Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
16 décembre 2021,
durée 1h30, calculatrices et documents autorisés

1 Améliorer un système (6 points, 25 mn)

Dans les systèmes Unix/Linux la création de processus et l'attente de la fin de ses fils passent par les deux fonctions `fork` et `wait` :

```
int main (void) {
    if (fork() == 0) {              // créer un processus fils
        printf("Fils\n");           // je suis le fils
        return EXIT_FAILURE;        // la fin du fils
    }
    printf("Père\n");               // je suis le père
    wait(NULL);                     // attendre la mort de son fils
    printf("Fin du père\n");        // le père a le dernier mot
    return EXIT_SUCCESS;
}
```

Nous souhaitons faire la même chose dans **notre mini-système développé en TP**.

A) Commençons par ajouter un appel système permettant à un processus d'attendre la mort d'un de ses fils (comme le `wait` de l'exemple). Donnez les deux étapes principales de la réalisation de cette attente ainsi que les données supplémentaires dont vous avez besoin ? [4 points].

```
Données:
- ajouter un état WAIT
- ajouter, pour chaque processus, le numéro de son père
WAIT:
- si ce processus a un fils, le mettre dans l'état WAIT
- choisir un autre processus
```

B) Comment modifier l'appel système `EXIT` afin de reveiller l'éventuel père en attente ? Donnez les trois étapes principales [2 points].

```
EXIT:
- chercher son père
- si il est WAIT, le passer à READY
- si le processus moribond a des fils, supprimer la liaison
```

2 Gestion de fichiers (6 points, 25 mn)

Dans un système très simplifié, la FAT d'un disque unique pourrait être représentée par les structures de données ci-dessous :

```
enum { FAT_EOF = -1, FAT_FREE = -2, FAT_RESERVED = -3 };

struct {
    int first;          // adresse du premier bloc (FAT_EOF si vide)
    int size;           // taille en octets (-1 si inutilisé)
} inodes[ NB_FILES ];   // les descripteurs

int fat[ NB_BLOCKS ];   // la FAT du disque unique
```

A) Écrivez la fonction `int count_free_blocks(void)` qui va calculer et renvoyer le nombre de blocs libres. [2 points]

```
int count_free_blocks() {
    int nb = 0;
    for(int i=0; i<NB_BLOCKS; i++) {
        if (fat[i] == FAT_FREE) nb++;
    }
    return nb;
}
```

B) Écrivez la fonction `int count_data_blocks(void)` qui va calculer et renvoyer le nombre de blocs **effectivement** occupés par tous les fichiers. [2 point]

```
int count_data_blocks() {
    int nb = 0;
    for(int i=0; i<NB_FILES; i++) {
        if (inodes[i].size < 0) continue;
        int first = inodes[i].first;
        while (first >= 0) {
            nb++;
            first = fat[first];
        }
    }
    return nb;
}
```

C) Finalement, écrivez une fonction qui calcule le nombre de blocs perdus (marqués comme utilisés mais qui n'apparaissent dans aucun fichier). [2 point]

```
int count_lost_blocks() {
    int nb = 0;
    for(int i=0; i<NB_BLOCKS; i++) {
        if (fat[i] >= 0) nb++;
        else if (fat[i] == FAT_EOF) nb++;
    }
    return nb - count_data_blocks();
}
```

3 Gestion des processus et synchronisation (8 points, 40 mn)

Considérons un système équipé d'un **seul disque** et d'un seul **processeur** géré en temps partagé par l'algorithme du **tourniquet**.

A) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur. Indiquez clairement le déroulement des opérations sur le processeur et le disque. [2 points]

```
faire quatre fois
  |  calculer pendant une seconde
  |  écrire sur un fichier pendant 3 secondes
fin-faire
calculer pendant une seconde
```

TR=17, TX=5/17

```
---- :   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
CPU1 : |CC|--|--|--|CC|--|--|--|CC|--|--|--|CC|--|--|--|CC|
DISK : |--|DD|DD|DD|--|DD|DD|DD|--|DD|DD|DD|--|DD|DD|DD|--|
```

B) Même question avec les mêmes détails si nous utilisons une écriture asynchrone gérée par un thread séparé. [2 points]

```
faire quatre fois
  |  calculer pendant une seconde
  |  créer un thread pour faire
  |    |  écrire sur un fichier pendant 3 secondes
  |  fin du thread
fin-faire
attendre la fin des threads
calculer pendant une seconde
```

TR=14, TX=5/14

```
---- :   1  2  3  4  5  6  7  8  9 10 11 12 13 14
CPU1 : |CC|CC|CC|CC|--|--|--|--|--|--|--|--|--|CC|
DISK : |--|DD|DD|DD|DD|DD|DD|DD|DD|DD|DD|DD|DD|--|
```

C) Reprenez le code de la question **B** et remplacez l'instruction d'attente des threads par une version basée sur un sémaphore (à créer, initialiser et utiliser). [2 points]

```
finThreads := semaphore( 0 );
faire quatre fois
  |  calculer pendant une seconde
  |  créer un thread pour faire
  |    |  écrire sur un fichier pendant 3s
  |    |  V(finThreads)
  |  fin du thread
fin-faire
faire 4 fois P(finThreads); fin-faire
calculer pendant une seconde
```

D) Normalement, nous ne devrions pouvoir faire qu'une seule écriture à la fois. Comment le garantir avec un sémaphore ? [1 point] Est-ce le travail du programmeur d'applications que de faire cette vérification ? [1 point]

```
mutex := semaphore( 1 );
...
  |    |  P(mutex)
  |    |  écrire sur un fichier pendant 3s
  |    |  V(mutex)
 ...

Non, ce n'est pas le travail du programmeur car d'autres
processus peuvent demander des E/S vers le disque. La synchro
est donc à la charge du système.
```

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
8 janvier 2021, première session
durée 1 heure, calculatrices et documents autorisés

1 Gestion de fichiers (6 points, 15 mn)

On vous donne une instance d'une FAT (-1 signale le dernier bloc d'un fichier, -2 un bloc réservé et -3 un bloc libre) :

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
FAT[i]	-2	-2	-2	-1	10	-3	-1	-3	13	11	-1	7	4	-1	8	5

A) Dans cette configuration combien avons-nous de fichiers et quels sont les blocs physiques de chaque fichier ? [3 points].

```
Fichier 1: 9, 13, 4
Fichier 2: 12, 7
Fichier 3: 16, 5, 10, 11
Fichier 4: 14
```

B) Le codage de cette FAT est-il erroné (indiquez clairement les erreurs et comment pouvons-nous les trouver) ? [1 point].

Le bloc 15 pointe vers le 8 qui est libre. C'est une erreur.

C) Que réalisent les opérations ci-dessous ? [1 point].

```
fat[6] = -1;
fat[14] = 6;
```

Ajout du bloc 6 à un fichier qui se terminait avant en 14.

D) Que contiennent les blocs réservés au début du disque ? [1 point]

Des descripteurs et la FAT.

2 Gestion des disques (4 points, 10 mn)

Considérons une machine dotée de 6 disques d'un téra-octet et des circuits d'E/S permettant à ces disques de réaliser des opérations en parallèle.

A) Quelle est la capacité de stockage pour une configuration RAID5 construite sur les six disques (que nous noterons par la suite RAID5(D1,D2,D3,D4,D5,D6)) ? [0,5 point]

capacité $= (5 \times \text{To})$

B) Même question pour un *striping* (RAID0) des disques : RAID0(D1,D2,D3,D4,D5,D6) ? [0,5 point]

capacité $= (6 \times \text{To})$

C) Quelle est la capacité de stockage et le nombre maximum de disques défaillants (sans altérer le bon fonctionnement du système) pour les trois configurations suivantes. Indiquez par un exemple les disques susceptibles d'être défaillants. [3 points]

C1 RAID0(RAID5(D1,D2,D3), RAID5(D4,D5,D6))

C2 RAID5(RAID0(D1,D2), RAID0(D3,D4), RAID0(D5,D6))

C3 RAID5(RAID1(D1,D2), RAID1(D3,D4), RAID1(D5,D6))

```
RAID1 = mirroring, capacité de RAID1(A,B) = 1 To

     | Capacité  | Fautes       | Exemple
-------------------------------------------------
C1 : | 4 To,     | 2 disques,   | D1 et D4
C2 : | 4 To,     | 2 disques,   | D5 et D6
C3 : | 2 To,     | 4 disques,   | D1, D3, D5 et D6
```

3 Gestion des processus et synchronisation (10 points, 35 mn)

Considérons un système équipé d'un **seul disque** et dans lequel les **deux processeurs** sont gérés en temps partagé par l'algorithme du **tourniquet**.

A) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation des processeurs sur la période considérée. Indiquez clairement le déroulement des opérations sur les processeurs et le disque. [2 points]

```
faire quatre fois
  |  calculer pendant une seconde
  |  écrire sur un fichier pendant 2 secondes
fin-faire
calculer pendant une seconde
```

TR=13, TX=5/26

```
---- :   1  2  3  4  5  6  7  8  9 10 11 12 13
CPU1 : |CC|--|--|CC|--|--|CC|--|--|CC|--|--|CC|
CPU2 : |--|--|--|--|--|--|--|--|--|--|--|--|--|
DISK : |--|DD|DD|--|DD|DD|--|DD|DD|--|DD|DD|--|
```

B) Même question avec les mêmes détails si nous utilisons des threads (version ci-dessous). [3 points]

```
faire quatre fois
  |  créer un thread pour faire
  |    |  calculer pendant une seconde
  |    |  écrire sur un fichier pendant 2 secondes
  |  fin du thread
fin-faire
attendre la fin des threads fils
calculer pendant une seconde
```

TR=11, TX=5/22

```
---- :   1  2  3  4  5  6  7  8  9 10 11
CPU1 : |CC|CC|--|--|--|--|--|--|--|--|CC|
CPU2 : |CC|CC|--|--|--|--|--|--|--|--|--|
DISK : |--|--|DD|DD|DD|DD|DD|DD|DD|DD|--|
```

C) Même question avec les mêmes détails si nous synchronisons les threads (version ci-dessous). Expliquez notamment la valeur initiale du compteur du sémaphore [3 points]

```
s := nouveau_sémaphore( 2 );
faire quatre fois
  |  créer un thread pour faire
  |    |  P(s)
  |    |  calculer pendant une seconde
  |    |  V(s)
  |    |  écrire sur un fichier pendant 2 secondes
  |  fin du thread
fin-faire
attendre la fin des threads fils
calculer pendant une seconde
```

TR=10, TX=5/20

```
---- :   1  2  3  4  5  6  7  8  9 10
CPU1 : |CC|CC|--|--|--|--|--|--|--|CC|
CPU2 : |CC|CC|--|--|--|--|--|--|--|--|
DISK : |--|DD|DD|DD|DD|DD|DD|DD|--|
```

D) Comment reprendre le code de la question **B** et remplacer l'instruction d'attente par une version basée sur un sémaphore (à créer, initialiser et utiliser) [2 points].

```
attente := semaphore( 0 );
faire quatre fois
  |  créer un thread pour faire
  |    |  calculer pendant une seconde
  |    |  écrire sur un fichier pendant 2s
  |    |  V(attente)
  |  fin du thread
fin-faire
faire quatre fois P(attente); fin-faire
calculer pendant une seconde
```

Système d'exploitation

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
7 janvier 2020, première session
durée 2 heures, calculatrices et documents autorisés

Responsable : Jean-Luc Massat

1 Choix d'une page victime (6 points, 35 mn)

Considérons un système de mémoire paginée. Nous disposons de 4 pages physiques qui sont toutes occupéees, le tableau donnant ci-dessous, pour chacune d'elles, la date du chargement de la page, la date du dernier accès à cette page et l'état des indicateurs de la page physique (accédée et modifiée).

