'adressage d'un processus ?

3.2. [1 pt] Donnez le nombre de pages qu'il y a dans un espace d'adressage virtuel de *4MiO*.

3.3. **[1 pt]** Donnez le numéro de page qui correspond à l'adresse virtuelle *0x00110A10.*

3.4. [3 pts] Supposez que les tables des pages d'un processus sont en mémoire physique durant l'exécution du processus.

Donnez pour chacune des configurations suivantes, la taille totale (en nombre de pages) des tables des pages à maintenir en mémoire durant l'exécution d'un processus de *4MiO* d'espace d'adressage virtuel contigu :

a) La table des pages du premier niveau a 1024 entrées.
b) La table des pages du premier niveau a 2048 entrées.
c) La table des pages du premier niveau a 512 entrées.

C'est pour quelle configuration a, b ou c, obtenez-vous la taille minimale ?

Question 4 (3 pts) : Ordonnancement de processus

Considérez un système monoprocesseur et les *3* processus *P1, P2* et *P3* suivants :

Processus	*Priorité de base*	*Temps d'exécution*
P1	*13*	*10 unités de CPU*
P2	*15*	*6 unités de CPU, 6 unités d'E/S, 2 unités de CPU*
P3	*6*	*5 unités de CPU*

Supposez ce qui suit :

- L'ordonnancement est préemptif et à files multiples avec *20* niveaux de priorité de *0* à *19*. La priorité *0* est la priorité la plus basse.
- Les processus de même priorité (d'une même file d'attente) ont un ordonnancement circulaire avec un quantum de *4* unités,
- La priorité d'un processus augmente de *4* lorsqu'il est mis en attente d'E/S et diminue de *4* lorsqu'il consomme son quantum, tout en maintenant sa priorité dans l'intervalle *[0,19]*.
- Le temps de commutation de contexte est égal à *0*.
- Tous les processus arrivent dans le système à l'instant *0*.
- L'ordonnanceur met à jour la priorité d'un processus à la fin de son quantum ou lorsqu'il se bloque.

Donnez le diagramme de Gantt montrant l'ordonnancement des processus. Pour chaque processus, indiquez l'évolution de sa priorité.

Le corrigé

Question 1 (5 pts) : Généralités

1.1 [3 pts] PP crée deux fils F1 et F2. F1 crée F11.

1 3 4 2 3 1 4 2 3 4 1 2

1.2 [2 pts] Pour les tâches périodiques indépendantes, une condition nécessaire et suffisante d'ordonnançabilité est celle de EDF (moins stricte que celles de RMA et DMA) :

c1/p1 + 2c1/2p1 + 4c1/2p1 <= 1

➔ 4 c1/p1 <= 1 ➔ p1 >= 4c1 => p1=4c1 et p2=p3=8c1.

Question 2 (6 pts) : Moniteurs et interblocage

2.1. [3 pts] *Moniteur LimiteCalls (const int N) ;*

```
{       int MaxCalls = N ;
        boolc wf ;
        int nb=0 ; // nombre d'appels en cours
        void getA()  { while(nb==MaxCalls) wait(wf) ; nb++ ; }
        void endA ( ){  if(nb>0) nb--; signal(wf); }
}
```

2.2. [3 pts]

Oui, pour M=2, N=3 et le scénario suivant : T1 appelle à F qui appelle à son tour F; T2 appelle à F ; T1 bloque suite à un autre appel récursif à F ; T2 bloque suite à un appel récursif à F ;

La valeur minimale de N est M * 2 + 1 car dans le pire cas chaque thread a besoin de 3 appels en cascade à F. Si chaque thread a déjà effectué 2 appels en cascade à F, il suffit de permettre un autre appel à F, afin que l'un des threads puisse terminer ses appels en cascade et permettre ensuite à d'autres de terminer les leurs.

Question 3 (6 pts) : Gestion de la mémoire

3.1. [1 pt] 1 page = 4KiO = 2^{12} octets ➔ Sur les 32 bits, 12 bits sont réservés au déplacement dans la page. Il reste donc 20 bits pour le numéro de page ➔ 2^{20} pages = 1Mi pages.

3.2. [1 pt] 4MiO / 4KiO pages = 1Ki pages = 1024 pages.

3.3. [1 pt] Le numéro de page de *0x00110A10 est donnée par 0x00110 c-à-d $2^8 + 2^4 = 256 + 16 = 272$. La page 272. Les 12 bits restants (A10) donnent le déplacement dans la page.*

3.4. [3 pts] 4MiO = 1024 pages.

a) **(10bits 1er niveau + 10 bits 2nd niveau) => 1024 entrées (table du 1er niveau) + 1024 entrées (1 table du 2ième niveau) = 2048 * 8 octets = 2 pages + 2 pages = 4 pages.**
   Une seule entrée de la table du 1er niveau est utilisée et elle pointe vers une table de 1024 entrées (du 2nd niveau).

b) **(11bits 1er niveau + 9 bits 2nd niveau) => 2048 entrées (table du 1er niveau) + 2 * 512 entrées (2 tables du 2ième niveau) = (2048 + 1024) * 8 octets = 4 pages + 2 pages = 6 pages.**
   Deux entrées de la table du 1er niveau sont utilisées et chacune pointe vers une table de 512 entrées (du 2nd niveau).

c) **(9bits 1er niveau + 11 bits 2nd niveau) => 512 entrées (table du 1er niveau) + 2048 entrées (1 table du 2ième niveau) = (512+2048) * 8 octets = 1 page + 4 pages = 5 pages.**
   Une seule entrée de la table du 1er niveau est utilisée et elle pointe vers une table de 2048 entrées (du 2nd niveau).

C'est la configuration a.

Question 4 (3 pts) : Ordonnancement de processus

P2	P1	P2	P1	P3	P2	P3	P1
0 4	4 8	8 10	10 14	14 16	16 18	18 21	21 23

À t=0 : Prio1=13 Prio2=15 Prio3=6

À t=4 : Prio1=13 Prio2=15-4=11 Prio3=6 (fin du quantum pour P2)

À t=8 : Prio1=13-4 =9 Prio2=11 Prio3=6 (fin du quantum pour P1)

À t=10 : Prio1=9 Prio2=11+4=15 (en E/S jusqu'à 16) Prio3=6 (P2 bloque en attente d'E/S).

À t=14 : Prio1=9-4=5 Prio2=15 (en E/S) Prio3=6 (fin du quantum pour P1)

À t=16 : Fin E/S de P2 -> Prio1=5 Prio2=15 Prio3=6

À t=18 : Prio1=5 Prio3=6 (fin de P2)

À t=21 : Prio1=5 (fin de P3).