Université Bordeaux I
Session de janvier 1999

U.F.R. Mathématiques et Informatique
Alain Griffault

Maîtrise d'Informatique
Systèmes d'exploitation
(Durée : 2 heures)
SUJET + CORRIGE

Documents interdits

Avertissement

Les exercices sont indépendants
L'ensemble est long, la notation en tiendra compte. (Le barème est indicatif de l'importance respective des exercices).

1 Cours : 2 points

Question 1.1 *Donnez les définitions des fragmentations interne et externe. Pour chacune, donnez un moyen pour l'améliorer, et la raison pour laquelle votre solution n'est pas retenue.*

Réponse à la question 1.1(2 points)

interne $F_i = \sum (tailles\ processus) / \sum (tailles\ allouees)$

La fragmentation interne existe dès lors que l'unité d'allocation de la mémoire diffère de l'unité minimale (ie l'octet).

Statistiquement $F_i = (N \times T)/2$ ou N est le nombre de processus en mémoire, et T la taille d'un bloc (une page). F_i est donc une fonction linéaire de la taille du bloc, il suffit donc de diminuer la taille des blocs pour diminuer la fragmentation interne.

Diminuer la taille des blocs va engendrer d'une part des sur-coûts de gestion des blocs; d'autre part un ralentissement des performances d'entrées-sorties.

Une fois de plus, il faut trouver le bon compromis.

externe $F_e = \sum (tailles\ trous) / (taille\ memoire)$

La fragmentation externe existe lorsque les allocations de mémoire sont contigües.

F_e dépend de l'algorithme d'allocations des pages (First Fit, Best Fit, Worst Fit, ...). Nous avons vu qu'il n'existe pas d'algorithme optimal pour ce problème, mais seulement des heuristiques.

Une amélioration possible est le compactage après chaque libération d'espace suite à la terminaison d'un processus.

Cette technique d'une part est coûteuse en temps; d'autre part nécessite que tous les codes soient relogeables, ce qui n'est pas toujours le cas.

Une autre amélioration possible (et souvent retenue) est l'utilisation d'allocations non contigües : pagination et/ou segmentation.

---

2 Interblocage : 4 points

Imaginez la politique suivante d'allocation de ressources. On autorise des requêtes et des libérations de ressources à tout moment. Si une requête ne peut pas être satisfaite parce que les ressources ne sont pas disponibles, nous examinons tous les processus bloqués en attente de ressources. S'ils possèdent les ressources désirées, on leur enlève donc ces ressources et on les donne au processus demandeur. On incrémente le vecteur des ressources pour lesquelles le processus bloqué attend afin d'inclure celles qu'on lui a enlevé.

Par exemple, considérez un système avec trois types de ressources et le vecteur `Disponible` initialisé à (4,2,2). Si le processus P_0 demande (2,2,1), il les obtient. Si P_1 demande (1,0,1), il les obtient. Ensuite, si P_0 demande (0,0,1), il est bloqué (ressource non disponible). Si maintenant P_2 demande (2,0,0), il obtient la ressource disponible (1,0,0) et celle allouée à P_0 (puisque P_0 est bloqué). Le vecteur `Tenu` de P_0 est décrémenté à (1,2,1) et son vecteur `Requête` incrémenté à (1,0,1).

Question 2.1 *Un interblocage peut-il se produire ? Si c'est le cas, donnez un exemple. Sinon, quelle condition nécessaire ne peut pas être satisfaite ?*

Réponse à la question 2.1(2 points) Supposons un interblocage. Cela signifie qu'il existe un ensemble $\{P_i\}$ de processus formant un cycle. Soit P_d le (dernier) processus bloqué du fait de sa requête qui ne peut être satisfaite, il demande donc une ressource qui vérifie :

1. aucun processus bloqué n'en détient. (sinon il y aurait réquisition).
2. plus aucune n'est disponible.