Page physique	Chargement	Accès	Accédée	Modifiée
0	126	279	0	0
1	230	260	1	0
2	120	272	1	1
3	160	280	1	1

A) En justifiant votre réponse, indiquez quelle sera la page remplacée (la victime) pour chacun des 3 algorithmes de remplacement suivants : LRU, FIFO et seconde chance (FINUFO). [3 points]

B) Quelle information vous manque-t-il pour pouvoir appliquer l'algorithme optimal ? [1 point]

C) L'algorithme NRU (*Not Recently Used*) consiste à choisir la page victime en utilisant les deux bits d'accès et de modification. Comment, d'après-vous, l'algorithme classe-t-il les quatre configurations possibles (justifiez votre choix) ? [2 points]

2 Système de gestion de fichiers (5 points, 30 mn)

Considérez un système de gestion de fichiers où le disque est géré à l'aide d'une FAT (File Allocation Table). La FAT est un tableau d'entier qui possède autant de lignes que des blocs sur le disque. Un bloc i est libre si `fat[i] = FAT_FREE`. Un bloc défectueux est marqué `FAT_BAD`. Si le bloc i est le dernier bloc d'un fichier, on a `fat[i] = FAT_EOF`. Toute autre valeur positive sur `fat[i]` indique le bloc suivant.

A) Écrivez la fonction d'allocation qui recherche un espace libre de `n` blocs consécutifs et renvoie l'adresse du premier bloc (ou -1 en cas d'échec). Cet espace libre peut contenir des blocs défectueux (sauf le premier). La FAT doit être parcourue une seule fois. [3 points]

```
int alloc_blocks(int fat[], int taille_fat, int n)
```

B) Écrivez la fonction de chaînage de la FAT lorsque les blocs sont alloués par la fonction d'allocation `alloc_blocks`. Le paramètre `debut` représente le premier bloc alloué et le paramètre `n` le nombre de blocs alloués. **Rappel** : le dernier bloc d'un fichier a la valeur `FAT_EOF`. [2 points]

```
void chainages(int fat[], int debut, int n)
```

Tournez la page SVP...

3 Gestion des processus et synchronisation (9 points, 55 mn)

Considérons un système équipé d'un **seul disque** et dans lequel le processeur **unique** est géré en temps partagé par l'algorithme du **tourniquet**.

A) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur sur la période considérée. [1 point]

```
int main (void) {
    DATA a, b;
    a = produireA();  /* 4 secondes de CPU  */
    ecrire(a);        /* E/S de 6 secondes  */
    b = produireB();  /* 2 secondes de CPU  */
    ecrire(b);        /* E/S de 4 secondes  */
    finir();          /* 2 secondes de CPU  */
    return 0;
}
```

B) Nous disposons maintenant d'une version asynchrone de la fonction d'écriture (voir ci-dessous). Proposez une version de `main` qui utilise cette fonction pour réduire le temps de réponse. Quel est le nouveau taux d'utilisation du processeur ? **Remarque** : utilisez le principe de l'attente active et limitez la consommation de CPU. [2 points]

```
void ecrire_asynchrone(DATA d, int* finie);
```

C) Le système d'exploitation nous offre deux fonctions pour gérer les threads (voir ci-dessous). Comment écrire la fonction `ecrire_asynchrone` en utilisant la fonction `ecrire` et la création de thread ? [2 points]

```
int  new_thread();   /* 0 chez le fils et > 0 chez le père */
void exit_thread();  /* fin du thread courant              */
```

D) Notre système dispose également de trois fonctions afin de manipuler des sémaphores (voir ci-dessous). Proposez une nouvelle version de l'écriture asynchrone basée sur un sémaphore : `SEMAPHORE ecrire_asynchrone_sem(DATA d)` . [2 points]

```
SEMAPHORE sem_create(int counter);
void sem_P(SEMAPHORE sem);
void sem_V(SEMAPHORE sem);
```

E) Proposez une nouvelle version de `main` basée sur `ecrire_asynchrone_sem`. [1 point]

F) Quel est le temps de réponse et le taux d'utilisation du processeur si les deux écritures (de A et de B) sont réalisées sur deux disques indépendants. [1 point]

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
11 janvier 2019, première session
durée 2 heures, calculatrices et documents autorisés

1 Gestion des processus et synchronisation *(8 points, 50 minutes)*

Considérons un système équipé d'un **seul disque** et dans lequel le processeur **unique** est géré en temps partagé par l'algorithme du **tourniquet**.

A) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur sur la période considérée. [1 point]

```
int main (void) {
    calculer();  /* 3 secondes          */
    ecrire();    /* E/S de 4 secondes   */
    finir();     /* calcul de 2 secondes */
    return 0;
}
```

B) Nous disposons maintenant d'une version asynchrone de la fonction d'écriture (voir ci-dessous). Proposez une version de `main` qui utilise cette fonction pour réduire le temps de réponse. Quel est le nouveau taux d'utilisation du processeur ? **Remarque** : utilisez le principe de l'attente active et limitez la consommation de CPU. [2 points]

```
void ecrire_asynchrone(int* finie);
```

C) Le système d'exploitation nous offre deux fonctions pour gérer les threads (voir ci-dessous). Comment écrire la fonction `ecrire_asynchrone` en utilisant la fonction `ecrire` et la création de thread ? [1,5 point]

```
int  new_thread();   /* 0 chez le fils et > 0 chez le père */
void exit_thread();  /* fin du thread courant              */
```

D) Notre système dispose également de trois fonctions afin de manipuler des sémaphores :

```
int  sem_create(int counter);
void sem_P(int sem_id);
void sem_V(int sem_id);
```

Proposez une nouvelle version de l'écriture asynchrone `int ecrire_asynchrone_sem(void)` basée sur un sémaphore (la fonction renvoie l'identifiant du sémaphore). [2 points]

E) Proposez une nouvelle version de `main` basée sur `ecrire_asynchrone_sem`. Quel est le temps de réponse et le taux d'utilisation du processeur de cette dernière version si l'écriture dure 6 secondes ? [1,5 point]

Tournez la page SVP...

2 Les régions mémoire de Linux *(7 points, 35 minutes)*

la carte mémoire des régions d'un processus Linux basée sur une mémoire paginée est détaillée ci-dessous (la taille est exprimée en pages) :

n°	adresse	taille	droits
0	0x10000000	256	r-x
1	0x12000000	512	rw-
2	0xEF800000	2048	rw-

Questions :

A) Donnez un sens à ces régions (plusieurs solutions sont possibles). [2 points]

B) La colonne **adresse** contient-elle des adresses logiques ou des adresses physiques ? [1 point]

C) Donnez un adresse qui ne correspond à aucun région. [1 point]

D) Quelle est la taille de la mémoire logique de ce processus ? [1 point]

E) Si nous décidons d'interdire l'écriture sur la quatrième page (de numéro 3) de la région 1, quel est le nouveau contenu de la table des régions ? **Remarque** : notez qu'une page mesure 2^{12} octets, soit 4096 octets en décimal et 0x1000 octets en hexadécimal. [2 points]

3 Gestion d'une mémoire virtuelle *(5 points, 35 minutes)*

Considérons une mémoire composée de trois pages physiques initialement vides. Lors de son exécution, un processus unique accède dans l'ordre aux pages virtuelles suivantes :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

Voila une trace de l'évolution de la mémoire sur les trois premiers accès :

	init	7	0	1	...
page physique 0	-	7	7	7	...
page physique 1	-	-	0	0	...
page physique 2	-	-	-	1	...
défaut de page		oui	oui	oui	...

Questions :

A) Devons-nous considérer que ce processus a effectué seulement 22 accès à la mémoire ? [1 point]

B) Pour chacun des algorithmes FIFO et LRU, donnez le contenu des pages physiques (c-à-d un numéro de page virtuelle) après chaque accès. Précisez également le nombre total de défauts de page provoqués par ces accès. [3 points]

C) Combien faut-il de pages physiques pour minimiser le nombre de défauts de page ? [1 point]

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
16 mai 2018, première session
durée 2 heures, calculatrices et documents autorisés

1 Questions... réponses (8 points)

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-programmé et doté d'une gestion mémoire performante (n'oubliez pas de justifier vos réponses). **Questions :** (un point par question sauf exceptions)

a) Sur une zone de 1024 ko gérée par l'algorithme du *buddy system*, quel est l'état des blocs libres et occupés après les allocations de 65 ko, 130 ko, 220 ko, 50 ko et 91 ko dans cet ordre ?

b) Si le programme $\ll P(s); P(t); \ldots; V(s); \ldots; P(t); \ldots; V(t); V(t); \ldots; P(s); V(s); \gg$ est exécuté en plusieurs exemplaires, existe-t-il un risque d'interblocage (les compteurs sont initialisés à deux) ?

c) Soient les cinq processus suivants qui démarrent en même temps. Combien faut-il de processeurs pour les exécuter sans contrainte (nous considérerons que les E/S se déroulent en parrallèle) ? Quel est le taux d'utilisation du processeur ? [2 points]

- calculer 3s, E/S sur 4s, calculer 3s
- calculer 2s, E/S sur 3s, calculer 1s, E/S de 3s, calculer 1s
- calculer 2s, E/S de 2s, calculer 2s, E/S de 2s, calculer 2s
- dormir sur 5s, calculer 2s, E/S de 3s
- E/S de 3s, calculer 3s, E/S de 4s

d) Donnez la matrice `max` de l'algorithme des banquiers pour ces trois processus :

- P_1 : allouer R_1 et R_2, ..., libérer R_1, ..., allouer R_3, R_2 et R_1, ..., libérer toutes les resources
- P_2 : allouer R_3 et R_1, ..., libérer R_3, ..., allouer R_2 et R_1, libérer toutes les ressources
- P_2 : allouer R_2 et R_3, ..., allouer R_2, ..., allouer R_3, ..., libérer toutes les ressources

e) Si le système reçoit les requêtes ci-dessous, donnez l'ordre de traitement en se basant sur l'algorithme SSTF (*Shortest Seek Time First*). La tête de lecture se trouve sur la piste 500.

200, 150, 320, 700, 210, 100, 600, 705, 490, 150, 310, 610, 150

f) Dans la FAT ci-dessous combien avons-nous de fichiers et quels sont les blocs physiques de chaque fichier ? [2 points]

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
FAT[i]	22	22	13	19	52	-1	20	17	52	15	6	3	-1	-1	4	52	18	-1	-1	8

2 Gestion des processus et synchronisation (8 points)

Considérons un système équipé d'un **seul disque** et dans lequel le processeur **unique** est géré en temps partagé par l'algorithme du **tourniquet**.

a) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur sur la période considérée. [2 points]

```
pour i ∈ {1, 2} faire
  | calculer pendant i seconde(s)
  | écrire sur un fichier pendant 4 secondes
fin-pour
calculer pendant 1 seconde
```

b) Même question que précédemment mais nous modifions le processus comme suit (indiquez clairement sur un schéma le déroulement de l'exécution). [2 points]

```
pour i ∈ {1, 2} faire
  |   créer un thread pour faire
  |     | calculer pendant i seconde(s)
  |     | écrire sur un fichier pendant 4 secondes
  |   fin du thread
fin-pour
attendre la fin des threads fils
calculer pendant 1 seconde
```

c) Si nous considérons que les deux E/S se déroulent sur deux disques différents, quel est l'impact sur le temps de réponse et le taux d'utilisation du processeur ? [2 points]

d) En gardant l'organisation en threads, comment modifier le code (avec l'utilisation d'un sémaphore) pour s'assurer que le calcul de 1 seconde s'exécute en premier et occupe toute la CPU ? [2 points]

3 Gestion des disques (4 points)

Considérons une machine dotée de 6 disques d'un téra-octet et des circuits d'E/S permettant à ces disques de réaliser des opérations en parallèle.

a) Quelle est la capacité de stockage pour une configuration RAID5 construite sur les six disques (que nous noterons par la suite RAID5(D1,D2,D3,D4,D5,D6)) ? [1 point]

b) Même question pour un *striping* (RAID0) des diques : RAID0(D1,D2,D3,D4,D5,D6) ? [1 point]

c) Quelle est la capacité de stockage et le nombre maximum de disques défaillants (sans altérer le bon fonctionnement du système) pour la configuration RAID0(RAID5(D1,D2,D3), RAID5(D4,D5,D6)) Indiquez par un exemple les disques susceptibles d'être défaillants. [2 points]

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
16 mai 2017, première session,
durée 2 heures, calculatrices et documents autorisés.

