

Systemes d'exploitation

Licence d'informatique — Faculté de Luminy
28 mars 1997,
durée 3 heures,
Tous les documents sont autorisés.

1 Codage des fichiers sur disque

(8 points)

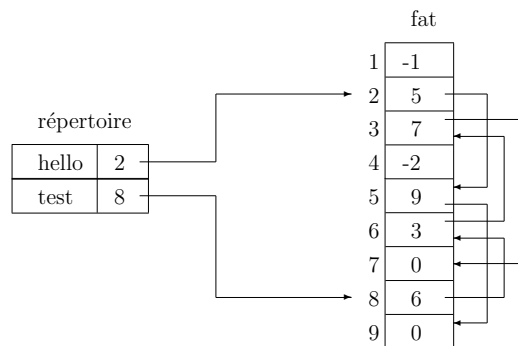
Le système MS-DOS représente l'état du disque au moyen d'une table d'entiers appelée la *FAT* pour *File Allocation Table*. Cette table est placée au début du disque. Chaque entrée de cette table est définie comme suit :

$$FAT[i] = \begin{cases} 0 & \text{si le bloc physique } i \text{ est le dernier du fichier auquel il appartient,} \\ n & \text{avec } n > 0, \text{ si le bloc physique } n \text{ suit le bloc } i \text{ dans le même fichier,} \\ -1 & \text{si le bloc physique } i \text{ est libre,} \\ -2 & \text{si le bloc physique } i \text{ est défectueux.} \end{cases}$$

Pour simplifier, partons du principe que la *FAT* et le répertoire du disque ont été amenés en mémoire centrale. On dispose donc des déclarations suivantes :

```
var rep : tableau [ 1 .. NB_FICHIERS ] de couples < nom : chaîne, adr : entier >  
fat : tableau [ 1 .. TAILLE_DISQUE ] d'entiers
```

Le schéma ci-dessous illustre le codage des fichiers au moyen de la *FAT*.



On dispose également de deux fonctions qui permettent la lecture ou l'écriture d'un bloc sur le disque :

```
lire_bloc(n : entier, var b : bloc) et ecrire_bloc(n : entier, b : bloc)
```

Dans ces conditions, traitez les questions suivantes :

- Donnez l'algorithme permettant de lire en accès direct un bloc dans un fichier défini par son nom : `sgf_lire_bloc(fichier : chaîne, nu : entier, var b : bloc) : entier` Cette routine renvoie 0 en cas de réussite, -1 si le fichier n'existe pas et -2 si le bloc n'existe pas.
- Donnez l'algorithme de la routine `sgf_ajouter_bloc(fichier : chaîne, b : bloc) : entier` qui ajoute un bloc à un fichier existant. Cette routine renvoie 0 en cas de réussite, -1 si le fichier n'existe pas et -2 si il n'y a plus de place disponible. Ne vous souciez pas de sauver sur le disque les modifications éventuelles de la *FAT* ou du répertoire.
- On désire maintenant allouer un espace contigu sur disque pour un fichier. Donnez l'algorithme de la routine `allocation_contigue(nbblocs : entier) : entier` qui alloue *nbblocs* blocs contigus en suivant la stratégie *best-fit*. Cette routine renvoie l'adresse du premier bloc en cas de réussite et -1 en cas d'échec.

d) On vous demande maintenant de définir un algorithme afin de vérifier la validité de la *FAT*. Cet algorithme doit dresser la liste

- des blocs occupés qui ne figurent dans aucun fichier,
- des blocs occupés qui apparaissent dans plusieurs fichiers,
- des blocs libres ou défectueux mais présents dans un fichier,
- des fichiers incorrects.

e) Discutez des avantages et des inconvénients liés à ce codage des fichiers sur le disque.

2 Choix de la victime

(4 points)

L'algorithme FINUFO choisit la page victime en se basant sur un bit d'utilisation qui est mis à jour automatiquement par le matériel. Cet algorithme n'utilise pas le bit `modif` et donc il ne choisit pas en priorité les pages qui n'ont pas été modifiées.

