Corrigé

1 Cours

1. Dans le système Unix, le partage des fichiers est obtenu grâce à l'utilisation d'une structure de graphes acycliques pour les répertoires. La destruction des fichiers dans une telle structure est plus délicate, car elle ne doit être effective que si tous les *propriétaires* sont d'accord. L'attribut *compteur de référence* contenu dans l'I-node d'un fichier Unix permet de comptabiliser les références à un fichier. Toute nouvelle référence incrémente le compteur, toute destruction le décrémente. La destruction est effective lorsque le compteur est à zéro.

```
NAME
     ln - make links

SYNOPSIS
     ln [ -s ] sourcename [ targetname ]
     ln [ -s ] sourcename1 sourcename2 [ sourcename3 ... ] targetdirectory

DESCRIPTION
     A link is a directory entry referring to a file; the same
     file (together with its size, all its protection informa-
     tion, etc.) may have several links to it.  There are two
     kinds of links: hard links and symbolic links.

     By default ln makes hard links.  A hard link to a file is
     indistinguishable from the original directory entry; any
     changes to a file are effective independent of the name used
     to reference the file.  Hard links may not span file systems
     and may not refer to directories.
```

   Les liens physiques incrémentent donc le compteur de références. Ils sont interdits sur les répertoires, ce qui garantit la structure acyclique du graphe des répertoires.

2. Dans le système Unix, un processus qui attend la terminaison d'une entrée/sortie sur disque est plus prioritaire qu'un processus qui attend qu'un tampon soit disponible, car un processus qui termine un transfert a des chances de libérer un tampon.

2 Algorithmes de remplacements de pages

1. Soit une mémoire comportant 3 cadres initialement vides, et la suite de références suivante : 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	1
cadre 3		2	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	1	3
cadre 3		2	2	2	2
défauts	p	p	p		p

   Ces cinq références provoquent le même nombre de défauts de pages, mais avec des cadres différents. Il est alors facile de répondre aux deux questions.

   (a) 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	1
cadre 3		2	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	1	3	3
cadre 3		2	2	2	2	1
défauts	p	p	p		p	p

(b) 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	1	0
cadre 3		2	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	1	3	3
cadre 3		2	2	2	2	2
défauts	p	p	p		p	

2. Construire 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.

FIFO, 3 cadres

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, 4 cadres

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

3 Synchronisation de processus

1. Les fourchettes sont représentées par des sémaphores.

`fourchette : tableau[0..(N-1)] de sémaphores initialisés à 1 ;`

(a) Quelques commentaires pour expliquer l'utilsation des sémaphores.

```
tant que vrai faire
     pense;
     P(fourchette[i]);                Le philosophe i obtient la fourchette i (droite),
                                      ou bien est mis dans la file d'attente de fourchette[i].
     P(fourchette[(i+1)modulo N]);    idem pour la fourchette gauche.
     mange;
     V(fourchette[i]);                Le philosophe i libere la fourchette i (droite), et,
                                      éventuellement débloque le philosophe i-1.
     V(fourchette[(i+1)modulo N]);    Le philosophe i libere la fourchette i-1 (gauche), et,
                                      éventuellement débloque le philosophe i+1.
     fait
```

(b) Le comportemnt suivant pour ($N = 4$) conduit à un blocage, et à la mort de tous les philosophes.

Philo 0	Philo 1	Philo 2	Philo 3	four. 0	four. 1	four. 2	four. 3
pense	pense	pense	pense	(1,vide)	(1,vide)	(1,vide)	(1,vide)
P(four.[0])				(0,vide)	(1,vide)	(1,vide)	(1,vide)
	P(four.[1])			(0,vide)	(0,vide)	(1,vide)	(1,vide)
		P(four.[2])		(0,vide)	(0,vide)	(0,vide)	(1,vide)
			P(four.[3])	(0,vide)	(0,vide)	(0,vide)	(0,vide)
P(four.[1])				(0,vide)	(-1,Philo 0)	(0,vide)	(0,vide)
	P(four.[2])			(0,vide)	(-1,Philo 0)	(-1,Philo 1)	(0,vide)
		P(four.[3])		(0,vide)	(-1,Philo 0)	(-1,Philo 1)	(-1,Philo 2)
			P(four.[0])	(-1,Philo 3)	(-1,Philo 0)	(-1,Philo 1)	(-1,Philo 2)
BLOCAGE							

2. Soit les structures de données suivantes :

```
un sémaphore mutex initialisé à 1.
philo : tableau[0..(N-1)] de sémaphores initialisés à 0 ;
etat : tableau[0..(N-1)] de (pense,mange,affame);
```

