Exercices dirigés

Systèmes & Réseaux Informatiques B4

2 - Systèmes

2003-2004

mise à jour du 25 mars 2004

Ce polycopié a été élaboré par les enseignants de l'équipe "Systèmes et Réseaux B" à partir d'exercices rédigés par Mmes Bouzefrane, Coquery, Costa, MM Berthelin, Cubaud, Kaiser, Peyre

ED 1
Le système en tant qu'environnement d'exécution exemple de LINUX

I Environnement d'un programme

Soit le programme "c" suivant (calcul_somme.c):

```
1.#include <stdio.h>
2.int VAR1 = 1;
3.int VAR2 = 1;
4.
5.// ======================A=========================
6.int calcul1( int n)
7.{
8.    int i, res = 0;
9.    for(i=0; i<=n; i++)
10.        res+=i;
11.    return res;
12.}
13.
14.// ======================B=========================
15.int calcul2( int n)
16.{
17.    int i, res = 0;
18.    for(i=0; i<=n; i++)
19.        res+=i*i;
20.    return res;
21.}
22.
23.// ======================C=========================
24.int main(int argn, char *argv[], char *env[])
25.{
26.  int nb_pas;
27.  int i;
28.  char *zone;
29.
30.  for (i=0; i < 8; i++)
31.    printf("VARIABLE ENVIRONNEMENT:%s\n", env[i]);
32.
33.  zone = (char *) malloc(1024*1024);
34.
35.  if (argn != 2){
36.    printf("il faut 1 argument : le nb de pas de calcul\n");
37.    exit (-2);
38.  }
39.
40.  sscanf(argv[1], "%d", &nb_pas);
41.
42.  printf("\n\n\n\nnombre de pas = %d \n", nb_pas);
43.  printf("somme = %d\n", calcul1(nb_pas));
44.  printf("somme = %d\n", calcul2(nb_pas));
45.
46.  printf("Adresse de   main       = %09lx\n", main);
```

```
47.  printf("Adresse de   VAR1       = %09lx\n", &VAR1);
48.  printf("Adresse de   VAR2       = %09lx\n", &VAR2);
49.  printf("Adresse de   nb_pas     = %09lx\n", &nb_pas);
50.  printf("Adresse de   zone       = %09lx\n", zone);
51.  printf("Adresse de   argv[1]    = %09lx\n", argv[1]);
52.
53.  // ====================D=========================
54.  sleep(2000);
55.  exit(nb_pas);
56.}
```

Question 1

*a)A quoi correspondent la fonction main et ses arguments (int argn, char *argv[], char *env[]) ?*

b)Définir les différentes composantes de ce programme?

Voici le début d'une exécution de ce programme. (calcul_somme 100)

```
VARIABLE ENVIRONNEMENT:LESSKEY=/etc/lesskey.bin
VARIABLE ENVIRONNEMENT:NNTPSERVER=news
VARIABLE ENVIRONNEMENT:INFODIR=/usr/local/info:/usr/share/info:/usr/info
VARIABLE ENVIRONNEMENT:HOSTNAME=bacchus
VARIABLE ENVIRONNEMENT:XKEYSYMDB=/usr/X11R6/lib/X11/XKeysymDB
VARIABLE ENVIRONNEMENT:HOST=bacchus
VARIABLE ENVIRONNEMENT:TERM=xterm
VARIABLE ENVIRONNEMENT:SHELL=/bin/bash

nombre de pas = 100
somme = 5050
somme = 338350
Adresse de   main       = 00804867c
....
```

c) A quoi correspondent ces variables d'environnement,à quoi servent-elles ?

d) Qui initialise ces variables ?

II Compilation d'un programme

Question 2.

a) Expliquer la chaine de production d'un programme.

On exécute les commandes suivantes :

```
#gcc -c calcul_somme.c
  #ls -l calcul_somme.o
       -rw-r--r--    1 peyre    23    2044 2003-02-24 14:35 calcul_somme.o
```

b) Rappeler les différentes composantes d'un ficher "objet".

c) Expliquer le résultat de la commande suivante:

```
#nm calcul_somme.o
  00000000 D VAR1
  00000004 D VAR2
  00000000 T calcul1
  00000038 T calcul2
  00000074 T main
           U exit
           U malloc
           U printf
           U sleep
           U sscanf
```

III Edition de liens

1.Edition de liens statique

On exécute les commandes suivantes :

```
#gcc -static calcul_somme.o -o calcul_somme.st
#ls -l calcul_somme.st
-rwxr-xr-x    1 peyre    23    381616 2003-02-24 16:54 calcul_somme.st*
```

Question 3.

a) Expliquer la différence de taille avec le fichier objet obtenu à la Question 2

b) Expliquer le resultat des commandes suivantes :

```
#ldd calcul_somme.st
        not a dynamic executable

#nm  calcul_somme.st| egrep 'printf|scanf|exit|calcul1|calcul2'

080493e0 T _IO_printf
08049410 T _IO_sscanf
08059410 T _IO_vfprintf
08060dc0 T _IO_vfscanf

0804f280 T _exit
08048180 T calcul1
080481f4 T calcul2
080491f0 T exit
08060d60 T fprintf
080493e0 T printf
08049410 T sscanf
```

2.Edition de liens dynamique

On refait les mêmes manipulations et on obtient;

```
#gcc calcul_somme.c -o calcul_somme
```

```
#ls -l calcul_somme
   -rwxr-xr-x    1 peyre    23      7200 2003-02-24 16:28 calcul_somme*

#nm calcul_somme| egrep 'printf|scanf|exit|calcul1|calcul2'
   080485cc T calcul1
   08048640 T calcul2
            U exit@@GLIBC_2.0
            U printf@@GLIBC_2.0
            U sscanf@@GLIBC_2.0

#ldd calcul_somme
   libm.so.6 => /lib/libm.so.6 (0x40026000)
   libc.so.6 => /lib/libc.so.6 (0x40049000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

c) Expliquer les différences entre les deux éditions de liens .

On s'intéresse à la librairie partagée "libc.so.6 ".

On effectue donc la commande suivante :

```
#nm /lib/libc.so.6 | egrep 'printf|scanf|exit'
     00058630 T _IO_printf
     00068150 T _IO_sprintf
     0005d120 T _IO_sscanf
     ...
     000a96e0 T _exit
     0002f740 T exit
     00058630 T printf
     0005d0e0 T scanf
```

d) Pourquoi dit-on que cette librairie est partagée; comment le système peut-il utiliser cette librairie ?

IV Lancement du programme

On se propose d'étudier le démarrage de notre programme dans le système. Pour cela on exécute les trois commandes suivantes :

```
#env | grep LD_LIBRARY_PATH
LD_LIBRARY_PATH=/usr/openwin/lib:/usr/X11R6/lib:/usr/local/lib

#strace calcul_somme.st 100
                      = ?
    1.execve("./calcul_somme.st", ["./calcul_somme.st", "100"], [/* 56 vars
    */]) = 0
    2.fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 7), ...}) = 0
    3.mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
    = 0x40000000
    4.write(1, "VARIABLE ENVIRONNEMENT:LESSKEY=/"..., 48) = 48
    5.... (autres write)
    6.mmap2(NULL, 1052672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
    1, 0) = 0x40001000
    7.write(1, "Adresse de   argv[1]    = 0bffff"..., 36) = 36
    8.nanosleep({2, 0}, {2, 0})               = 0
    9.munmap(0x40000000, 4096)                = 0
    10._exit(100)
```

```
#strace calcul_somme 100
   1.execve("./calcul_somme", ["./calcul_somme", "100"], [/* 56 vars */]) =
   = 0x804a6c8
   2.mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
   0) = 0x40013000
   3.open("/etc/ld.so.cache", O_RDONLY)      = 3
   4.fstat64(3, {st_mode=S_IFREG|0644, st_size=71175, ...}) = 0
   5.mmap2(NULL, 71175, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40014000
   6.close(3)                                = 0
   7.open("/lib/libm.so.6", O_RDONLY)        = 3
   8.read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0@7\0\000"...,
   1024) = 1024
   9.fstat64(3, {st_mode=S_IFREG|0755, st_size=176463, ...}) = 0
   10.mmap2(NULL, 140960, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
   0x40026000
   11.mmap2(0x40048000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3,
   0x21) = 0x40048000
   12.close(3)                               = 0
   13.open("/lib/libc.so.6", O_RDONLY)       = 3
   14..read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\300\205"...,
   1024) = 1024
   15.fstat64(3, {st_mode=S_IFREG|0755, st_size=1312470, ...}) = 0
   16.mmap2(NULL, 1169856, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
   0x40049000
   17.mmap2(0x4015d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED,
   3, 0x113) = 0x4015d000
   18.mmap2(0x40163000, 14784, PROT_READ|PROT_WRITE,
   MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x40163000
   19.close(3)                               = 0
   20.munmap(0x40014000, 69719)              = 0
   21.fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 7), ...}) = 0
   22.mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
   0) = 0x40014000
   23.write(1, "VARIABLE ENVIRONNEMENT:LESSKEY=/"..., 48) = 48
   24....
   25.mmap2(NULL, 1052672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
   1, 0) = 0x40167000
   26.write(1, "Adresse de   argv[1]    = 0bffff"..., 36) = 36
   27.nanosleep({2, 0}, {2, 0})              = 0
   28.munmap(0x40014000, 4096)               = 0
   29._exit(100)             =?
```

Question 4.

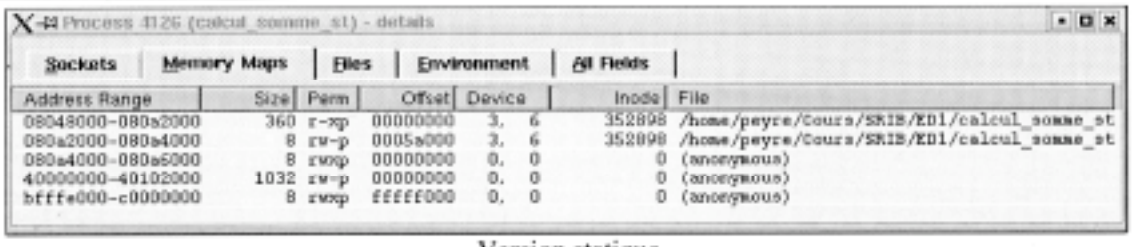
a) Rappeler le rôle de la primitive execve; qui l'exécute et comment ?

b) Détailler les différentes étapes mises en évidence par la commande "strace

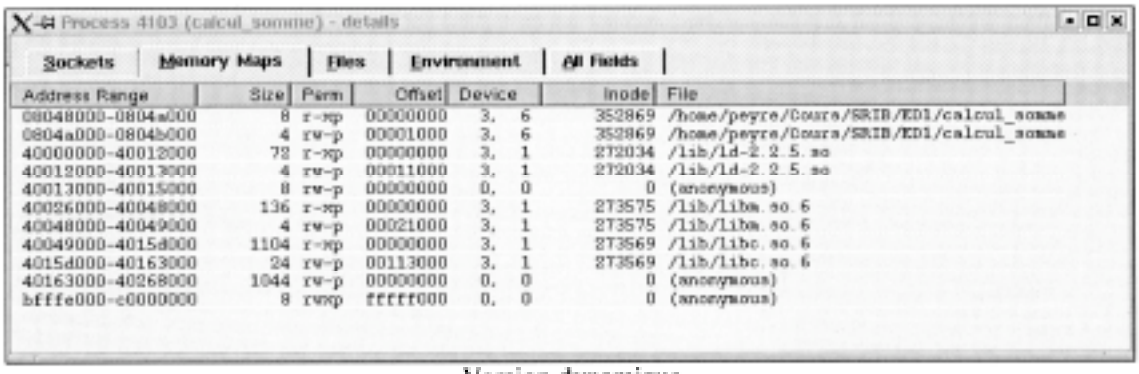
V Exécution

On s'intéresse à l'exécution des deux versions de notre exécutable (édition de liens statique ou dynamique). Un outils nous permet de retrouver les zones mémoires allouées par le système à chacun des processus. De plus on connait les adresses des symboles suivants :

symbole	version statique	pour dynamique
main	008048230	00804867c
nb_pas (local main)	0bffff3d4	0bffff3d4
VAR 1 (global)	0080a2008	00804a3b4
argv[1]	0bffff5b3	0bffff5b9
zone	040001008	040167008



Version statique



Version dynamique

ED 2
Introduction aux thèmes abordés en Systèmes B

Gestion des *processus*⊡

- Mise en évidence de parallélismes (pseudo ou réels)
- Mise en évidence de l'existence d'acteurs systèmes

Gestion des *ressources*⊡

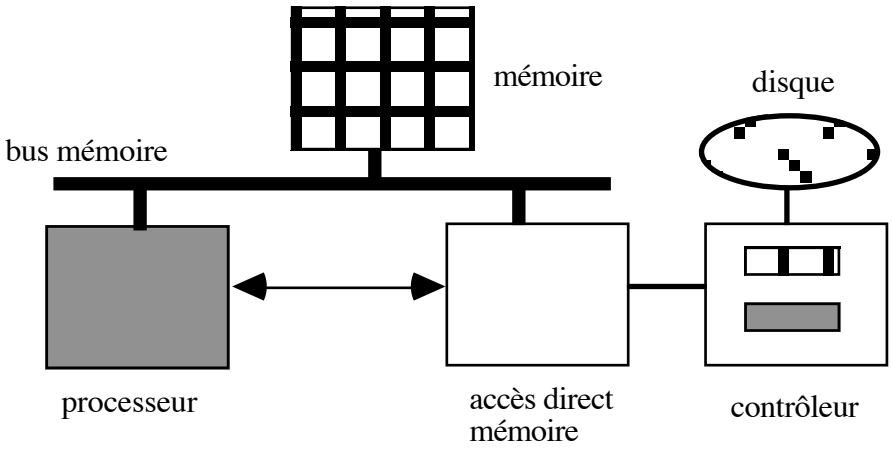
- Exécuter des travaux dont les besoins sont supérieurs aux ressources du système.
- Optimiser le système en partageant des ressources.
- Sécuriser les partages de ressources (ex. : éviter l'interblocage)

Exercice 1⊡Un petit système multi-tâches

A partir de l'étude d'un sous-système d'entrée-sortie, nous rappellerons des fonctionnalités d'un système d'exploitation et dégagerons le concept de parallélisme, puis celui de processus et nous aborderons les problèmes liés à la synchronisation des processus.

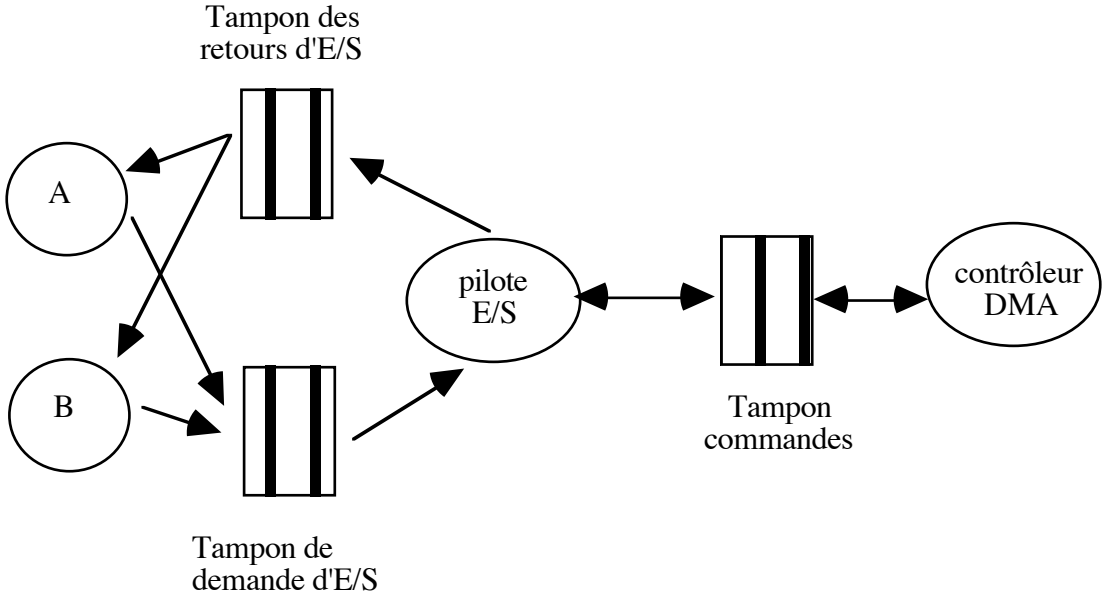
Question 1⊡Présentation du système

Le schéma ci-dessous représente un système d'entrée-sortie que l'on commmentera, et dont on expliquera le fonctionnement en dégageant la notion de parallélisme "réel". Supposant qu'il y ait deux utilisateurs "simultanés" de ce système, on donnera un exemple de fonctionnement en introduisant la notion de pseudo-parallélisme.



Question 2⊡Découpage du système en processus.

Le schéma ci-dessous représente la coopération entre les processus.



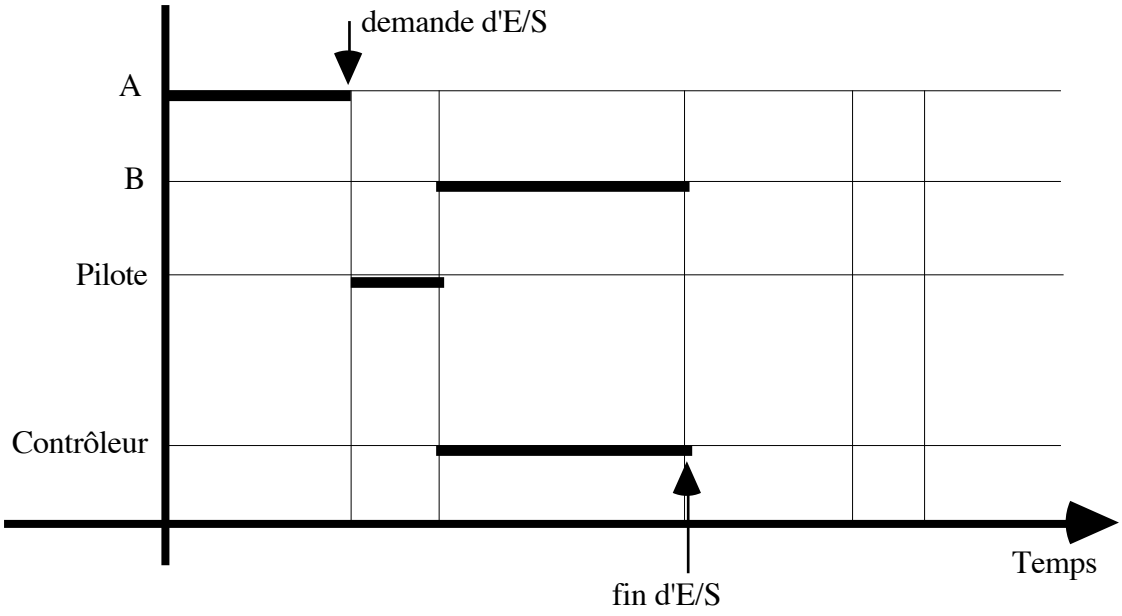
Commenter cette organisation

Question 3⊡Représentation du parallélisme entre processus.

Le chronogramme ci-dessous donne un exemple de diagramme du fonctionnement avec deux utilisateurs du système en mettant en évidence le parallélisme entre les processus.

Compléter le chronogramme

On soulignera les 3 états d'un programme (élu, prêt, bloqué), le parallélisme réel (entre le DMA et B, par exemple) et le pseudo-parallélisme (entre A et B, par ex.)



Exercice 2⊡Exemple de processus UNIX

Soit le programme C suivant :