1 Exécution d'un processus — 9 points

Considérons le programme ci-dessous :

```
// Exécution : faire a fois ( 1 minute de CPU puis 2 minutes d'E/S )
int fa(int a) { ... }

// Exécution : faire b fois ( 1 minute de CPU puis 3 minutes d'E/S )
int fb(int b) { ... }

// Exécution : 1 minute de CPU
int fc(int s) { ... }

int main() { int a = fa(2); int b = fb(3); return fc(a+b); }
```

Questions (donnez à chaque fois la trace de l'exécution) :

A) Si nous avons un seul processeur, quel est le temps de réponse de ce programme et le taux d'utilisation de la CPU ? [1 point]

B) Même question si nous disposons de deux processeurs. [1 point]

C) Revenons au processeur unique et considérons que les E/S sont des écritures. Nous disposons d'un tampon de sortie de 4 minutes. L'opération de vidage de ce tampon est assurée de manière asynchrone par le système d'exploitation. La fin du processus implique l'attente du **vidage du tampon**. Dans ces conditions quel est le temps de réponse et le taux d'utilisation du processeur ? [2 points]

D) Nous avons maintenant **deux disques** (et donc deux tampons de 4 minutes). La fonction `fa()` utilise un disque et `fb()` l'autre. Calculez le temps de réponse et le taux d'utilisation du processeur. [2 points]

E) En plus des deux disques (et donc des deux tampons), nous disposons maintenant de **trois processeurs** et nous ajoutons à notre système des fonctions de gestion des threads :

```
int  th_create();    // 0: fils; numéro:père
void th_exit();      // fin du thread courant
void th_join(int n); // attente de la fin d'un thread
```

Modifiez la fonction `main()` pour exécuter les deux fonctions dans deux threads et calculez le temps de réponse et le taux d'utilisation du processeur. [3 points]

Tournez la page SVP...

2 Les philosophes et les banquiers — 5 points

Un beau jour de mai, 5 philosophes décident d'aller manger ensemble des spaghettis. Le restaurateur dresse une table ronde, mais place seulement 5 fourchettes (une entre chaque assiette). Pour manger, un philosophe a besoin de deux fourchettes : celle placée à sa droite et celle placée à sa gauche.

Questions :

A) Montrez qu'il existe un risque d'interblocage et proposez une solution simple à ce problème. [1 point]

B) Pour éviter les interblocages, les philosophes décident d'appliquer l'algorithme des banquiers. Donnez la matrice `Max` et le vecteur `Dispo`. [1 point]

C) Pour **ce problème bien particulier** donnez l'algorithme simplifié permettant de déterminer si oui ou non, le système est dans un état sain. [2 points]

D) Le restaurateur place maintenant les cinq fourchettes au **centre de la table** (chaque philosophe a toujours besoin de deux fourchettes). Donnez la nouvelle version de la matrice `Max` et du vecteur `Dispo`. [1 point]

3 Système de gestion de fichiers — 6 points

Considérez un système de gestion de fichiers où le disque est géré à l'aide d'une FAT (File Allocation Table). La FAT est un tableau d'entier qui possède autant de lignes que des blocs sur le disque. Un bloc i est libre si `fat[i] = FAT_FREE`. Si le bloc i est le dernier bloc d'un fichier, on a `fat[i] = FAT_EOF`. Toute autre valeur positive sur `fat[i]` indique le bloc suivant.

A) Écrivez la fonction d'allocation qui recherche un espace libre de `n` blocs consécutifs et renvoie l'adresse du premier bloc (ou -1 en cas d'échec) en se basant sur le principe de meilleur ajustement (best-fit). La FAT doit être parcourue une seule fois. [2 points]

```
int alloc_bloc_best_fit (int fat[], int taille_fat, int n)
```

B) Écrivez la fonction de chaînage de la FAT lorsque les blocs sont alloués par la fonction d'allocation `alloc_bloc_best_fit`. Le paramètre `debut` représente le premier bloc alloué et le paramètre `n` le nombre de blocs alloués. **Rappel** : le dernier bloc d'un fichier a la valeur `FAT_EOF`. [2 points]

```
void chainages(int fat[], int debut, int n)
```

C) Écrivez la fonction d'allocation `alloc_bloc_frac` qui alloue `n` blocs non-consécutifs et retourne l'adresse du premier (ou -1 s'il n'existe pas assez de blocs libres). **Rappel** : n'oubliez pas d'enchaîner correctement les blocs sur la `fat`. [2 points]

```
int alloc_bloc_frac (int fat[], int taille_fat, int n)
```

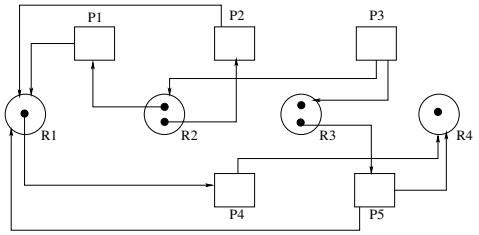
Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
11 mai 2016, première session,
durée 2 heures, calculatrices et documents autorisés.

1 Gestion de ressources — 5 points

Considérez le graphe d'allocation des ressources ci-dessous :



Dans ce cadre, répondez aux questions suivantes :

A) Le système est-il *bloqué* ? Justifiez votre réponse.

B) Proposez des structures de données et écrivez (sommairement) un algorithme permettant de détecter automatiquement si le système est dans un état de blocage.

2 Système de gestion de fichiers — 6 points

Considérez un système de gestion de fichiers où le disque est géré à l'aide d'une FAT (File Allocation Table). La FAT est un tableau d'entier qui possède autant de lignes que des blocs sur le disque. Un bloc i est libre si `fat[i] = FAT_FREE`. Si le bloc i est le dernier bloc d'un fichier, on a `fat[i] = FAT_EOF`. Toute autre valeur positive sur `fat[i]` indique le bloc suivant.

A) Écrivez la fonction d'allocation qui recherche un espace libre de `n` blocs consécutifs et renvoie l'adresse du premier bloc (ou -1 en cas d'échec) en se basant sur le principe de meilleur ajustement (best-fit).

```
int alloc_bloc_best_fit (int fat[], int taille_fat, int n)
```

B) Écrivez la fonction de chaînage de la FAT lorsque les blocs sont alloués par la fonction d'allocation `alloc_bloc_best_fit`. `debut` représente le premier bloc alloué et `n` le nombre de blocs alloués. **Rappel** : le dernier bloc d'un fichier a la valeur `FAT_EOF`.

```
void chainages(int fat[], int debut, int n)
```

C) Écrivez la fonction d'allocation `alloc_bloc_frac` qui alloue `n` blocs non-consécutifs et retourne l'adresse du premier (ou -1 s'il n'existe pas assez de blocs libres). **Rappel** : n'oubliez pas d'enchaîner correctement les blocs sur la `fat`.

```
int alloc_bloc_frac (int fat[], int taille_fat, int n)
```

3 Ordonnancement de processus — 9 points

Considérez un système d'ordonnancement qui utilise les structures de données ci-dessous.

```
#define EMPTY (0)
#define READY (1)
#define HIGH (0)
#define LOW  (1)

struct {
  PSW cpu;    // Process State Word
  int state;  // EMPTY or READY
  int time;   // Estimated execution time
} process[2][MAX_PROCESS]; // process[0] = high priority process
                           // process[1] = low priority process

int current_high_process = -1; int current_low_process = -1;
int current_priority = HIGH;
```

L'algorithme d'ordonnancement de ce système est basé sur le principe de **Files de priorité**. Les processus sont classés sur 2 niveaux de priorités distincts : `HIGH` et `LOW`. La file `HIGH` gère les processus de priorité **haute**. Cette file utilise l'algorithme *SJF - (Shortest Job First)* pour ordonnancement. La file `LOW` gère les processus de priorité **basse** et utilise l'algorithme *RR - (Round Robin ou Tourniqué)*. Les processus de priorité `HIGH` sont, bien sûr, prioritaire par rapport aux processus de priorité `LOW`.

A) Écrivez la fonction d'ordonnancement pour les processus de priorité *haute*. La fonction doit retourner l'indice du processus choisi ou -1 s'il y a aucun processus à ordonnancer.

```
int ordonnancement_HIGH ();
```

B) Écrivez la fonction d'ordonnancement pour les processus de priorité *basse*. La fonction doit retourner l'indice du processus choisi ou -1 s'il y a aucun processus à ordonnancer.

```
int ordonnancement_LOW ();
```

C) Écrivez la fonction générale d'ordonnancement basée sur la structure et le principe des **Files de priorité** décrits ci-dessus. Cette fonction doit également sauvegarder le processus courant dans la bonne file de priorité, restaurer et retourner le processus choisi. Cette fonction utilise les deux fonctions précédentes. Elle reçoit en paramètre le processus courant et la priorité du processus. La fonction fait un `exit` s'il n'y a plus aucun processus à ordonnancer.

```
PSW ordonnancement (PSW cpu, int prior);
```

D) En se basant sur le principe de **Files de priorités** décrits ci-dessus et pour un quantum égal à 5 unités de temps. Donnez l'ordonnancement des processus et le temps moyen d'attente pour les processus suivants :

Process	Date	Time	Priority
P1	0	18	LOW
P2	8	25	LOW
P3	12	10	HIGH
P4	18	2	HIGH
P5	20	23	LOW
P6	21	3	HIGH

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
20 mai 2015, première session,
durée 2 heures, calculatrices et documents autorisés.

1 Gestion des processus et synchronisation *(8 points)*

Considérons un système dans lequel le processeur **unique** est géré **sans réquisition**. La machine possède **un seul** disque. Traitez les questions suivantes :

A) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur sur la période considérée. [2 points]

```
pour i variant de 1 à 3 faire
  | calculer pendant i seconde(s)
  | écrire sur un fichier pendant (2 × i) secondes
  | si (i = 2) alors calculer pendant une seconde
fin-pour
calculer pendant une seconde
```

B) Même question que précédemment mais nous modifions le processus comme suit (indiquez sur un schéma le déroulement de l'exécution pour **le cas le plus favorable** et **le cas le moins favorable**). [2 points]

```
pour i variant de 1 à 3 faire
  | créer un thread pour faire
  |   | calculer pendant i seconde(s)
  |   | écrire sur un fichier pendant (2 × i) secondes
  |   | si (i = 2) alors calculer pendant une seconde
  | fin du thread
fin-pour
attendre la fin des threads fils
calculer pendant une seconde
```

C) Visiblement l'ordre d'exécution des threads a un impact sur les performances. Comment modifier le code pour s'assurer que le bon thread s'exécute en premier ? [2 points]
Conseil : utilisez un sémaphore.

D) En partant de la question B, si nous utilisons un algorithme de type **tourniquet** avec une tranche de temps très fine et que nous négligeons les temps de commutation d'un thread à une autre, quel ordonnancement obtenons-nous (faites un schéma et donnez le temps de réponse) ? [2 points]

Tournez la page SVP...

2 Gestion de fichiers à trous *(12 points)*

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*) :

```
struct {
    int suiv;          /* adr. physique sur bloc suivant (-1 si dernier) */
    int num;           /* numéro du bloc logique                         */
} fat[ NB_BLOCS ];     /* la FAT chargée en mémoire                      */
```

Le numéro permet de coder des fichiers à trous (tous les numéros logiques ne correspondent pas à des blocs physiques). Si `fat[`P`].num =` L, alors le bloc logique L est stocké dans le bloc physique P. **Attention** : le chaînage de la FAT ne suit pas forcément l'ordre logique des blocs.

Un *descripteur* est un enregistrement qui permet l'accès aux données d'un fichier.

```
typedef struct {
    int premier;    /* adr. du premier bloc physique (-1 si vide) */
    int nb_blocs;   /* nombre de blocs physiques                  */
} descripteur;
```

Dans l'exemple ci-dessous vous pouvez retrouver un fichier qui commence à **l'adresse 4** et qui est constitué de **trois** blocs physiques. Les blocs libres sont signalés par le champ `suiv` à -2.