Proposez une modification de FINUFO pour choisir en priorité

- les pages propres et non utilisées,
- les pages sales et non utilisées,
- etc.

3 Synchronisation de processus

(8 points)

On se propose de tirer partie d'une machine multi-processeurs pour accélérer l'exécution d'un processus. Ce processus est composé de sept portions de code notées C_1 à C_7 . L'analyse de ces portions nous révèle les contraintes suivantes :

- l'exécution de C_3 doit se placer après C_5 et C_1 ,
- l'exécution de C_2 doit se placer après C_4 et C_7 ,
- l'exécution de C_4 doit se placer après C_5 , C_3 et C_6 ,
- l'exécution de C_7 doit se placer après C_1 et C_6 ,

- Donnez un ordre d'exécution séquentiel qui respecte les contraintes énoncées.
- Imaginons que dans un processus on puisse utiliser les notations suivantes :

```
cobegin P1; ... Pn; coend  
begin S1; ... Sm; end
```

pour indiquer au S.E. que les instructions P_1, \dots, P_n peuvent se dérouler en parallèle tandis que les instructions S_1, \dots, S_m 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 sept portions de code. Tout le parallélisme potentiel est-il utilisé ? (justifiez votre réponse).

- En utilisant des sémaphores, les `cobegin/coend` et les `begin/end`, pouvez vous reformuler l'algorithme suivi par le processus pour utiliser tout le parallélisme potentiel. N'oubliez pas de préciser, pour chaque sémaphore, la valeur initiale du compteur.
- Même question que précédemment si la contrainte 2 est modifiée comme suit :
 - l'exécution de C_2 doit se placer après C_1 et C_6 ou bien après C_5 .

Systemes d'exploitation

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

1 Utilisation des fichiers

(8 points)

Sur certains systemes d'exploitation, il est possible de « monter » des fichiers en memoire. Cette operation passe generalement par un appel systeme qui a la forme suivante :

```
void* mmap( char* fichier, int debut, int longueur);
```

Lorsqu'un processus utilisateur appelle cette routine systeme **tout se passe comme si** une partie du fichier (definie par debut et longueur) avait ete chargee en memoire a partir de l'adresse renvoyee par mmap. On peut, a partir de la, utiliser cette adresse pour exploiter le fichier en lecture **ou** en ecriture comme le montre l'exemple ci-dessous

```
{  
int k; char* c;  
c = mmap("mon_fichier", 100, 1024); /* monter le fichier */  
for(k = 0; k < 1024; k++) putchar(c[k]); /* afficher son contenu */  
for(k = 0; k < 1024; k++) c[k] = 'A'; /* modifier le contenu */  
ummap(c); /* demonter le fichier */  
}
```

Dans ce cadre, repondez aux questions suivantes :

a) Quels sont les modules du systeme d'exploitation qui sont touches par cette fonctionnalite ?

La fonction mmap() n'est presente que si le systeme d'exploitation est capable de gerer une memoire virtuelle paginee. **Indication** : Sur ce type de systeme, chaque entree de la table des pages virtuelles a la forme suivante

(presente, modif, protection, nu. page physique, information).
1 bit 1 bit 3 bits 20 bits 7 bits

Le champ information est libre et vous pouvez l'utiliser comme vous le voulez.

- b) Quelles sont les modifications operees par l'appel systeme mmap() pour preparer l'association entre un fichier et une portion de la memoire d'un processus ?
- c) Proposez un mecanisme permettant au systeme d'exploitation d'amener les informations du fichier en memoire centrale quand le processus en a besoin.
- d) Comment repercuter sur le fichier (stocke sur disque) les modifications effectuees en memoire centrale par le processus ?

2 Allocation contigue de la memoire

(6 points)

On considere la suite de demandes d'allocation (+) et de liberation (-) suivantes, dans un espace memoire 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 memoire est initialement vide, comment le S.E. realise les allocations en partant du principe qu'il applique la strategie de la premiere zone libre. On complete cette strategie par l'application d'un algorithme de tassage si aucune des zones libres ne convient.
- b) Meme question si le systeme applique la strategie du meilleur ajustement. Comparez ces deux strategies en utilisant le nombre de tassages effectues et la quantite de donnees deplacees.
- c) L'algorithme par subdivision est-il interessant sur cet exemple (justifiez votre reponse) ?