```
#include <unistd.h>

void main() {
pid_t p1, p2, p3, p4;
int i;

if ((p1=fork())==0)
   if ((p2=fork())==0)
printf("je suis le processus p2, mon numero est %d \n", getpid());
   else  execlp("a", "a", NULL);

if ((p3=fork()==0) { execlp("b","b", NULL); }
if ((p4=fork()==0) { execlp("c","c", NULL); }
sleep(5);
/* attente de terminaison des fils*/
while((i=waitpid(-1,NULL,0))>0) printf(" \nFils %d termine\n ",i) ;
}
```

Question 1

Tracer l'arborescence des processus créés par ce programme si les programmes a, b et c se terminent tous par l'instruction : exit(2);

Pour simplifier, nous supposons que :

- les programmes a, b, c durent respectivement 2s, 3s et 1s
- tous les processus partagent le même processeur
- un processus qui obtient le processeur le garde jusqu'à ce qu'il se termine sauf dans le cas où il passe à l'état endormi (exécute la fonction `sleep()`).

Question 2

Dans ces conditions précises, représenter à l'aide d'un diagramme le comportement de chaque processus de l'arborescence.

ED 3
Gestion du processeur

Exercice 1 : Politiques d'ordonnancement des processus

Exemple de Linux

Il existe trois politiques d'ordonnancement dans le système Linux : la première est utilisée pour ordonnancer des processus ordinaires et les deux autres pour ordonnancer des processus temps réel. Afin de connaître la politique d'ordonnancement à utiliser pour un processus, chaque processus possède un type qui peut être égal à :

`SCHED_FIFO` pour un processus temps réel non préemptible,

`SCHED_RR` pour un processus temps réel préemptible,

`SCHED_OTHER` pour un processus ordinaire (non temps réel).

Trois files différentes accueilleront les processus prêts appartenant aux trois types. Les processus de la file `SCHED_FIFO` sont plus prioritaires que ceux de la file `SCHED_RR` qui eux-mêmes sont plus prioritaires que ceux de la file `SCHED_OTHER`.

Quel que soit son type, un processus Linux possède une priorité et est inséré dans la file associée à son type dans l'ordre décroissant de sa priorité. A tout moment, le processus de type *x* le plus prioritaire se trouve en tête de la file du même type.

L'ordonnanceur gère la file `SCHED_FIFO` en utilisant la stratégie non préemptive. Il choisit d'élire le processus de type `SCHED_FIFO` le plus prioritaire pour s'exécuter. N'étant pas préemptible, ce processus s'exécute jusqu'à la fin sans libération du processeur, excepté dans les cas suivants :

un autre processus de type `SCHED_FIFO` plus prioritaire vient d'être inséré dans la file ; il est alors exécuté à la place du processus courant qui est inséré en fin de file.

le processus demande à faire une entrée-sortie;

le processus abandonne le processeur en appelant la primitive *`sched_yield()`*.

La file `SCHED_RR` est gérée par la technique du tourniquet avec une seule file d'attente. En effet, tout processus de type `SCHED_RR` est exécuté pour la durée d'un quantum de temps (égale à 60 ms). A l'expiration du quantum de temps, le processus le plus prioritaire parmi les processus de type `SCHED_FIFO` est choisi. Si la file d'attente associée est vide, le processus `SCHED_RR` le plus prioritaire est élu.

La priorité pour les processus des files `SCHED_FIFO` et `SCHED_RR` varie entre 1 et 99. Le plus prioritaire étant celui qui a la plus grande valeur.

La file `SCHED_OTHER` accueille les processus ordinaires c'est-à-dire non temps réel. Les processus temps réel étant plus prioritaires, les processus de type `SCHED_OTHER` ne pourront s'exécuter que si les files de type `SCHED_FIFO` et `SCHED_RR` sont vides.

La file `SCHED_OTHER` est elle aussi gérée avec la technique du tourniquet à une seule file d'attente. L'insertion des processus se fait en fonction de leurs priorités mais lorsque la priorité est identique (en général égale à 0), l'insertion se fait en FIFO.

La priorité d'un processus (quel que soit son type) se calcule à partir :

d'une partie modifiable par l'utilisateur (à l'aide des primitives `nice` ou `setpriority`). Un processus utilisateur ne pourra que diminuer sa priorité; il ne pourra l'augmenter que s'il est un processus privilégié.

et d'une partie fixée par le système et qui baisse lorsque le processus consomme un certain nombre de cycles d'horloge ; ceci pour permettre à ceux de moindre priorité de s'exécuter rapidement.

Soit un programme C lancé sous Linux par le super-utilisateur, et qui crée presque en même temps 4 processus p1, p2, p3 et p4.

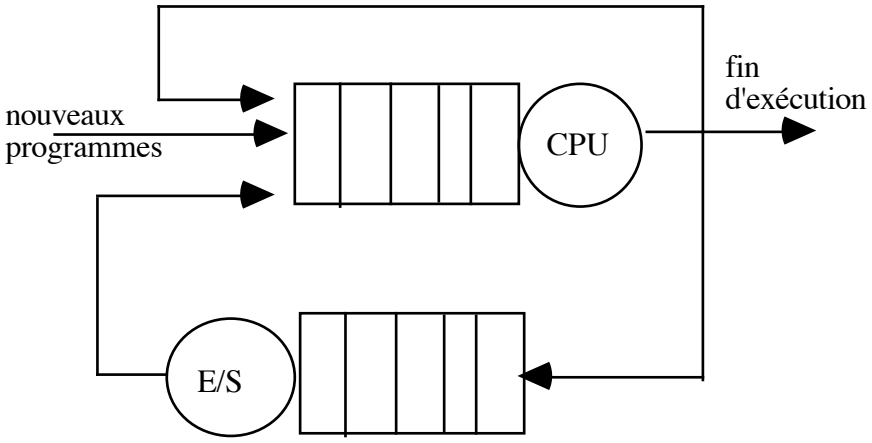
Question

En supposant que ces processus ne font pas d'entrée-sortie, que leurs priorités ne changent pas durant l'exécution et que leurs durées d'exécution sont celles décrites ci-dessous, déterminer les temps de réponses de chaque processus en appliquant les politiques d'ordonnancement adéquates.

Nom du processus	durée d'exécution	actions particulières réalisées
P0	60ms	à la fin de son calcul, il crée les 4 processus : p1, p2, p3 et p4 - met p1 dans la file `SCHED_RR` avec la priorité 2 - met p2 dans la file `SCHED_FIFO` avec la priorité 3 - met p3 dans la file `SCHED_RR` avec la priorité 4 - met p4 dans la file `SCHED_FIFO` avec la priorité 4
	30ms	calcul + attente de la fin des processus
P1	100ms	calcul
P2	90ms	calcul
	30ms	exécute `usleep`(30)
	10ms	calcul
P3	110ms	calcul + exécute `sched_setscheduler()` à la fin du calcul pour se mettre dans la file SCHED_OTHER avec la priorité 1
	20ms	calcul
P4	70ms	calcul + exécute `sched_yield()` en fin de calcul pour libérer le processeur
	50ms	

Exercice 2□
Priorité des processus fonction du temps d'accès à l'UC

On considère un système dans lequel les seules ressources partagées sont un disque géré par un canal d'entrée sortie et un processeur.



Les requêtes disques sont gérées à l'ancienneté (FIFO). Par contre, l'allocation du processeur est réalisée selon la politique suivante : on définit un quantum de temps q. Un processus allocateur est activé à chaque début et fin de traitement d'une entrée-sortie disque ou après un quantum si une entrée-sortie ne s'est pas produite dans le dernier quantum.

L'allocateur alloue le processeur au processus Pi en attente qui a le plus fort rapport Ti/Tcpui. Ti représente la durée totale écoulée depuis le début de l'exécution du processus Pi (c'est-à-dire le temps écoulé depuis l'instant de première requête du processeur par le processus Pi).

Tcpui est le cumul des durées pendant lesquelles le processeur a été alloué au processus i.

Lorsque le rapport Ti/Tcpui est égal à 0/0, celui-ci est interprêté comme $+\infty$.

Lorsque plusieurs processus ont même priorité supérieure à celle de tous les autres processus, c'est le processus d'indice le plus fort qui obtient le processeur.

L'allocateur utilise un délai de garde (temporisation) de durée q. Il est activé□

- soit lorsque le délai est écoulé

- soit lorsque le processus actif fait une requête d'entrée-sortie

- soit lorsque l'exécution d'une telle requête se termine.

Dans tous les cas, il commence par réarmer le délai de garde puis procède à l'allocation quand celle-ci est possible. A l'instant initial trois processus sont présents dans le système et commencent à s'exécuter en faisant leur première requête à l'allocateur.

Le tableau suivant donne (en nombre de quantums) la durée totale de CPU nécessaire à leur exécution, le nombre d'entrée-sortie disque réalisé par chaque processus et la date, mesurée en temps CPU écoulé depuis le début d'exécution, des requêtes d'entrée-sortie.

N° processus	Temps total CPU (quantum)	Nombre d'accès disque	Dates des accès (tps CPU relatif)
1	5,5	1	1,5
2	5	0	—
3	1,5	2	0,5 puis 1

La durée d'un service disque est deux quantums et les demandes sont servies à l'ancienneté (FIFO).

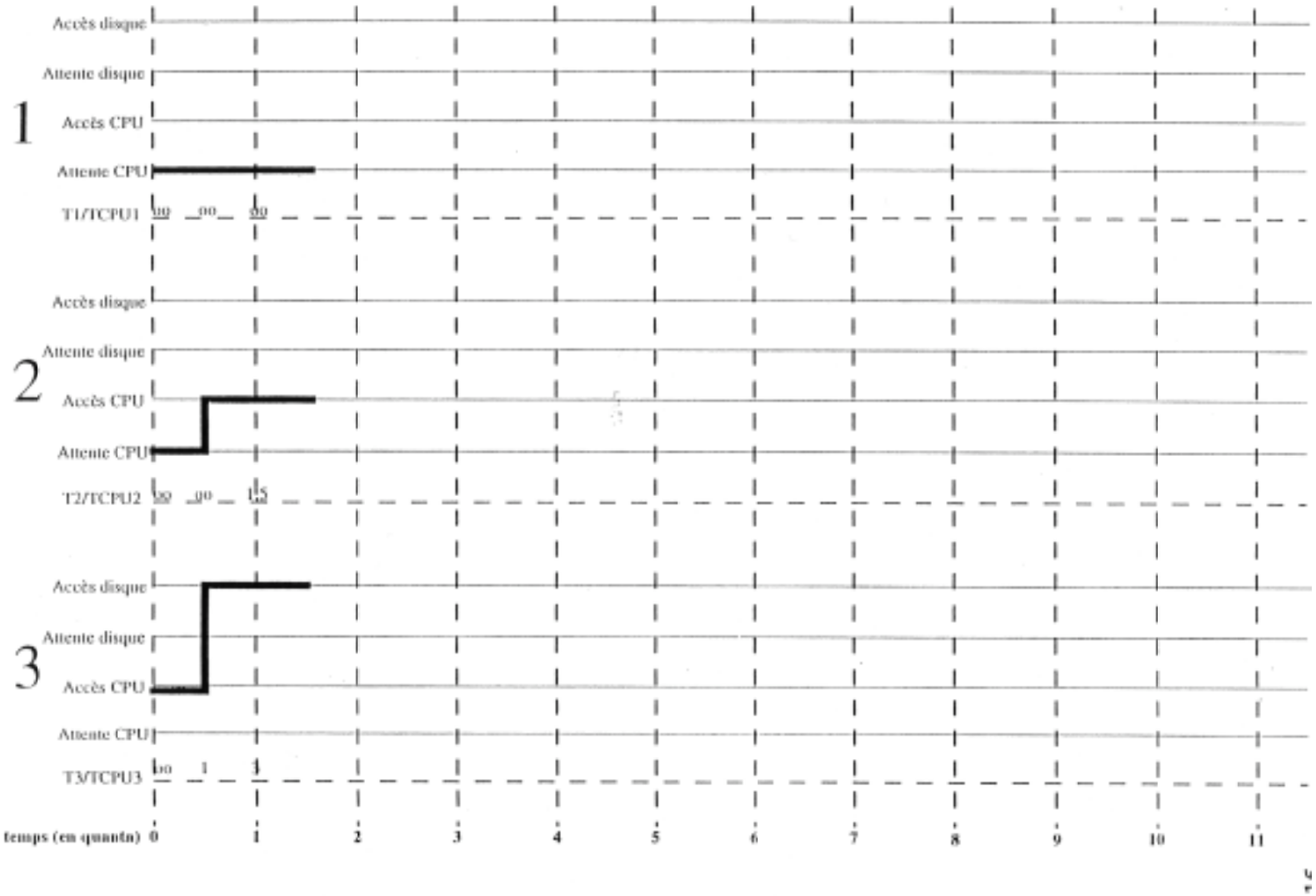
Question 1

Compléter le chronogramme en annexe qui donne l'état des différents processus à chaque instant ainsi que la valeur des rapports Ti/Tcpui lors de chaque changement d'état.

Note : la durée de fonctionnement de l'allocateur est négligée.

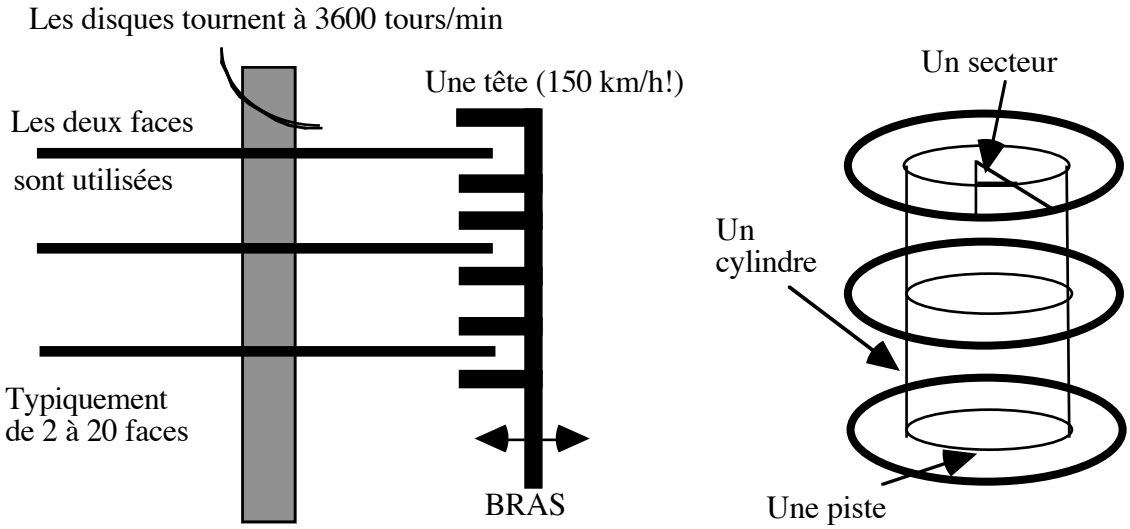
Question 2

Commenter le chronogramme en donnant les avantages et inconvénients de cette gestion du processeur.


1
Accès disque
Attente disque
Accès CPU
Attente CPU
T1/TCPU1
2
Accès disque
Attente disque
Accès CPU
Attente CPU
T2/TCPU2
3
Accès disque
Attente disque
Accès CPU
Attente CPU
T3/TCPU3
temps (en quanta) 0 1 2 3 4 5 6 7 8 9 10 11

ED 4
Gestion des disques et de la mémoire

Exercice 1⊡
Comparaison des politiques de gestion du disque



On se propose d'étudier diverses politiques de déplacement du bras, dans le service d'un ensemble de requêtes à un disque. A un instant donné, la file d'attente des requêtes est définie par le tableau suivant :

N° de cylindre demandé	20	16	13	40	12	1	11
Ordre d'arrivée	1	2	3	4	5	6	7

Supposant qu'aucune autre requête n'arrive pendant le service des requêtes en attente et que la position initiale du bras est sur le cylindre 15, on étudiera les politiques suivantes⊡service à l'ancienneté (FCFS), service sur le cylindre le plus proche (SSTF), service selon la stratégie de l'ascenseur (SCAN) sens initial montant.

Question 1

Rappeler le fonctionnement des différentes politiques.

Question 2

Les résultats des requêtes sont, par souci de cohérence, délivrés dans l'ordre d'arrivée de celles-ci.

Donner l'ordre de service des requêtes et le nombre de cylindres parcourus.

Pour chaque requête et pour les trois stratégies, donner en nombre de cylindres parcourus, la date de délivrance du résultat.

Exercice 2

Adressage dans une mémoire virtuelle paginée

Soit le programme suivant:

```
Program Machin;
chaine  : array[0..1999] of Byte;
i   : integer; -- i sur 2 octets
begin
    For i:= 0 to 1999 Do chaine[i]:= '*';
end;
```

Les instructions de ce programme tiennent sur 20 octets. Ce programme est exécuté sur une machine à mémoire paginée (taille de la mémoire réelle = 1 MégaOctets et taille d'une page = 512 octets). Les instructions à référence mémoire ont un champ adresse de 24 bits.

Question 1

a) Donner la taille de l'espace adressable, le nombre de bits du déplacement, le nombre de bits d'une adresse virtuelle, le nombre de bits d'une adresse réelle, le nombre de bits du numéro de page virtuelle, le nombre de bits du numéro de page réelle, le nombre d'entrées de la table des pages.

b) Le chargement de ce programme en mémoire engendre-t-il une fragmentation interne? Si non, pourquoi ? Si oui, la chiffrer.

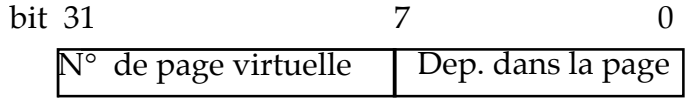
Les pages de ce programme (numérotées 0,1,2 et 3) sont respectivement implantées en mémoire dans les pages réelles 9, 17, 65 et 33.

c) Donner l'adresse virtuelle et l'adresse réelle correspondante [en décimal, sous la forme (numéro de page, déplacement)] générées lors de l'exécution de chacune des instructions suivantes:

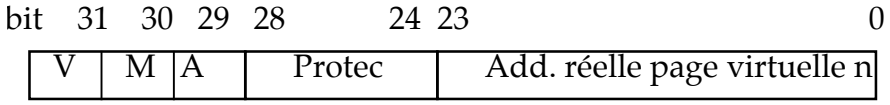
chaine[10]:= '*'; chaine[515]:= '*'; chaine[1026]:= '*'; chaine[1999]:= '*'

Mécanisme de gestion d'une mémoire virtuelle paginée

Dans un ordinateur, dont la mémoire est gérée selon le principe d'une mémoire virtuelle paginée, une adresse a la structure suivante :



A chaque processus est associée une table des pages dont la n-ième entrée a la structure suivante :



avec :

V : bit de validité = 1 si la page est résidente ; = 0 sinon

M : bit de modification = 1 si la page a été modifiée depuis son chargement ; = 0 sinon

A : bit de dernier accès ; mis à 1 à chaque accès à une page. Remis à 0 périodiquement par un processus système

Protec : mode de protection d'accès à la page.

Question 2

Expliquer l'utilité de ces différents champs.

Question 3

Décrire sous forme algorithmique les opérations réalisées par le matériel lors du décodage d'une adresse. (Faire abstraction de la gestion de la protection.)

Question 4

Quelle est a priori *la taille de la table des pages d'un processus? Rappeler les techniques qui sont mises en oeuvre pour réduire cette taille.*

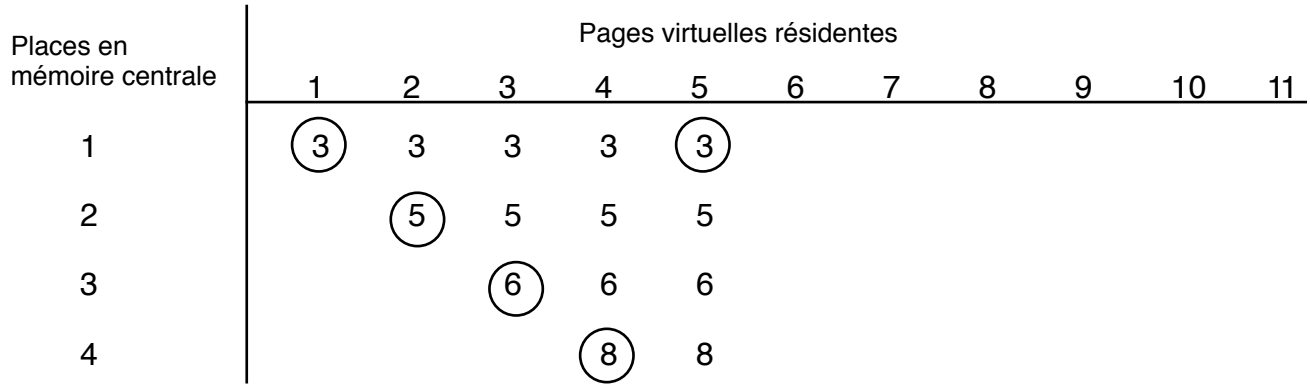
Question 5

Rappeler le fonctionnement des algorithmes de remplacement de pages FIFO ("First In-First Out"), c'est à dire à l'ancienneté de chargement, et LRU ("Least Recently Used"), c'est à dire à l'ancienneté de référence.

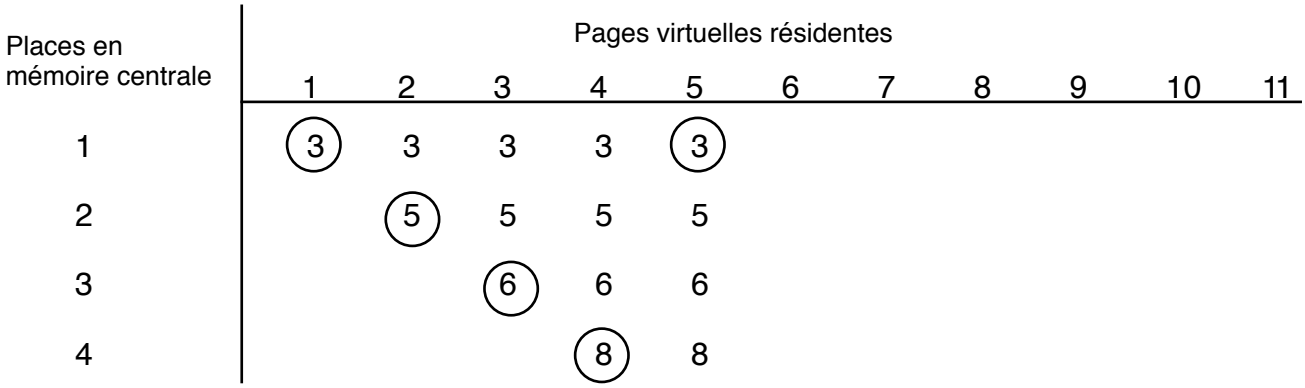
Soit la liste des pages virtuelles référencées aux instants t = 1, 2, …,11:

3, 5, 6, 8, 3, 9, 6, 12, 3, 6, 10

Sachant qu'il n'y a que 4 places en mémoire centrale, compléter les deux tableaux suivants représentant les pages présentes en mémoire centrale pour chaque politique.



Politique FIFO



Politique LRU

Comparer le nombre de défauts de pages provoqués , dans l'exemple, par les deux politiques.

Commenter le résultat trouvé.

Exercice 3 Gestion d'une mémoire par zones

On se propose de définir des algorithmes de gestion par zones d'une mémoire. Cette mémoire est gérée par un allocateur qui utilise les deux procédures ALLOUER_ZONE(T,A) et LIBERER_ZONE(T,A) où T est la taille de la zone et A l'adresse d'implantation en mémoire de la zone.

L'allocateur entretient une liste des zones non utilisées de la mémoire (appelées zones libres). Chaque zone libre comporte un en-tête de deux mots qui contient la taille de la zone et l'adresse de la zone libre suivante dans la liste. On suppose que cette liste est ordonnée suivant les adresses croissantes d'implantation en mémoire de ces zones.

Pour que l'allocateur puisse allouer une zone libre, il faut évidemment que cette zone soit de taille supérieure ou égale à celle qui est requise. D'autre part, quand la taille de la zone libre est supérieure à la taille demandée, le résidu est récupéré et forme une nouvelle zone libre.

Question 1

Représenter sous forme d'un schéma la structure de la liste des zones libres en mémoire.

Question 2

On suppose que l'algorithme d'allocation est "first fit", c'est-à-dire que la zone libre choisie est celle qui apparaît la première dans la liste.

Ecrire la procédure ALLOUER_ZONE.

Question 3

On suppose que l'algorithme d'allocation est "best fit", c'est-à-dire que la zone libre choisie est celle dont la taille est la plus proche de celle demandée.

Ecrire la procédure ALLOUER_ZONE.

Question 4

Quels sont les inconvénients du choix qui a été fait quant à l'ordre de la liste des zones libres ?

Comment y remédier ?

Question 5

On impose qu'il n'y ait pas de zones contiguës en mémoires. Lorsqu'une zone est libérée et qu'il s'avère qu'elle est contigüe à une autre zone libre, elles sont fusionnées pour ne plus en former qu'une.

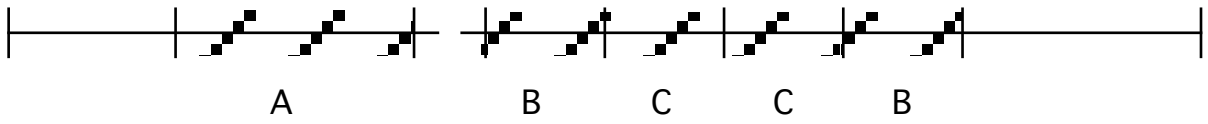
Ecrire la procédure LIBERER_ZONE.

Question 6

Soit N le nombre de zones allouées et M le nombre de zones libres.

Montrer que lorsque le système est en équilibre (c'est-à-dire que le nombre moyen de zones libres est constant) et que N est grand, on a la relation : $M \sim N/2$.

Les zones allouées peuvent être réparties en 3 classes : A, B, C. La classe A correspond aux zones allouées qui sont contigües à 2 zones libres. La classe B, celles qui n'ont qu'une zone libre à gauche ou à droite. La classe C regroupe les autres cas. On notera N_A, N_B, N_C, le nombre de zones de chacune de ces classes.



ED 5
Système multi-tâches sous UNIX

On se propose de voir comment les principes d'un système multi-tâches vus aux Exercices Dirigés précédents sont mis en œuvre par UNIX. On dispose pour cela de deux programmes⊡

`calcul`: qui effectue uniquement des calculs trigonométriques

`disque`: qui écrit uniquement un gros fichier sur le disque

Le source de ces deux programmes est donné ci-dessous. Les constantes numériques ont été choisies afin de rendre la durée de ces programme assez proche (30s environ sur un PC SX 25).

```
/*--- calcul.c ----*/
#include <math.h>
main(){
double s,x;
int i,j;

for (j=1;j<=28;j++){
 s=0;
 for(i=1;i<=50000;i++){
  x=sin(3.141592654*s);
  x=cos(3.141592654*s);
  x=tan(3.141592654*s);
  s+=0.1;
 }
}
}
```

```
/*--- disque.c ----*/
#include <fcntl.h>
#define BSIZE 8192
#define FSIZE 1000
char buf[BSIZE];
int fdes,i,j;
main(){
 for (j=0;j<6;j++){
 fdes=open("./fichiertest",O_WRONLY|O_CREAT,0777);
 for (i=0;i<FSIZE;i++) write(fdes,buf,BSIZE);
 close(fdes);
 }
}
```

Exercice 1⊡Gestion des processus

La commande `ps` permet de connaître quels sont les processus en cours de traitement. L'option ps -u affiche (entre autre) les informations suivantes⊡

- USER : l'utilisateur qui a lancé le processus
- PID : le Process IDentification number. Chaque processus a son propre PID.
- %CPU, %MEM : Le taux d'utilisation de la CPU et de la mémoire pour ce processus
- STAT: Le statut du processus.
- TIME : la quantité totale de temps CPU consommé

Les cas les plus courants de STAT sont⊡

R En cours d'exécution (ie, dans la "run queue" - pas en attente)

S Le processus est dormant ("sleeping") depuis moins de 20s

I Le processus est dormant ("idle") depuis plus de 20s

(S et I sont souvent en attente de saisie au clavier.)

Question 1

L'interpréteur des commande permet de lancer des commandes en "arrière plan", ce qui permet à l'utilisateur de lancer une nouvelle commande sans attendre la fin de la première. Il suffit pour cela de terminer la commande par le caractère `&.` On peut également lancer plusieurs commandes en séquence en les séparant par un point-virgule.

Lancer en parallèle 3 processus `calcul` par la commande⊡

```
calcul&;calcul&;calcul&
```

puis lancer `ps -u` régulièrement (la commande !! permet de repéter la dernière commande sans la retaper)

Interpréter les résultats affichés

Question 2

On souhaite voir l'évolution du statut d'un processus (avec le champs `STAT` de `ps`).

Lancer un processus calcul, puis l'interrompre avec `CTRL-Z.`

Que devient son statut ?

Lancer la commande `bg`

Que se passe-t'il ?

Détruisez le processus avec la commande `kill -9` suivie du PID du processus

Question 3

L'option -`aux` de `ps` permet de visualiser tous les processus en cours de traitement sur la machine.

Commenter cette liste en identifiant les processus système

Exercice 2 : Parallélisme calcul et E/S

La commande `time` permet de connaître le temps d'exécution d'un programme. La version de `time` utilisée ici affiche dans l'ordre les résultats suivants⊡

- U: (user) temps CPU pour le processus (précision 0.5s)
- S: (system) temps CPU que le noyau a consacré au processus (précision 0.5s)
- E: (elapsed) temps total d'exécution du processus (précision 1s)
- P: rapport (U+S)/E en pourcentage (parfois >100% du fait de l'imprécision)
- X+D: mémoire partagée (X) ou non (D) utilisée en moyenne par le processus (en Ko)
- I+O: nombre de lectures (I) et d'écritures (0) de blocs sur le disque
- pF: nombre de défauts de page

Question 1

Lancer `time calcul`

Interpréter les résultats affichés par `time`. Répéter une ou deux fois l'opération. Que constate-t'on ?

Question 2

Lancer `calcul&;time calcul`

Interpréter les résultats. Calculer le taux d'occupation de la CPU

Question 3

Lancer `time disque`

Calculer le temps passé en E/S

Question 4

effacer le fichier produit et lancer `time calcul&;time disque &`

Interpréter les résultats

Exercice 3□L'écroulement (thrashing)

Les programmes `gentil et mechant` ci-dessous utilisent une matrice de 800*4096 caractères (codés sur un octet), soit 3.2 Mo.

