Support de cours

le cnam

EXERCICES TUBES ANONYMES

Joelle Delacroix

V1

05/08/2024

Conservatoire national des arts et métiers

Table des matières

I Préambule

1. Rappels

Généralités sur les tubes Rappel

Un tube est un tuyau dans lequel un processus peut écrire des données qu'un autre processus peut lire. La communication dans le tube est unidirectionnelle et une fois le sens d'utilisation du tube choisi, celui-ci ne peut plus être changé. En d'autres termes un processus lecteur du tube ne peut devenir écrivain dans ce tube et vice-versa. Lors de la création d'un tube, deux descripteurs sont créés, permettant respectivement de lire et écrire dans le tube. Le principe d'écriture/lecture dans un fichier à l'aide d'un descripteur est repris ici, mais au lieu que l'utilisateur gère la position du curseur dans le fichier en lecture et écriture, c'est le système qui s'en charge via les deux descripteurs associé au tube. Les données dans le tube sont gérées en flots d'octets, sans préservation de la structure des messages déposés dans le tube, selon une politique de type « Premier entré, Premier servi » (FIFO), c'est-à-dire que le processus lecteur reçoit les données les plus anciennement écrites. Par ailleurs, les lectures sont destructives c'est-à-dire que les données lues par un processus disparaissent du tube. Le tube a une capacité finie qui est celle du tampon qui lui est alloué. Cette capacité est définie par la constante PIPE_BUF dans le fichier . Un tube peut donc être plein et amener de ce fait les processus écrivains à s'endormir en attendant de pouvoir réaliser leur écriture

Tubes anonymes Rappel

Le tube anonyme est géré par le système au niveau du système de gestion de fichiers et correspond à un fichier au sein de celui-ci, mais un fichier sans nom. Du fait de cette absence de nom, le tube ne peut être manipulé que par les processus ayant connaissance des deux descripteurs en lecture et en écriture qui lui sont associés. Ce sont donc le processus créateur du tube et tous les descendants de celui-ci créés après la création du tube et qui prennent connaissance des descripteurs du tube par héritage des données de leur père

2. Les fonctions utiles pour les exercices

Création d'un tube anonyme

Un tube anonyme est créé par la primitive pipe() dont le prototype est :

```
1 #include <unistd.h>
2 int pipe (int desc[2]);
```

La primitive retourne deux descripteurs placés dans le tableau desc :

- desc[0] : correspond au descripteur utilisé pour la lecture dans le tube,
- desc[1] : correspond au descripteur pour l'écriture dans le tube.

Le tube est représenté au sein du système par un objet inode auquel n'est associé aucun bloc de données, les données transitant dans le tube étant placées dans un tampon alloué dans une case de la mémoire centrale. Tout processus ayant connaissance du descripteur desc[0] peut lire depuis le tube. De même tout processus ayant connaissance du descripteur desc[1] peut écrire dans le tube.

Fermeture d'un tube anonyme

Un tube anonyme est considéré comme étant fermé lorsque tous les descripteurs en lecture et en écriture existants sur ce tube sont fermés. Un processus ferme un descripteur de tube fd en utilisant la primitive close() :

```
1 int close(int fd);
```

À un instant donné, le nombre de descripteurs ouverts en lecture détermine le nombre de lecteurs existants pour le tube. De même, le nombre de descripteurs ouverts en écriture détermine le nombre d'écrivains existants pour le tube. Un descripteur fermé ne permet plus d'accéder au tube et ne peut pas être régénéré.

Lecture dans un tube anonyme

La lecture dans un tube anonyme s'effectue par le biais de la primitive read() dont le prototype est :

```
1 int read(int desc[0], char *buf, int nb);
```

La primitive permet la lecture de nb caractères depuis le tube desc, qui sont placés dans le tampon buf. Elle retourne en résultat le nombre de caractères réellement lus. L'opération de lecture répond à la sémantique suivante :

- si le tube n'est pas vide et contient taille caractères, la primitive extrait du tube nb caractères qui sont lus et placés à l'adresse buf ;
- si le tube est vide et que le nombre d'écrivains est non nul, la lecture est bloquante. Le processus est mis en sommeil jusqu'à ce que le tube ne soit plus vide ;
- si le tube est vide et que le nombre d'écrivains est nul, la fin de fichier est atteinte. Le nombre de caractères rendu est nul