Tous les exemplaires de cette ressource sont donc détenus par les processus non bloqués $\{Q_j\}$, deux cas possibles :

1. le processus Q_r qui détient la ressource termine en libérant la ressource. Cela n'est pas précisé dans le texte, mais il est fortement probable que lors d'une libération de ressources, les requêtes bloquantes soient réexaminées. Il n'y a pas d'interblocage dans cette situation.
2. le processus Q_r émet une nouvelle requête qui le bloque. Mais Q_r ne peut pas être interbloqué, car cela contredirait l'hypothèse que P_d est le **dernier**. Q_r terminera donc son exécution en libérant fatalement la ressource pour P_d. Il n'y a donc pas d'interblocage dans cette situation.

Pour résumer, la réquisition rend l'interblocage impossible.

Question 2.2 *Le blocage indéfini (la famine) peut-il se produire ? Si c'est le cas, donnez un exemple. Sinon, expliquez brièvement pourquoi ?*

Réponse à la question 2.2(2 points) Il est possible que plusieurs processus se liguent afin d'affamer indéfiniment un tiers.

Soit un système comportant trois ressources en un seul exemplaire, et trois processus : P1 désire (0,1,1), P2 désire (1,0,1) et P3 (1,1,0).

	Processus	Requête	Etats			Réquisition	Requêtes effectives			Disponibles
			P1	P2	P3		P1	P2	P3	
			.	.	.					(1,1,1)
1	P3	(1,0,0)	.	.	R					(0,1,1)
2	P1	(0,1,0)	R	.	R					(0,0,1)
3	P3	(0,1,0)	R	.	S				(0,1,0)	(0,0,1)
4	P1	(0,0,1)	R	.	S				(0,1,0)	(0,0,0)
5	P2	(1,0,0)	R	R	S	P3			(1,1,0)	(0,0,0)
6	P1	libère	.	R	S				(1,0,0)	(0,0,1)
7	P2	(0,0,1)	.	R	S				(1,0,0)	(0,0,0)
8	P1	(0,1,0)	R	R	S	P3			(1,1,0)	(0,0,0)
9	P2	libère	R	.	S				(0,1,0)	(0,0,1)
4+6 etc	P1	(0,0,1)	R	.	S				(0,1,0)	(0,0,0)

Sur cet exemple, les processus P1 et P2 interdisent au processus P3 d'avoir simultanément les deux ressources qu'il désire. Il faut noter qu'il n'y a aucune hypothèse supplémentaire par rapport au sujet. Cette situation n'est pas du fait d'algorithmes particuliers, mais uniquement de la politique d'allocation des ressources.

---

3 Algorithmes de remplacements de pages : 4 points

Question 3.1 *Construisez, pour une mémoire comportant 3 cadres initialement vides, des suites de références les plus simples (courtes) possibles pour lesquelles :*

1. *le nombre de défauts de pages est plus petit pour l'algorithme FIFO que pour LRU.*
2. *le nombre de défauts de pages est plus petit pour l'algorithme LRU que pour FIFO.*

Réponse à la question 3.1(2 points) Soit une mémoire comportant 3 cadres initialement vides, et la suite de références: 0 1 2 0 3.

FIFO

références	0	1	2	0	3
cadre 1	0	0	0	0	3
cadre 2		1	1	1	1
cadre 3			2	2	2
défauts	p	p	p		p

LRU

références	0	1	2	0	3
cadre 1	0	0	0	0	0
cadre 2		1	1	1	3
cadre 3			2	2	2
défauts	p	p	p		p

Ces cinq références provoquent le même nombre de défauts de pages, mais laisse un contexte différent. Il est alors facile de répondre aux deux questions.

1. le nombre de défauts de pages est plus petit pour l'algorithme FIFO que pour LRU en ajoutant la référence 1.

