ED 7
Interblocage

Exercice 1 : Caractérisation de l'interblocage et prévention

On dispose d'un certain nombre de ressources critiques nécessaires à l'exécution de programmes : table traçante, imprimante, modem…
Chaque tâche, lors de son exécution, demande l'allocation de ressources lorsque celles-ci sont nécessaires. Les ressources sont libérées au bout d'un temps fini et au pire en fin d'exécution de la tâche.
Un exemple de programmation de trois tâches est le suivant ;

Tâche P1	Tâche P2	Tâche P3
demander(table_traçante)	demander(modem)	demander(imprimante)
demander(modem)	demander(imprimante)	demander(table_traçante)
--exécution	--exécution	--exécution
libérer(modem)	libérer(imprimante)	libérer(table_traçante)
libérer(table_traçante)	libérer(modem)	libérer(imprimante)

Question 1

Montrer qu'il y a risque d'interblocage pour ce comportement des tâches.

On analysera en particulier les quatre conditions nécessaires à l'apparition d'un interblocage.

Question 2

Proposer une solution de prévention statique de l'interblocage pour ce cas.

Exercice 2 : Analyse de condition d'interblocage

Un système comporte 11 ressources d'une classe de ressources banalisées (par exemple des blocs disque) ; 5 tâches se partagent ces ressources en les demandant une à une, chaque tâche ne peut demander plus de 3 ressources au total et restitue ses ressources au bout d'un temps fini.

Question 1

Montrer qu'il ne peut y avoir interblocage dans un tel système.

On généralise le problème précédent : N tâches se partagent M ressources d'une classe de ressources banalisées qu'ils demandent une à une, chaque tâche ne peut demander plus de T ressources. On a les relations suivantes :

T <M + 1 et N*T<M+N

Question 2

Montrer que l'interblocage est impossible dans un tel système.

On suppose maintenant que les 5 tâches se partagent 11 ressources d'une classe A de ressources banalisées, la demande maximale Ta pour chaque tâche est 3 et 6 ressources d'une classe B de resssources banalisées, la demande maximale Tb pour chaque tâche est 2. Les ressources sont acquises une à une et restituées au bout d'un temps fini.

Question 3

Montrer que contrairement au cas où il n'y a qu'une classe de ressources, il y a risque d'interblocage dans un tel système.

Question 4

Montrer que si le système dispose de 14 ressources de classe A et 7 ressources de classe B, il n'y a plus de risque d'interblocage.

Exercice 3 : Détection et prévention dynamique de l'interblocage

On considère un système comprenant 16 cases de mémoire (ressources banalisées) partagées par 3 processus(tâches).

A l'instant t1, on a l'état suivant :

processus	Allocation	Requête	Disponible
P0	5	0	0
P1	5	2	
P2	6	2	

A l'instant t2, on a l'état suivant :

processus	Allocation	Requête	Disponible
P0	5	1	0
P1	5	2	
P2	6	3	

A l'instant t3, on a l'état suivant :

processus	Allocation	Requête	Disponible
P0	3	1	3
P1	7	2	
P2	3	3	

Question 1

Pour les instants t1, t2 et t3, dire s'il y a ou non interblocage. On montrera que l'état du système est sain ou non.

On suppose que la demande maximale de chacun des processus est de 10 cases mémoire.

Question 2

Pour les instants t1, t2 et t3, dire s'il y a ou non risque d'interblocage . On montrera que l'état du système est fiable ou non.

Exercice 4 : Interblocage par verrouillage de parties d'un fichier

Sous Unix/Linux, il existe une fonction système appelée `fcntl`() qui permet de verrouiller et de déverrouiller une section (ou la totalité) d'un fichier. Cette fonction utilise la structure de données flock définie dans la librairie <`fcntl.h`>. La structure `flock` contient les champs suivants :

```
struct flock {
      short l_type ;    /* type: F_RDLCK, F_WRLCK ou F_UNLCK */
      short l_whence ;  /* portée: SEEK_SET, SEEK_CUR ou SEEK_END */
      off_t l_start ;   /* position par rapport à l_whence */
      off_t l_len ; /* longueur: si =0 <==> jusqu'à la fin du fichier */
      pid_t l_pid ;     /* pid du processus qui pose ce verrou */
}
```

Le champ l_type peut prendre l'une des valeurs suivantes :

- F_RDLCK : verrou de lecture, partagé par plusieurs processus,
- F_WRLCK : verrou en écriture, verrou exclusif (accès interdit par d'autres processus),
- F_UNLCK : déverrouillage.