(a) Quelques commentaires afin d'expliquer le fonctionnement des sémaphores.

```
tant que vrai faire
   pense;
   prendre_couverts(i);  Le philosophe i prend deux fourchettes.
   mange;
   poser_couverts(i);    Le philosophe i repose les deux fourchettes.
   fait
```

avec les procédures suivantes :

```
prendre_couverts(i);
   debut
   P(mutex);             Afin d'etre seul pour lire ou modifier les etats.
   etat[i] := affame;    Le philosophe i signale ses intentions.
   test(i);              Le philosophe essaie d'acquerir la ressource.
   V(mutex);
   P(philo[i]);          Le philosophe est soit :
                          - bloque s'il n'a pas obtenu la ressource dans test,
                          - non bloque et peut ainsi manger.
   fin

poser_couverts(i);
   debut
   P(mutex);             Afin d'etre seul pour lire ou modifier les etats.
   etat[i] := pense;     Le philosophe retourne a ses cheres etudes.
   test((i-1) modulo N); Il debloque eventuellement sont voisin de gauche.
   test((i+1) modulo N); Il debloque eventuellement sont voisin de droite.
   V(mutex);
   fin

test(i)                  Cette procedure : soit ne fait rien,
                         soit donne au philosophe i une ressource (via le semaphore).
   debut
   si (etat[i]=affame) et (etat[(i-1) modulo N]<>mange) et (etat[(i+1) modulo N]<>mange) alors
       debut
       etat[i] := mange;
       V(philo[i]);
       fin
   fin
```

(b) La répétition de la seconde partie du comportemnt suivant pour $(N = 4)$ n'est pas équitable.

Etat 0	Etat 1	Etat 2	Etat 3	Philo 0	Philo 1	Philo 2	Philo 3	Remarques
pense	pense	pense	pense	(0,vide)	(0,vide)	(0,vide)	(0,vide)	
affame				(1,vide)	(0,vide)	(0,vide)	(0,vide)	
mange				(0,vide)	(0,vide)	(0,vide)	(0,vide)	
	affame			(0,vide)	(-1,P 1)	(0,vide)	(0,vide)	1 est bloque par 0
			affame	(0,vide)	(-1,P 1)	(0,vide)	(-1,P 3)	3 est bloque par 0
		affame		(0,vide)	(-1,P 1)	(1,vide)	(-1,P 3)	
		mange		(0,vide)	(-1,P 1)	(0,vide)	(-1,P 3)	
pense				(0,vide)	(-1,P 1)	(0,vide)	(-1,P 3)	0 ne debloque ni 1, ni 3
affame				(1,vide)	(-1,P 1)	(0,vide)	(-1,P 3)	
mange				(0,vide)	(-1,P 1)	(0,vide)	(-1,P 3)	
		pense		(0,vide)	(-1,P 1)	(1,vide)	(-1,P 3)	2 ne debloque ni 1, ni 3
Les philosophes 0 et 2 empêchent leurs *camarades* de manger								

3. Soit les structures de données :

```
un sémaphore mutex initialisé à N-1.
fourchette : tableau[0..(N-1)] de sémaphores initialisés à 1 ;
```

La première solution est fausse (blocage) car tous les philosophes peuvent posseder au même instant une fourchette. Une correction consiste à interdir la présence de tous les philosophes au même instant autour de la table. Une mise en œuvre possible est de *retirer une chaise*, ainsi seulement $N-1$ philosphes pourront être ensemble autour de la table. De ce fait, un parmi eux obtiendra deux fourchettes, mangera, puis posera ses fourchettes afin que d'autres puissent s'en saisir.

Le sémaphore `mutex` représente les chaises.

```
tant que vrai faire
     pense;
     P(mutex);                        Le philosophe i prend un chaise.
     P(fourchette[i]);                Le philosophe i obtient la fourchette i (droite),
                                      ou bien est mis dans la file d'attente de fourchette[i].
     P(fourchette[(i+1)modulo N]);    idem pour la fourchette gauche.
     mange;
     V(fourchette[i]);                Le philosophe i libere la fourchette i (droite), et,
                                      éventuellement débloque le philosophe i-1.
     V(fourchette[(i+1)modulo N]);    Le philosophe i libere la fourchette i-1 (gauche), et,
                                      éventuellement débloque le philosophe i+1.
     V(mutex);                        Le philosophe i libère une chaise, en la donnant
                                      éventuellement à un philosophe affamé.
     fait
```

Cette solution corrige non seulement le problème du blocage, mais est également équitable. En effet un philosphe qui désire manger, s'il n'obtient pas de chaise, est bloqué par `mutex`. Mais ensuite dès qu'un de ses collègues retournera penser, il sera débloqué et obtiendra une chaise. A cet instant, si ses fourchettes sont prises, il suffira qu'il attende, car elles finiront nécessairement par être posées.

Remarque : Le philosophe peut manger debout, s'il a obtenu une chaise auparavant, cela ne change pas la validité de la solution, soit :

```
tant que vrai faire
     pense;
     P(mutex);                        Le philosophe i prend un chaise.
     P(fourchette[i]);                Le philosophe i obtient la fourchette i (droite),
                                      ou bien est mis dans la file d'attente de fourchette[i].
     P(fourchette[(i+1)modulo N]);    idem pour la fourchette gauche.
     V(mutex);                        Le philosophe i libère une chaise, en la donnant
                                      éventuellement à un philosophe affamé.
     mange;
     V(fourchette[i]);                Le philosophe i libere la fourchette i (droite), et,
                                      éventuellement débloque le philosophe i-1.
     V(fourchette[(i+1)modulo N]);    Le philosophe i libere la fourchette i-1 (gauche), et,
                                      éventuellement débloque le philosophe i+1.
     fait
```