	0	1	2	3	4	5	6	7	8	9	10	11	12
suivant	-2	-1	-2	11	8	-2	-1	-2	1	-2	-2	12	6
numéro	-	0	-	3	9	-	10	-	7	-	-	4	5

Questions :

A) Retrouvez dans l'exemple l'autre fichier, son nombre de blocs logiques et physiques. **[2 points]**

B) Écrivez `int nb_trous(descripteur d)` qui calcule le nombre de trous (des blocs logiques qui ne correspondent à aucun bloc physique). **[2 points]**

C) Écrivez `int adr_physique_pour_lecture(descripteur d, int num)` qui renvoie l'adresse physique d'un bloc logique **afin de le lire**. Cette fonction renvoie -1 si le bloc en question n'existe pas (tentative de lecture dans un trou). **[2 points]**

D) Écrivez `int allouer_bloc()` qui permet d'allouer un bloc sur disque. Cette fonction renvoie -1 si le disque est plein. **[1 point]**

E) Écrivez `int adr_physique_pour_ecriture(descripteur *d, int num)` qui renvoie l'adresse physique d'un bloc logique **afin de le modifier**. Si ce bloc n'existe pas, la fonction doit allouer un nouveau bloc et mettre à jour les chaînages. Cette fonction renvoie -1 si le disque est plein. **[2 points]**

F) Écrivez `void creer_un_trou(descripteur *d, int num)` qui supprime un bloc logique **si il existe**. **[3 points]**

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
Faculté des Sciences, site de Luminy
Aix-Marseille Université
15 mai 2014, première session,
durée 2 heures, calculatrices et documents autorisés.

1 Gestion des processus et synchronisation *(7 points)*

Considérons un système dans lequel le processeur **unique** est géré **sans réquisition**.

Questions :

a) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur sur la période considérée. [1 point]

```
pour i ∈ {1, 2, 3} faire
  | calculer pendant i seconde(s)
  | écrire sur un fichier pendant (2 × i) secondes
fin-pour
calculer pendant 1 seconde
```

b) Même question que précédemment mais nous modifions le processus comme suit (indiquez clairement sur un schéma le déroulement de l'exécution pour le cas le **plus favorable**). [2 points]

```
pour i ∈ {1, 2, 3} faire
  |   créer un thread pour faire
  |     | calculer pendant i seconde(s)
  |     | écrire sur un fichier pendant (2 × i) secondes
  |   fin du thread
fin-pour
attendre la fin des threads fils
calculer pendant 1 seconde
```

c) Si nous considérons que les E/S de 2s et 4s se déroulent sur un disque A et que celle de 6s se déroule sur un autre disque B, quel est l'impact sur le temps de réponse et le taux d'utilisation du processeur ? Étudiez le cas le plus favorable et le cas le moins favorable. [2 points]

d) Visiblement l'ordre d'exécution des threads a un impact sur les performances. Comment modifier le code pour s'assurer que le calcul de 3 secondes s'exécute en premier ? [2 points]
Conseil : utilisez un sémaphore.

2 Questions... réponses *(6 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante (n'oubliez pas de justifier vos réponses).

Questions : (un point par question)

a) Sur une zone de 1024 ko gérée par l'algorithme du *buddy system*, quel est l'état des blocs libres et occupés après les allocations de 65 ko, 130 ko, 220 ko, 50 ko et 91 ko dans cet ordre ?

b) Si le programme « $P(s); P(t); \ldots; V(s); \ldots; P(t); \ldots; V(t); V(t); \ldots; P(s); \ldots; V(s);$ » est exécuté en plusieurs exemplaires, existe-t-il un risque d'interblocage (les compteurs sont initialisés à deux) ?

c) soit les cinq processus suivants qui démarrent en même temps. Combien faut-il de processeurs pour les exécuter sans contrainte ? Quel est le taux d'utilisation du processeur ?
- calculer 3s, E/S sur 4s, calculer 3s
- calculer 2s, E/S sur 3s, calculer 1s, E/S de 3s, calculer 1s
- calculer 2s, E/S de 2s, calculer 2s, E/S de 2s, calculer 2s
- dormir sur 5s, calculer 2s, E/S de 3s
- E/S de 3s, calculer 3s, E/S de 4s

d) Donnez la matrice `max` pour ces trois processus :
- P_1 : allouer R_1 et R_2, ..., libérer R_1, ..., allouer R_3, R_2 et R_1, ..., libérer toutes les resources
- P_2 : allouer R_3 et R_1, ..., libérer R_3, ..., allouer R_2 et R_1, libérer toutes les ressources
- P_2 : allouer R_2 et R_3, ..., allouer R_2, ..., allouer R_3, ..., libérer toutes les ressources

e) Si le système recoit les requêtes ci-dessous, donnez l'ordre de traitement en se basant sur l'algorithme SSTF (*Shortest Seek Time First*). La tête de lecture se trouve sur la piste 500.

200, 150, 320, 700, 210, 100, 600, 705, 490, 150, 310, 610, 150

f) Même question mais pour l'algorithme de balayage (la position initiale est 500 et le sens est descendant).

3 Mémoire paginée *(7 points)*

Considérons une machine 32 bits (pointeurs et données) dotée d'un mécanisme de pagination à un seul niveau (une table des pages par processus et des pages de données). Chaque entrée de la table de pages mesure 32 bits (dont seulement 28 sont utilisables). **Questions :**

a) Pourquoi perdons-nous quatre bits dans chaque entrée de la table des pages ? [1 point]

b) Si les pages mesurent 2^{10} octets, quelle est la taille de mémoire centrale susceptible d'être gérée par le système d'exploitation ? [1 point]

c) Si les pages mesurent 2^{10} octets, donnez la composition d'une adresse paginée (parties et tailles en bits) [1 point]

Considérons maintenant que nous avons une pagination à deux niveaux (une table des hyper-pages par processus, des tables de pages et des pages de données). Chaque entrée de la table des hyper-pages mesure 32 bits. **Questions :**

d) Partons du principe que nous avons des pages de 2^{11} octets. Précisez la forme d'une adresse paginée (parties et tailles en bits). [1 point]

e) Toujours avec des pages de 2^{11} octets, quelle est la taille maximale de la table des hyper-pages ? Quel est le problème posé par cette taille ? [1 point]

f) Répondez à la question d) avec des pages de 2^{12} octets. Expliquez pourquoi le problème évoqué à la question e) est résolu. [2 points]

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
Faculté des Sciences, site de Luminy
Aix-Marseille Université
15 mai 2013, première session,
durée 2 heures, calculatrices et documents autorisés.

1 Gestion des processus et synchronisation *(6 points)*

Questions :

a) Donnez le temps de réponse du processus ci-dessous et le taux d'utilisation du processeur (qui est unique) sur la période considérée. [1 point]

```
pour i ∈ {1, 2, 3} faire
  | calculer pendant 2 secondes
  | écrire sur un fichier pendant 4 secondes
fin-pour
calculer pendant 1 seconde
```

b) Même question que précédemment mais nous modifions le processus comme suit (indiquez clairement sur un schéma le déroulement de l'exécution). [2 points]

```
pour i ∈ {1, 2, 3} faire
  | créer un thread pour faire
  |   | calculer pendant 2 secondes
  |   | écrire sur un fichier pendant 4 secondes
  | fin du thread
fin-pour
attendre la fin des threads fils
calculer pendant 1 seconde
```

c) Dans la version précédente il existe un risque de ne pas construire exactement le même fichier. Pouvez-vous expliquer ce phénomène et proposer une correction. [2 points]
Conseil : utilisez un tableau de sémaphores défini par

```
soit sems : tableau[1..3] de sémaphores initialisés à zéro
```

d) Finalement, l'instruction d'attente de la fin des *threads* fils n'existe pas ! Comment obtenir (élégamment) le même comportement en utilisant un sémaphore ? [1 point]

2 Questions... réponses *(7 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante (n'oubliez pas de justifier vos réponses).

Tournez la page SVP...

Questions : (un point par question)

a) Comment le système peut-il retirer de la mémoire à un processus et sur quels critères ?
b) A quels moments le système s'exécute-t-il ? et à qui est « facturé » cette exécution ?
c) Comment le système assure-t-il la sécurité des données stockées sur un disque ?
d) Si le programme « $P(s); P(t); \ldots; V(s); \ldots; P(t); \ldots; V(t); V(t)$ » est exécuté en plusieurs exemplaires, existe-t-il un risque d'interblocage ?
e) Sur une zone de 1024 ko gérée par l'algorithme du *buddy system*, quel est l'état des blocs libres et occupés après les allocations de 70 ko, 150 ko, 200 ko, 60 ko et 71 ko dans cet ordre ?
f) Les pseudo-partitions de répartition (RAID 0 ou *stripping*) améliorent-elles la sécurité ou l'efficacité ?
g) Pourquoi peut-on, dans le système UNIX, agir sur plusieurs sémaphores (par P ou V) en une seule opération atomique ?

3 Gestion de fichiers *(7 points)*

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de cases que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Un *descripteur* est un enregistrement qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
typedef struct {
    int taille;            /* nombre de blocs                       */
    int premier, dernier;  /* adr. physiques des blocs (-1 si vide) */
} descripteur;
```

Questions :

a) Écrivez la fonction qui insère un nouveau bloc (d'adresse `adr_bloc`) à la position `nu_bloc` (0 dénote le début du fichier). [2 points]

```
void inserer_bloc(descripteur *d, int fat[], int adr_bloc, int nu_bloc)
```

b) Écrivez la fonction qui vide un fichier de son contenu. [1 point]

```
void vider(descripteur *d, int fat[])
```

c) Écrivez une fonction qui crée un fichier de n blocs (utilisez les fonctions des questions a et b). Cette fonction renvoie 1 en cas de réussite et 0 sinon. [2 points]

```
int creation(descripteur *d, int fat[], int taille_fat, int n)
```

d) Écrivez la fonction qui liste sur la sortie standard l'adresse physique du premier bloc de chaque fichier. A quoi peut servir cette fonction ? [2 points]

```
void lister_1er_blocs(int fat[], int taille_fat)
```

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
Faculté des Sciences, site de Luminy
Aix-Marseille Université
25 juin 2013, deuxième session,
durée 2 heures, calculatrices et documents autorisés.

1 Gestion des processus *(7 points)*

On dispose d'une machine dotée d'un processeur et d'un seul disque. Considérons un processus unique qui réalise des calculs et des entrées/sorties d'une seconde chacune :

1s de CPU, deux E/S, 1s de CPU, trois E/S, 1s de CPU, quatre E/S, 1s de CPU.

Questions :

a) Donnez le temps de réponse et le taux d'utilisation du processeur sur la période considérée. [1 point]

b) Considérons que les E/S sont des écritures et que nous disposons d'un tampon de sortie capable de stocker deux E/S. Dans ces conditions, donnez le déroulé des opérations et calculez le nouveau taux d'utilisation du processeur. [2 points]

c) Quelle est la plus petite taille du tampon permettant que les quatre secondes de CPU s'exécutent sans interruption (donc en quatre secondes) ? Pourquoi voulons-nous une telle configuration et quel est le temps réponse obtenu ? [2 points]

d) Si chacun des trois cycles d'E/S est réalisé sur un disque différent (chacun ayant un tampon de deux secondes), quel est l'impact sur le temps de réponse ? Justifiez votre réponse et indiquez clairement à quelle(s) condition(s) vous obtenez ce résultat [2 points]

Tournez la page SVP...

2 Allocation de zones mémoire contiguës *(7 points)*

On considère une mémoire d'entiers composée de zones occupées et de zones libres. Chaque zone débute par sa taille (positive pour les zones libres et négative pour les zones occupées). Nous avons donc un chainage qui permet de retrouver toutes les zones.

Dans l'exemple ci-dessous, il y a deux zones libres qui totalisent une taille de 8 et une zone occupée de taille 4 :

3	0	-1	-4	10	11	-2	5	0	1	3	5

Questions :

a) Écrivez la fonction qui calcule la taille des zones libres de la mémoire. N'oubliez pas qu'il existe, en C, une fonction `abs(`v`)` qui permet de calculer la valeur absolue d'un entier v. [1 point]