Le verrouillage en écriture nécessite au préalable l'ouverture du fichier en écriture.

Le champ l_whence peut prendre une des valeurs suivantes pour définir la portée du fichier :

- SEEK_SET : à partir du début du fichier,
- SEEK_CUR : à partir de la position courante du curseur sur le fichier,
- SEEK_END : depuis la fin du fichier.

La portée commence à l_whence + l_start et sur l_len octets. Si l_len = 0, alors la portée va jusqu'à la fin du fichier et suit la fin du fichier si la taille de celui-ci augmente.

La fonction suivante :
fcntl(d, F_SETLKW, &verrou);
permet de poser un verrou sur une section d'un fichier de descripteur d. Le mode de verrouillage ainsi que la portée sont définis dans la variable verrou.
Si la section spécifiée dans la variable verrou est déjà verrouillée, il y a blocage du processus appelant jusqu'à déverrouillage de cette section.

Supposons que l'on dispose d'un fichier Fiche.txt qui dont le contenu est le suivant :
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

Soient les opérations suivantes :
a- verrouiller en écriture une suite de 5 caractères du fichier Fiche.txt à partir du 10ème caractère.
b- verrouiller en écriture une suite de 5 caractères du fichier Fiche.txt à partir du 20ème caractère.

Supposons que le processus P1 après qu'il ait effectué l'opération a, fait un traitement quelconque qui prend 100ms avant de tenter de faire l'opération b, sachant que le processus P2 se contente de faire l'opération b suivi d'un traitement quelconque qui prend 200ms.
Le verrouillage sera possible même si c'est P2 qui commence à s'exécuter avant P1.

Question 1

Décrire le comportement de P1 et de P2 vis-à-vis des opérations de verrouillage si P1 démarre son exécution en premier.

Nous complétons le scénario précédent en supposant que P2 après l'opération b et le traitement de 200ms tente d'effectuer l'opération a.

Question 2

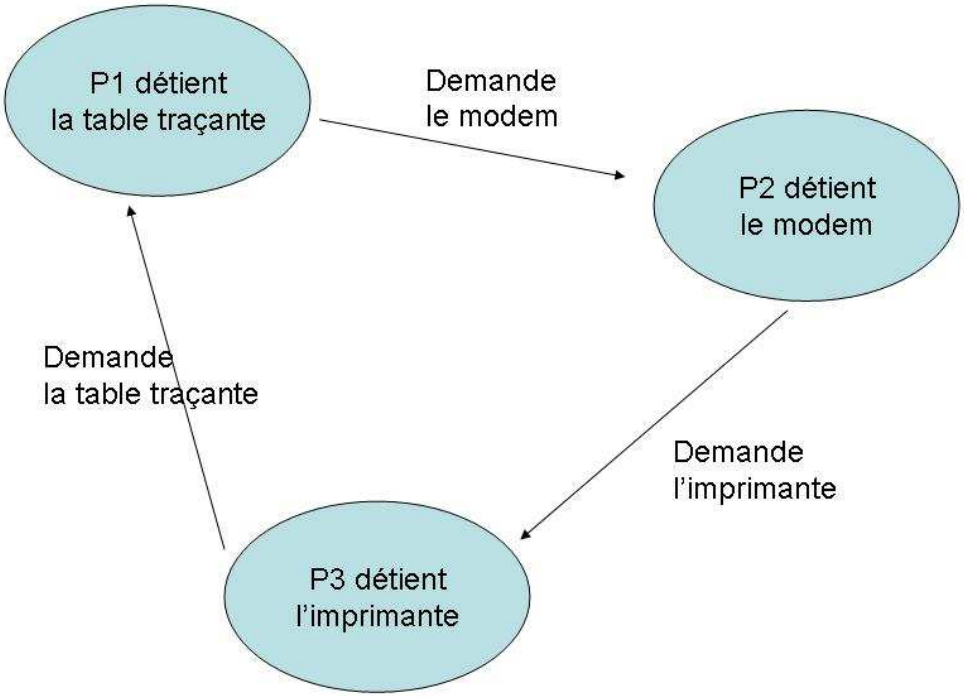
Quelle situation obtient-on dans ce cas ?