3 Synchronisation de processus

(6 points)

On se propose de tirer partie d'une machine multi-processeurs pour acclereler l'execution d'un processus. En etudiant ce processus, nous avons trouve la ligne suivante :

```
x = foo(2, gnu(gnat(zip(a,b),flup(c,d)) + flup(e,f))) + foo(zip(b,a),flup(d,c))
```

- a) En partant du principe que les fonctions zip et flup sont commutatives, decomposez et simplifiez cette ligne de code en utilisant autant de variables intermediaires que necessaire.
- b) Imaginons que dans un processus on puisse utiliser les notations suivantes :

```
cobegin P1; ... Pn; coend  
begin S1; ... Sm; end
```

pour indiquer au S.E. que les instructions P₁, ..., P_n peuvent se derouler en parallele tandis que les instructions S₁, ..., S_m doivent etre executees de maniere sequentielle. Une instruction **cobegin/coend** se termine quand toutes les instructions qui la composent sont terminees.

Utilisez ces notations pour repartir au mieux le calcul sur les processeurs disponibles.

- c) En utilisant des semaphores, les **cobegin/coend** et les **begin/end**, pouvez vous reformuler l'algorithme suivi par le processus pour utiliser tout le parallelisme potentiel. N'oubliez pas de preciser, pour chaque semaphore, la valeur initiale du compteur.
- d) L'analyse du code de gnu() et foo() nous revele que ces fonctions utilisent un espace global commun lors de leur execution. Quel est l'effet de cette constatation sur le code de la question c). Proposez une correction.

Systemes d'exploitation

Licence d'informatique — Faculté de Luminy
5 mai 1999,
durée 3 heures,
Tous les documents sont 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. Chaque entrée a la forme suivante

```
< id-processus, nu-page-virtuelle, protection, modif, util >
```

Une adresse virtuelle générée par le processeur a maintenant la forme suivante

```
< id-processus, nu-page-virtuelle, déplacement >
```

Dans ce cadre, répondez aux questions suivantes :

- A-t-on, du point de vue d'un processus, les mêmes capacités d'adressage mémoire que dans un système plus classique ?
- Décrivez, sous la forme d'un algorithme, le processus de transformation des adresses virtuelles en adresses physiques. Je vous rappelle qu'une adresse physique est un couple

```
< nu-page-physique, déplacement >.
```

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

```
defaut_de_page( a : adresse-virtuelle ).
```

- 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.
- Donnez l'algorithme suivi par 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 routines :

```
lire_page( p :id-processus, src :nu-page-virt, dest :nu-page-phys)  
écrire_page( p :id-processus, src :nu-page-phys, dest :nu-page-virt)
```

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

2 Synchronisation de processus

(6 points)

Certaines rues d'Amsterdam sont tellement étroites, qu'il n'y a pas assez de place pour faire passer deux voies de tramway. Pour régler ce problème, il existe un tronçon de voie unique comme le montre le schéma ci-dessous.



Pour étudier ce problème, nous allons simuler le comportement des tramways en utilisant un processus par véhicule. Nous utiliserons également des sémaphores pour régler les problèmes de synchronisation. Dans ce cadre, répondez aux questions suivantes :

- Donnez l'algorithme appliqué au passage du point *A* et du point *B* par les tramways qui assurent le trajet Sud-Nord. L'objectif est simple : il faut éviter les **collisions**.
- L'objectif est toujours le même, mais pour éviter l'apparition d'un bouchon nous changeons un peu la stratégie. Si il existe des tramways en attente des deux cotés, la voie unique doit être utilisée **alternativement** dans les deux sens. N'oubliez pas de préciser la valeur initiale des variables partagées et des sémaphores que vous utilisez. **Conseil** : utilisez une variable partagée qui représente l'état de la voie unique.
- Visiblement, l'activité alternative de la voie unique ne permet pas d'obtenir un rendement suffisant. Pour palier ce défaut, modifiez votre algorithme de façon à respecter l'alternance, mais tous les cinq passages.