Écriture dans un tube anonyme

L'écriture dans un tube anonyme s'effectue par le biais de la primitive write() dont le prototype est :

```
1 int write(int desc[1], char *buf, int nb);
```

La primitive permet l'écriture de nb caractères placés dans le tampon buf dans le tube desc. Elle retourne en résultat le nombre de caractères réellement écrits. L'opération d'écriture répond à la sémantique suivante :

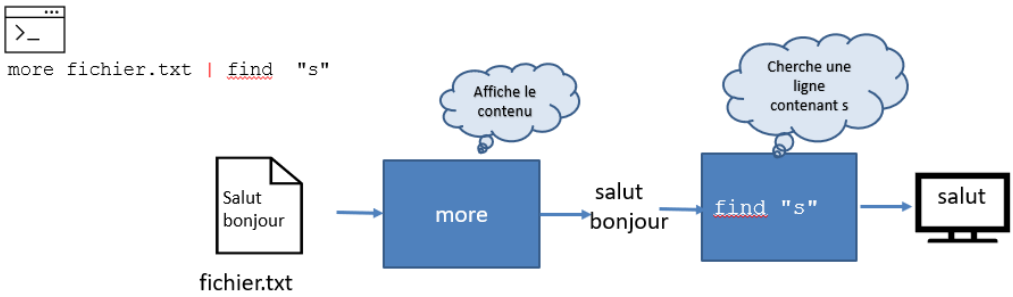
- si le nombre de lecteurs dans le tube est nul, alors une erreur est générée et le signal SIGPIPE est délivré au processus écrivain, et le processus se termine. L'interpréteur de commandes shell affiche par défaut le message « Broken pipe » ;
- si le nombre de lecteurs dans le tube est non nul, l'opération d'écriture est bloquante jusqu'à ce que les nb caractères aient effectivement été écrits dans le tube. Dans le cas où le nombre nb de caractères à écrire est inférieur à la constante PIPE_BUF (4 096 octets), l'écriture des nb caractères est atomique, c'est-à-dire que les nb caractères sont écrits les uns à la suite des autres dans le tube. Si le nombre de caractères est supérieur à PIPE_BUF, la chaîne de caractères à écrire peut au contraire être arbitrairement découpée par le système

3. Retour sur le Pipe en langage de commandes

Rediriger une sortie d'une commande sur l'entrée d'une autre commande

Dans le cours sur le langage de commandes SGF, nous avons vu que :

| (pipe) permet de rediriger la sortie de la commande more vers l'entrée de la commande grep.



Cette redirection consiste en fait en la mise en place d'un tube entre le processus exécutant la commande « more » et le processus exécutant le commande « find ». Le résultat de la commande « more » est écrit dans le tube par le processus exécutant cette commande et le processus exécutant le commande « find » lit ces données.

```
joelle@ubuntu:~$ more fichier.txt
salut
bonjour
joelle@ubuntu:~$ more fichier.txt | grep "s"
salut
joelle@ubuntu:~$
```

II Exercices corrigés

1. EXERCICE 1

Soit le programme C suivant ; cochez les affirmations qui sont vraies par rapport à celui-ci.