FIFO

références	0	1	2	0	3	1
cadre 1	0	0	0	0	3	3
cadre 2		1	1	1	1	1
cadre 3			2	2	2	2
défauts	p	p	p		p	

LRU

références	0	1	2	0	3	1
cadre 1	0	0	0	0	0	0
cadre 2		1	1	1	3	3
cadre 3			2	2	2	1
défauts	p	p	p		p	p

2. le nombre de défauts de pages est plus petit pour l'algorithme LRU que pour FIFO en ajoutant la référence 0.

FIFO

références	0	1	2	0	3	0
cadre 1	0	0	0	0	3	3
cadre 2		1	1	1	1	0
cadre 3			2	2	2	2
défauts	p	p	p		p	p

LRU

références	0	1	2	0	3	0
cadre 1	0	0	0	0	0	0
cadre 2		1	1	1	3	3
cadre 3			2	2	2	2
défauts	p	p	p		p	

Question 3.2 *Construisez une suite de références la plus simple possible qui met en évidence l'anomalie de Belady :* **pour l'algorithme FIFO, l'augmentation de la mémoire peut augmenter le nombre de défauts de pages.** *Vous pourrez chercher à continuer la suite* **0 1 2 3 0 1 4** *sur une mémoire constituée de 3 pages, puis de 4 pages.*

Réponse à la question 3.2(2 points)

FIFO

références	0	1	2	3	0	1	4	0	1	2	3	4
cadre 1	0	0	0	3	3	3	4	4	4	4	4	4
cadre 2		1	1	1	0	0	0	0	0	2	2	2
cadre 3			2	2	2	1	1	1	1	1	3	3
défauts	p	p	p	p	p	p	p			p	p	

FIFO

références	0	1	2	3	0	1	4	0	1	2	3	4
cadre 1	0	0	0	0	0	0	4	4	4	4	3	3
cadre 2		1	1	1	1	1	1	0	0	0	0	4
cadre 3			2	2	2	2	2	2	1	1	1	1
cadre 4				3	3	3	3	3	3	2	2	2
défauts	p	p	p	p			p	p	p	p	p	p

---

4 Synchronisation avec des sémaphores : 5 points

On dispose de processus `a, b, c` et `init` qui sont lancés au même instant. Le but est de contrôler l'ordonnancement des actions des processus `a, b` et `c`. Pour cela on dispose d'un langage de spécification comportant les opérations de concaténation séquentielle `'*'` et `'.'`, de choix exclusif `'+'` et de mise en parallèle `'||'`. Les processus `a, b` et `c` exécutent le code suivant :

```
Processus x() /* les  codes sont différents pour x = a, b ou c */
debut
   boucle { avantx ; actionsx ; apresx ; }
fin           /* actionsx n'est pas forcément atomique */
```

Remarquons que si `avantx` et `apresx` sont vides pour un processus donné, celui-ci exécute indéfiniment `actionsx` , et ce de façon séquentielle mais parallèlement aux autres processus.

Question 4.1 *Donnez, pour les spécifications suivantes (où x représente* `actionsx` *) le code (qui est toujours une séquence de* `wait()` *et/ou de* `signal()`*) du processus* `init`*, et de* `avantx` *et* `apresx` *pour chaque processus* `a, b` *et* `c`*.*

1. `(a.b.c)* soit : a . b . c . a . b . c .` *etc*
2. `(a+b+c)* exemple : a . b . c . a . a . c . c . b` *etc*
3. `(a||b||c)* soit : a||b||c . a||b||c . a||b||c .` *etc*

Rappels :

```
type semaphore = (n : entier; l : liste de processus);
```

manipulée uniquement par les actions atomiques `wait` (≡ P) et `signal` (≡ V).

```
procedure wait(var S : semaphore)
debut
S.n := S.n - 1;
si (S.n < 0) alors debut
        ajouter(moi,S.l);
        block;
        fin
fin
```

```
procedure signal(var S : semaphore)
debut
S.n := S.n + 1;
si (S.n <= 0) alors debut
        enlever(P,S.l);
        wakeup(P);
        fin
fin
```