3 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					Max					Dispo				
	R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4
P_0	0	0	1	2	P_0	0	0	1	2					
P_1	1	0	0	0	P_1	1	7	5	0					
P_2	1	3	5	4	P_2	2	3	5	6		1	5	2	0
P_3	0	6	3	2	P_3	0	6	5	2					
P_4	0	0	1	4	P_4	0	6	5	6					

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

- Donnez la matrice des besoins.
- Le système est-il dans un état sain ? justifiez votre réponse.
- Si le processus P_1 dépose la requête $(0, 4, 2, 0)$, cette requête est-elle acceptée immédiatement ? (justifiez votre réponse).
- A partir de la situation initiale, si le processus P_4 dépose la requête $(1, 4, 2, 0)$, quel est le résultat ?

Systemes d'exploitation

Licence d'informatique — Faculté de Luminy
11 mai 2000,
durée 3 heures,
Tous les documents sont autorisés.

1 Synchronisation de l'accès aux ressources

(7 points)

Partons du principe qu'il existe deux ressources R_1 et R_2 ayant les mêmes capacités mais de qualité différente (par exemple une imprimante rapide et une autre plus lente). On se propose de réaliser les fonctions permettant à un processus de demander l'allocation d'une ressource et sa libération. Pour gérer ces opérations, nous allons utiliser une version adaptée des sémaphores, les *bi-sémaphores* :

bi-sémaphore = \langle libre : tableau[1..2] de booléens; F1,F2 : Files de processus \rangle

libre[i] est vrai ssi la ressource R_i est libre. Avec un **bi-sémaphore**, l'utilisateur va être capable d'effectuer une demande générale (il a besoin de R_1 ou R_2), ou une demande particulière pour la ressource R_1 (la plus rapide). Les deux files sont là pour représenter les processus en attente sur chaque type de demande.

- Donnez la valeur initiale d'un **bi-sémaphore** pour le problème qui nous intéresse.
- Donnez l'algorithme suivi par la fonction d'allocation générale. Cette fonction bloque le processus appelant si les deux ressources sont occupées.

Fonction allouer_Ri(var s : **bi-sémaphore**) : **nu-de-ressource** ;

- Donnez l'algorithme suivi par la fonction d'allocation particulière. Cette fonction bloque le processus appelant si la ressource R_1 est occupée.

Procédure allouer_R1(var s : **bi-sémaphore**) ;

- Donnez l'algorithme suivi par la fonction de libération. Cette fonction favorise toujours les processus qui effectuent des demandes générales.

Procédure libérer(var s : **bi-sémaphore**; i : **nu-de-ressource**) ;

- Quels reproches peut-on faire à ce système d'allocation ? Comment pourrait-on l'améliorer ?

2 Lecture d'un fichier sur disque

(6 points)

On vous donne les éléments suivants : le bloc élémentaire d'entrée/sortie est fixé à 0,5 ko (512 octets), un numéro de bloc est codé sur quatre octets et les fichiers sont représentés sur disque à l'aide de tables d'allocation à un seul niveau.

- Si un processus effectue les lectures suivantes – dans cet ordre – dans un fichier déjà ouvert, quels sont les numéros des blocs **logiques** lus par le système ? (justifiez votre réponse)

1000 octets à partir de l'adresse 500 (de 500 à 1499) ;
11000 octets à partir de l'adresse 120000 (de 120000 à 130999) ;
3000 octets à partir de l'adresse 260000 (de 260000 à 262999) ;

- Si le système d'exploitation ne conserve aucun bloc en mémoire centrale entre les accès (hormis le descripteur du fichier ouvert), ces trois lectures dans un fichier nécessitent combien d'opérations élémentaires de lecture d'un **bloc physique** ? (précisez votre calcul pour chacun des trois cas)
- Même question que précédemment si le système conserve deux blocs en mémoire centrale entre les accès (précisez quels sont les blocs conservés).
- Sans changer le codage des fichiers sur disque, proposez une amélioration des accès aléatoires qui consomme peu de mémoire centrale (quelques centaines d'octets).

