

Système d'exploitation

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

Responsable : Jean-Luc Massat

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 réveiller 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

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

Responsable : Jean-Luc Massat

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 $\text{RAID5}(D1, D2, D3, D4, D5, D6)$) ? [0,5 point]

capacité = $(5 \times T_0)$

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

capacité = $(6 \times T_0)$

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 $\text{RAID0}(\text{RAID5}(D1, D2, D3), \text{RAID5}(D4, D5, D6))$

C2 $\text{RAID5}(\text{RAID0}(D1, D2), \text{RAID0}(D3, D4), \text{RAID0}(D5, D6))$

C3 $\text{RAID5}(\text{RAID1}(D1, D2), \text{RAID1}(D3, D4), \text{RAID1}(D5, D6))$

RAID1 = mirroring, capacité de $\text{RAID1}(A, B) = 1 T_0$

	Capacité	Fautes	Exemple
C1 :	4 T_0 ,	2 disques,	D1 et D4
C2 :	4 T_0 ,	2 disques,	D5 et D6
C3 :	2 T_0 ,	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|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 occupées, 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]

```
LRU : page 1 (dernier accès le plus vieux 260)
FIFO : page 2 (chargement le plus vieux 120)
FINUFO : page 0 (bit d'accès à zéro)
```

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

```
La date du prochain accès dans le futur.
```

- 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]

```
A M | Priorité pour choisir la victime
-----+-----
1 1 | utilisée et modifiée -> à garder (priorité 0)
1 0 | utilisée mais non modifiée -> à garder si possible (priorité 2)
0 1 | non utilisée mais modifiée -> à garder si possible (priorité 1)
0 0 | non utilisée et non modifiée -> à supprimer (priorité 3)
```

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)
```

```
alloc_blocks(int fat[], int taille_fat, int n)
    resultat = 0
    taille_libre = 0
    pour i variant de 0 a taille_fat-1
        si (fat[i] == FAT_FREE) alors
            taille_libre++;
            si (taille_libre == n) renvoyer resultat;
        sinon-si (fat[i] == FAT_BAD) alors
            si (taille_libre == 0) resultat = (i + 1);
    sinon
        resultat = i+1;
        taille_libre = 0;
    fin-si
fin-pour
renvoyer -1
```

- 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)
```

```
void chainages(int fat[], int debut, int n)
    precedent = debut
    n--
    tant que (n > 0)
        debut++
        si (fat[debut] == FAT_FREE) alors
            fat[precedent] = debut
            precedent = debut
        n--
    fin-si
fin-tant que
fat[precedent] = FAT_EOF
```

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;
}
```

```
Exécution : ++++-----+-----+ ( + pour CPU et - pour E/S)
Temps de réponse = 18s
Taux d'utilisation de la CPU = 8S / 18s = 44%
```

- 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);
```

```
int main (void) {
    int finie = 0;
    DATA a = produireA(); /* 4 secondes de CPU */
    ecrire_asynchrone(a, &finie); /* E/S asynchrone de 6 secondes */
    DATA b = produireB(); /* 2 secondes de CPU */
    while (finie == 0) { /* attendre la fin de l'E/S */
        usleep(1000); /* ne pas consommer trop de CPU */
    }
    ecrire_asynchrone(b, &finie); /* E/S asynchrone de 4 secondes */
    finir(); /* 2 secondes de CPU */
    while (finie == 0) { /* attendre la fin de l'E/S */
        usleep(1000); /* ne pas consommer trop de CPU */
    }
    return 0;
}
```

```
trace CPU : ++++++...+.. (. pour l'attente)
trace E/S : ...----- (. pas d'E/S à faire)
Temps de réponse = 14s
Taux d'utilisation de la CPU = 8S / 14s = 57%
```

- 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 */

void ecrire_asynchrone(DATA d, int* finie) {
    *finie = 0;
    if (new_thread() == 0) {
        ecrire(d); /* le thread fils réalise l'E/S */
        *finie = 1;
        exit_thread();
    }
}
```

- 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);
```

```
SEMAPHORE ecrire_asynchrone(DATA d) {
    SEMAPHORE s = sem_create(0); /* Sémaphore de droit de passage */
    if (new_thread() == 0) {
        ecrire(d); /* le thread fils réalise l'E/S */
        sem_V(s); /* donner le droit de passage */
        exit_thread();
    }
    return s;
}
```