Réponse à la question 4.1(1,5 + 1,5 + 2 points)

1. Langage de spécification : `(a.b.c)*`
   Noms des sémaphores : A, B et C
   Code du processus init : V(A)

	a	b	c
avantx	P(A)	P(B)	P(C)
apresx	V(B)	V(C)	V(A)

2. Langage de spécification : `(a+b+c)*`
   Noms des sémaphores : D
   Code du processus init : V(D)

	a	b	c
avantx	P(D)	P(D)	P(D)
apresx	V(D)	V(D)	V(D)

3. Langage de spécification : `(a||b||c)*`
   Noms des sémaphores : A, B et C
   Code du processus init : V(A)V(A)V(B)V(B)V(C)V(C)

	a	b	c
avantx	P(A)P(A)	P(B)P(B)	P(C)P(C)
apresx	V(B)V(C)	V(C)V(A)	V(A)V(B)

5 Exclusion mutuelle répartie : 5 points

Les horloges logiques définissent un temps $C(e)$ pour chaque événement e tel que :

1. Si a est avant b pour un processus X, alors $C(a) < C(b)$.
2. Si un processus X envoie un message M à un processus Y, alors $C(envoi(M)) < C(reception(M))$.

L'algorithme de Lamport permet l'obtention d'horloges logiques.

1. Pour chaque processus X incrémenter localement $C_X()$ pour chaque nouvel événement e.
2. Pour chaque processus X, pour chaque envoi de message M, ajouter l'estampille $C_X()$.
3. Lorsqu'un processus Y reçoit un message (M, E) ou M est un message et E une estampille, alors faire $C_Y() = max(C_Y(), E) + 1$

Rappel : Une relation R d'ordre strict (anti-réflexive, anti-symétrique et transitive) est :

totale ssi $\forall a, b$: $(a, b) \in R$ *ou* $(b, a) \in R$.

partielle ssi $\exists a, b$ *tel que* : $(a, b) \notin R$ *et* $(b, a) \notin R$.

Soit $X = X_1, ..., X_N$ un ensemble de machine, soit $P = \{P_{i_j}\}$ un ensemble de processus (P_{i_j} s'exécute sur X_i) et soit $E = \{e^k_{i_j}\}$ un ensemble d'événements ($e^k_{i_j}$ est un événement du processus P_{i_j}).

Question 5.1 *Montrez que la relation R définie par : $((X_p, P_{p_j}, e^k_{p_j}), (X_q, P_{q_j}, e^k_{q_j})) \in R$ ssi $C_{P_{p_j}}(e^k_{p_j}) < C_{P_{q_j}}(e^k_{q_j})$ défini un ordre partiel sur $X \times P \times E$.*

Réponse à la question 5.1(1 points) La relation $<$ sur les entiers défini un ordre strict, il suffit de montrer que R est partielle, c'est à dire que deux événements peuvent avoir la même valeur d'horloge associée.
Considérons la suite d'événements suivante pour deux processus P_1 et P_2.

P_1		P_2	
événement	C_{P_1}	événement	C_{P_2}
envoi(M)	to		t1
	to	réception(M)	max(to,t1)+1
	to	envoi(M')	max(to,t1)+2
réception(M')	max(to,t1)+3		max(to,t1)+2
		a	max(to,t1)+3

Les deux événements réception(M')$_{P_1}$ et a$_{P_2}$ ont la même estampille et sont donc incomparable par la relation R.
La relation R est donc partielle.