3 Mémoire segmentée

(7 points)

Soit un système basé sur une mémoire segmentée. La taille maximale de chaque segment est fixée à 128 ko (soit 2^{17} octets). La taille d'un segment est toujours multiple de 2^8 . La mémoire logique d'un processus est définie par la table des segments suivante :

Segment	Base	Taille
0	100	3
1	250	1
2	10	5
3	80	10
4	92	6

Dans ce cadre, traitez les questions suivantes :

- Pourquoi la taille des segments est-elle multiple de 256 ? Comment faut-il lire les valeurs présentes dans la table des segments ?
- Quelles sont les rapports entre la taille de la mémoire centrale et la taille (en nombre de bits) de chaque colonne de la table des segments ?
- Quelles sont les adresses physiques pour les adresses logiques suivantes ?

430, 131082, 263474, 393616, 524400

- Quel est l'impact de la mémoire segmentée, sur l'étape de compilation d'un programme ? Ou, en d'autres termes, comment le compilateur peut-il prendre en compte la segmentation pour générer un programme plus performant ?

Systemes d'exploitation

Licence d'informatique — Faculté de Luminy
4 mai 2001,
durée 3 heures,
Tous les documents sont autorisés.

1 Implantation physique des fichiers

(7 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 a la structure suivante :

- Les cases de 0 à 234 contiennent des informations diverses.
- Les cases de 235 à 244 contiennent respectivement les adresses physiques des blocs de données 0 à 9 du fichier en question.
- Les cases de 245 à 254 contiennent respectivement les adresses physiques des blocs d'index 0 à 9 du fichier en question.
- La case 255 contient l'adresse physique du dernier bloc d'index.

Chaque entrée d'un bloc index contient l'adresse physique d'un bloc de données sauf la dernière (n° 255) qui contient l'adresse du bloc d'index suivant et l'avant dernière (n° 254) qui contient l'adresse du bloc d'index précédent.

Questions :

- Quelle est la nature de ce codage ?
- Existe-t-il une limite pour la taille des fichiers avec cette organisation ?
- 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 préoccupez pas des problèmes de débordement ou des erreurs d'entrée/sortie.

- Si on cherche à lire 1000 entiers à partir de l'adresse 782800 (en entiers), combien de blocs physiques le système d'exploitation doit-il lire en partant du principe qu'aucun bloc n'est gardé en mémoire (à l'exception du descripteur) ?
- Comment améliorer les accès séquentiels avec cette architecture ?

2 Gestion des processus

(6 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 :

- Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ?
- 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.
- 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) ?
- 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 Mémoire segmentée et paginée

(7 points)

Soit un système basé sur une mémoire segmentée paginée. La taille des pages est fixée à 2^{10} octets. La taille maximale de chaque segment est fixée à 256 pages. La mémoire logique d'un processus est définie par la table des segments suivante :

N° de Segment	table des pages	Taille
0	1200	254
1	700	71
2	750	3

Voilà un extrait des tables de pages

adresse 1200 :

n°	p. phys.
0	100
1	250
⋮	⋮
253	80
254	-
255	-

adresse 700 :

n°	p. phys.
0	600
⋮	⋮
70	900
71	-
⋮	⋮

adresse 750 :

n°	p. phys.
0	1000
1	1100
2	1150
3	-
⋮	⋮

Questions :

- Expliquez le sens des valeurs 1200, 700 et 750 présentes dans la table des segments.
- Quelle est la taille totale de la mémoire occupée par ce processus ?
- Décrivez les composantes (taille et nature) d'une adresse logique segmentée et paginée dans un tel système.
- Quelles sont les adresses physiques pour les adresses logiques suivantes ?
262154, 2024, 335048, 525324, 259972
- Pourquoi avoir choisi la limite de 256 pages pour la taille des segments ? Quelles en sont les avantages ?

Systemes d'exploitation

Licence d'informatique — Faculté de Luminy
6 septembre 2001,
durée 3 heures,
Tous les documents sont autorisés.