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 int pip[2];
6 int main(void)
7 {
8 int nb_ecrit;
9 int pid;
10
11 if(pipe(pip))
12 {
13   perror("pipe");
14   exit;
15 }
16 pid = fork();
17 if (pid == 0)
18 {
19   close(pip[0]);
20   close(pip[1]);
21   printf("Je suis le fils\n");
22   exit;
23 }
24 else
25 {
26   close(pip[0]);
27   for(;;){
28   if ((nb_ecrit = write(pip[1], "ABC", 3)) == -1)
29   {
30     perror ("pb write");
31     exit;
32   }
33   else
34     printf ("retour du w
```

A Le processus père créé un fils

B Le processus père créé un tube anonyme

C Le fils recouvre son code avec la fonction « printf("Je suis le fils\n"); »

D Le père est écrivain sur le tube

E Le fils est lecteur sur le tube

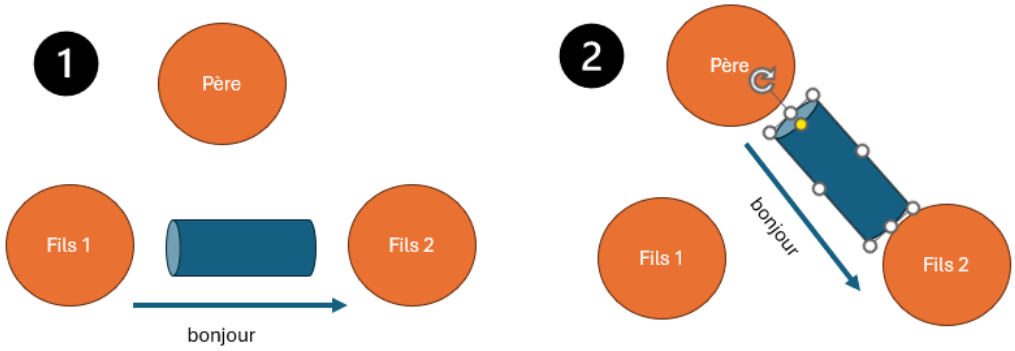
F Le père ferme le descripteur du tube en lecture

G Le tube est brisé car il n'y a pas de lecteurs sur le tube

H Le tube est brisé car il n'y a pas d'écrivains sur le tube

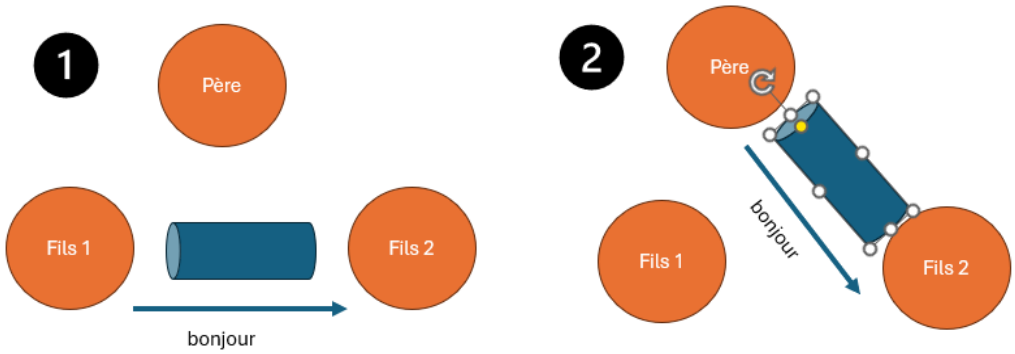
2. ENONCE EXERCICE 2

Ecrivez les programmes correspondant aux deux cas suivants :



3. CORRIGE EXERCICE 2

Ecrivez les programmes correspondant aux deux cas suivants :



CAS 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main ()
{
pid_t ret, ret1;
int tub[2];
char message[7];

pipe(tub);
ret = fork();
```

```
12 if (ret == 0)
13 {
14        printf ("je suis le fils 1; mon pid est %d\n", getpid());
15        /*écrivain sur le tub, il ferme son descripteur en lecture */
16        close (tub[0]);
17        write (tub[1], "bonjour", 7);
18        close (tub[1]);
19        exit;
20 }
21 else
22 {
23     ret1 = fork();
24     if (ret1 == 0)
25     {
26        printf ("je suis le fils 2; mon pid est %d\n", getpid());
27        /*lecteur sur le tub, il ferme son descripteur en écriture */
28        close (tub[1]);
29        read (tub[0], message, 7);
30        printf(" Fils 2, Le message reçu est : %s\n", message);
31        close (tub[0]);
32        exit;
33     }
34    else
35     {
36        close (tub[0]);
37        close (tub[1]);
38        printf ("je suis le père; mon pid est %d\n", getpid());
39        printf ("pid de mon fils 1, %d\n", ret);
40        printf ("pid de mon fils 1, %d\n", ret1);
41        wait();
42        wait();
43     }
44 }
45 }
```

(cf. exo2tubecas1.c)

CAS 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 main ()
5 {
6 pid_t ret, ret1;
7 int tub[2];
8 char message[7];
9
10 pipe(tub);
11 ret = fork();
12 if (ret == 0)
13 {
14     close (tub[0]);
15     close (tub[1]);
16     printf ("je suis le fils 1; mon pid est %d\n", getpid());
17     exit;
18 }
19 else
```