```
int taille_libre(int mem[], int taille_mem)
```

b) Écrivez la fonction qui recherche et renvoie la position de la zone (occupée ou libre) qui précède la zone z. En cas d'erreur, cette fonction renvoie -1. [1,5 point]

```
int predecesseur(int z, int mem[], int taille_mem)
```

b) Écrivez la fonction qui fusionne la zone z et sa suivante si elles sont contiguës et libres. Cette fonction renvoie 1 si une modification est faite et zéro sinon. [2 points]

```
int fusionner_si_besoin(int z, int mem[], int taille_mem)
```

b) Écrivez la fonction qui libère la zone z en essayant de reconstituer la zone libre la plus grande. Utilisez les deux fonctions précédentes (même si vous n'avez pas répondu aux questions). Cette fonction 0 en cas de problème et 1 si la libération est faite. [2,5 points]

```
int liberer(int z, int mem[], int taille_mem)
```

3 Gestion d'une mémoire virtuelle *(6 points)*

Exercice déjà posé.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
Faculté des Sciences
Aix-Marseille Université
9 mai 2012, première session,
durée 2 heures,
calculatrices et documents autorisés.

1 Gestion des processus

(6 points, 35 minutes)

On dispose d'une machine dotée d'un processeur et d'un seul disque. Considérons un processus unique qui réalise des calculs et des entrées/sorties d'une seconde chacune :

2s de CPU, trois E/S, 2s de CPU, quatre E/S, 2s de CPU, une E/S, 1s de CPU.

Questions :

a) Donnez le temps de réponse et le taux d'utilisation du processeur sur la période considérée. [1 point]

b) Considérons que les E/S sont des écritures et que nous disposons d'un tampon de sortie capable de stocker deux E/S. Dans ces conditions, donnez le déroulé des opérations et calculez le nouveau taux d'utilisation du processeur. [1,5 point]

c) Quelle est la plus petite taille du tampon permettant d'obtenir le temps de réponse le plus court ? (justifiez votre réponse) [1,5 point]

d) Si chaque E/S est réalisée sur un disque différent, quel est l'impact sur le temps de réponse ? (justifiez votre réponse et indiquez clairement à quelle(s) condition(s) vous obtenez ce résultat) [2 points]

2 Gestion de fichiers

(8 points, 50 minutes)

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de cases que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Un *descripteur* est un enregistrement qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
typedef struct {
    int taille;    /* nombre de blocs         */
    int premier;   /* adresse du premier bloc */
    int dernier;   /* adresse du dernier bloc */
} descripteur;
```

Questions :

a) Écrivez une fonction qui ajoute un bloc à un fichier en mettant à jour les chaînages et le descripteur. [1 point]

```
void ajouter_bloc(descripteur *d, int fat[], int adr_bloc)
```

b) Écrivez la fonction qui insère un nouveau bloc (d'adresse `adr_bloc`) à la position `nu_bloc` (0 dénote le début du fichier). [2 points]

```
void inserer_bloc(descripteur *d, int fat[], int adr_bloc, int nu_bloc)
```

c) Écrivez la fonction qui vide un fichier de son contenu. [1 point]

```
void vider(descripteur *d, int fat[])
```

d) Écrivez une fonction qui crée un fichier de n blocs (utilisez les fonctions des questions a et c). Cette fonction renvoie 1 en cas de réussite et 0 sinon. [2 points]

```
int creation(descripteur *d, int fat[], int taille_fat, int n)
```

e) Écrivez la fonction qui liste sur la sortie standard l'adresse physique du premier bloc de chaque fichier. A quoi peut servir cette fonction ? [2 points]

```
void lister_1er_blocs(int fat[], int taille_fat)
```

3 Mémoire paginée

(6 points, 35 minutes)

Exercice déjà posé.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
Faculté des Sciences - Aix-Marseille Université
18 juin 2012, deuxième session,
durée 2 heures,
calculatrices et documents autorisés.

1 Gestion des processus

(6 points, 35 minutes)

On dispose d'une machine dotée d'un processeur et d'un seul disque. Considérons un processus unique qui réalise des calculs et des entrées/sorties d'une seconde chacune :

1s de CPU, deux E/S, 1s de CPU, trois E/S, 1s de CPU, quatre E/S, 1s de CPU.

Questions :

a) Donnez le temps de réponse et le taux d'utilisation du processeur sur la période considérée. [1 point]

b) Considérons que les E/S sont des écritures et que nous disposons d'un tampon de sortie capable de stocker deux E/S. Dans ces conditions, donnez le déroulé des opérations et calculez le nouveau taux d'utilisation du processeur. [1,5 point]

c) Quelle est la plus petite taille du tampon permettant que les quatre secondes de CPU s'exécutent sans interruption (donc en quatre secondes) ? Pourquoi voulons-nous une telle configuration et quel est le temps réponse obtenu ? [1,5 point]

d) Si chacun des trois cycles d'E/S est réalisé sur un disque différent (chacun ayant un tampon de deux secondes), quel est l'impact sur le temps de réponse ? Justifiez votre réponse et indiquez clairement à quelle(s) condition(s) vous obtenez ce résultat [2 points]

2 Gestion de fichiers

(7 points, 45 minutes)

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de cases que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Un *descripteur* est un enregistrement qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
typedef struct {
    int taille;    /* nombre de blocs                      */
    int premier;   /* adresse du premier bloc (-1 si vide) */
    int dernier;   /* adresse du dernier bloc (-1 si vide) */
} descripteur;
```

Tournez la page SVP...

Questions :

a) Écrivez une fonction qui ajoute un bloc à un fichier en mettant à jour les chaînages et le descripteur. [1,5 point]

```
void ajouter_bloc(descripteur *d, int fat[], int adr_bloc)
```

b) Écrivez la fonction qui supprime le premier bloc d'un fichier et met à jour le descripteur et la FAT. [2 points]

```
void supprimer_premier_bloc(descripteur *d)
```

c) Écrivez la fonction qui vide un fichier de son contenu et met à jour le descripteur. [1,5 point]

```
void vider(descripteur *d, int fat[])
```

d) Écrivez une fonction qui crée un fichier de n blocs contigus (placés les uns après les autres). Cette fonction renvoie le descripteur de ce nouveau fichier. La taille est forcée à -1 si l'opération a échoué. [2 points]

```
descripteur creation_contigue(int fat[], int taille_fat, int n)
```

3 Mémoire paginée

(7 points, 40 minutes)

Exercice déjà posé.

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
13 mai 2011, première session,
durée 2 heures,
calculatrices et documents autorisés.

1 Parallélisme et allocation de la CPU

(10 points, 1 heure)

Considérons le programme suivant

$$(A_1, \{B_1, A_2\}, A_3, \{C_1, (A_4, \{C_2, B_2\})\}, \{A_5, A_6, A_7\})$$

où les parenthèses dénotent une exécution séquentielle et les accolades une exécution parallèle. Une instruction $\{X_1, \ldots, X_n\}$ se termine quand toutes les instructions X_i sont terminées. Les instructions A_i durent une seconde, les B_i deux secondes et les C_i trois secondes.

Questions :

a) Donnez le temps d'exécution de ce programme et le taux d'utilisation du processeur sur une machine mono-processeur. [1 point]

b) Même question pour une machine bi-processeurs. Indiquez clairement la répartition de l'exécution sur les processeurs. [1,5 point]

c) Même question pour une machine dotée de trois processeurs. [2 points]

d) Si les instructions C_i sont composées, pour moitié de calcul processeur et pour l'autre moitié d'opérations d'E/S, quel est le taux d'utilisation du processeur sur une machine mono-processeur ? [1,5 point]

e) On vous donne les fonctions suivantes :

```
int   thread_create();           /* création d'un thread          */
void  thread_exit();             /* suicide d'un thread           */
sem_t sema_create(int compteur); /* création d'un sémaphore       */
void  sema_wait(sem_t s);        /* prise d'un sémaphore (P)      */
void  sema_signal(sem_t s);      /* libération d'un sémaphore (V) */
```

Le *thread* fils est créé par recopie du *thread* père. Les deux *threads* reviennent donc de l'appel à `thread_create`. Cette fonction renvoie 0 chez le fils et 1 chez le père.
Avec l'aide de ces fonctions, comment écrire un programme dans un langage proche du C qui soit équivalent à $\{\{U, V\}, (W, T), X\}$? [2 points]

e) Même question pour le programme $\{(A, B), (C, B)\}$ avec la contrainte supplémentaire : le code B ne doit être exécuté qu'une seule fois. [2 points]

2 Mémoire paginée

(5 points, 20 minutes)

Exercice déjà posé.

3 Les philosophes et les banquiers

(5 points, 40 minutes)

Un beau jour de mai, 5 philosophes décident d'aller manger ensemble des spaghettis. Le restaurateur dresse une table ronde, mais place seulement 5 fourchettes (une entre chaque assiette). Pour manger, un philosophe a besoin de deux fourchettes : celle placée à sa droite et celle placée à sa gauche.

Questions :

a) Montrez qu'il existe un risque d'interblocage et proposez une solution simple à ce problème. [1 point]

b) Pour éviter les interblocages, les philosophes décident d'appliquer l'algorithme des banquiers. Donnez la matrice `Max` et le vecteur `Dispo`. [1 point]

c) Pour **ce problème bien particulier** donnez l'algorithme simplifié permettant de déterminer si oui ou non, le système est dans un état sain. [1,5 point]

d) Sans vous préoccuper des problèmes de parallélisme et de synchronisation, donnez l'algorithme appliqué par le $i$ème philosophe avant de manger. [1,5 point]

Système d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
16 juin 2011, deuxième session,
durée 2 heures,
calculatrices et documents autorisés.

1 Parallélisme et allocation de la CPU *(9 points)*

Considérons un programme composé de quatre parties : A, B, C et D. Les parties A, B et C sont indépendantes. Pour s'exécuter, la partie D a besoin des résultats des trois autres parties.

Questions :

a) Si A, B, C et D durent respectivement 1 seconde, 3 secondes, 3 secondes et 1 seconde, donnez le meilleur temps d'exécution et le meilleur taux d'utilisation de la ressource processeur sur une machine dotée d'un processeur, puis de deux, puis de trois. Indiquez clairement la répartition de l'exécution sur les processeurs. [3 points]

b) On vous donne les fonctions suivantes :

```
int   thread_create();          /* création d'un thread          */
void  thread_exit();            /* suicide d'un thread           */
```

Le *thread* fils est créé par recopie du *thread* père. Les deux *threads* reviennent donc de l'appel à `thread_create`. Cette fonction renvoie 0 chez le fils et 1 chez le père.
Avec l'aide de ces fonctions, et **sans connaître le temps d'exécution** des quatre parties, comment écrire un programme dans un langage proche du C qui exploite le parallélisme de cet exemple ? **Attention** : vous ne disposez que de ces deux fonctions, mais les *threads* partagent les variables globales [2 points].

c) On vous donne les fonctions suivantes :

```
sem_t sema_create(int compteur); /* création d'un sémaphore       */
void  sema_wait(sem_t s);        /* prise d'un sémaphore (P)      */
void  sema_signal(sem_t s);      /* libération d'un sémaphore (V) */
```

Comment utiliser ces fonctions pour améliorer le code de la question précédente ? Exliquer l'amélioration attendue. [2 points]

c) Vous voulez maintenant brider votre programme pour ne pas utiliser plus de deux processeurs simultanément. Comment faire en utilisant les sémaphores ? [2 points]

2 Mémoire paginée *(5 points)*

Exercice déjà posé.

3 Questions... réponses *(6 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante.

Questions :

a) Entre *First-Fit*, *Best-Fit* et *Buddy-System* quel est l'algorithme le plus rapide ?
b) Pourquoi dit-on que les systèmes modernes sont dirigés par les interruptions ?
c) Dans un système de mémoire virtuelle paginée, pourquoi le système garde-t-il quelques pages physiques libres en réserve ?
d) Dans certains systèmes paginés, la PMMU n'a pas accès à la table des pages. Dans ces conditions, comment fonctionne la transformation des adresses ?
e) Le principe du tassage est-il utilisable pour l'algorithme par subdivision (*buddy system*) ?
f) Dans le problème des philosophes (simulés par des processus) et des spaghettis, donnez la matrice des besoins (matrice `max` de l'algorithme des banquiers). Pour ce problème, à quelle condition un état est-il sain ?

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
18 mai 2010, première session,
durée 2 heures,
calculatrices et documents autorisés.

1 Gestion de fichiers *(8 points)*

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de cases que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Questions :

a) Écrivez une fonction qui recherche un espace libre de n blocs consécutifs et renvoie son adresse (ou -1 en cas d'échec) en se basant sur la stratégie du premier trouvé (*first-fit*) [2 points].