Question 5.2 *Donnez une relation R' sur $X \times P \times E$ qui soit totale et qui respecte les horloges logiques.*

Réponse à la question 5.2(1 points) Intuitivement, il faut donner un moyen pour ordonner deux estampilles égales. Une solution consiste à comparer les "numéros" de processus, et s'ils sont identiques, les numéros de machines. Ainsi, il semble évident que la relation R' suivante soit totale.

$$((X_{p1}, P_{p2_{j2}}, e^{k3}_{p3_{j3}}), (X_{p4}, P_{p5_{j5}}, e^{k6}_{p6_{j6}})) \in R' \Leftrightarrow \begin{cases} C_{P_{p3_{j3}}}(e^{k3}_{p3_{j3}}) < C_{P_{p6_{j6}}}(e^{k6}_{p6_{j6}}) \\ \vee \\ C_{P_{p3_{j3}}}(e^{k3}_{p3_{j3}}) = C_{P_{p6_{j6}}}(e^{k6}_{p6_{j6}}) \wedge p2_{j2} < p5_{j5} \\ \vee \\ C_{P_{p3_{j3}}}(e^{k3}_{p3_{j3}}) = C_{P_{p6_{j6}}}(e^{k6}_{p6_{j6}}) \wedge p2_{j2} = p5_{j5} \wedge p1 < p4 \end{cases}$$

Cette relation respecte les horloges logiques au sens suivant :
$C(a) < C(b) \Rightarrow ((X_{p1}, P_{p2_{j2}}, a), (X_{p4}, P_{p5_{j5}}, b)) \in R'$
Cela signifie que toutes les machines et tous les processus "voient" toujours deux événements ayant des estampilles différentes dans le même ordre.

Soit l'algorithme réparti suivant pour le problème de l'exclusion mutuelle.

1. Si un processus P d'une machine X désire entrer en section critique, il envoie un message $< demande, C_P() >$ ($C_P()$ est l'horloge locale de P) à tous les processus de toutes les machines qui sont également concernés par cette section critique, puis il attend l'autorisation de chacun.

2. Lorsqu'un processus Q reçoit un message $< demande, E >$ de la part d'un processus P, il réagit ainsi :

   Non concerné il renvoie un message $< OK, C_Q() >$ à P.

   En section critique il attend sa sortie, puis envoie un message $< OK, C_Q() >$ à P.

   Concerné Si $(E, C_Q()) \in R$ alors il renvoie un message $< OK, C_Q() >$ à P. Sinon rien.

Question 5.3 *L'algorithme est-il correct si la relation R est celle définie par la relation d'ordre partielle sur $X \times P \times E$? Si oui, expliquez pourquoi, sinon donnez la propriété non satisfaite et un exemple de telle situation.*

Réponse à la question 5.3(1,5 points) L'algorithme n'est pas correct.

Il y a un risque de famine généralisée (qui se traduit par de l'interblocage dans ce cas).

Considérons deux processus et la séquence d'événements suivante :

P_1		P_2	
événement	$C_{P_1}()$	événement	$C_{P_2}()$
envoi$< demande, 0 >$	0		0
	0	envoi$< demande, 0 >$	0
reception$< demande, 0 >$	1		0
		reception$< demande, 0 >$	1

Dans cette situation P_1 (resp P_2) n'envoie pas de message $< OK, C_{P_1}() >$ à P_2 (resp $< OK, C_{P_2}() >$ à P_1) car $((P_2, 0), (P_1, 0)) \notin R$ (resp $((P_1, 0), (P_2, 0)) \notin R$).

Les deux processus s'attendent donc mutuellement.

Question 5.4 *L'algorithme est-il correct si la relation R est la votre qui définie une relation d'ordre totale sur $X \times P \times E$? Si oui, expliquez pourquoi, sinon donnez la propriété non satisfaite et un exemple de telle situation.*

Réponse à la question 5.4(1,5 points) L'algorithme est correct avec la relation R'.

En effet, R' étant totale, dans tous les conflits, toutes les demandes seront ordonnées de la même manière par tous les processus.

Ainsi seul le processus ayant émis la demande avec la plus petite estampille ne donnera pas son accord, il recevra obligatoirement l'accord de tous les autres, et pourra ainsi entrer en section critique.