```
/*--- gentil.c ---- */
#define NBRE 2000
#define PAGE 4096
char t[NBRE][PAGE];
main(){
int j,k;
 for (j=0;j<NBRE;j++)
  for (k=0;k<PAGE;k++) t[j][k]=1;
}
```

```
/*--- mechant.c ---- */
#define NBRE 2000
#define PAGE 4096
char t[NBRE][PAGE];
main(){
int j,k;
 for (k=0;k<PAGE;k++)
  for (j=0;j<NBRE;j++) t[j][k]=1;
}
```

Si la mémoire physique libre est limitée à 8 Mo, le système ne pourra pas gérer plus de 2 ou 3 processus sans faire fréquemment appel à la mémoire virtuelle. Quand toute la mémoire physique est occupée, le gestionnaire de mémoire virtuelle doit choisir des pages (ici, des morceaux de 4096 octets) à retirer, qu'il recopie sur le disque. En gros, c'est la page la moins récemment utilisée qui est choisie (algorithme LRU).

Question 1

Déterminer à l'aide de la commande `free`, la quantité de mémoire physique disponible et la taille de l'espace disque disponible pour la mémoire

virtuelle. (variable selon le type de PC utilisé)

En déduire le nombre de processus MAX pouvant s'exécuter sans appels fréquents à la mémoire virtuelle.

Question 2

La commande `vmstat` permet de voir l'évolution des processus et du système dans le temps. Elle affiche à l'intervalle de temps choisi divers paramètres utiles pour cet exercice⊡

- r : nombre de processus "runnable" (`vmstat` lui-même n'est pas compté)
- b: nombre de processus bloqués en attente disque
- fre: mémoire libre disponible (en Ko)
- si: "swap in" lecture de pages sur le disque (en Ko/s)
- so: "swap out" écriture de pages sur le disque (en Ko/s)
- id: "idle" pourcentage de temps pendant lequel la CPU est inutilisée

Lancer MAX+1 processus `gentil` en parallèle avec la commande vmstat 1 (le 1 signfie 1 affichage par seconde)

Commenter l'évolution des champs r, b, free, si, so, id

Calculer le taux d'occupation du CPU (cf Exercice 2, question 2)

Question 3

Idem avec MAX+1 processus `mechant`

Pourquoi y-a t'il de telles différences ?

Question 4

Idem avec MAX+2 processus `gentil` puis `mechant`

Que constate-t'on?

ED 5
Complément

Exercice 1□Gestion des processus

Question 1

```
> ps -u
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907 12.6  1.9  157  612 pp3 S    13:27   0:00 -sh
cubaud     908  0.0  1.0   55  336 pp3 R    13:27   0:00 ps -u
```

On lance 3 calculs en parallèle□

```
> ca&;ca&;ca&
[1] 909
[2] 910
[3] 911

> ps -u
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907  3.2  1.9  157  612 pp3 S    13:27   0:00 -sh
cubaud     909 63.5  0.5    7  176 pp3 R    13:28   0:01 ca
cubaud     910 63.0  0.5    7  176 pp3 R    13:28   0:01 ca
cubaud     911 62.5  0.5    7  176 pp3 R    13:28   0:01 ca
cubaud     912  0.0  1.0   59  340 pp3 R    13:28   0:00 ps -u

> !!
ps -u
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907  2.4  1.9  157  616 pp3 S    13:27   0:00 -sh
cubaud     909 39.5  0.5    7  176 pp3 R    13:28   0:02 ca
cubaud     910 38.5  0.5    7  176 pp3 R    13:28   0:02 ca
cubaud     911 39.2  0.5    7  176 pp3 R    13:28   0:02 ca
cubaud     913  0.0  1.0   59  340 pp3 R    13:28   0:00 ps -u
```

Question 2

```
> ca
**** taper ici CTRL Z *****
Suspended
> ps -u
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907  0.1  2.3  209  748 pp3 S    13:27   0:02 -sh
cubaud     991 59.2  0.5    7  176 pp3 T    14:00   0:02 ca
cubaud     992  0.0  1.0   55  336 pp3 R    14:00   0:00 ps -u

> bg
[1]    ca &
```

bg a pour effet de reprendre la dernière commande suspendue en arrière plan. C'est une astuce utile si on a lancé une commande en oubliant de la faire suivre par un &.

```
> ps -u
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907  0.1  2.3  209  748 pp3 S    13:27   0:02 -sh
cubaud     991 50.2  0.5    7  176 pp3 R    14:00   0:08 ca
cubaud     993  0.0  1.0   55  336 pp3 R    14:01   0:00 ps -u
> kill -9 991
```

Question 3

```
> ps -aux
USER       PID %CPU %MEM SIZE  RSS TTY STAT START   TIME COMMAND
cubaud     907  0.0  2.3  209  748 pp3 S    13:27   0:02 -sh
cubaud    1004  0.0  1.0   63  344 pp3 R    14:24   0:00 ps -aux
root         1  0.0  1.0   56  324  ?  S    11:01   0:00 init [5]
root         6  0.7  0.2   27   72  ?  S    11:01   1:32 (update)
root         7  0.0  0.3   27  112  ?  S    11:01   0:03 update (bdflush)
root        52  0.0  0.7   45  224  ?  S    11:01   0:00 /usr/sbin/crond -l10
root        66  0.0  0.6   49  212  ?  S    11:01   0:00 /usr/sbin/syslogd
root        68  0.0  0.2   40   84  ?  S    11:01   0:00 /usr/sbin/klogd
root        72  0.0  0.4   62  152  ?  S    11:01   0:00 /usr/sbin/inetd
root        74  0.0  0.4   31  148  ?  S    11:01   0:00 /usr/sbin/lpd
root        78  0.0  0.5   75  164  ?  S    11:01   0:00 /usr/sbin/rpc.mountd
root        80  0.0  0.2   84   92  ?  S    11:01   0:00 /usr/sbin/rpc.nfsd
root        96  0.0  0.7   48  220 v03 S    11:01   0:00 /sbin/agetty 38400 tty3
root       899  0.0  1.3   62  424  ?  S    13:25   0:02 in.telnetd
>
```

Exercice 2 : Parallélisme calcul et E/S

Question 1

```
> time ca
23.040u 0.110s 0:23.15 100.0% 0+0k 0+0io 19pf+0w
> !!
time ca
23.050u 0.110s 0:23.16 100.0% 0+0k 0+0io 19pf+0w
> !!
time ca
23.060u 0.090s 0:23.15 100.0% 0+0k 0+0io 19pf+0w
```

Question 2

```
> ca &;time ca
[1] 1012
23.760u 0.160s 0:48.19 49.6% 0+0k 0+0io 15pf+0w
[1]  + Exit 32                ca
>
```

Question 3

```
> time di
0.040u 10.110s 0:23.26 43.6% 0+0k 0+0io 15pf+0w
> rm fich*
> time di
0.050u 10.120s 0:23.94 42.4% 0+0k 0+0io 15pf+0w
> rm fich*
> time di
0.130u 9.770s 0:23.34 42.4% 0+0k 0+0io 15pf+0w
```

Question 4

```
> time ca&;time di&
[1] 1203
[2] 1204
>
[2]  + Done                   di
0.030u 9.110s 0:31.11 29.3% 0+0k 0+0io 17pf+0w
[1]  + Exit 32                ca
24.300u 1.100s 0:47.22 53.7% 0+0k 0+0io 21pf+0w
```

Exercice 3□L'écroulement (thrashing)

Question 1

```
> free
             total       used       free     shared    buffers
Mem:         31376      30608        768      21688      16756
-/+ buffers:            13852      17524
Swap:        66020        732      65288
>
```

Question 2

```
> ge&;ge&;ge&;vmstat 1
*** je retire ici l'affichage résultant ***
> !!
> ge&;ge&;ge&;vmstat 1
[1] 1405
[2] 1406
[3] 1407
 procs                  memory    swap        io    system         cpu
 r b w   swpd   free   buff  si   so   bi    bo   in   cs  us  sy  id
 3 0 0   8624 17636   5204   0    1    2    66  253   33   9   5  87
 3 0 0   8624 15232   5204   0    0    0     0  117   12  83  17   0
 3 0 0   8624 12856   5204   0    0    0     0  125   13  82  18   0
 3 0 0   8624 10508   5204   0    0    0    26  174   14  84  16   0
 3 0 0   8624  8104   5204   0    0    0     0  122   13  87  13   0
 3 0 0   8624  5708   5204   0    0    0     0  126   11  84  16   0
 3 0 0   8624  3308   5204   0    0    0     0  122   12  86  14   0
 3 0 0   8624   920   5204   0    0    0     2  128   15  85  15   0
 3 0 0   8632   512   3576   0   60    0     0  243   27  72  27   1
 3 0 0   9120   512   3236   0  592    0     0 1318  161  37  44  18
 3 0 0   9828   512   2980   4  708    0     0 1556  197  29  55  15
 0 0 0   9372 23460   2892  44  304    1     0  841  125  44  27  29
 0 0 0   9364 23416   2908  28    0   16     0  213   31   5   5  89
 0 0 0   9364 23416   2908   0    0    0     0  121    6   5   3  92
*** taper CTRL-C pour arrêter vmstat quand tout est fini (r=0) ***
```

Question 3

```
> me&;me&;me&;vmstat 1
[1] 1420
[2] 1421
[3] 1422
 procs                  memory    swap        io    system         cpu
 r b w   swpd   free   buff  si   so   bi    bo   in   cs  us  sy  id
 3 0 0   9044 14052   3408   0    1    2    65  252   33   9   5  86
 3 0 0   9116   512   3408   0   72    0     0  266   35  32  68   0
 3 0 0   9352   512   3172   0  336    0     0  798  118  88  13   0
 3 0 0   9764   512   2988   0  416    0     0  972  163  89  11   0
 3 0 0  10140   512   2548   0  388    0     0  912  119  81  19   0
 3 0 0  10020   512   2356 160   40    0     0  525  102  95   5   0
 3 0 0  10020   512   2352  12    8    4     5  184   25  94   6   0
 3 0 0  10020   512   2352   0    0    0     0  122   12  93   7   0
 3 0 0  10020   512   2352   0    0    0     0  131   13  95   5   0
 3 0 0  10028   512   2352   0    8    0     0  142   13  96   4   0
 3 0 0  10060   512   2364  24   36   12     0  267   47  94   6   0
 3 0 0  10060   512   2364   0    0    0     0  121   13  95   5   0
 3 0 0  10060   512   2364   0    0    0     0  123   12  95   5   0
 3 0 0  10072   516   2364   4   12    0     0  151   18  96   4   0
 1 0 0   9996 16584   2384  56   16   24     0  318   56  86  14   0
 0 0 0   9996 24676   2384   0    0    0     0  133   10  28   8  64
 0 0 0   9996 24676   2384   0    0    0     0  117    6   5   3  92
```

Question 4

```
> ge&;ge&;ge&;ge&;vmstat 1
[1] 1425
[2] 1426
[3] 1427
[4] 1428
 procs            memory    swap        io    system         cpu
 r b w  swpd  free  buff  si  so   bi   bo   in   cs  us  sy  id
 4 0 0  9840 21180  2728   0   1    2   64  251   32   9   5  87
 4 0 0  9840 18808  2728   0   0    0    0  132   12  86  14   0
 4 0 0  9840 16064  2728   0   0    0    0  145   16  84  16   0
 4 0 0  9840 13688  2728   0   0    0    0  144   13  87  13   0
 4 0 0  9840 11300  2728   0   0    0    0  123   13  85  15   0
 4 0 0  9840  8916  2728   0   0    0    0  135   11  89  11   0
 4 0 0  9840  5996  2728   0   0    0    5  165   15  82  18   0
 4 0 0  9840  3620  2728   0   0    0    0  137   13  84  16   0
 4 0 0  9840  1232  2728   0   0    0    0  133   12  90  10   0
 4 0 0 10036   512  2072   0 228    0    0  584   83  61  24  15
 4 0 0 10680   512  1732  16 648   16    0 1492  165  43  48   9
 4 0 0 11476   512  1668  28 816   20    0 1880  236  29  44  27
 4 0 0 12288   512  1568   4 816    0    0 1762  180  35  62   3
 4 0 0 13244   512  1560  16 956    0    0 2081  207  38  57   5
 4 0 0 14108   512  1552   0 880    0    0 1880  177  37  63   0
 4 0 0 14780   512  1556  28 696    4    0 1589  196  30  46  24
 4 0 0 15712   512  1548  16 932    0    0 2035  203  33  66   1
 4 0 0 16540   512  1548   8 852    0    0 1861  183  34  64   3
 0 0 0 10452 25964  1576  68  20   28    0  353   63  28  19  53
 0 0 0 10452 25964  1576   0   0    0    0  119    6   5   3  92
```

La différence de comportement de mechant est maintenant assez spectaculaire. Le thrashing est proche !!

```
> me&;me&;me&;me&;vmstat 1
[1] 1434
[2] 1435
[3] 1436
[4] 1437
 procs            memory    swap        io    system         cpu
 r b w  swpd  free  buff  si  so   bi   bo   in   cs  us  sy  id
 4 0 0  9996  7408  2376   0   1    2   64  251   32   9   5  86
 4 0 0 10232   512  1956   8 348    0    0  861  165   7  56  37
 4 0 0 10788   512  1912  32 564   56    0 1453  237   6  38  57
 2 2 0 11380   512  1864  64 652    4    0 1583  240   4  37  58
 0 4 0 11860   512  1804 108 596    4    0 1595  315   6  35  59
 0 4 0 12344   512  1720  92 576    0    0 1497  266   4  34  63
 0 4 0 12900   512  1700 108 652    0    0 1711  302   4  31  65
 4 0 0 13364   512  1704  32 488    4    5 1207  149  58  30  12
 5 0 1 14700   512  1648  28 1356   40    0 3030  333  43  57   0
 1 3 0 15240   512  1692 104 608   52    0 1726  270  84  16   0
 4 1 0 16040   512  1708  48 832   12    0 1986  277  55  45   0
 1 3 0 16544   512  1712  76 604   12    0 1550  257  73  25   2
 4 0 0 16820   512  1712  72 320    0    0  967  205  84  16   0
 0 4 0 16960   512  1712 128 248    0    0  890  174  33   7  60
 0 5 0 16836  7672  1732 164  40   20    0  565  124   5  16  79
 0 4 0 16652  7432  1752 220   0   20    0  634  162   0  14  86
 0 3 0 16460  7208  1772 200   0   20    0  569  119   7  10  83
 0 3 0 16172  6920  1772 288   0    0    0  704  153   5  10  85
 0 3 0 15872  6620  1772 304   0    0    0  740  172   7   7  86
 0 3 0 15588  6336  1772 280   0    0    0  684  146   4   8  88
 0 3 0 15324  6072  1772 268   0    0    0  679  186   5  11  83
 procs            memory    swap        io    system         cpu
 r b w  swpd  free  buff  si  so   bi   bo   in   cs  us  sy  id
 0 4 0 15080  5804  1784 260   0   12    0  661  147   5  10  84
 0 3 0 14784  5500  1784 304   0    0    0  739  159   5  11  84
 0 3 0 14480  5196  1784 300   0    0    0  721  159   4   8  88
 0 3 0 14160  4876  1784 320   0    0    0  767  166   7   8  85
 0 3 0 13788  4504  1784 376   0    0    0  876  192   7  12  81
 0 3 0 13488  4204  1784 296   0    0    0  715  152   7  11  82
```