1 Synchronisation de processus

(7 points)

On vous propose d'étudier le code de synchronisation suivant. Il utilise deux variables publiques (`disponible` et `tour`) et une variable privée (`ticket`).

```
<init>    disponible = 1
          tour = 1

<prologue> ticket = disponible
          disponible = disponible + 1
          répéter
          jusqu'à (ticket = tour)

<épilogue> tour = tour + 1
```

Questions :

- Ce code est sensé régler un défaut lié aux solutions d'attente active. Quel est ce défaut et comment est-il supprimé?
- Ce code est-il une solution correcte au problème de l'exclusion mutuelle? Expliquez votre réponse.
- Proposez une nouvelle version correcte basée sur le même principe. Vous pouvez utiliser une nouvelle variable publique et l'instruction XRM.

2 Mémoire segmentée et paginée

(8 points)

Soit un système basé sur une mémoire segmentée paginée. La taille des pages est fixée à 2^{10} octets. La taille maximale de chaque segment est fixée à 256 pages. La mémoire logique d'un processus est définie par la table des segments suivante :

N° de Segment	table des pages	Taille
0	1200	127
1	700	200
2	750	50
3	850	20
4	600	20

Voilà un extrait des tables de pages

adresse 1200 :

	n° p. phys.
0	100
⋮	⋮
125	80
126	81
⋮	⋮
255	⋮

adresse 700 :

	n° p. phys.
0	655
⋮	⋮
10	110
11	120
⋮	⋮
255	⋮

adresse 750 :

	n° p. phys.
0	210
1	215
2	220
⋮	⋮
⋮	⋮
255	⋮

Questions :

- Expliquez le sens des valeurs 1200, 700 et 750 présentes dans la table des segments.
- Quelle est la taille totale de la mémoire occupée par ce processus?
- Décrivez les composantes (taille et nature) d'une adresse logique segmentée et paginée dans un tel système.
- Quelles sont les adresses physiques pour les adresses logiques suivantes?
 $129124, 808970, 272404, 200, 526546$
- Pourquoi avoir choisi la limite de 256 pages pour la taille des segments? Quelles en sont les avantages?
- Est-il possible de voir apparaître plusieurs fois la même valeur dans les tables de pages? Que signifie cette répétition?

3 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					Max					Dispo				
	R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4
P_0	1	0	0	2	P_0	1	0	0	2					
P_1	0	0	1	0	P_1	5	7	1	0					
P_2	5	3	1	4	P_2	5	3	2	6		2	5	1	0
P_3	3	6	0	2	P_3	5	6	0	2					
P_4	1	0	0	4	P_4	5	6	0	6					

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

- Donnez la matrice des besoins.
- Le système est-il dans un état sain? Justifiez votre réponse.
- Si le processus P_1 dépose la requête $(2, 4, 0, 0)$, cette requête est-elle acceptée immédiatement? (justifiez votre réponse).
- A partir de la situation initiale, si le processus P_4 dépose la requête $(2, 4, 1, 0)$, quel est le résultat?

Systemes 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 :

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

- Quels sont les états possibles de la ressource R_1 (au sens du graphe d'allocation de ressources) ?
- Quels sont les états possibles du processus P_1 indépendamment de P_2 ?
- 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) ?
- Si on applique l'algorithme des banquiers, combien reste-t-il d'états possibles pour le système d'allocation de ressources ?
- 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 :

- Sur ces processeurs il existe une instruction

```
ajouter_en_mémoire_associative <npv,npp>
```

Que fait, d'après vous, cette instruction ?

- Même si le processeur n'utilise plus la table des pages virtuelles, cette structure de données est-elle toujours utile (justifiez votre réponse) ?
- Donnez les grandes lignes du traitant associé à l'interruption « défaut de mémoire associative ».
- Quels sont les avantages et les inconvénients de cette implantation de la pagination par rapport à la formule classique ?
- Cette implantation pose un problème dans le choix d'une victime. Décrivez le problème et proposez une solution.

Systemes d'exploitation

Licence d'informatique — Faculté des Sciences de Luminy
19 Juin 2003,
durée 3 heures,
Tous les documents sont autorisés.