```
{
    ret1 = fork();
    if (ret1 == 0)
    {
       printf ("je suis le fils 2; mon pid est %d\n", getpid());
       /*lecteur sur le tub, il ferme son descripteur en écriture */
       close (tub[1]);
       read (tub[0], message, 7);
       printf(" Fils 2, Le message reçu est : %s\n", message);
       close (tub[0]);
       exit;
    }
   else
    {
     /*écrivain sur le tub, il ferme son descripteur en lecture */
       printf ("je suis le père; mon pid est %d\n", getpid());
       close (tub[0]);
       write (tub[1], "bonjour", 7);
       close (tub[1]);

       printf ("pid de mon fils 1, %d\n", ret);
       printf ("pid de mon fils 2, %d\n", ret1);
       wait();
       wait();
    }
}
}
```

(cf. exo2tubecas2.c)

4. ENONCE EXERCICE3

Ecrivez le programme permettant de réaliser la communication client-serveur suivante :

un processus père créé un processus fils. Ces deux processus communique via des tubes anonymes.

- Le père demande deux entiers à l'utilisateur et il les envoie à son fils.
- Le fils calcule la somme des deux entiers et envoie le résultat à son père.
- Le père affiche le résultat.
- Les deux processus se terminent.

Fonctions utiles pour l'exercice

Complément

Les données transitant dans un tube anonymes sont des chaines de caractères.

Lorsque le père envoie les entiers nb1 et nb2 à son fils via un tube anonyme, il faut donc les convertir au préalable en chaines de caractères.

```
int nb1;
char nbc1[8];
/* conversion de nb1 en chaine de caractères avant écriture dans le tube*/
sprintf(nbc1, "%d", nb1);
```

Lorsque le fils récupère les données envoyées par le père, celles-ci sont de type chaine de caractères. Pour pouvoir les additionner et calculer le résultat, il faut convertir ces chaines en entier.

```
int nb1;
char nbc1[8];
/* conversion des éléments lus dans le tube de chaine de caractères vers entier
 */
nb1=atoi(nbc1);
```

Remarque

Il faut évidemment faire de même pour le résultat res.

5. CORRIGE EXERCICE 3

Pour faire communiquer le fils et le père en bidirectionnel, il faut deux tubes anonymes.

int tubreq[2]; du père vers le fils

int tubrep[2]; du fils vers le père

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main ()
{
pid_t ret;
int nb1, nb2,res;
int tubreq[2];
int tubrep[2];
char nbc1[8],nbc2[8], cres[8];;

pipe(tubreq);
pipe(tubrep);
ret = fork();
if (ret == 0)
{

       /*écrivain sur le tubrep, il ferme son descripteur en lecture */
       close (tubrep[0]);
        /*lecteur sur le tubreq, il ferme son descripteur en écriture */
       close (tubreq[1]);
       /* lecture des deux nombres dans le tube tubreq */
       read (tubreq[0], nbc1, 2);
       read (tubreq[0], nbc2, 2);
        /* conversion des éléments lus dans le tube de chaine de caractères vers
entier */
       nb1=atoi(nbc1);
       nb2=atoi(nbc2);
       printf("je suis le fils; les deux nombres envoyés par mon père sont %d,
%d\n", nb1, nb2);
       res = nb1 + nb2;
       /* conversion de res en chaine de caractères avant écriture dans le tube*/
       sprintf(cres, "%d", res);
       write (tubrep[1], cres, 2);
       close (tubrep[1]);
       exit;
```

```
35      }
36     else
37      {
38         /*lecteur sur le tubrep, il ferme son descripteur en écriture */
39         close (tubrep[1]);
40          /*écrivain sur le tubreq, il ferme son descripteur en lecture */
41         close (tubreq[0]);
42         printf("Donnez un premier nombre entier : ");
43         scanf("%d",&nb1);
44         printf("Donnez un second nombre entier : ");
45         scanf("%d",&nb2);
46         /* conversion de nb1 et nb 2 en chaine de caractères avant écriture dans le
   tube*/
47         sprintf(nbc1, "%d", nb1);
48         sprintf(nbc2, "%d", nb2);
49         write (tubreq[1], nbc1, 2);
50         write (tubreq[1], nbc2, 2);
51         /* lecture du résultat envoyé par le fils */
52         read (tubrep[0], cres, 2);
53         close (tubrep[0]);
54         /* conversion en entier */
55         res=atoi(cres);
56         printf("Le résultat est %d \n: ", res);
57         wait();
58      }
59 }
60
```