```
0 3 0 13188  3904  1784 300   0    0    0  725  159   6   8  86
0 3 0 12892  3608  1784 296   0    0    0  743  177   5   9  86
0 3 0 12592  3308  1784 300   0    0    0  741  201   7  10  83
0 3 0 12300  3016  1784 296   0    0    0  749  209   6  10  84
0 4 0 12016  2728  1792 276   0    8    0  679  147   5   9  86
0 5 0 11816  2468  1808 248   0   16    0  645  139   7   7  86
0 5 0 11648  2212  1832 232   0   24    0  641  136   3  14  83
0 3 0 11404  1940  1836 260   0    4    0  672  143   4   9  87
0 3 0 11140  1676  1836 268   0    0    0  661  139   5   6  89
3 0 0 10888  1428  1836 244   0    0    0  632  127  88  12   1
3 0 0 10632  1172  1836 256   0    0    0  640  129  85  15   0
3 0 0 10404   944  1836 228   0    0    0  576  116  88  12   0
3 0 0 10368   916  1836  28   0    0    0  180   21  96   4   0
3 0 0 10368   912  1836   0   0    0    0  124   12  96   4   0
3 0 0 10368   912  1836   0   0    0    0  122   13  94   6   0
procs            memory    swap        io    system         cpu
r b w  swpd  free  buff  si  so   bi   bo   in   cs  us  sy  id
4 0 0 10368   912  1836   0   0    0    0  142   12  90  10   0
3 0 0 10368   912  1836   0   0    0    0  134   12  96   4   0
2 0 0 10368  8992  1836   0   0    0    0  129   14  96   4   0
2 0 0 10368  8988  1836   0   0    0    0  140   14  95   5   0
2 0 0 10368  8984  1836   0   0    0    0  132   12  95   5   0
2 0 0 10368  8984  1836   0   0    0    0  131   13  95   5   0
0 0 0 10368 25160  1836   0   0    0    0  127   14  81   8  11
0 0 0 10368 25160  1836   0   0    0    0  122    6   5   5  91
```

ED 6
Interblocage

Il s'agit de montrer sur des exemples simples comment en utilisant les propriétés des états sains et fiables, on peut démontrer la présence ou non d'interblocage.

Exercice 1

Un système S accepte cinq utilisateurs, S = {U1, U2, U3, U4, U5}. Chaque utilisateur peut déclarer des processus, des fichiers, des tampons (qui sont utitlisés pour les entrées-sorties de fichiers : un tampon par fichier ouvert) et des messages.

On veut étudier l'utilisation des ces quatre classes de ressources, lors de divers phases du système, et pour cela on relève certains compteurs à différents instants:

à l'instant t :

- l'utilisateur U1 a obtenu 5 processus, 30 fichiers, 10 tampons et 30 messages, et il demande 10 processus, 100 fichiers, 30 tampons et 40 messages.
- l'utilisateur U2 a obtenu 5 processus, 10 fichiers, 10 tampons et 5 messages, et il demande 30 processus, 150 fichiers, 40 tampons et 40 messages.
- l'utilisateur U3 a obtenu 10 processus, 50 fichiers, 5 tampons et 5 messages, et il demande 20 processus, 60 fichiers, 10 tampons et 10 messages.
- l'utilisateur U4 a obtenu 10 processus, 10 fichiers, 10 tampons et 5 messages, et il demande 20 processus, 60 fichiers, 15 tampons et 5 messages.
- l'utilisateur U5 a obtenu 10 processus, 50 fichiers, 10 tampons et 5 messages, et il demande 20 processus, 60 fichiers, 15 tampons et 5 messages.

à l'instant v :

- l'utilisateur U1 a obtenu 10 processus, 20 fichiers, 5 tampons et 5 messages, et il demande 20 processus, 70 fichiers, 20 tampons et 15 messages.
- l'utilisateur U2 a obtenu 10 processus, 20 fichiers, 5 tampons et 15 messages, et il demande 40 processus, 100 fichiers, 50 tampons et 40 messages.
- l'utilisateur U3 a obtenu 10 processus, 20 fichiers, 5 tampons et 5 messages, et il demande 30 processus, 150 fichiers, 10 tampons et 40 messages.
- l'utilisateur U4 a obtenu 10 processus, 20 fichiers, 5 tampons et 10 messages, et il demande 20 processus, 40 fichiers, 10 tampons et 30 messages.
- l'utilisateur U5 a obtenu 10 processus, 20 fichiers, 10 tampons et 5 messages, et il demande 10 processus, 40 fichiers, 30 tampons et 15 messages.

On précise que la demande d'un utilisateur est sa demande totale, à l'instant considéré, qu'elle inclut les ressources qui lui ont déjà été attribuées, qu'un utilisateur qui ne recoit pas toutes ses demandes est bloqué en attente, et que le système comporte au maximum 50 processus, 200 fichiers, 50 tampons et 50 messages.

Question 1

Notation :

- le vecteur X représente le nombre total de ressources dans chaque classe,
- le vecteur R représente les ressources non allouées dans chaque classe,

- la matrice A représente les ressources de chaque classe allouées à chaque utilisateur,
- la matrice D représente les ressources de chaque classe demandées par chaque utilisateur.

Présenter les états du système aux instants t et v du modèle faisant intervenir les vecteurs X et R, ainsi que les matrices A et D

Déterminer s'il y a interblocage à l'instant t ou à l'instant v.

Question 2

Pour prévenir l'interblocage, on décide de fixer la demande maximale de chaque utilisateur à 30 processus, 150 fichiers, 40 tampons et 40 messages.

Déterminer si les états du système aux instants t et v sont fiables

Pour répondre aux demandes, on envisage de placer le système à l'instant w dans l'état suivant à l'instant w :

- l'utilisateur U1 a obtenu 5 processus, 10 fichiers, 2 tampons et 0 message,
- l'utilisateur U2 a obtenu 5 processus, 10 fichiers, 3 tampons et 0 messages,
- l'utilisateur U3 a obtenu 5 processus, 20 fichiers, 2 tampons et 0 messages,
- l'utilisateur U4 a obtenu 5 processus, 40 fichiers, 10 tampons et 5 messages,
- l'utilisateur U5 a obtenu 5 processus, 10 fichiers, 3 tampons et 10 messages.

Déterminer si cet état est fiable

Question 3

Pour éviter plus facilement l'interblocage, on décide de limiter la demande de l'utilisateur à 10 processus, 40 fichiers, 10 tampons et 10 messages. Dans ce cas, il n'y a jamais pénurie de ressources.

Comme cela semble trop restrictif, quelqu'un propose de réorganiser le système pour qu'il comporte 51 processus, 200 fichiers, 50 tampons et 50 messageset ensuite de limiter la demande de chaque utilisateur à 11 processus, 40 fichiers, 10 tampons et 10 messages.

Déterminer si l'interblocage est possible (en envisageant le cas le plus défavorable).

Une autre proposition est de réorganiser le système pour qu'il comporte 51 processus, 201 fichiers, 50 tampons et 50 messageset ensuite de limiter la demande de chaque utilisateur à 11 processus, 41 fichiers, 10 tampons et 10 messages.

Déterminer si l'interblocage est possible

Exercice 2

Quatre processus P1, P2, P3 et P4 se partagent 50 granules sur un disque. Chacun garde les granules acquis et attend l'arrivée de sa nouvelle demande avant de continuer. Il n'y a pas de réquisition.

- P1 demande successivement 1 granule, puis 4, puis 3, puis 6, puis 7, puis 1, puis rend les 22 granules acquis.
- P2 demande successivement 4 granules, puis 3, puis 9, puis 5, puis 5, puis rend les 26 granules acquis.
- P3 demande successivement 2 granules, puis 3, puis 4, puis 7, puis 2, puis 2, puis 1, puis rend les 21 granules acquis.

- P4 demande successivement 7 granules, puis 2, puis 1, puis 2, puis 2, puis 9, puis rend les 23 granules acquis.

Question1

La situation à t1 est la suivante⊡

R(t1) = 6

A(t1) = (14, 7, 9, 14)

D(t1) = (21, 16, 16, 23)

Montrer que l'état correspond à un interblocage du système :

Donner un autre exemple d'état t2 d'interblocage du système.

Question 2

On prend la valeur de 26 granules comme annonce maximale pour chacun des processus du système. En garantissant au processus le mieux pourvu en granules la possibilité d'atteindre cette valeur maximale, on peut simplifier la politique de prévention dynamique de l'interblocage.

Rappeler cette politique et expliquer pourquoi cette simplification est valable.

L'allocateur est confronté dans l'état t3 aux demandes D(t3) :

R(t3) = 18

A(t3) = (14, 7, 3, 8)

D(t3) = (14, 16, 9, 16)

C - A(t3) = (12, 19, 23, 18)

S' il applique cette politique (simplifiée), que doit répondre l'allocateur ?

L'allocateur peut-il satisfaire n'importe lequel des processus demandeurs (autre que P1) en étant assuré de ne jamais laisser le système aller vers un état d'interblocage ? Examiner chaque processus.

ED 7
Programmation concurrente avec le langage Ada

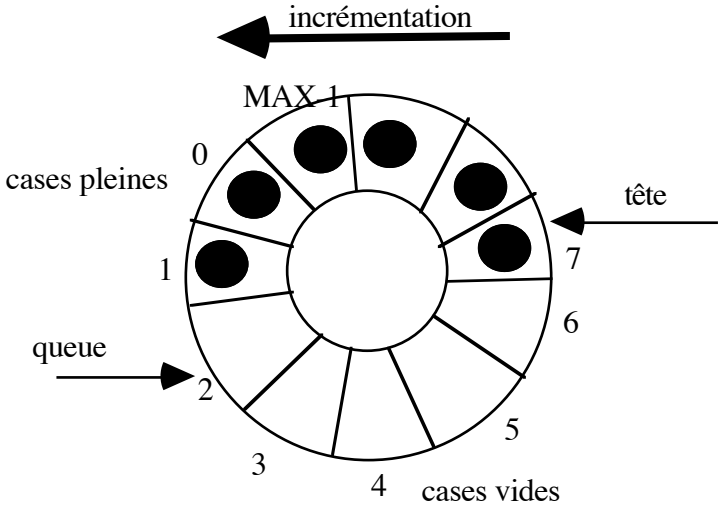
Revue des caractéristiques du langage Ada à partir d'un petit exemple□

- Les structures de données, les sous-programmes (procédures et fonctions) et les instructions de base
- Les paquetages et la généricité
- Les tâches

Exercice□Gestion et utilisation d'un tampon de messages

Deux opérations seulement sont autorisées sur le tampon : déposer un message et retirer un message. Seuls le type `message` (structure d'un élément du tampon) et les spécifications des deux opérations `déposer` et `retirer` sont connues de l'extérieur. Tous les détails de gestion□nom, structure du tampon, sa taille, … doivent rester cachés.

On ne se préoccupe pas à ce stade des problèmes de synchronisation vis à vis du tampon.



Question 1

Ecrire un paquetage Ada effectuant la gestion d'un tampon géré de façon circulaire et pouvant contenir au maximum 20 messages.

Question 2

Ecrire une procédure Ada utilisant le paquetage de la question 1 et dont le but est de lancer en parallèle deux tâches. Le rôle de la première tâche est de préparer les messages et de les déposer, celui de la deuxième est de retirer les messages et de les traiter.

Question 3

En fait les tampons peuvent contenir autre chose que des messages (des entiers, des réels, etc.) et être de taille variable.

Comment modifier la solution de la question 1 pour que le paquetage puisse être utilisé quelque soit le type des éléments du tampon et sa taille ?

Annexe⊡programmes Ada pour la question 2

```
-------------------------------------------------------------
-- paquetage de gestion du tampon
-------------------------------------------------------------

-- partie spécification

package TAMPON is
type MESSAGE is string(1..100);
procedure DEPOSER(M : in MESSAGE; OK : out boolean);
procedure RETIRER(M : out MESSAGE; OK : out boolean);
end TAMPON;

-- corps du paquetage

package body TAMPON is
MAX : constant := 20;
TETE, QUEUE : integer range 0.. MAX-1;
NBMES : integer range 0..MAX;
TAMP : array (0..MAX-1) of MESSAGE;

procedure DEPOSER(M : in MESSAGE; OK : out boolean) is
begin
     --- A COMPLETER ! ---
end DEPOSER;

procedure RETIRER(M : out MESSAGE; OK : out boolean) is
begin
     --- A COMPLETER ! ---
end RETIRER;

begin
     -- initialisation des variables du paquetage
     NBMES:=0; TETE:=0; QUEUE:=0;
end TAMPON;
```

```
------------------------------------------------------------
-- programme utilisateur du paquetage, sans synchronisation !!
------------------------------------------------------------

with TAMPON; use TAMPON;
procedure DEPOT-RETRAIT is

    task PRODUCTEUR is
    task body PRODUCTEUR is
    M : MESSAGE; OK : boolean;
    begin
      loop
      PREPARER(M);
      DEPOSER(M, OK);
      if not OK then -- action spécifique du processus
      end if;
      endloop;
    end PRODUCTEUR;

    task CONSOMMATEUR is
    task body CONSOMMATEUR is
    M : MESSAGE; OK : boolean;
    begin
      loop
      RETIRER(M, OK);
      if not OK then -- action spécifique du processus
      end if;
      UTILISER(M);
      endloop;
    end CONSOMMATEUR;

begin -- activation des deux tâches en parallèle
null;
end;
end DEPOT-RETRAIT;
```

ED 8
Exclusion mutuelle par variables communes

L'objectif de l'ED est de construire une solution respectant certaines propriétés décrites ci-dessous. L'aboutissement est l'algorithme de Dekker (étendu à N processus par Dijkstra). La construction est progressive, les premières questions aboutissent à des solutions ne satisfaisant qu'à une partie des propriétés.

Rappels

• Définitions⊡

- Ressource critique : une ressource partageable à un seul point d'accès (qui ne peut être attribuée qu'à un seul instant donné) est dite ressource critique.
- Section critique : la phase d'utilisation par un processus d'une ressource critique est dite section critique.

• Hypothèses⊡

- Les vitesses des processus sont quelconques et inconnues
- Tout processus sort de la section critique au bout d'un temps fini.

• La solution doit comporter les propriétés suivantes :

a) A tout instant un processus au plus peut se trouver en section critique.

b) Si plusieurs processus sont bloqués en attente de la ressource critique, alors qu'aucun processus ne se trouve en section critique, l'un d'eux doit pouvoir y entrer au bout d'un temps fini (pas de blocage mutuel indéfini).

c) Si un processus est bloqué hors d'une section critique ce blocage ne doit pas empêcher l'entrée d'un autre processus en section critique.

d) La solution doit être la même pour tous les processus (aucun processus ne doit jouer de rôle privilégié).

• La programmation de l'exclusion mutuelle se décompose en trois parties :

- entrée
- section critique
- sortie

L'entrée et la sortie assurent le respect des propriétés a), b), c) et d).

Exercice 1⊡Algorithme de Dekker

On se propose de programmer l'exclusion mutuelle entre deux processus parallèles pour l'accès à une section critique : les seules opérations indivisibles sont l'affectation d'une valeur à une variable et le test de la valeur d'une variable.

Principe de la solution⊡

- Définir un ensemble de variables d'état, communes aux contextes des deux processus
- L'autorisation d'entrée en Section Critique sera définie par des tests sur ces variables, et

l'attente éventuelle sera programmée comme une attente active (répétition cyclique des tests).

Question 1

On utilise une seule variable booléenne C telle que C = vrai si un des processus se trouve dans sa section critique, faux sinon.

Ecrire le programme

Vérifier que l'exclusion mutuelle ne peut être ainsi programmée.

Question 2

On utilise une variable commune unique T telle que T = i si et seulement si le processus Pi est autorisé à entrer en sa section critique (i = 1, 2).

Écrire le programme d'un processus.

Vérifier que la solution ne vérifie pas la condition c) (le blocage d'un processus hors de sa section critique peut empêcher l'autre d'entrer en sa section critique) mais vérifie les autres conditions.

Question 3

On utilise c(i) variable booléenne attachée au processus Pi (i = 1, 2)

c(i) = vrai si Pi est dans sa section critique ou demande à y entrer

c(i) = faux si Pi est hors de sa section critique

Pi peut lire et modifier c(i), peut lire seulement c(j) si j différent de i.

Écrire le programme du processus Pi

Vérifier qu'on ne peut obtenir qu'une solution satisfaisant aux conditions a), c), d) ou b), c), d) mais non aux quatre.

Question 4

On peut obtenir une solution correcte en combinant les solutions précédentes et en introduisant une variable supplémentaire T servant à régler les conflits à l'entrée de la section critique, T n'est modifiée qu'en fin de section critique.

L'ensemble des variables est:

- c(i) avec la signification précédente (Question 3)

- T avec la signification précédente (Question 2)

S'il y a conflit (c(i) = c(j) = vrai), Pi et Pj exécutent une séquence d'attente où T a une valeur constante.

Si t = j, alors Pi annule sa demande en faisant c(i) := faux, Pj peut alors entrer en section critique. Pi attend que t = i et refait sa demande par c(i) := vrai

Écrire le programme de Pi. Vérifier les 4 conditions.

Exercice 2 : Algorithme de Peterson

On vous propose les deux algorithmes suivants, A et B, pour traiter l'exclusion mutuelle entre deux processus. Pouvez-vous dire s'ils sont corrects et les analyser sous les deux points de vue suivants :

a- Respect de l'exclusion mutuelle

b- Absence d'interblocage

Pour donner une réponse plus claire, ne pas hésiter à numéroter les instructions des processus.

Algorithme A :