```
int alloc_contigue(int fat[], int taille_fat, int n)
```

b) Même question pour la stratégie du meilleur ajustement (*best-fit*) [2 points].

c) Un descripteur est un enregistrement qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
typedef struct {
    int taille_en_blocs;
    int adr_premier_bloc;
    int adr_dernier_bloc;
} descripteur;
```

Écrivez la fonction « `verifier` » qui vérifie le codage d'un fichier (descripteur, taille et chaînage) et renvoie 1 en cas de succès et 0 sinon. [2 points]

```
int verifier(descripteur d, int fat[], int taille_fat)
```

d) Écrivez la fonction « `tronquer` » qui réduit la taille du fichier à n blocs (ou moins si la taille d'origine est inférieure à n blocs). [2 points]

```
void tronquer(descripteur *d, int fat[], int n)
```

2 Gestion des processus *(5 points)*

On dispose d'une machine dotée d'un processeur (géré sans réquisition) et d'un seul disque. Considérons deux processus A (arrivé en premier) et B (arrivé une seconde après) définis par :

A : 2s de CPU, 3s d'E/S, 2s de CPU, 4s d'E/S et 1s de CPU ;
B : 2s de CPU, 3s d'E/S, 1s de CPU, 3s d'E/S et 2s de CPU ;

Questions :

a) Donnez le taux d'utilisation du processeur pour chaque processus et de manière globale sur la période considérée. [1,5 point]

b) Considérons que les E/S sont des écritures et que nous disposons d'un tampon de sortie capable de stocker une seconde d'écriture. Dans ces conditions, donnez le schéma de l'ordonnancement des processus et calculez les nouveaux taux d'utilisation du processeur. [1,5 point]

c) A quelles conditions l'ajout d'un nouveau disque permet-il d'améliorer les performances ? [2 points]

3 Questions... réponses *(7 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante.

Questions :

a) Pourquoi faut-il ajuster la taille de la tranche de temps ? (0,5 point) et comment le faire ? (0,5 point)

b) Pourquoi et sur quels critères le système retire-t-il de la mémoire à un processus ? (1 point)

c) Pourquoi la taille des pages est-elle toujours une puissance de deux ? (0,5 point)

e) A quel moment un processus s'exécute-t-il en mode système ? (1 point)

g) L'algorithme par subdivision (*buddy system*) provoque-t-il de la fragmentation interne ? externe ? (0,5 point)

h) Si nous disposons de disques capables de lire ou d'écrire des blocs de 512 octets, pourquoi le système va-t-il utiliser des blocs de 1ko ou plus ? (1 point)

i) Pourquoi peut-on, dans le système UNIX, agir sur plusieurs sémaphores (par P ou V) en une seule opération atomique ? (1 point)

j) A quoi peut servir une fonction système **non bloquante** permettant de récupérer le compteur d'un sémaphore ? (1 point)

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
16 juin 2010, deuxième session,
durée 2 heures,
calculatrices et documents autorisés.

1 Gestion de fichiers *(8 points)*

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de cases que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Un *descripteur* est un enregistrement qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
typedef struct {
    int taille_en_blocs;
    int adr_premier_bloc;
    int adr_dernier_bloc;
} descripteur;
```

Questions :

a) Écrivez une fonction qui va créer un fichier de n blocs (les chainages et le descripteur doivent être préparés). Cette fonction renvoie 1 en cas de réussite et 0 sinon. (2 points)

```
int creation(descripteur *d, int fat[], int taille_fat, int n)
```

b) Écrivez une fonction qui ajoute un bloc à un fichier en mettant à jour les chaînages et le descripteur. (1 point)

```
void ajouter_bloc(descripteur *d, int fat[], int adr_bloc)
```

c) Écrivez la fonction qui insère un nouveau bloc (d'adresse `adr_bloc`) à la position `nu_bloc` (0 dénote le début du fichier). (2 points)

```
void inserer_bloc(descripteur *d, int fat[], int adr_bloc, int nu_bloc)
```

d) Écrivez la fonction qui vide un fichier de son contenu. (1 point)

```
void vider(descripteur *d, int fat[])
```

e) Écrivez la fonction qui liste sur la sortie standard l'adresse physique du premier bloc de chaque fichier. A quoi peut servir cette fonction ? (2 points)

```
void lister_1er_blocs(int fat[], int taille_fat)
```

2 Gestion des processus *(5 points)*

Prenons cinq processus notés de A à E ayant les caractéristiques suivantes :

processus	date d'arrivé	durée
A	0	3
B	1	5
C	7	1
D	6	3
E	4	2

Questions :

a) Donnez le temps de réponse cumulé si le système applique la stratégie FIFO pour ordonnancer les processus. (1 point)

b) Même question pour les stratégies SJF, SRT et Tourniquet (avec une tranche de temps de une unité). (3 points)

c) Pourquoi ne parle-t-on pas d'entrées/sorties dans la description de ces processus ? (1 point)

3 Questions... réponses *(7 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante.

Questions :

a) Entre *First-Fit*, *Best-Fit* et *Buddy-System* quel est l'algorithme le plus rapide ?

b) Pourquoi dit-on que les systèmes modernes sont dirigés par les interruptions ?

c) Dans un système de mémoire virtuelle paginée, pourquoi le système garde-t-il quelques pages physiques libres en réserve ?

d) Dans certains systèmes paginés, la PMMU n'a pas accès à la table des pages. Dans ces conditions, comment fonctionne la transformation des adresses ?

e) Si un processus exécute la séquence suivante : P(s) ;P(s) ;V(s) ;P(s) ;P(s) ;V(s) ;P(s) à quelle condition peut-on garantir l'absence de blocage ?

f) Le principe du tassage est-il utilisable pour l'algorithme par subdivision (*buddy system*) ?

g) Dans le problème des philosophes (simulés par des processus) et des spaghettis, donnez la matrice des besoins (matrice `max` de l'algorithme des banquiers). Pour ce problème, à quelle condition un état est-il sain ?

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
15 mai 2009, première session,
durée 2 heures,
calculatrices et documents autorisés.

1 Synchronisation de processus *(7 points)*

Exercice déjà posé.

2 Allocation contigüe de la mémoire *(6 points)*

On considère la suite de demandes d'allocation ($+$) et de libération ($-$) suivantes, dans un espace mémoire de 1000 blocs.

$$+300,\ +200,\ +240,\ -200,\ +100,\ -300,\ +250,\ +400,\ -240,\ +150,$$
$$+95,\ -100,\ -95,\ +200,\ -150,\ -250,\ +100,\ -400,\ -100,\ -200$$

a) Si la mémoire est initialement vide, comment le S.E. réalise les allocations en partant du principe qu'il applique la stratégie de la première zone libre. On complète cette stratégie par l'application d'un algorithme de tassage si aucune des zones libres ne convient.
b) Même question si le système applique la stratégie du meilleur ajustement. Comparez ces deux stratégies en utilisant le nombre de tassages effectués et la quantité de données déplacées.

3 Questions... réponses *(7 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante.

Questions :

a) Faut-il plusieurs processus pour exploiter pleinement une machine bi-processeurs ?
b) Comment le système peut-il retirer de la mémoire à un processus ?
c) A quels moments le système s'exécute-t-il ?
d) Comment le système assure-t-il la sécurité des données stockées sur un disque ?
e) Quel est le mécanisme permettant à un processus de réaliser des écritures asynchrones sur disque ?
f) Quelle est la définition du tampon utilisateur et son rôle ?
g) L'algorithme par subdivision (*buddy system*) provoque-t-il de la fragmentation interne ? externe ?
h) Si nous disposons de disques capables de lire ou d'écrire des blocs de 512 octets, pourquoi le système va-t-il utiliser des blocs de 1ko ou plus ?
i) Pourquoi peut-on, dans le système UNIX, agir sur plusieurs sémaphores (par P ou V) en une seule opération atomique ?
j) A quoi peut servir une fonction système non bloquante permettant de récupérer le compteur d'un sémaphore ?

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
18 juin 2009, deuxième session,
durée 2 heures,
calculatrices et documents autorisés.

1 Gestion de fichiers *(7 points)*

Exercice déjà posé.

2 Questions... réponses *(7 points)*

Les questions ci-dessous doivent être traitées dans le cadre d'un système d'exploitation moderne, multi-threads, multi-programmé et doté d'une gestion mémoire performante. N'oubliez pas de justifier vos réponses.

Questions :

a) Soit un processus qui effectue 5 fois, 1 seconde de calcul et 2 secondes d'écriture sur un fichier, donnez le taux d'utilisation du processeur sur la période considérée.
b) Si nous ajoutons à la question a) un tampon de sortie pouvant contenir 1 seconde d'entrée/sortie, quel est le gain espéré ?
c) A quels moments dit-on que le système s'exécute pour le compte d'un processus ?
d) Les pseudo-partitions de répartition (RAID 0 ou *stripping*) améliorent-elles la sécurité ou l'efficacité ?
e) Pourquoi, pour un sémaphore s donné, les procédures P(s) et V(s) s'exécutent-elles en exclusion mutuelle ?
f) Dans le problème des philosophes et des spaghettis, comment appliquer une règle simple (mais efficace) pour éviter les interblocages ?
g) Quels sont les critères utilisés par le système d'exploitation pour réduire ou augmenter la taille des tranches de temps alloués à un processus ?
h) Quels sont les critères utilisés par le système d'exploitation pour réduire ou augmenter la mémoire physique alloué à un processus.

3 Gestion d'une mémoire virtuelle *(6 points)*

Exercice déjà posé.

Systèmes d'exploitation

Licence d'informatique — Faculté de Luminy
2 septembre 1998,
durée 3 heures,
Tous les documents sont autorisés.

1 Gestion d'une mémoire virtuelle *(6 points)*

Considérons une machine qui comporte une mémoire composée de trois pages physiques. Lors de son exécution, un programme accède (dans l'ordre) aux pages virtuelles suivantes :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

Initialement, les trois pages physiques ne contiennent aucune page virtuelle et le programme est le seul à s'exécuter.

a) Pour chacun des algorithmes FIFO, FINUFO et LRU, donnez le contenu des pages physiques (c-à-d un numéro de page virtuelle) après chaque accès. Précisez également le nombre total de défauts de page provoqués par ces accès. (3 points)

b) Donnez la courbe « nombre de pages physiques / nombre de défauts de page » de l'algorithme LRU pour l'exemple ci-dessus. (3 points)

2 Allocation du processeur *(6 points)*

On vous donne les six processus suivants :

processus	arrivée	durée
1	0	10
2	3	10
3	5	3
4	11	2
5	14	5
6	15	3

Questions :

a) Si le « scheduler » suit les stratégies FIFO, SJF, SRT et RR (avec un quanta de 2), comment alloue-t-il la CPU aux processus ? Comparez les stratégies en utilisant le temps de réponse comme critère. (3 points)

b) Même question si le scheduler applique la stratégie HRN avec réquisition de la CPU à l'arrivée d'un nouveau processus (3 points).

Tournez la page SVP...

3 Implantation physique des fichiers *(8 points)*

Le *bloc* est l'unité élémentaire d'entrée/sortie sur disque. Il est défini comme suit :