Sachant que le système refuse une opération de verrouillage qui présente un risque d'interblocage en retournant dans `fcntl()` *-1 avec* `EDEADLK` *dans* `errno`*, quel est le processus qui verra sa tentative de verrouillage échouée ?*

Solution

Exercice 1 :

Question 1 : attente circulaire possible.



Les 4 conditions nécessaires à l'apparition de l'interblocage :

1. accès exclusif de chaque ressource
2. les processus qui demandent de nouvelles ressources gardent celles qu'ils ont déjà acquises et attendent l'arrivée de leur demande
3. pas de réquisition possible de ressources déjà allouées
4. les processus en attente des ressources déjà allouées forment une chaîne circulaire.

Question 2 :

Une solution pour ce cas : P3 doit demander la table traçante avant l'imprimante.

Solution générale :

- les processus demandent les ressources une après l'autre dans le même ordre.
- ou bien les processus demandent toutes les ressources en même temps (demande globale).

Exercice 2 :

Question 1 :

Au pire, chaque processus a obtenu déjà 2 ressources, ce qui a mobilisé 10 ressources. Le premier qui demandera sa 3ème ressource pourra terminer et rendre une ressource au moins ; cette ressource pourra permettre à un autre processus de terminer, etc.

Généralisation : Au pire, les N processus ont chacun T-1 ressources ; il doit en rester une pour que l'un après l'autre tous les processus puissent terminer, donc M=N(T-1) + 1 ou encore M + N = N*T + 1 ou encore M + N > N*T.

D'autre part, le nombre de ressources maximal qu'un processus peut demander ne doit pas dépasser le nombre de ressources disponibles. D'où : T<= M c'est-à-dire : T< M+1.

Question 2 :

Selon la formule précédente, il n'y a pas d'interblocage. En effet :

Pour N=5, M=11, T=3, M+N=16 < 5*3=15 et T=3 < M+1=12

Question 3 :

Soit le tableau donnant les allocations avec Ta=3 et Tb=2

Processus	P1	P2	P3	P4	P5	Réserve
Ressource A	2	2	2	2	2	1
Ressource B	1	1	1	1	1	1

Si P1 reçoit la dernière ressource A et P2 la dernière ressource B, on est en interblocage dès que tous les processus demandent une nouvelle ressource qu'elle soit de la classe A ou de la classe B.

Question 4 :

Pour éviter l'interblocage, il faut que, dans le pire cas, la dernière ressource puisse permettre aux processus de finir l'un après l'autre. Il vient :

Processus	P1	P2	P3	P4	P5	Réserve
Ressource A	2	2	3	3	3	1
Ressource B	2	2	1	1	1	0

Exercice 3 :

Question 1

A l'instant t1, on a l'état suivant :

Processus	Allocation	Requête	Disponible
P0	5	0	0
P1	5	2	
P2	6	2	

L’état est sain car P0 peut s’exécuter sans demande supplémentaire de ressources. A la fin de P0, les 5 ressources sont libérées et sont utilisées pour satisfaire les demandes de P1 et P2.

A l’instant t2, on a l’état suivant :

processus	Allocation	Requête	Disponible
P0	5	1	0
P1	5	2	
P2	6	3	

L’état est non sain car aucune demande ne pourra être satisfaite.

A l’instant t3, on a l’état suivant :

Processus	Allocation	Requête	Disponible
P0	3	1	3
P1	7	2	
P2	3	3	

L’état est sain. Il y a suffisamment de ressources (3) pour satisfaire les demandes de P0 et P1 en même temps ou bien celle de P2. Après libération des ressources, d’autres demandes seront satisfaites.