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

```
int main (void) {
    SEMAPHORE s;
    DATA a = produireA(); /* 4 secondes de CPU */
    s = ecrire_asynchrone_sem(a); /* E/S asynchrone de 6 secondes */
    DATA b = produireB(); /* 2 secondes de CPU */
    sem_P(s); /* attendre la fin de l'E/S */
    s = ecrire_asynchrone(b); /* E/S asynchrone de 4 secondes */
    finir(); /* 2 secondes de CPU */
    sem_P(s); /* attendre la fin de l'E/S */
    return 0;
}
```

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]

```
trace CPU      : ++++++.. (. pour l'attente)
trace disk 1   : ....----- (. pas d'E/S à faire)
trace disk 2   : .....----- (. pas d'E/S à faire)
Temps de réponse = 10s
Taux d'utilisation de la CPU = 8s / 10s = 80%
```

Les sujets plus anciens sont basés sur une version du cours assez différente (6 crédits, 60 heures et des chapitres en plus). De ce fait, les corrections ne sont pas disponibles.

Système d'exploitation

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

Responsable : Jean-Luc Massat

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.

Voilà 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

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

Responsable : Jean-Luc Massat

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 $\ll P(s); P(t); \dots; V(s); \dots; P(t); \dots; V(t); V(t); \dots; P(s); V(s); \gg$ 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 parallèle) ? 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 ressources
- 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  $i \in \{1, 2\}$  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  $i \in \{1, 2\}$  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 $\boxed{\text{RAID5}(D1, D2, D3, D4, D5, D6)}$) ? [1 point]

- b) Même question pour un *striping* (RAID0) des disques : $\boxed{\text{RAID0}(D1, D2, D3, D4, D5, D6)}$? [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 $\boxed{\text{RAID0}(\text{RAID5}(D1, D2, D3), \text{RAID5}(D4, D5, D6))}$? 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]

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)
```

Tournez la page SVP...

Système d'exploitation

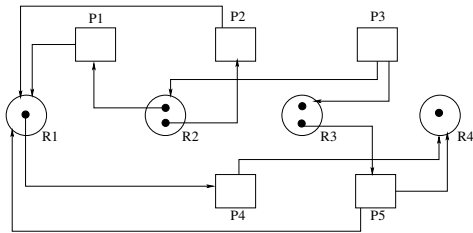
Troisième année de la licence d'informatique
UFR Sciences, site de Luminy, Aix-Marseille Université
11 mai 2016, première session,
durée 2 heures, calculatrices et documents autorisés.

Responsable : Jean-Luc Massat

1 Gestion de ressources

5 points

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



Dans ce cadre, répondez aux questions suivantes :

- Le système est-il *bloqué*? Justifiez votre réponse.
- 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 $\text{fat}[i] = \text{FAT_FREE}$. Si le bloc i est le dernier bloc d'un fichier, on a $\text{fat}[i] = \text{FAT_EOF}$. Toute autre valeur positive sur $\text{fat}[i]$ indique le bloc suivant.

- 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)
```

- 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)
```

- 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.

- 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 ();
```

- 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 ();
```

- 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);
```

- 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

Systeme 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]

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]

Systemes d'exploitation

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.

Responsable : Jean-Luc Massat

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 « $P(s); P(t); \dots; V(s); \dots; P(t); \dots; V(t); V(t); \dots; P(s); \dots; V(s);$ » 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 ressources
 - 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) 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]

Systemes 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).

Questions : (un point par question)

- Comment le système peut-il retirer de la mémoire à un processus et sur quels critères ?
- A quels moments le système s'exécute-t-il ? et à qui est « facturé » cette exécution ?
- Comment le système assure-t-il la sécurité des données stockées sur un disque ?
- Si le programme « $P(s); P(t); \dots; V(s); \dots; P(t); \dots; V(t); V(t)$ » est exécuté en plusieurs exemplaires, existe-t-il un risque d'interblocage ?
- 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 ?
- Les pseudo-partitions de répartition (RAID 0 ou *stripping*) améliorent-elles la sécurité ou l'efficacité ?
- 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 $\text{fat}[i] = 0$. 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 $\text{fat}[i] = j$. Si le bloc k est le dernier, alors $\text{fat}[k] = -1$.

```
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)
```

Tournez la page SVP...

Systemes 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 :

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

- 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)
```
- 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)
```
- 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)
```
- 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 renvoie 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.

Tournez la page SVP...