(cf. exoserveuradditiontube.c)

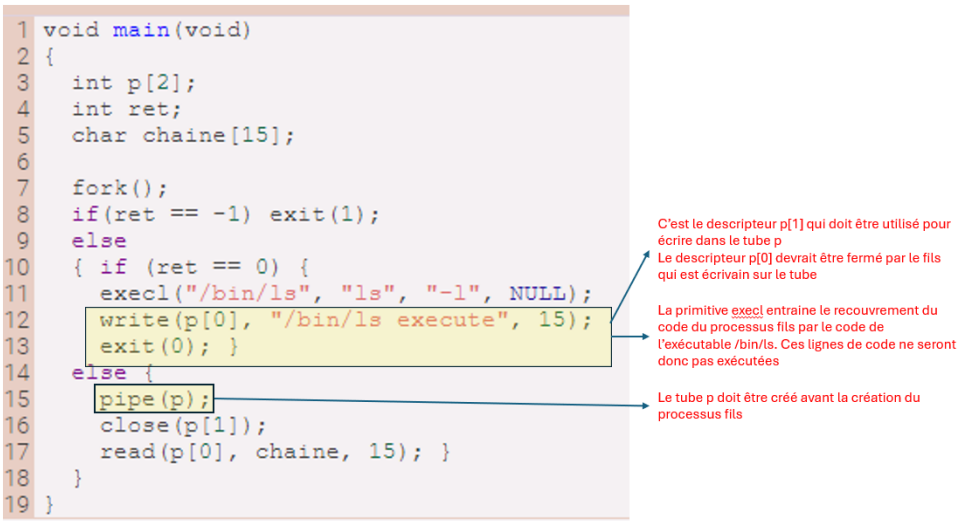
6. EXERCICE 4

ENONCE

Le programme C donné ci-dessous comporte plusieurs erreurs au niveau de l'emploi des primitives liées aux processus et aux outils de communication mis en œuvre. Quelles sont-elles ?

```
1 void main(void)
2 {
3   int p[2];
4   int ret;
5   char chaine[15];
6
7   fork();
8   if(ret == -1) exit(1);
9   else
10  {  if (ret == 0) {
11    execl("/bin/ls", "ls", "-l", NULL);
12    write(p[0], "/bin/ls execute", 15);
13    exit(0); }
14  else {
15    pipe(p);
16    close(p[1]);
17    read(p[0], chaine, 15); }
18  }
19 }
```

CORRECTION



Un code corrigé peut être le suivant :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>void main(void)
{
  int p[2];
  int ret;
  char chaine[15];

  pipe(p);
  fork();
  if(ret == -1) exit(1);
  else
  {  if (ret == 0) {
   close(p[0]);
   write(p[1], "/bin/ls execute", 15);
   execl("/bin/ls", "ls", "-l", NULL);
     }
  else {
    close(p[1]);
    read(p[0], chaine, 15);
    wait();}
  }
}
```

7. ENONCE EXERCICE 5

Soit le programme suivant (pseudo code incluant des primitives linux).

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main(void)
{
 int p1[2], p2[2];
 int c;  /* c est un caractère */
```

```
9
10 pipe(p1); pipe(p2);
11
12 if( fork() == 0 )
13 {
14       close(p1[1]); close(p2[0]);
15       while(read(p1[0], c, 1) == 1)
16       {
17          write(p2[1], c, 1);
18       }
19       exit(0);
20        }
21 else
22 {    if( fork() == 0 )
23       {
24             close(p2[1]); close(p1[0]);
25             while(read(p2[0], c, 1) == 1)
26             {
27              write(p1[1], c, 1);
28             }
29             exit(0);
30       }
31
32 close(p1[0]); close(p1[1]); close(p2[0]); close(p2[1]);
33 wait();
34 exit(0);
35    }
36 }
37
```

Question 1 - Représentez à l'aide d'un schéma les processus créés et les outils de communication qui les relient.

Question 2 - Expliquez pourquoi les processus de ce programme tombent dans une situation de blocage.