1 Parallélisme et utilisation des « threads » (10 points)

Vous venez d'acquérir une nouvelle machine dotée de quatre 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 */
```

Cette fonction 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, -1 en cas d'erreur et le numéro du fils (un entier positif) chez le père.

Questions :

- 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.
- 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 ?
- Le programme B est composé de quatre parties. Le code B_2 a besoin du résultat de B_1 , tandis que B_4 a besoin de B_2 ou B_3 . Proposez une version parallèle et éventuellement montrez les défauts de votre solution.
- La nouvelle version du système vous offre les trois fonctions ci-dessous. Dans ce cadre, répondez à la question c).

```
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 modifier la réponse à la question précédente pour être sûr que le « thread » initial est bien le dernier à terminer son exécution (utilisez les sémaphores).

2 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 :

- Montrez qu'il existe un risque d'interblocage et proposez une solution simple à ce problème.
- Pour éviter les interblocages, les philosophes décident d'appliquer l'algorithme des banquiers. Donnez la matrice `Max` et le vecteur `Dispo`.
- Pour **ce problème bien particulier** donnez l'algorithme simplifié permettant de déterminer si oui ou non, le système est dans un état sain.
- Sans vous préoccuper des problèmes de parallélisme et de synchronisation, donnez l'algorithme appliqué par le i ème philosophe avant de manger.
- 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 ?
- Donnez la nouvelle version de la matrice `Max` et du vecteur `Dispo`.

3 Mémoire virtuelle paginée (4 points)

Soit un système basé sur une mémoire virtuelle paginée. La taille des pages est fixée à 2^{10} octets. Dans un programme C on déclare la tableau ci-dessous. Pour l'exécution de ce programme, le système a réservé 8 pages physiques qui sont initialement vides.

```
char tableau[32][2048];
```

Questions :

- Dans ce cadre, combien de défauts de page sont générés par le code suivant (on part du principe que les indices sont dans des registres) :

```
for(i=0; i<32; i++) for(j=0; j<2048; j++) tableau[i][j] = 'A';
```


Justifiez votre réponse. Quel est l'impact d'un doublement de la mémoire physique (8 à 16) ?
- Même question si le code devient (la mémoire physique est de 16 pages) :

```
for(j=0; j<2048; j++) for(i=0; i<32; i++) tableau[i][j] = 'A';
```
- De combien faut-il augmenter le nombre de pages physiques pour que le nombre de défaut de page baisse de manière significative ?

Systemes 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 :

- 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.
- 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.
- 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.
- 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).
- 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 :

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

- Montrez qu'il existe un risque d'interblocage et proposez une solution simple à ce problème.
- Pour éviter les interblocages, les philosophes décident d'appliquer l'algorithme des banquiers. Donnez la matrice Max et le vecteur $Dispo$.
- Pour **ce problème bien particulier** donnez l'algorithme simplifié permettant de déterminer si oui ou non, le système est dans un état sain.
- Sans vous préoccuper des problèmes de parallélisme et de synchronisation, donnez l'algorithme appliqué par le i ème philosophe avant de manger.
- 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 ?
- Donnez la nouvelle version de la matrice Max et du vecteur $Dispo$.

Systemes 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 :

- 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)
- Si le nombre de pages physiques passe à quatre, quel est le nombre de défauts de page pour l'algorithme LRU (2 points) ?
- Quel est le meilleur compromis pour la taille de la mémoire physique (1 point) ?
- 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, \dots, 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 :

- Combien de processus exécutent leur section critique simultanément (justifiez votre réponse) ?
- 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 ?
- 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 :