Question 2

Etat sain : signifie que le système n'est pas en interblocage. Une suite saine s'obtient en plaçant dans la séquence fictive d'abord tous les processus servis, puis tous les processus en attente de ressources.

Etat fiable : signifie que le système ne peut évoluer vers un interblocage.

Un état fiable implique un état sain et un état sain n’implique pas forcément un état fiable.

Un état fiable se calcule comme pour l’état sain sauf que l’on raisonne par rapport à l’annonce qui est de 10.

A l’instant t1, on a l’état suivant :

Processus	Allocation	Requête	Disponible
P0	5	10-5=5	0
P1	5	10-5=5	
P2	6	10-6=4	

L’état est non fiable car en raisonnant par rapport à l’annonce, aucune demande ne peut être satisfaite.

A l'instant t2, on a l'état suivant :

processus	Allocation	Requête	Disponible
P0	5	1	0
P1	5	2	
P2	6	3	

L'état n'étant pas sain, il n'est donc pas fiable.

A l'instant t3, on a l'état suivant :

processus	Allocation	Requête	Disponible
P0	3	1	3
P1	7	2	
P2	3	3	

L'état est fiable car en raisonnant par rapport à l'annonce, la demande de P1 peut être satisfaite. La libération des ressources de P1 permettra de satisfaire les demandes de P0 et de P2.

Exercice 4 :

L'opération a se traduit par :

```
struct flock verrou;

descript=open("Fiche.txt",O_RDWR);

/* verrou exclusif sur les 5 caracteres a partir du 10eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 10;
verrou.l_len=   5;
verrou.l_pid= getpid();
fcntl(descript, F_SETLKW, &verrou) ;
```

L'opération b se traduit par :

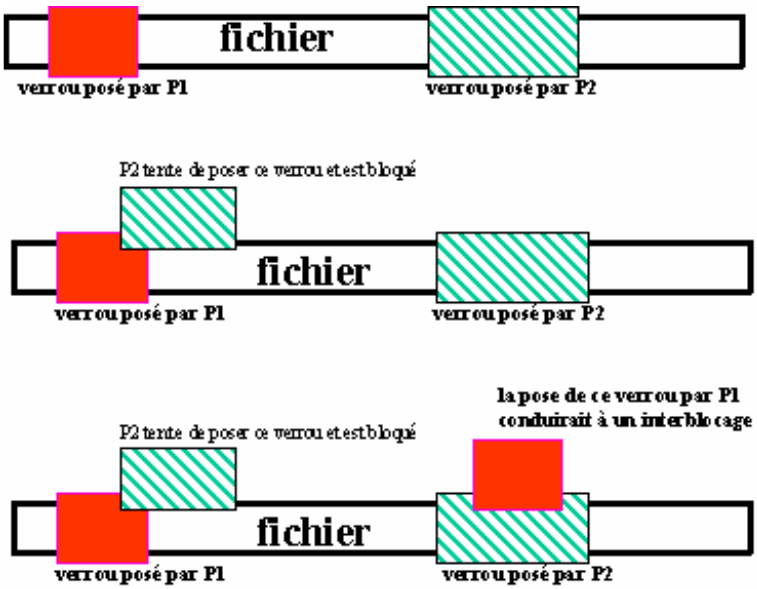
```
/* verrou exclusif sur les 5 caracteres a partir du 20eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 20;
verrou.l_len=   5;
verrou.l_pid= getpid();
```

Question 1 :

Si P1 démarre en premier, il verrouille la zone a et commence un traitement de 100ms. En supposant un quantum de temps inférieur à 100ms (ce qui est souvent le cas), P2 prend la main, démarre son exécution en verrouillant la zone b du fichier Fiche.txt. Lorsque P1 reprend la main, il tente de verrouiller la zone b qui est déjà verrouillée. P1 se bloque alors jusqu'à la fin de P2, après quoi P1 sera débloqué et pourra verrouiller la zone b.