```
bloc = tableau [ 0..255 ] d'entiers;
```

Les lectures de blocs se font au moyen de la fonction `lire_bloc( nu:entier; var b:bloc )`. Un descripteur de fichiers est un bloc qui contient les informations relatives à l'implantation physique du fichier. Ce descripteur à la structure suivante :

— Les cases de 0 à 243 contiennent des informations diverses.
— Les cases de 244 à 253 contiennent respectivement les adresses physiques des blocs de données 0 à 9 du fichier en question.
— La case 254 contient l'adresse physique d'un index. Chaque entrée de cet index contient l'adresse physique d'un bloc de données.
— La case 255 contient l'adresse physique d'un maître-index. Chaque entrée de ce maître-index contient l'adresse physique d'un index.

Questions :

a) Sur quel système retrouve-t-on ce type d'organisation ?

b) Si l'entier correspond à quatre octets, quelle est la taille maximum d'un fichier avec ce type d'organisation ?

c) Écrivez la fonction

```
fonction adresse_physique( desc :bloc ; nubloc :entier ) : entier ;
```

qui rend l'adresse physique du bloc de données d'adresse logique `nubloc` dans le fichier décrit par `desc`. Vous ne vous préoccuperez pas des problèmes de débordement ou des erreurs d'entrée/sortie.

d) Si on cherche à lire 1000 octets à partir de l'adresse 272000, combien de blocs le système d'exploitation doit-il lire en partant du principe qu'aucun bloc n'est gardé en mémoire ?

Systèmes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
12 juin 2008, deuxième session,
durée 3 heures,
calculatrices et documents autorisés.

1 Gestion d'une mémoire virtuelle *(9 points)*

Certaines machines (comme le processeur IBM RS-6000) n'utilisent pas une table des pages virtuelles pour chaque processus. Sur ce type de système, l'état de la mémoire est décrit par une seule table qui dispose d'une entrée pour chaque page physique. Cette table a la forme suivante

```
struct {
    int idp;                /* nu de processus        */
    int npv;                /* nu de page virtuelle   */
    int protection;         /* codage des permissions */
    int modif;              /* bit de modification    */
} dpp[NB_PAGES_PHYSIQUES];
```

Les pages physiques libres sont signalées par le champ `idp` égal à zéro. Les adresses virtuelles et physiques ont maintenant la forme suivante :

```
struct ADRV { int idp; int npv; int deplacement; };
struct ADRP { int npp; int deplacement; };
```

Questions :

a) A-t-on, du point de vue d'un processus, les mêmes capacités d'adressage mémoire que dans un système plus classique ?

b) Ecrivez la fonction de transformation des adresses virtuelles en adresses physiques.

```
struct ADRP transformer(struct ADRV a) { ... }
```

Dans cet algorithme, le défaut de page va correspondre à l'appel de la fonction :

```
struct ADRP defaut_de_page(struct ADRV a);
```

c) Quel est l'impact de la taille des pages sur la performance du processus de transformation des adresses. Justifiez votre réponse en présentant quelques exemples chiffrés. Quels sont les moyens, matériels et/ou logiciels, qui permettent de réduire le coût de cette transformation.

d) Écrivez la fonction `defaut_de_page`. Pour simplifier, vous pouvez considérer qu'il existe une zone de pagination pour chaque processus. Vous disposez alors des deux fonctions :

```
charger_page( int idp, int npv, int npp);
sauver_page ( int idp, int npp, int npv);
```

e) Comment le partage de pages peut-il s'organiser dans ce système ?

2 Gestion des processus *(5 points)*

Soit un processus dont le travail consiste à calculer (pendant 1 seconde) et à sauver le résultat dans un fichier (pendant 4 secondes) le tout dix fois.

Questions :

a) Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ? [1 point]

b) Admettons que le système alloue un tampon de sortie pouvant contenir 2 secondes d'entrée/sortie. Dans ces conditions répondez à la question a) et donnez la trace de l'exécution. [2 points]

c) Du point de vue du système d'exploitation, est-il intéressant d'allouer un tampon de sortie dont la taille est supérieur à 4 secondes (justifiez votre réponse) ? [2 points]

3 Mémoire paginée à deux niveaux *(6 points)*

Considérons un système basé sur une mémoire paginée à deux niveaux. La taille des pages est fixée à 2^{10} octets et la machine manipule des adresses et des entiers de 32 bits. La mémoire du processus courant est décrite par les paramètres `RL=31076`, `RB=600` et les pages ci-dessous.

Adresse 150 :

	n° p. phys.
0	950
⋮	⋮
100	500
101	190
102	100
⋮	⋮
255	550

Adresse 400 :

	n° p. phys.
⋮	⋮
50	800
51	450
52	350
⋮	⋮

Adresse 600 :

	n° p. phys.
0	180
⋮	⋮
10	150
11	170
12	900
⋮	⋮
120	400
121	401

Adresse 900 :

	n° p. phys.
⋮	⋮
200	200
201	700
202	300
⋮	⋮

Questions :

a) Donnez la composition d'une adresse paginée dans ce système (nature et taille en nombre de bits de chaque composante) ? [1 point]

b) Quelle est la taille maximum de la mémoire alloué à un processus ? Comment procéder pour augmenter cette limite ? [2 points]

c) Quelle est la taille de la mémoire allouée au processus courant. [1 point]

d) Donnez les adresses physiques des adresses paginées 2621440, 2726554, 3351702 et 31511528. [2 points]

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
16 mai 2006,
durée 3 heures,
calculatrices et documents autorisés.

Responsable : Jean-Luc Massat

1 Synchronisation de processus *(9 points)*

On se propose de tirer parti d'une machine multi-processeurs pour accélerer l'exécution d'un processus. Ce processus est composé de six portions de code notées A, B, C, D, E et F. L'analyse de ces portions nous révèle les contraintes suivantes :

1. l'exécution de B doit se placer après C, G et A,
2. l'exécution de D doit se placer après A et G,
3. l'exécution de F doit se placer après B ou bien D,
4. l'exécution de C doit se placer après A et E,

Nous n'avons aucune information sur la durée d'exécution de ces portions de code.

Questions :

a) Donnez un ordre d'exécution séquentiel qui respecte les contraintes énoncées.

b) Imaginons que dans un processus on puisse utiliser les notations suivantes :

cobegin $P_1;\ \ldots\ P_n;$ **coend**
begin $S_1;\ \ldots\ S_m;$ **end**

pour indiquer au système que les instructions P_i peuvent se dérouler en parallèle tandis que les instructions S_i doivent être exécutées de manière séquentielle. Une instruction **cobegin**/**coend** se termine quand toutes les instructions qui la composent sont terminées.
Utilisez ces notations pour rendre **le plus parallèle possible** l'exécution des six portions de code. Tout le parallélisme potentiel est-il utilisé ? (justifiez votre réponse).

c) En utilisant des sémaphores, les **cobegin**/**coend** et les **begin**/**end**, pouvez vous reformuler le processus pour utiliser tout le parallélisme potentiel. N'oubliez pas de préciser, pour chaque sémaphore, la valeur initiale du compteur.

d) Même question que précédemment si la contrainte 2 est modifiée comme suit :

2. l'exécution de D doit se placer après A et G **ou bien** après E.

e) Si chaque portion calcule pendant une seconde, quel est le nombre de processeurs dont vous avez besoin ?

2 Traitement des interblocages *(5 points)*

Considérons un système d'allocation de ressources portant sur 5 processus et 4 classes de ressources. L'état du système est donné par les informations suivantes :

Allocation

	R_1	R_2	R_3	R_4
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

Max

	R_1	R_2	R_3	R_4
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

Dispo

R_1	R_2	R_3	R_4
1	5	2	0

Dans ce cadre, répondez aux questions suivantes en utilisant l'algorithme des banquiers :

a) Combien faut-il ajouter de ressources (pour chaque R_i) de manière à ce que les processus puissent s'exécuter sans contrainte.

b) Le système est-il dans un état sain ? justifiez votre réponse.

c) Si le processus P_1 dépose la requête $(0, 4, 2, 0)$, cette requête est-elle acceptée immédiatement ? (justifiez votre réponse).

d) A partir de la situation initiale, si le processus P_4 dépose la requête $(1, 4, 2, 0)$, quel est le résultat ?

3 Gestion des processus *(6 points)*

Soit le processus ci-dessous exécuté sur une machine mono-processeur

```
faire 3 fois
  |  calculer pendant 1 seconde
  |  écrire sur le fichier F1 pendant 2 secondes
fin-faire
faire 2 fois
  |  calculer pendant 1 seconde
  |  écrire sur le fichier F1 pendant 3 secondes
fin-faire
```

Les deux séries d'écritures sont indépendantes.

Questions :

a) Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ?

b) Si le processus est maintenant structuré sous la forme de deux *threads*, chacun s'occupant d'une boucle, quel est le nouveau temps de réponse et le taux d'utilisation de la CPU ? Dessinez une trace de l'exécution des threads.

c) Pour améliorer son code, le programmeur décide de garder les *threads*, mais de produire deux fichiers différents (F1 et F2 par exemple). A quelles conditions peut-on obtenir une amélioration ? Evaluez cette amélioration en calculant le temps de réponse et le taux d'utilisation de la CPU.

e) Admettons que le système alloue deux tampons de sortie pouvant contenir chacun 2 secondes d'entrée/sortie (un pour chaque *thread*). Dans ces conditions, répondez à la question a) et donnez la trace de l'exécution.

Systèmes d'exploitation

Troisième année de la licence d'informatique — U.F.R. des Sciences de Luminy
24 mai 2005,
durée 3 heures,
Tous les documents sont autorisés.

1 Gestion d'une mémoire virtuelle *(8 points)*

Considérons une machine qui comporte une mémoire composée de trois pages physiques initialement vides. Lors de son exécution, un processus unique accède dans l'ordre aux pages virtuelles suivantes :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

Questions :

a) Pour chacun des algorithmes FIFO, LRU et OPT, donnez le contenu des pages physiques (c-à-d un numéro de page virtuelle) après chaque accès. Précisez également le nombre total de défauts de page provoqués par ces accès. (3 points)
b) Si le nombre de pages physiques passe à quatre, quel est le nombre de défauts de page pour l'algorithme LRU (2 points) ?
c) Quel est le meilleur compromis pour la taille de la mémoire physique (1 point) ?
d) Donnez la nouvelle trace de l'exécution du processus en prenant comme hypothèse que la taille des pages virtuelles a doublé (2 points).

2 Synchronisation de processus *(6 points)*

Soient n processus numérotés $P_0, P_1, \ldots, P_{n-1}$. Chaque processus accède aux variables partagées suivantes :

```
attente : tableau [ 0 ... n-1 ] de booléens;
verrou : booléen;
```

Ces variables sont toutes initialisées à la valeur faux. Les autres variables sont propres à chaque processus. Par ailleurs, nous disposons sur cette machine d'une instruction câblée de synchronisation définie par

```
instruction Test-And-Set ( var copie:booléen, var verrou:booléen )
début
  en exclusion mutuelle faire
      copie := verrou;
      verrou := vrai;
   fin-faire
fin
```

Le processus P_i a la forme suivante :

```
attente[i] := vrai;
répéter
   Test-And-Set(copie, verrou)
jusqu'à (copie = faux) ou (attente[i] = faux)

< section critique >

j := (i + 1) mod n;
tant-que (j <> i) et (attente[j] = faux) faire j := (j + 1) mod n;
attente[i] := faux
si (i = j) alors verrou := faux
sinon attente[j] := faux
```

Questions :

a) Combien de processus exécutent leur section critique simultanément (justifiez votre réponse) ?
b) Quel est le comportement particulier de ces processus et en quoi cette solution est-elle meilleur que la programmation classique du prologue et de l'épilogue ?
c) Malgré ses qualités, il reste un défaut à cette solution. Lequel ?

3 Codage des fichiers *(6 points)*

Dans un système de fichiers, la position physique d'un fichier sur disque est définie par un descripteur :

```
struct descripteur {
  int nbblocs;         /* taille du fichier en blocs */
  int data[10];        /* adresses des 10 premiers blocs de données */
  int premierIndex;    /* adresse du premier bloc d'index */
  int dernierIndex;    /* adresse du dernier bloc d'index */
  }
```

Les blocs d'index sont chaînés et peuvent contenir `TAILLE_INDEX` références aux blocs de données. Nous disposons de deux fonctions pour manipuler ces index :

`int creerIndex(int bloc, int precedent)` Création d'un nouvel index contenant une seule référence. Cette fonction renvoie l'adresse physique du nouvel index sur disque.

`void ajouterReference(int index, int bloc)` Ajout d'un référence à un bloc d'index dont l'adresse physique est passée en paramètre. Un emplacement doit être disponible dans l'index en question.

Dans ce cadre, répondez aux questions suivantes :

a) Donnez une explication de ce codage (2 points).
b) Donnez une implantation de la fonction `ajouterBloc` qui a la charge d'ajouter un bloc de donnée à un descripteur en prenant soin, le cas échéant, de créer un nouvel index. Attention : quatre cas sont à étudier (4 points).