```
procedure VERSION_A is
-- Déclaration des variables globales
       CANDIDAT: array(1..2) of boolean;
       PRIORITE: integer;

task type PROCESSUS1;
task body PROCESSUS1 is
begin
       loop
              CANDIDAT(1):= true;
              while CANDIDAT(2) and PRIORITE = 2 loop
                     null;
              endloop;
              -- Section Critique
              PRIORITE = 2;
              CANDIDAT(1):= false;
              -- Suite
       endloop;
end PROCESSUS1;

task type PROCESSUS2;
task body PROCESSUS2 is
begin
       loop
              CANDIDAT(2):= true;
              while CANDIDAT(1) and PRIORITE = 1 loop
                     null;
              endloop;
              -- Section Critique
              PRIORITE = 1;
              CANDIDAT(2):= false;
              -- Suite
       endloop;
end PROCESSUS2;

begin
-- Initialisations
       CANDIDAT(1):= false;
       CANDIDAT(2):= false;
       PRIORITE:= 1;
       declare
              P1: PROCESSUS1;
              P2: PROCESSUS2;
       begin
              null;
       end;
end VERSION_A;
```

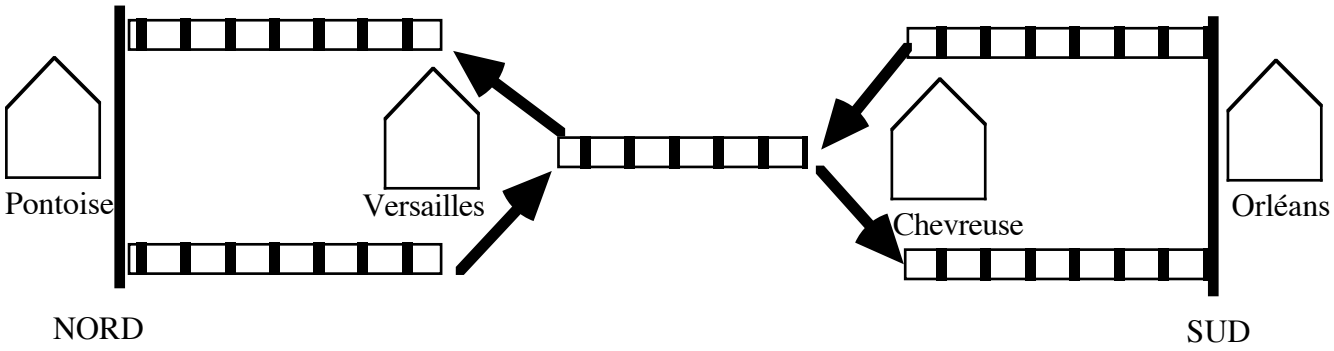
Algorithme B

```
procedure VERSION_B is

-- Déclaration des variables globales
       CANDIDAT: array(1..2) of boolean;
       PRIORITE: integer;

task type PROCESSUS1;
task body PROCESSUS1 is
begin
       loop
              CANDIDAT(1):= true;
              PRIORITE:= 2;
              while CANDIDAT(2) and PRIORITE = 2 loop
                     null;
              endloop;
              -- Section Critique
              CANDIDAT(1):= false;
              -- Suite
       endloop;
end PROCESSUS1;

task type PROCESSUS2;
task body PROCESSUS2 is
begin
       loop
              CANDIDAT(2):= true;
              PRIORITE:= 1;
              while CANDIDAT(1) and PRIORITE = 1 loop
                     null;
              endloop;
              -- Section Critique
              CANDIDAT(2):= false;
              -- Suite
       endloop;
end PROCESSUS2;
begin
-- Initialisations
       CANDIDAT(1):= false;
       CANDIDAT(2):= false;
       PRIORITE:= 1;
       declare
              P1: PROCESSUS1;
              P2: PROCESSUS2;
       begin
              null;
       end;
end VERSION_B;
```

ED 9
Compétition d'accès entre deux classes de processus

Le problème étudié est celui de l'exclusion mutuelle entre deux classes de processus, les processus d'une même classe ne s'excluant pas mutuellement. On pourra se référer au problème classique des "lecteurs et rédacteurs", en considérant deux classes de "lecteurs" en compétition d'accès pour une ressource.

La ligne de chemin de fer PONTOISE-ORLEANS comporte un tronçon à voie unique de VERSAILLES à CHEVREUSE.



Les règles de circulation sur la voie unique sont les suivantes :

1. Le tronçon ne doit jamais être emprunté simultanément par deux trains allant en sens inverse;
2. Le tronçon peut être emprunté par un ou plusieurs trains allant tous dans le même sens;
3. Il n'y a pas de sens de parcours de parcours prioritaire.

Pour étudier la rentabilité de cette ligne, on désire effectuer une simulation du trafic. pour cela, on introduit deux classes de processus : les trains PONTOISE-ORLEANS et les trains ORLEANS-PONTOISE, qui se décrivent comme suit :

```
task type PONTOISE-ORLEANS;
task body PONTOISE-ORLEANS is

begin
       parcours(PONTOISE, VERSAILLES) ;
       entrée_nord;
       parcours(VERSAILLES, CHEVREUSE);
       sortie_sud;
       parcours(CHEVREUSE, ORLEANS) ;
end PONTOISE-ORLEANS;

task type ORLEANS- PONTOISE;
task body ORLEANS- PONTOISE is
begin
       parcours(ORLEANS , CHEVREUSE ) ;
       entrée_sud;
       parcours(CHEVREUSE , VERSAILLES ) ;
       sortie_nord;
       parcours(VERSAILLES , PONTOISE ) ;
end ORLEANS- PONTOISE;
```

Question 1⊡Coalition possible des processsus d'une même classe

On considère le cas où toute coalition est autorisée. Les trains allant dans un sens peuvent attendre indéfiniment, s'il y a constamment des trains dans l'autre sens sur le tronçon.

En utilisant la synchronisation par sémaphores, on demande de programmer les quatre procédures : `entrée_nord, sortie_sud, entrée_sud, sortie_nord,` *de façon à ce que les processus respectent les règles de circulation sur la voie unique.*

Question 2⊡Limitation de la coalition

A partir du moment où un train attend dans un sens, dix nouveaux trains au plus peuvent emprunter la voie dans l'autre sens, avant de laisser passer le ou les trains en attente. On évite ainsi l'attente indéfinie du cas précédent.

On étudiera les états suivants pour l'arrivée d'un train à CHEVREUSE :

- pas d'attente à CHEVREUSE
- attente à CHEVREUSE et moins de 11 trains entrés à VERSAILLES depuis le début de l'attente
- attente à CHEVREUSE et plus de 10 trains entrés à VERSAILLES depuis le début de l'attente et les états analogues pour l'attente à VERSAILLES. On étudiera les transitions entre ces états, dues aux arrivées de trains à CHEVREUSE et à VERSAILLES et aux autres sorties des trains de la section de voie unique.

On pourra utiliser les variables d'état suivantes :

N1 (N2) : nombre de trains sur la voie unique sens P.O.(O.P.)

M1 (M2) : nombre de trains en attente entrée nord (entrée sud)

T1 (T2) : nombre de trains entrés à VERSAILLES depuis le début de l'attente à CHEVREUSE

Programmer la nouvelle version

ED 10
Tampons d'entrée-sortie

Cet exercice montre une mise en oeuvre du schéma PRODUCTEUR-CONSOMMATEUR appliquée à trois processus, dont l'un joue les deux rôles vis à vis de deux tampons différents.

La cadence de transfert des entrées et des sorties d'un programme peut être augmentée en utilisant en parallèle l'unité centrale et les périphériques et en contrôlant leur communication par le schéma producteur-consommateur.

Un tel fonctionnement met en jeu trois processus :

- un *processus d'entrée* d'information de périphérique à mémoire centrale, qui remplit un tampon d'entrée (producteur d'entrées),
- un *processus usager* qui puise ses données dans le tampon d'entrée (consommateur d'entrées), qui réalise le traitement d'un programme donné, établit des résultats à l'aide d'une zone de travail et qui vide ces résultats dans le tampon de sortie (producteur de sorties),
- un *processus de sortie* d'information de la mémoire centrale à périphérique, qui vide un tampon de sortie (consommateur de sorties).

La structure algorithmique des processus est la suivante⊡

```
task body ENTREE
begin
loop
       Lire le périphérique d'entrée;
       Remplir la prochaine place libre du tampon d'entree;
endloop
end ENTREE

task body USAGER
begin
loop
       Prendre un message du tampon d'entrée;
       Le traiter dans la zone de travail;
       Mettre le résultat dans la prochaine place libre du tampon de sortie;
endloop
end USAGER

task body SORTIE
begin
loop
       Prendre un message du tampon de sortie;
       Ecrire sur le périphérique de sortie;
endloop
end SORTIE
```

On suppose fournies les procédures suivantes :

```
procedure lire(u: out zone) -- transfert du périphérique d'entrée vers une zone u,
procedure écrire(v:in zone) -- transfert d'une zone v vers le périphérique de sortie,
procedure copier(a: in zone, b: out zone) -- transfert d'une zone a dans une zone b,
procedure traitement(c: in out zone) -- traitement quelconque sur une zone c.
```

On suppose aussi que les cases des tampons et les zones de travail ont tous la même taille.

Donc, pour tous les u, v, a, b,c des procédures :

taille(u) = taille(v) = taille(a) = taille(b) = taille(c)

Exercice 1⊡
Utilisation de deux tampons distincts

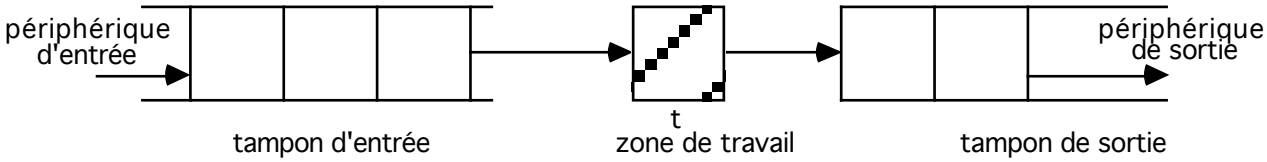
On introduira deux index i et j pour la gestion du tampon d'entrée, et deux index k et l pour celle du tampon de sortie. Les tampons sont implantés sous forme de tableaux.

On demande d'expliciter la gestion des cases des tampons et de donner le programme des processus `ENTREE`, `USAGER` *et* `SORTIE` *dans les divers cas suivants :*

Cas n° 1

Le tampon d'entrée a M places, celui de sortie N places. Le processus `USAGER` utilise une zone de travail t qui lui est réservée tout le temps. Il y recopie le message d'entrée et il en copie le message vers le tampon de sortie.

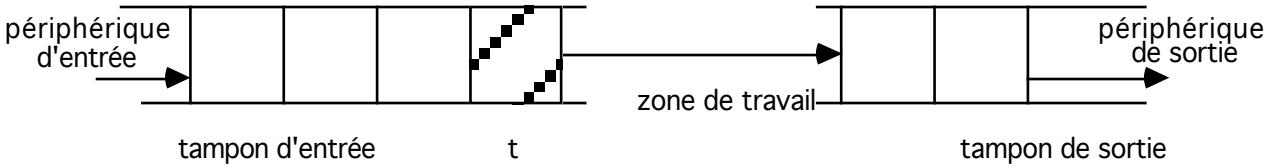
Le flot de données est :



Cas n° 2

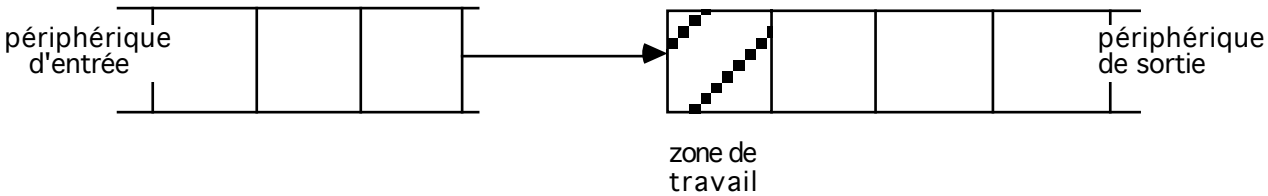
Le tampon d'entrée a M places, celui de sortie N places. Le processus `USAGER` utilise comme zone de travail la case du tampon d'entrée où il a puisé le message.

Le flot de données devient :



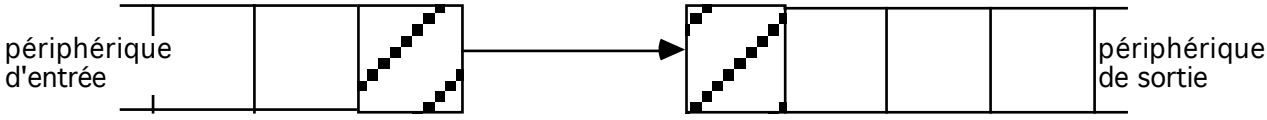
Cas n° 3

Même situation, mais cette fois la zone de travail est la place du tampon de sortie utilisée après la copie du tampon d'entrée vers le tampon de sortie. On libère ainsi, dès que possible la place du tampon d'entrée.



Cas n° 4

Le processus `USAGER` utilise simultanément des places des tampons d'entrée et de sortie (par exemple pour faire du transcodage de caractères).



Cas n° 5

Le processus USAGER utilise comme zone de travail, d'abord la place du tampon d'entrée, puis ensuite la place du tampon de sortie, sans jamais réutiliser la place d'entrée.

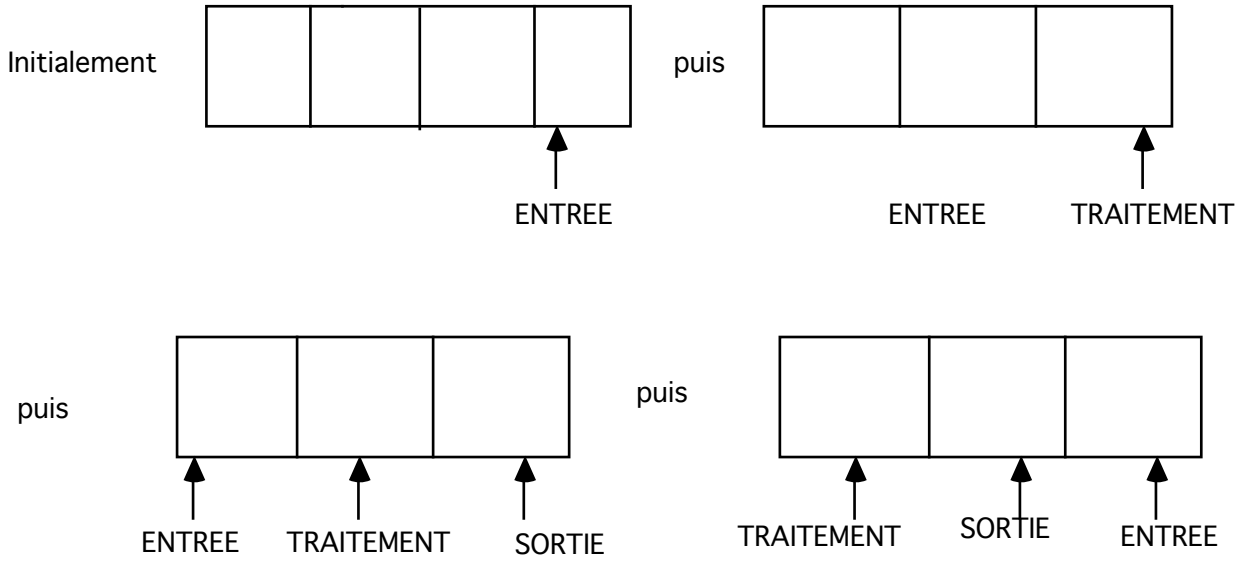
Dans ce cas, le flot de données est identique au schéma précédent, mais la libération de la case du tampon d'entrée se fait plus tôt.

Exercice 2⊡Utilisation d'un seul tampon

On veut éviter toute copie de données d'une zone dans une autre. Pour cela, on utilise un seul tampon de N cases dans lequel chaque case sert successivement pour entrer un message, le traiter et le sortir (gestion circulaire).

Question 1

On suppose ces cases consécutives en mémoire et on utilise des pointeurs ENTREE, TRAITEMENT et SORTIE. Un exemple de gestion d'un tampon de trois cases serait :



Reprogammer les processus dans ce cas .

Question 2

Que devient la solution précédente quand les N cases du tampon sont fixes, mais non consécutives en mémoire ?

Question 3

Donner une solution qui demeure valable quand le nombre de cases du tampon est variable dans le temps.

Indiquer sous quelle condition est permise la variation du nombre de cases du tampon .

ED 11
Synchronisation de processus dans un forum

Cet exercice montre la mise en oeuvre de schémas de synchronisation de processus serveurs et utilisateurs dans une plateforme réalisant un forum.

On souhaite implanter un service de gestion d'une liste d'utilisateurs participant à un forum de discussions par messages . Un service de messagerie est chargé d'acheminer les messages vers les destinataires, ce service ne fait pas l'objet de notre étude. A chaque utilisateur, est associée une tâche dont le schéma est le suivant :