Question 2 :

P1 verrouille la zone a. P2 verrouille la zone b. Après des traitements respectifs, P1 demande à verrouiller la zone b, il sera bloqué en attente de déverrouillage de la zone b. P2 à son tour demande à verrouiller la zone a => situation d'interblocage.
Pour éviter une situation d'interblocage, le système détecte cette situation et refuse la demande de verrouillage de P2 en retournant -1 à l'appel fcntl() ainsi que EDEADLK dans errno.
C'est donc P2 qui va échouer dans sa demande de verrouillage.



Annexe : Programme complet

Dans ce programme, il y a un processus père et un processus fils ; chacun pose un 1er verrou sur une portion différente du fichier. Puis le processus père tente de poser un autre verrou sur la partie verrouillée par le processus fils; il est donc bloqué. Ensuite le processus fils tente de poser un verrou sur la partie verrouillée par le processus père et provoque donc une situation d'interblocage (voir Figure).

L'appel à fcntl() retourne -1 et le processus qui détecte l'interblocage doit se terminer pour libérer ses verrous et permettre à l'autre processus de poser son deuxième verrou.

```
/* pgme.c */
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>

main(){
int descript;
struct flock verrou;
if((descript=open("Fiche.txt",O_RDWR))==-1) {perror("erreur sur open"); exit(1);}

if(fork()==0) {    /* Processus FILS */
/* verrou exclusif sur les 5 caracteres a partir du 10eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 10;
verrou.l_len=   5;
verrou.l_pid= getpid();

/* pose bloquante */
if(fcntl(descript, F_SETLKW, &verrou)==-1){perror("ERR: pose verrou");}
printf("1er verrou pose par %d\n", getpid());

sleep(rand()%3);

/* verrou exclusif sur les 5 caracteres a partir du 20eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 20;
verrou.l_len=   5;
verrou.l_pid= getpid();

/* pose bloquante 2eme verrou*/
printf("%d tente de poser le 2eme verrou\n", getpid());
if(fcntl(descript, F_SETLKW, &verrou)==-1){
       printf("Refus de la pose de verrou pour %d : Risque d'interblocage \n",
       getpid());
       if(errno==EDEADLK){
              fcntl(descript, F_GETLK, &verrou);
              printf("interblocage avec le processus %d\n",verrou.l_pid);
              printf("arret du processus %d\n", getpid()); exit(1);
       }
}
printf("2eme verrou pose par %d\n", getpid());
sleep(5); /* pour laisser les autres subir le verrou */
printf("Fin du processus %d\n",getpid()); /* suppression automatique des verrous*/
}

else {    /* Processus PERE */
/* verrou exclusif sur les 5 caracteres a partir du 20eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 20;
verrou.l_len=   5;
verrou.l_pid= getpid();

/* pose bloquante */
```

```
if(fcntl(descript, F_SETLKW, &verrou)==-1){perror("ERR: pose verrou");exit(1);}
printf("1er verrou pose par %d\n", getpid());

sleep(rand()%3);

/* verrou exclusif sur les 5 caracteres a partir du 10eme */
verrou.l_type=  F_WRLCK;
verrou.l_whence=SEEK_SET;
verrou.l_start= 10;
verrou.l_len=   5;
verrou.l_pid= getpid();

/* pose bloquante 2eme verrou*/
printf("%d tente de poser le 2eme verrou\n", getpid());
if(fcntl(descript, F_SETLKW, &verrou)==-1){
       printf("Refus de la pose de verrou pour %d : Risque d'interblocage \n",
       getpid());
       if(errno==EDEADLK){
              fcntl(descript, F_GETLK, &verrou);
              printf("interblocage avec le processus %d\n",verrou.l_pid);
              printf("arret du processus %d\n", getpid()); exit(1);
       }
}
printf("2eme verrou pose par %d\n", getpid());
sleep(5); /* pour laisser les autres subir le verrou */
printf("Fin du processus %d\n",getpid()); /* suppression automatique des verrous*/
}
}
```

Exécution

```
$cc pgme.c -o pg2
$./pg2
1er verrou pose par 1568
1er verrou pose par 1569
1568 tente de poser le 2eme verrou
1569 tente de poser le 2eme verrou
Refus de la pose de verrou pour 1569 : Risque d'interblocage
interblocage avec le processus 1568
arret du processus 1569
2eme verrou pose par 1568
Fin du processus 1568
$
```