```
void ajouterBloc(struct descripteur* d, int bloc)
```

Systèmes d'exploitation

Licence d'informatique
Faculté des Sciences de Luminy
9 Juin 2004,
durée 3 heures,
Tous les documents sont autorisés.

1 Producteur et consommateur *(9 points)*

L'activité de gestion d'un entrepôt est formalisée par les variables partagées définies ci-dessous. Un processus producteur a la charge de remplir le stock tandis qu'un processus consommateur à la charge de le vider.

```
stock : tableau[0 à Max-1] de produits;
nbTotalProduits : entier;
nbTotalConsommés : entier;
```

Questions :

a) proposez une version de l'algorithme du producteur et du consommateur basée sur **l'attente active** pour régler les problèmes de synchronisation. N'utilisez ni sémaphore ni instruction particulière du processeur et indiquez clairement pourquoi votre solution d'attente active est correcte.

b) Pour éviter une perte de CPU, les concepteurs décident d'utiliser les sémaphores à compteur pour régler les problèmes de synchronisation. Donnez la nouvelle version de l'algorithme du producteur et du consommateur.

c) Les concepteurs observent que le stock est toujours plein. Ceci est dû au fait que le processus producteur est plus rapide que le processus consommateur. Pour régler ce problème, ils décident de placer deux consommateurs pour un producteur. Donnez la nouvelle version de producteur et des consommateurs.

d) Si nous avons deux consommateurs, nous pourrions avoir deux stocks (un pour chaque consommateur) et le producteur pourrait placer sa production alternativement dans les deux stocks. Expliquez pourquoi la solution du stock unique est meilleure (donnez au moins deux bonnes raisons).

e) Sur une machine mono-processeur quelle doit être la nature du traitement effectué par les consommateurs pour que la multiplication de ces derniers soit intéressante.

2 Gestion des processus *(5 points)*

Soit un processus dont le travail consiste à calculer (pendant 1 seconde) et à sauver le résultat dans un fichier (pendant 4 secondes) le tout dix fois.

Questions :

a) Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ?

b) Admettons que le système alloue un tampon de sortie pouvant contenir 2 secondes d'entrée/sortie. Dans ces conditions répondez à la question a) et donnez la trace de l'exécution.

c) Du point de vue du système d'exploitation, est-il intéressant d'allouer un tampon de sortie dont la taille est supérieur à 4 secondes (justifiez votre réponse) ?

d) Admettons maintenant que le fichier ainsi créé soit stocké sur une pseudo partition de répartition construite à l'aide de deux disques. Quel est l'impact de cette organisation sur l'exécution du processus ?

3 Les philosophes et les banquiers *(6 points)*

Un beau jour de juin, n philosophes décident d'aller manger ensemble des spaghettis. Le restaurateur dresse une table ronde, mais place seulement n fourchettes (une entre chaque assiette). Pour manger, un philosophe a besoin de deux fourchettes : celle placée à sa droite et celle placée à sa gauche.

Questions :

a) Montrez qu'il existe un risque d'interblocage et proposez une solution simple à ce problème.

b) Pour éviter les interblocages, les philosophes décident d'appliquer l'algorithme des banquiers. Donnez la matrice `Max` et le vecteur `Dispo`.

c) Pour **ce problème bien particulier** donnez l'algorithme simplifié permettant de déterminer si oui ou non, le système est dans un état sain.

d) Sans vous préoccuper des problèmes de parallélisme et de synchronisation, donnez l'algorithme appliqué par le $i$ème philosophe avant de manger.

e) Face à ce problème, le restaurateur décide de placer une assiette au centre de la table dans laquelle il pose les n fourchettes. Les philosophes vont chercher les fourchettes sur cette assiette et les y reposent après avoir mangé. Dans ce cadre, existe-t-il un risque d'interblocage ?

f) Donnez la nouvelle version de la matrice `Max` et du vecteur `Dispo`.

Systèmes d'exploitation

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
16 mai 2008, première session,
durée 3 heures,
calculatrices et documents autorisés.

Responsable : Jean-Luc Massat

1 Gestion de fichiers *(10 points)*

Considérons un disque géré à l'aide d'une FAT (*File Allocation Table*). Ce tableau d'entiers possède autant de lignes que de blocs sur le disque. Un bloc i est libre si fat$[i] = 0$. Dans les autres cas, il est utilisé.

Questions :

a) Écrivez une fonction qui recherche un espace libre de n blocs consécutifs et renvoie son adresse (ou -1 en cas d'échec) en se basant sur la stratégie du premier trouvé (*first-fit*) [2 points].

```
int alloc_contigue(int fat[], int taille_fat, int n)
```

b) Même question pour la stratégie du meilleur ajustement (*best-fit*) [3 points].

Descripteur. Un descripteur est un bloc qui permet l'accès aux données d'un fichier. Pour un fichier donné, si les blocs i et j se suivent, alors fat$[i] = j$. Si le bloc k est le dernier, alors fat$[k] = -1$.

```
struct descripteur {
    int taille_en_blocs;
    int adr_premier_bloc;
    int adr_dernier_bloc;
};
```

Questions :

c) Écrivez la fonction « `int verifier(struct descripteur d)` » qui vérifie le codage d'un fichier (descripteur, taille et chaînage) et renvoie 1 en cas de succès et 0 sinon. [2 points]

d) Sachant que les descripteurs sont marqués -2 dans la FAT, écrivez une fonction qui affiche les blocs de données qui n'appartiennent à aucun fichier (erreur de codage). [3 points]

```
void verifier_disque(int fat[], int taille_fat);
```

Vous avez à votre disposition une fonction de lecture des descripteurs :

```
struct descripteur lire_descripteur(int adr);
```

Conseil : utilisez un tableau supplémentaire qui donne l'état de chaque bloc du disque.

2 Parallélisme et utilisation des « threads » *(6 points)*

Vous venez d'acquérir une nouvelle machine dotée de plusieurs processeurs. Pour l'exploiter au mieux, vous décidez de modifier vos anciens programmes afin de les adapter à cette architecture. Pour ce faire, le système d'exploitation vous offre **deux fonctions** pour gérer des « threads » :

```
int thread_create();  /* création d'un thread */
void thread_exit();   /* suicide d'un thread  */
```

La fonction `thread_create` crée un nouveau « thread » fils par recopie à l'identique du « thread » courant (le père). Le « thread » fils démarre son exécution comme si il revenait de l'appel à la fonction `thread_create`. La fonction renvoie 0 chez le fils, et 1 chez le père.

Questions :

a) Le programme A est composé de trois parties (notées A_1, A_2 et A_3). A_3 a besoin des résultats de A_1 et A_2 et l'exécution de A_1 est toujours plus longue que celle de A_2. Dans ce cadre, donnez une version qui utilise le parallélisme de A et expliquez le gain attendu.

b) Même question, mais on ne connaît pas le plus long des deux codes A_1 et A_2. J'insiste : vous ne disposez de rien d'autre que les deux fonctions présentées ci-dessus. Par contre, je vous rappelle que les « threads » partagent les variables globales. Quelle critique pouvez-vous faire de la solution que vous proposez ?

c) Le programme B est composé de trois parties. Le code B_3 a besoin des résultats de B_1 **ou** B_2. Proposez une version parallèle **sans perte de CPU**. Pour vous aider, on vous donne une fonction **non interruptible** qui incrémente son argument et renvoie son ancienne valeur :

```
int test_and_set(int* verrou); /* renvoyer puis incrémenter  */
```

3 Mémoire virtuelle paginée *(4 points)*

Soit un système basé sur une mémoire paginée. La taille des pages est fixée à 2^{12} octets et la machine manipule des adresses et des entiers de 32 bits. La mémoire du processus courant est décrite par les paramètres ci-contre.

```
RL = 120
RB = 300
```

	n° p. phys.
0	600
⋮	⋮
70	900
71	950
⋮	⋮
119	210
120	350
⋮	⋮

Questions :

a) Si la table des pages mesure au maximum une page, quelle est la taille maximun de la mémoire alloué à un processus. Pourquoi cette limitation ? [1 point]

b) Quelle est la taille de la mémoire allouée au processus courant. [0,5 point]

c) Quelle est la taille maximum de la mémoire physique que le système puisse gérer. [1 point]

d) Donnez les adresses physiques des adresses paginées 3977, 288820 et 494640. [1,5 point]

N'oubliez pas de justifier vos réponses.

Systèmes d'exploitation

Licence d'informatique — Faculté des Sciences de Luminy
12 Juin 2002,
durée 3 heures,
Tous les documents sont autorisés.

1 Gestion des processus *(6 points)*

Soit le processus ci-dessous

```
faire 4 fois
  |  calculer pendant 1 seconde
  |  écrire sur le fichier F1 pendant 2 secondes
fin-faire
faire 4 fois
  |  calculer pendant 1 seconde
  |  écrire sur le fichier F1 pendant 3 secondes
fin-faire
```

Les deux séries d'écritures sont indépendantes.

Questions :

a) Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ?
b) Si le processus est maintenant structuré sous la forme de deux *threads*, chacun s'occupant d'une boucle, quel est le nouveau temps de réponse et le taux d'utilisation de la CPU ?
c) Pour améliorer son code, le programmeur décide de garder les *threads*, mais de produire deux fichiers différents (F1 et F2 par exemple). A quelles conditions peut-on obtenir une amélioration et laquelle (temps de réponse et taux de CPU) ?
e) Admettons que le système alloue deux tampons de sortie pouvant contenir 2 secondes d'entrée/sortie (un pour chaque *thread*). Dans ces conditions, répondez à la question a) et donnez la trace de l'exécution.

2 Allocation de ressources *(8 points)*

Soit deux processus (P_1 et P_2) et deux ressources non partageables (R_1 et R_2). Les deux processus vont utiliser les deux ressources (la matrice `Max` de l'algorithme des banquiers ne contient que des 1).

Questions :

a) Quels sont les états possibles de la ressource R_1 (au sens du graphe d'allocation de ressources) ?
b) Quels sont les états possibles du processus P_1 indépendamment de P_2 ?
c) Quel est le nombre d'états possibles pour le système d'allocation de ressources (il est conseillé de faire une étude de cas sur l'état de chaque ressource) ?
d) Si on applique l'algorithme des banquiers, combien reste-t-il d'états possibles pour le système d'allocation de ressources ?
e) Pour éviter les interblocages, on peut associer un ordre à chaque ressource (par exemple $R_1 < R_2$) et interdire les demandes d'allocation qui ne respectent pas cet ordre. Dans ces conditions, combien d'états sont encore possibles ?

3 Mémoire virtuelle paginée *(6 points)*

Sur certains processeurs, l'étape de transformation des adresses virtuelles paginées en adresses physiques est particulièrement simple. En effet, ces processeurs utilisent seulement leur mémoire associative sans effectuer d'accès à la table des pages (voir ci-dessous) :

```
<npv,dép> := adresse_virtuelle_paginée
si il existe un couple <npv,npp> dans la mém. associative
alors
    adresse_physique := <npp,dép>
sinon
    générer une interruption de défaut de mémoire associative
fin si
```

Questions :

a) Sur ces processeurs il existe une instruction
`ajouter_en_mémoire_associative <npv,npp>`
Que fait, d'après vous, cette instruction ?
b) Même si le processeur n'utilise plus la table des pages virtuelles, cette structure de données est-elle toujours utile (justifiez votre réponse) ?
c) Donnez les grandes lignes du traitant associé à l'interruption « défaut de mémoire associative ».
d) Quels sont les avantages et les inconvénients de cette implantation de la pagination par rapport à la formule classique ?
e) Cette implantation pose un problème dans le choix d'une victime. Décrivez le problème et proposez une solution.