```
task type utilisateur;
task body utilisateur is
moi : numero;-- numero est un type identifiant l'utilisateur comme participant au forum
begin
       attribuer_nom(moi);
       rejoindre_forum (moi);
       while "non_termine" loop
       -- discuter avec les participants :
       -- recevoir messages
       -- ou
       -- consulter liste des participants
       -- envoyer message à des participants
       end loop;
       quitter_forum(moi);
       liberer_nom(moi);
end utilisateur;
```

Problème A Etude du service de noms

Un utilisateur souhaitant se joindre au forum doit se faire attribuer un nom unique et le libérer lorsqu'il a quitté le forum. Le nombre de participants au forum est limité.

On définit le paquetage suivant, qui gère l'attribution et le libération des noms :

```
package nom is
type numero is integer range 1..max;-- nombre maximum de participants
procedure attribuer_nom(moi : out numero);
procedure liberer_nom(moi : in numéro);
end nom;
package body nom is
-- ce paquetage gère l'attribution d'un nom unique de type numero et d'un tampon : boite -
- pour le processus appelant
etat_nom : array(numero) of (libre, occupe):=(others=>libre);
-- à compléter

procedure attribuer_nom(moi : out numero) is
begin
-- bloquer l'appelant tant qu'il n'y a pas de numero libre
-- choisir un numero libre : moi, dans etat_nom, ce qui a pour effet de réserver
-- boite(moi) destiné à contenir la liste des participants pour moi
end attribuer-nom;

procedure liberer_nom(moi:in numero) is
begin
-- mettre à jour etat_nom(moi)
-- réveiller le cas échéant un processus en attente d'attribution d'un nom
end_liberer_nom;
begin -- initialisation des variables ; end nom;
```

Question 1

Ecrire le code des procédures attribuer_nom et liberer_nom en utilisant des sémaphores pour la synchronisation, compléter les déclarations du paquetage, initialiser les variables .

Problème B Etude du service de gestion de la liste des participants

Un processus "démon" `gerant_liste` est chargé de mettre à jour la liste des participants au forum : `liste_participants,` lors de chaque demande `rejoindre_forum` et `quitter_forum`, puis de diffuser la nouvelle liste à chaque participant via une boîte à lettres : `boite`, enfin il autorise l'utilisateur demandeur à rejoindre ou à quitter le forum. Avant tout envoi de message, un utilisateur doit consulter la liste des participants dans sa `boite`. On souhaite que tous les participants disposent de la même liste avant d'envoyer un message; le processus `gerant_liste` doit alors bloquer toute nouvelle consultation de la liste par les utilisateurs, éventuellement attendre la fin des consultations en cours avant de diffuser la nouvelle liste.

On dispose de la structure de donnée suivante :

```
boite : array(numero) of liste;
-- le type liste est prédéfini, il supporte les opérations
-- d'affectation, d'ajout et de retrait d'éléments
-- boite(num) contient la liste des participants pour l'utilisateur num
```

Schéma de comportement du processus `gerant_liste :`

```
task body gerant_liste is
liste_participants : liste := liste_vide;
begin
       loop
       -- attendre une demande "rejoindre" ou "quitter le forum"
       -- modifier liste_participants
       -- bloquer l'accès aux boites et attendre la fin de consultation des boites
       -- diffuser dans la boite de chaque participant la nouvelle liste
       -- autoriser le demandeur à rejoindre ou à quitter le forum;
       end loop;
end gerant_liste;
```

Problème B1 Etude de la gestion des demandes rejoindre et quitter le forum

Les utilisateurs déposent leur demande sous forme d'une `requete` par `deposer_req` dans un tampon à l'intention de `gerant_liste.`

On déclare :

```
N:constant:= taille_du_tampon;
type type_req is (rejoindre, quitter);
type requete is record

num : numero;
req : type_req;
end record;
type tampon_req is array(0..N-1) of requete;
tampon : tampon_req;
```

a) `tampon` avec lecture d'une `requete`

Le processus `gerant_liste` lit une `requete` dans `tampon` par `extraire_req`, puis il traite cette `requete`.

```
procedure deposer_req(r : in requete) is
-- à programmer

procedure extraire_req(r : out requete) is
-- à programmer
```

Question 2

Ecrire les procédures `deposer_req`, `extraire_req` en utilisant des sémaphores pour la synchronisation et une gestion circulaire du tampon (à l'ancienneté). Ne pas oublier d'initialiser les les variables de gestion du `tampon` et les sémaphores.

b) `tampon` avec lecture de plusieurs `requete`

On souhaite maintenant modifier la gestion de `tampon` de la manière suivante :

au lieu d'extraire une seule `requete` de `tampon`, le processus `gerant-liste` extrait toutes les `requete` présentes dans `tampon` et les recopie dans un tampon local. Pour chaque `requete`, il modifie `liste_participants`, mémorise le demandeur (`r.num`). Il diffuse ensuite, la nouvelle liste à tous les participants et enfin, autorise tous les demandeurs à rejoindre ou à quitter le forum.

On remplace la procédure `extraire_req` par la procédure `vider_req`.

On déclare :

```
nbreq : 0..N :=0; nombre de requete  présentes dans tampon.
procedure vider_req(t_req : out tampon_req; nret :out natural) is
nr : 0..N;
begin
       -- prendre une autorisation d'extraire une requete
       nr := nbreq;-- nombre de requete presentes
       nret := nbreq;-- mémorise le nombre de demandeurs
       nbreq :=0;
       -- pour toute requete (on en extrait nr) présente dans tampon :
              -- recopier la requete du tampon dans t_req
              -- ajouter une autorisation de déposer dans tampon
              -- prendre une autorisation d'extraire une nouvelle requete, si nécessaire
end vider_req;
```

La procédure `deposer_req` devient :

```
procedure deposer_req(r : in requete) is
begin
       -- blocage tant que le tampon est plein
       -- prendre une autorisation de déposer dans tampon
       nbreq :=nbreq + 1;
       -- déposer la requete dans tampon
       -- ajouter une autorisation de prendre une requete
end deposer_req;
```

Question 3

Dire en quoi cette politique est intéressante .

Modifier la procédure `deposer_req` *et écrire la procédure* `vider_req`*. La solution devra respecter le schéma producteurs-consommateur utilisant des sémaphores, elle devra aussi assurer la cohérence de la variable* `nbreq`*. On vérifiera que la solution est sans interblocage et qu'il n'y a pas famine de processus. On étudiera les cas où* `tampon` *contient 0, 1, 2, N* `requete`*.*

Problème B2 Diffusion de la liste des participants

Le processus `gerant_liste,` après prise en compte d'une ou plusieurs requêtes modifie `liste_participants` en ajoutant ou en enlevant un ou plusieurs participants.

Avant tout envoi de message, un utilisateur consulte dans sa `boite` la liste des partipants.

Après modification de `liste_participants,` le processus `gerant_liste` recopie dans la `boite` de chaque participant la nouvelle liste, il ne peut le faire que si le participant n'est pas en train de consulter sa `boite` ; enfin, on suppose que l'envoi de messages n'est pas bloquant pour l'utilisateur.

On déclare

```
mutex : array(numero) of semaphore;
for i in numero loop E0(mutex(i),1);end loop;
```

Un utilisateur i souhaitant envoyer un message exécute:

```
P(mutex(i)); consulter_boite(i); choisir_destinataires;
envoyer_message(destinataires); V(mutex(i));
```

Pour `gerant_liste,` on envisage les stratégies suivantes pour la diffusion de la nouvelle liste par `gerant_liste`, en supposant qu'il n'y a pas de panne des utilisateurs :

a)
```
for i in "liste_participants"
loop
  P(mutex(i)); boite(i):= liste_participants; V(mutex(i));
end loop;
```
b)
```
for i in "liste_participants"
loop
P(mutex(i));
end loop;
   for i in "liste_participants"
loop
boite(i):= liste_participants; V(mutex(i));
end loop;
```

Question 4.

Pourquoi la modification de `liste_participants` *ne nécessite-t-elle pas de synchronisation?*

Montrer que, pendant la diffusion, les utilisateurs peuvent encore utiliser des listes différentes avec la stratégie a) et que ce n'est pas le cas avec la stratégie b).

Problème B3 Gestion de l'autorisation à rejoindre ou à quitter le forum

Un utilisateur appelle les procédures `rejoindre_forum` et `quitter_forum :`

```
procedure rejoindre_forum (moi : in numero) is
r : requete;
begin
       r.req := rejoindre;
        r.num := moi;
        deposer_req(r);
        attendre_autorisation(moi)
end rejoindre_forum;

procedure quitter_forum (moi : in numero) is
r : requete
begin
       r.req := quitter;
       r.num := moi;
       deposer_req(r);
       attendre_autorisation(moi)
end quitter_forum;
```

Le processus `gerant_liste` autorise chaque participant r.num, dont il a traité la `requete` par : `autoriser(r.num)`

Question 5.

Déclarer et initialiser les sémaphores nécessaires pour l'ensemble des utilisateurs.

Ecrire les instructions `attendre autorisation(moi), autoriser(r.num)`

Problème C Etude du service de gestion de la liste des participants par plusieurs processus `gerant_liste` .

On implante quatre processus `gerant_liste` ayant le même comportement et activés au même moment.

Afin d'éviter de diffuser plusieurs fois la même `liste_participants`, on maintient un numéro de version `nv`; un processus `gerant_liste` ne sera autorisé à diffuser `liste_participants` que si sa modification est la plus récente, sinon il devra attendre la fin de la diffusion par un autre processus `gerant_liste`.

On déclare :

```
type id_proc is natural range 1..4;
-- identifiant des 4 processus gerant_liste
liste_participants : liste := liste_vide;
-- liste_participants est maintenant une variable partagée
nv : natural :=0;
bloqué : array(id_proc) of boolean := (others=>false);
task type gerant_liste;
gerant : array(id_proc) of gerant_liste;

task body gerant_liste is
n : natural; r : type_req; ego : id-proc;
attente : boolean := false;
begin
      ego:=-- identifiant du processus;
      loop
      extraire_req(r);--on extrait une requete de tampon
      I1;
      nv := nv+1;
      -- modifier_liste;
      n : = nv;
      I2;
      I3;
      if nv>n then
      -- la modification n'est pas la plus récente; attente mémorise le fait
      --  que ego devra attendre la diffusion par un autre processus gerant_liste
       bloque(ego):=true; attente := true;
       else diffuser_liste;
       for i in id_proc loop
       if bloqué(i) then bloqué(i):=false; I4; end if;
       end loop;
      end if;
      I5;
      if attente then attente := false; I6; end if;
      -- on est sûr que l'utilisateur r.num a reçu la nouvelle liste
      autoriser(r.num);
      end loop;
end gerant_liste;
```

La diffusion de liste_participants est celle définie en **B2** b) :

```
procedure diffuser_liste is
begin
       for i in "liste_participants" loop
              P(mutex(i)); end loop;
       for i in "liste_participants" loop
              boite(i):= liste_participants; V(mutex(i)); end loop;
end diffuser_liste;
```

Question 6.

a) Modifier la procédure `extraire_req` *(un processus* `gerant_liste` *n'extrait qu'une* `requete`*, stratégie* ***B1****a), pour tenir compte du fait qu'il y a maintenant plusieurs processus* `gerant_liste`*.*

b) Un processus `gerant_liste` *est successivement "consommateur" sur* `tampon`*, puis "rédacteur" sur* `liste_participants`*, enfin, le cas échéant, "lecteur" sur* `liste_participants`*.*

Pourquoi ne peut-on accepter que plusieurs processus `gerant_liste` *soient lecteurs simultanément sur* `liste_participants`*?*

c) Déclarer les sémaphores nécessaires à la synchronisation sur `liste_participants` *et à la synchronisation de chaque processus* `gerant_liste` *vis à vis de la diffusion de la nouvelle liste. Ecrire les instructions I1, I2, I3, I4, I5, I6.*

d) Dans une première version, `task body gerant_liste` *n'utilisait pas le booléen* `attente`*."* `if attente then attente := false; I6; end if;"` *était remplacé par* `"if bloque(ego) then I6; end if;"`*.*

Expliquez pourquoi cette solution était fausse.

ED 12
RÉVISIONS

INTERBLOCAGE

Un processus espion dont le travail consiste à mesurer l'activité du système est cyclique et utilise de 1 à 5 granules disques de la manière suivante⊡

```
task body espion is
begin
       loop
       demander 1 granule (attente si pas disponible) et travailler avec 1 granule ;
       demander 1 autre granule (attente si indisponible) et travailler avec 2 granules ;
       demander 1 autre granule (attente si indisponible) et travailler avec 3 granules ;
       demander 1 autre granule (attente si indisponible) et travailler avec 4 granules ;
       demander 1 autre granule (attente si indisponible) et travailler avec 5 granules ;
       envoyer le résultat sur le réseau et restituer les 5 granules;
       end loop;
end espion;
```

On réserve treize granules pour les mesures et on voudrait bien installer plusieurs espions.

On constate qu'avec deux espions, il y a toujours assez de granules car 5 + 5 < 13.

Avec cinq espions, il peut y avoir interblocage. Considérons plusieurs états possibles d'allocation représentés par la matrice d'allocation A(t)⊡

a) A(t1) = (3 3 2 2 1)

b) A(t2) = (3 3 3 2 1)

Question 1

a) *Montrer qu'avec 5 espions, A(t1) est un état fiable et que A(t2) n'est pas un état fiable.*

b) *Peut-on avoir interblocage si on installe 3 espions? Expliquer votre réponse (avec éventuellement un exemple de cas d'interblocage quand il existe).*

c) *Peut-on avoir interblocage si on installe 4 espions? Expliquer votre réponse (avec éventuellement un exemple de cas d'interblocage quand il existe).*

S'il y a risque d'interblocage, expliquer comment faire pour l'éviter?

REMPLACEMENT DE PAGE

On dispose d'un système de mémoire paginée à la demande et de deux algorithmes A et B. On observe l'exécution d'un programme auquel le système alloue 3 cases de mémoire centrale et qui accède successivement aux pages 1, 9, 3, 8, 9, 1, 3, 9, 8.

Avec l'algorithme A, on constate qu'il y a successivement en mémoire les pages suivantes⊡

1	1	1	3	3	1	1	1	3
	9	3	8	8	8	3	3	8
		9	9	9	9	9	9	9

Avec l'algorithme B, on constate qu'il y a successivement en mémoire les pages suivantes⊡

1	1	1	3	3	1	1	1	1
	9	3	8	8	3	3	8	8
		9	9	9	8	8	9	9

Question 2

a) Lequel des deux algorithmes correspond à l'algorithme FIFO et lequel correspond à l'algorithme LRU? Justifier votre raisonnement

Déterminer dans chaque cas le nombre total de défauts de page

b) Quelles auraient été les pages en mémoire avec l'algorithme Optimal?

et quel serait le nombre total de défauts de page?

CONCURRENCE ET COHÉRENCE

On veut mettre au point le programme suivant :

```
procedure Essai is
Total : Integer := 0;
Sema : semaphore -- E1
task T1;
task T2;

task body T1 is
X, TotalT1 : Integer;
begin
       for I in 1..4 loop
              P(Sema);-- I1
              X := Total;-- I2
              exit when X >= 4 ;-- I3
              X := X + 1;-- I4
              Total := X;-- I5
              V(Sema);-- I6
              end loop;
       imprimer(X); -- I7
       TotalT1 := Total; imprimer(TotalT1);--I8
end T1 ;

task body T2 is
Y, TotalT2 : Integer;
begin
       for J in 1..4 loop
              P(Sema);-- J1
              Y := Total;-- J2
```