- Donnez une explication de ce codage (2 points).
- 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)
```

Systemes d'exploitation

Responsable : Jean-Luc Massat

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.

1 Synchronisation de processus

(9 points)

On se propose de tirer parti d'une machine multi-processeurs pour accélérer 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 $P_1; \dots P_n$; **coend**
begin $S_1; \dots S_m$; **end**

pour indiquer au système que les instructions P_i peuvent se dérouler en parallèle tandis que les instructions S_j 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					Max				
	R_1	R_2	R_3	R_4		R_1	R_2	R_3	R_4
P_0	0	0	1	2	P_0	0	0	1	2
P_1	1	0	0	0	P_1	1	7	5	0
P_2	1	3	5	4	P_2	2	3	5	6
P_3	0	6	3	2	P_3	0	6	5	2
P_4	0	0	1	4	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? Évaluez 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.

Systemes d'exploitation

Responsable : Jean-Luc Massat

Troisième année de la licence d'informatique
U.F.R. Sciences de Luminy
Université de la Méditerranée
16 mai 2007,
durée 3 heures,
calculatrices et documents autorisés.

1 Parallélisme et utilisation des « threads » (10 points)

Vous venez d'acquérir une nouvelle machine dotée de quatre 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 */
```

Cette fonction 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 :

- 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.
- 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 ?
- Le programme B est composé de quatre parties. Le code B_2 a besoin du résultat de B_1 , tandis que B_4 a besoin de B_2 ou B_3 . Proposez une version parallèle et éventuellement montrez les défauts de votre solution.
- La nouvelle version du système vous offre les trois fonctions ci-dessous. Dans ce cadre, répondez à la question c).

```
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 modifier la réponse à la question précédente pour être sûr que le « thread » initial est bien le dernier à terminer son exécution (utilisez les sémaphores).

2 Gestion d'une mémoire virtuelle (6 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 :

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

3 Mémoire virtuelle paginée (4 points)

Soit un système basé sur une mémoire virtuelle paginée. La taille des pages est fixée à 2^{10} octets. Dans un programme C on déclare la tableau ci-dessous. Pour l'exécution de ce programme, le système a réservé 8 pages physiques qui sont initialement vides.

```
char tableau[32][2048];
```

Questions :

- Dans ce cadre, combien de défauts de page sont générés par le code suivant (on part du principe que les indices sont dans des registres) :

```
for(i=0; i<32; i++) for(j=0; j<2048; j++) tableau[i][j] = 'A';
```

Justifiez votre réponse. Quel est l'impact d'un doublement de la mémoire physique (8 à 16) ?

- Même question si le code devient (la mémoire physique est de 16 pages) :

```
for(j=0; j<2048; j++) for(i=0; i<32; i++) tableau[i][j] = 'A';
```

- De combien faut-il augmenter le nombre de pages physiques pour que le nombre de défaut de page baisse de manière significative ?

Systemes d'exploitation

Responsable : Jean-Luc Massat

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.

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 $\text{fat}[i] = 0$. 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 $\text{fat}[i] = j$. Si le bloc k est le dernier, alors $\text{fat}[k] = -1$.

```
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 maximum 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.

Systemes 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-t-on, du point de vue d'un processus, les mêmes capacités d'adressage mémoire que dans un système plus classique ?
- 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 default_de_page(struct ADRV a);
```

- 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.
- Écrivez la fonction default_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);
```

- 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 :

- Quel est le temps de réponse de ce processus et le taux d'utilisation de la CPU sur la période considérée ? [1 point]
- 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]
- 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 :		Adresse 400 :		Adresse 600 :		Adresse 900 :	
	n° p. phys.		n° p. phys.		n° p. phys.		n° p. phys.
0	950	:	:	0	180	:	:
:	:	50	800	:	:	:	:
:	:	51	450	10	150	:	:
100	500	52	350	11	170	:	:
101	190	:	:	12	900	200	200
102	100	:	:	:	:	201	700
:	:	:	:	120	400	202	300
:	:	:	:	:	:	:	:
255	550	:	:	121	401	:	:

Questions :

- Donnez la composition d'une adresse paginée dans ce système (nature et taille en nombre de bits de chaque composante) ? [1 point]
- Quelle est la taille maximum de la mémoire alloué à un processus ? Comment procéder pour augmenter cette limite ? [2 points]
- Quelle est la taille de la mémoire allouée au processus courant. [1 point]
- Donnez les adresses physiques des adresses paginées 2621440, 2726554, 3351702 et 31511528. [2 points]

N'oubliez pas de justifier vos réponses.