8. CORRIGE EXERCICE 5

Question 1 - Représentez à l'aide d'un schéma les processus créés et les outils de communication qui les relient.

Voilà les actions réalisées par ce programme :

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main(void)
{
 int p1[2], p2[2];
 int c;  /* c est un caractère */

 pipe(p1); pipe(p2);

 if( fork() == 0 )
 {
       close(p1[1]); close(p2[0]);
       while(read(p1[0], c, 1) == 1)
       {
          write(p2[1], c, 1);
       }
       exit(0);
       }
else
 {    if( fork() == 0 )
       {
             close(p2[1]); close(p1[0]);
             while(read(p2[0], c, 1) == 1)
             {
              write(p1[1], c, 1);
             }
             exit(0);
       }

 close(p1[0]); close(p1[1]); close(p2[0]); close(p2[1]);
 wait();
 exit(0);
   }
}
```

Création de deux tubes p1 et p2

Création d'un fils F1

Le fils F1 ferme le descripteur en écriture sur p1 et le descripteur en lecture sur p2
Il lit sur p1 un caractère c qu'il écrit sur p2

On est dans le père; Création d'un fils F2

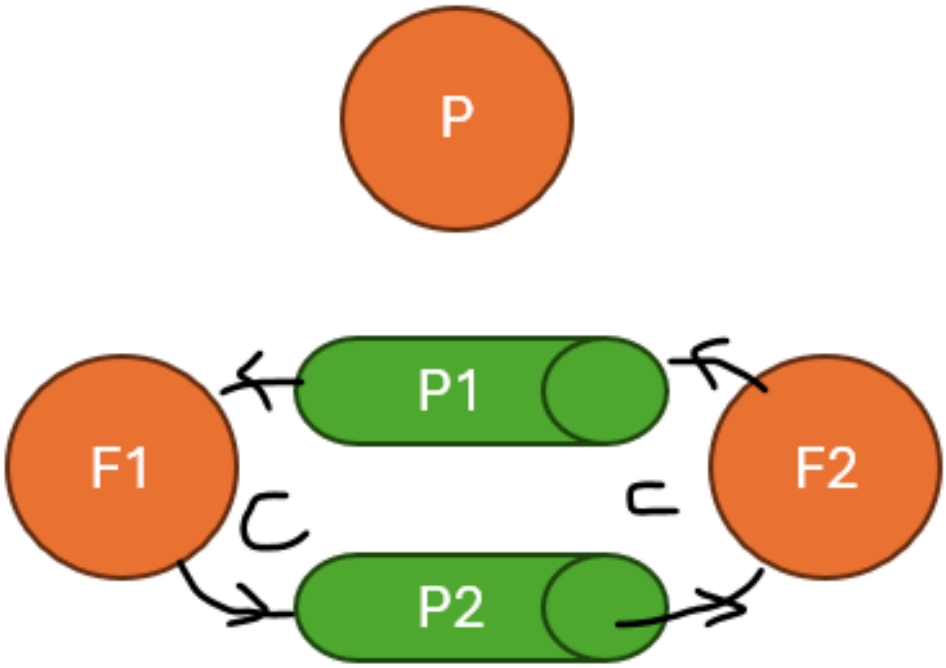
Le fils F2 ferme le descripteur en écriture sur p2 et le descripteur en lecture sur p1
Il lit sur p2 un caractère c qu'il écrit sur p1

On est dans le père; il ferme tous les descripteurs et attend ces fils

Le père créé donc deux tubes et deux fils F1 et F2.

Le fils F1 ferme le descripteur en écriture sur p1 et le descripteur en lecture sur p2. Il lit sur p1 un caractère c qu'il écrit sur p2

Le fils F2 ferme le descripteur en écriture sur p2 et le descripteur en lecture sur p1. Il lit sur p2 un caractère c qu'il écrit sur p1



Question 2 - Expliquez pourquoi les processus de ce programme tombent dans une situation de blocage.

Dans chacune des boucles, F1 et F2 se mettent en attente en lecture. Les deux tubes sont initialement vides et la lecture sur un tube vide est bloquante. Chacun des processus s'endort donc en attente d'un caractère c. et aucun d'eux ne peut écrire. ils sont donc bloqués.