```
            exit when Y >= 4 ;-- J3
            Y := Y + 1;-- J4
            Total := Y; -- J5
            V(sema);-- J6
            end loop;
      imprimer(Y); --J7
      TotalT2 := Total ; imprimer(TotalT2); --J8
end T2 ;

begin
      E0(Sema,2);   -- E2
end Essai;
```

--**Nota** : la notation exit (instructions I3 et J3) fait sortir de la boucle et aller derrière end loop

Après un premier essai où on a fait tourner T1 seul, on imprime X = Total1 = 4 sans problème.

Quelle surprise de voir qu'en exécutant T1 et T2 en concurrence, on imprime des valeurs différentes d'une exécution à une autre.

Pour expliquer les différences, supposer que chaque instruction commentée I1, I2, ..., J7, J8 compte pour une unité, et que l'allocateur de l'unité centrale alloue celle-ci par nombre d'unités à chaque tâche.

1ère exécution : 3 unités pour T1, 3 unités pour T2, 3 unités pour T1, 3 unités pour T2, etc

(on a la séquence I1 I2 I3 J1 J2 J3 I4 I5 I6 J4 J5 J6 I1 I2 I3 J1 J2 J3 I4 I5 I6 J4 J5 J6....)

2ème exécution : 3 unités pour T1, 18 unités pour T2, 3 unités pour T1, 3 unités pour T2, 20 unités pour T1, 5 unités pour T2, (on a la séquence I1 I2 I3 J1 J2 J3 J4 J5 J6 J1 J2 J3 J4 J5 J6 J1 J2 J3 J4 J5 J6 I4 I5 I6 J1 J2 J3 I1 I2 I3 I4 I5 I6 I1 I2 I3 I4 I5 I6 I1 I2 I3 I4 I5 I6 I7 I8 J4 J5 J6 J7 J8).

3ème exécution : on exécute T1 en entier, puis T2

Question 3

a) *Donner les valeurs imprimées : X, TotalT1, Y, TotalT2, lors de chacune des trois exécutions.*

b) *Pour obtenir toujours le résultat de la 3ème exécution, résultat correct, il y a une instruction à corriger. Laquelle et comment?*

c) On va fonctionner à l'alternat, c'est à dire une itération de T1 (I1 à I6) suivie d'une itération de T2 (J1 à J6) etc., et pour cela :

E1 est remplacée par : Sema1, Sema2 : semaphore;

E2 est remplacée par : E0(Sema1, 1); E0(Sema2, 0);

I1, J1, I6 et J6 sont modifiées

Programmer les instructions I1, J1, I6 et J6 pour obtenir l'exécution à l'alternat. Qu'obtient-on comme valeurs imprimées. Cela peut-il remplacer l'exclusion mutuelle dans tous les cas? Expliquer votre réponse.

DUO DE PRODUCTEURS POUR UN CONSOMMATEUR

On veut installer une variante du producteur consommateur qui permette à deux producteurs P1 et P2 (processus clients) de partager un même consommateur C (processus serveur), tout en ayant chacun son tampon de dépôt de message (chacun a un tampon de 5 cases). On ajoute en plus que les messages déposés par P1 sont prioritaires par rapport aux messages déposés par P2. Cela donne la gestion de données suivante :

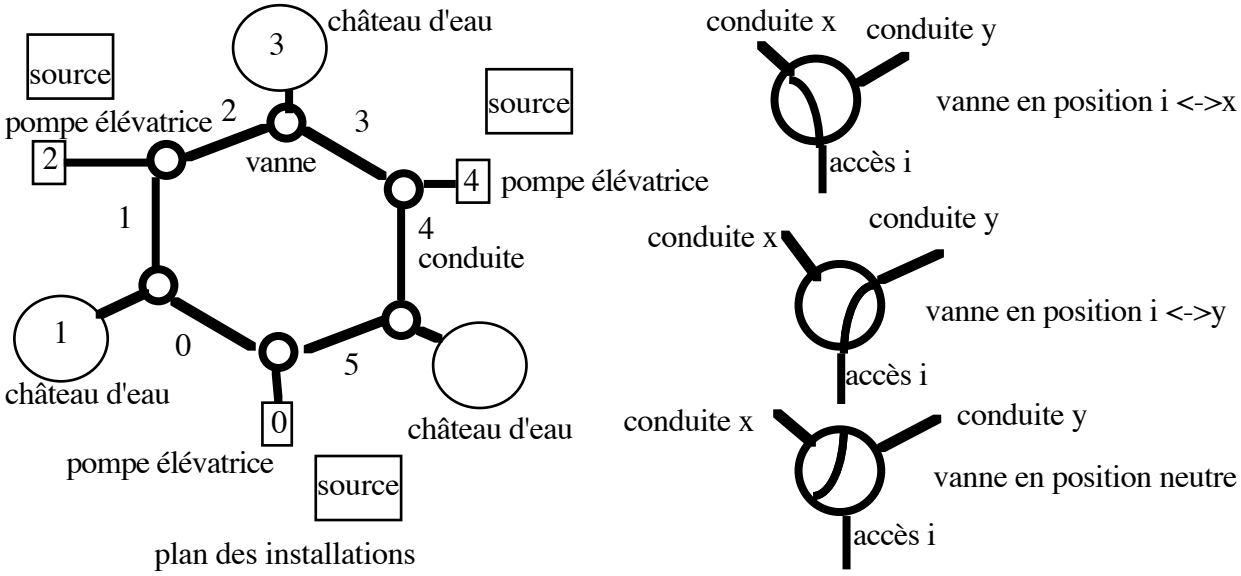
```
procedure Duo is

T1, T2 : array(0..4) of Message; -- T1 pour P1 et T2 pour P2
Nombre: Integer := 0; -- pour P1 et C
-- déclaration des sémaphores;     -- à faire
task P1; -- producteur prioritaire
task P2;  -- deuxième producteur

task body P1 is
X1: Message;
Queue1 : Integer;:= 0
begin
       loop
              preparer(X1);
              T1(Queue1) := X1;
              Queue1 := (Queue1 + 1) mod 5;
              Nombre := Nombre+ 1; -- P2 n'incrémente jamais Nombre
       end loop;
end P1 ;
task body P2 is
X2 : Message;
Queue2: Integer := 0;
begin
       loop
              preparer(X2);
              T2(Queue2) := X2;
              Queue2 := (Queue2 + 1) mod 5;
       end loop;
end P2 ;

task  C ;-- c'est le consommateur
task  body C is
Prioritaire : Boolean;-- permet de choisir entre T1 et T2
Tete1, Tete2 : Integer := 0; Y : Message;
begin
       loop
              Prioritaire := Nombre > 0; -- vrai s'il y a au moins un message dans T1
              if Prioritaire then Nombre := Nombre - 1;  end if;-- et C va le consommer
              if Prioritaire then
                     Y := T1(Tete1); Tete1 := (Tete1 + 1) mod 5;
              else
                     Y := T2(Tete2); Tete2 := (Tete2 + 1) mod 5;
              end if;
       end loop;
end C;

begin
       -- Initialisation  obligatoire des sémaphores-  -- à faire
end Duo;
```

Question 4

Compléter le programme Duo en ajoutant le contrôle de concurrence et la synchronisation par sémaphores (ne mettre que ceux qui sont nécessaires).

ED 13
Sémaphores privés

Dans une commune de grande étendue il y a 3 châteaux d'eau qui sont alimentés par 3 pompes élévatrices qui pompent l'eau de 3 points d'eau (sources ou réservoirs naturels). Pour éviter une pénurie d'eau en cas de tarissement d'un des points d'eau, le conseil municipal décide de construire des conduites pour pouvoir alimenter un château d'eau à partir de deux points d'eau. Mais par souci d'économie, il ne veut pas doubler les pompes élévatrices ; par contre installer les conduites n'est pas onéreux et au contraire apporte de l'emploi dans la commune. Le conseil municipal fait donc l'achat de 3 vannes et fait construire 6 conduites d'eau selon le plan ci-dessous.



Question1.

Le conseil municipal confie l'entretien des 6 conduites d'eau à 6 employés communaux. chacun d'eux est responsable d'une conduite et de son utilisation. Chaque employé peut donc actionner les deux vannes qui sont à chaque bout de la conduite, l'une vers un château d'eau, l'autre vers une pompe. Les vannes comportent trois positions (neutre, affectée à l'une ou l'autre des conduites), ce qui interdit à deux pompes d'alimenter le même château d'eau (il y aurait une surpression dangereuse), ou à une même pompe d'alimenter deux châteaux d'eau (il y aurait une dépression et arrêt de la pompe).

Montrer que si les actions des 6 employés communaux ne sont pas coordonnées, il peut y avoir interblocage.

Montrez que cet interblocage peut être évité par une gestion simple de l'emploi du temps de ces 6 employés.

Question 2.

Un membre informaticien du conseil municipal (ancien auditeur du CNAM, qui n'a pas tout oublié du cours système), décide d'étudier plusieurs règles d'action par ces employés sur

les vannes et d'analyser leur efficacité par un programme de simulation .

Pour cela, il lance l'exécution de 6 processus analogues. Les 6 processus peuvent se dérouler en parallèle, il faut donc contrôler l'utilisation qu'ils font des vannes et compléter le comportement, ci-dessous, de chaque processus par un prélude et un postlude réalisant l'évitement de l'interblocage :

```
task body PROCESSUS is
I: ID_PROC:= association d'un nom unique au processus;
Z : conduite; X, Y : vanne;
begin
Z := I ; X := vanne à un bout de Z ; Y := vanne à l'autre bout de Z ;
       loop
              prélude ;
              connecter X à la conduite Z ;
              connecter Y à la conduite Z ;
              simuler l'utilisation de la conduite pendant un délai aléatoire ;
              comptabiliser les durées d'attente et d'utilisation de Z ;
              déconnecter Y, déconnecter X ;
              postlude ;
              stocker les résultats du cycle de simulation ;
              -- procédure S, voir question 8
              attendre un délai aléatoire;
       endloop;
end PROCESSUS;
```

Compléter le programme ci-dessus en introduisant des variables d'état et en appliquant la technique classique des sémaphores privés, avec un tableau de 6 sémaphores privés SEMPRIV[Z] où Z correspond au numéro de conduite.

Question 3

On sait que cette méthode des sémaphores privés permet d'éviter l'interblocage, mais qu'elle ne prévient pas l'attente indéfinie d'un processus par suite de coalition involontaire entre d'autres processus.

Montrer un exemple de coalition possible.

Question 4

Pour éviter cette coalition, on donne à chaque conduite une priorité qui croît avec l'attente du processus, et on ne connecte une vanne à une conduite que si la conduite voisine n'a pas une priorité plus forte.

Par exemple la priorité p[Z] d'une conduite Z vaut zéro quand la conduite est libre ou occupée. Que se passe-t-il quand la conduite Z est demandée ? Si les conduites Z-1 (modulo 6) et Z + 1 (modulo 6) ne sont pas occupées ou ne sont pas demandées avec des priorités plus grandes que p[Z], alors la conduite Z peut être attribuée et sa priorité p[Z] est remise à zéro. Si la conduite Z ne peut être attribuée, on en tient compte en augmentant de 1 la priorité p[Z] chaque fois que l'une des conduites Z-1 ou Z + 1 est libérée et lors de la demande de Z.

Reprendre le programme de la question 2 pour y introduire les priorités.

Question 5

Votre solution conduit-elle à interblocage ? Expliquer pourquoi.

Question 6

On suppose maintenant que plusieurs processus peuvent demander à utiliser une même conduite Z. On veut contrôler l'emploi de cette conduite Z de telle façon qu'un seul processus puisse s'en servir à la fois et que lorsqu'il restitue la conduite, on essaie d'utiliser les conduites Z + 1 et Z-1 avant de réutiliser la conduite Z.

La première instruction du processus n'est plus `Z:=I`, mais `Z := tirage aléatoire d'un entier prenant une valeur comprise entre 1 et 6`.

Compléter le programme de la question 2 en conséquence, en ajoutant les sémaphores nécessaires.

Question 7

Comme dans la question 6, on suppose que plusieurs processus peuvent demander une même conduite, mais qu'un seul a le droit de s'en servir à un instant donné. Toutefois, on veut maintenant éviter de trop nombreuses manipulations des vannes aussi donne t-on, lorsqu'un processus a fini d'utiliser une conduite, priorité aux processus qui veulent la réutiliser, avant d'essayer d'utiliser les lignes Z+1 ou Z-1. On admet que les processus utilisateurs de la ligne Z se coalisent pour monopoliser l'emploi de celle-ci. Le postlude comprend donc la déconnexion de X et de Y.

Compléter le programme de la question 6 en conséquence, en ajoutant les sémaphores et les variables d'état nécessaires.

Question 8

Les résultats de simulation sont placés dans un tableau à 1000 entrées. Les processus y écrivent à la fin de chaque exécution d'un de leur cycle. Tous les mille cycles le tableau est plein et il faut le vider sur disque ou l'imprimer avant de continuer la simulation.

Pour cela on utilise un compteur de cycle NC qui est une variable d'état de la simulation, qui est commune à tous les processus et qui est incrémentée en fin de chaque cycle.

Quand cette variable atteint la valeur 1000, l'impression est lancée. Au préalable tous les autres processus se sont bloqués au fur et à mesure qu'ils remplissent les 6 dernières entrées du tableau. Le dernier processus, celui qui remplit la millième entrée commande l'impression, puis réactive les autres processus pour reprendre la simulation.

Une procédure S est exécutée en fin de cycle de chaque processus permet de réaliser la synchronisation nécessaire tous les 1000 cycles. On utilisera le compteur NC et la méthode des sémaphores privés ou bien le compteur NC, un sémaphore de blocage SBLOC et un sémaphore d'exclusion mutuelle MUTEXBLOC. On suppose donnée les procédures "ranger les résultats dans l'entrée NC" et "vider le tableau".

Programmer la procédure S

ED 14
Programmation par tâches et rendez-vous Ada

On souhaite implanter un paquetage qui gère l'échange de messages entre processus. Ce paquetage est utilisé concurremment par des processus qui en appellent les procédures⊡

```
procedure POSER(X : in MESSAGE);
procedure OTER(Y : out MESSAGE);
```

La programmation de la concurrence sera réalisée par tâches et rendez-vous, c'est-à-dire sans mémoire commune.

Exercice 1 : Tampon bicase

On veut construire un paquetage gérant deux cases . L'interface de ce paquetage `BICASE` s'écrit :

```
package BICASE is
       procedure POSER(X : in MESSAGE);
       procedure OTER(Y : out MESSAGE);
end BICASE;
```

Le corps du paquetage contient deux cases T1 et T2 . Le comportement du paquetage est le suivant :

- pour `POSER` :

  quand la case `T1` est vide, l'action est : `T1 := X; -- ceci remplit T1`

  une tâche `T_T1` réalise la gestion de `T1`.

- pour `OTER` :

  quand la case `T2` est pleine, l'action est : `Y := T2; -- ceci vide T2`

  une tâche `T_T2` réalise la gestion de `T2`;

- Quand `T1` est pleine et `T2` est vide, l'action est : `T2 := T1; -- ceci vide T1 et remplit T2`

  une tâche `CANAL` réalise cette action.

Question

Compléter, au moyen de rendez-vous ADA, la programmation donnée ci-dessous.

```
package BICASE is
       procedure POSER(X : in MESSAGE);
       procedure OTER(Y : out MESSAGE);
end BICASE;
package body BICASE is
       task T_T1 is
              entry POSE(X : in MESSAGE);
       end T_T1;
       task T_T2 is
              entry OTE(X : out MESSAGE);
       end T_T2;
```

```
task CANAL is
       entry EMETTRE(EMIS : in MESSAGE);
       entry RECEVOIR(RECU : out MESSAGE);
end CANAL;
procedure POSER(X : in MESSAGE) is
       begin T_T1.POSE(X); end POSER;
procedure OTER(Y : out MESSAGE) is
       begin T_T2.OTE(Y); end POSER;
task body T_T1 is
*** à programmer ***
task body CANAL is
*** à programmer ***
task body T_T2 is
*** à programmer ***
begin null; end BICASE;
```

Exercice 2 : Tampon multiple pour entrées lentes

On suppose que les dépôts de messages sont des opérations lentes (par exemple elles sont effectuées par des processus pilotes de périphériques qui reçoivent des messages caractère par caractère sur des lignes asynchrones lentes) et que les retraits de message sont rapides (par exemple les processus font des mouvements de mémoire à mémoire). On souhaite modifier le paquetage précédent afin d'accepter des dépôts de messages en parallèle. L'interface du nouveau paquetage `LENT_RAPIDE` s'écrit :

```
package LENT_RAPIDE is
       procedure POSER(X : in MESSAGE);
       procedure OTER(Y : out MESSAGE);
end LENT_RAPIDE;
```

Pour permettre des accès parallèles aux processus qui appellent `POSER`, un tampon `T1` de 10 cases est réalisé comme suit : chaque case est gérée par une tâche de `type T_T1.`Lors de l'appel de `POSER`, une case disponible de `T1` est allouée au processus appelant ; celui-ci la remplit à son rythme (phase de remplissage, phase lente), pendant le remplissage d'autres processus peuvent accéder à d'autres cases du tampon `T1`. Lorsque la case est remplie, la tâche `T_T1` correspondante demande son transfert dans un tampon `T2` de 4 cases, géré par une tâche `T_T2` selon un schéma producteurs-consommateurs classique. Lorsque la case a été recopiée dans `T2`, elle est alors "libérée".

Une tâche `T_NUMERO` gère l'allocation et la libération des cases de `T1`.

Question

Programmer le paquetage `LENT_RAPIDE` à l'aide de tâches et de rendez-vous Ada . On établira auparavant une structuration en tâches de ce paquetage.