La mémoire virtuelle, l'exclusion mutuelle, les tubes nommées

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

1er décembre 2004

Plan

Plan

Les limites du swap

Problèmes avec le swap :

- coût,
- pas possible avoir des processus plus grand que la mémoire vive.

Remarques :

- portions du code très souvent utilisés, portions du code très peu souvent utilisés,
- de même, pour les données.

Les limites du swap

Problèmes avec le swap :

- coût,
- pas possible avoir des processus plus grand que la mémoire vive.

Remarques :

- portions du code très souvent utilisés, portions du code très peu souvent utilisés,
- de même, pour les données.

Solutions

Les « overlays » :

- morceaux d'un programme
  chargé en mémoire de façon séquentielle.

Chargement dynamique :

- une fonction est chargée en mémoire seulement à son appel.

Problèmes:

- le travail est laissé au programmeur.

Solutions

Les « overlays » :

- morceaux d'un programme
  chargé en mémoire de façon séquentielle.

Chargement dynamique :

- une fonction est chargée en mémoire seulement à son appel.

Problèmes:

- le travail est laissé au programmeur.

Solutions

Les « overlays » :

- morceaux d'un programme
  chargé en mémoire de façon séquentielle.

Chargement dynamique :

- une fonction est chargée en mémoire seulement à son appel.

Problèmes:

- le travail est laissé au programmeur.

Plan

Le demand paging ou swappeur paresseux

La mémoire virtuelle est découpée en pages.

Un nombre restreint de pages est chargé en mémoire physique.

On demande l'accès à un adresses logique :

- tant que sa page se trouve en mémoire continuer,
- sinon, lever une interruption `Page Fault`, et
- traiter l'interruption,
- une fois que la page demandé est en place, recommencer l'opération interrompue.

Remarque :
une instruction doit être interruptible, par exemple :

```
add A B in C
```

avec un Page Fault sur l'accès à C.

Le demand paging ou swappeur paresseux

La mémoire virtuelle est découpée en pages.

Un nombre restreint de pages est chargé en mémoire physique.

On demande l'accès à un adresses logique :

- tant que sa page se trouve en mémoire continuer,
- sinon, lever une interruption `Page Fault`, et
- traiter l'interruption,
- une fois que la page demandé est en place, recommencer l'opération interrompue.

Remarque :
une instruction doit être interruptible, par exemple :

```
add A B in C
```

avec un Page Fault sur l'accès à C.

Le demand paging ou swappeur paresseux

La mémoire virtuelle est découpée en pages.

Un nombre restreint de pages est chargé en mémoire physique.

On demande l'accès à un adresses logique :

- tant que sa page se trouve en mémoire continuer,
- sinon, lever une interruption `Page Fault`, et
- traiter l'interruption,
- une fois que la page demandé est en place, recommencer l'opération interrompue.

Remarque :
une instruction doit être interruptible, par exemple :

```
add A B in C
```

avec un Page Fault sur l'accès à C.

Le demand paging ou swappeur paresseux

La mémoire virtuelle est découpée en pages.

Un nombre restreint de pages est chargé en mémoire physique.

On demande l'accès à un adresses logique :

- tant que sa page se trouve en mémoire continuer,
- sinon, lever une interruption `Page Fault`, et
- traiter l'interruption,
- une fois que la page demandé est en place, recommencer l'opération interrompue.

Remarque :
une instruction doit être interruptible, par exemple :

```
add A B in C
```

avec un Page Fault sur l'accès à C.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur,
  restauration du contexte du processus

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur,
- restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

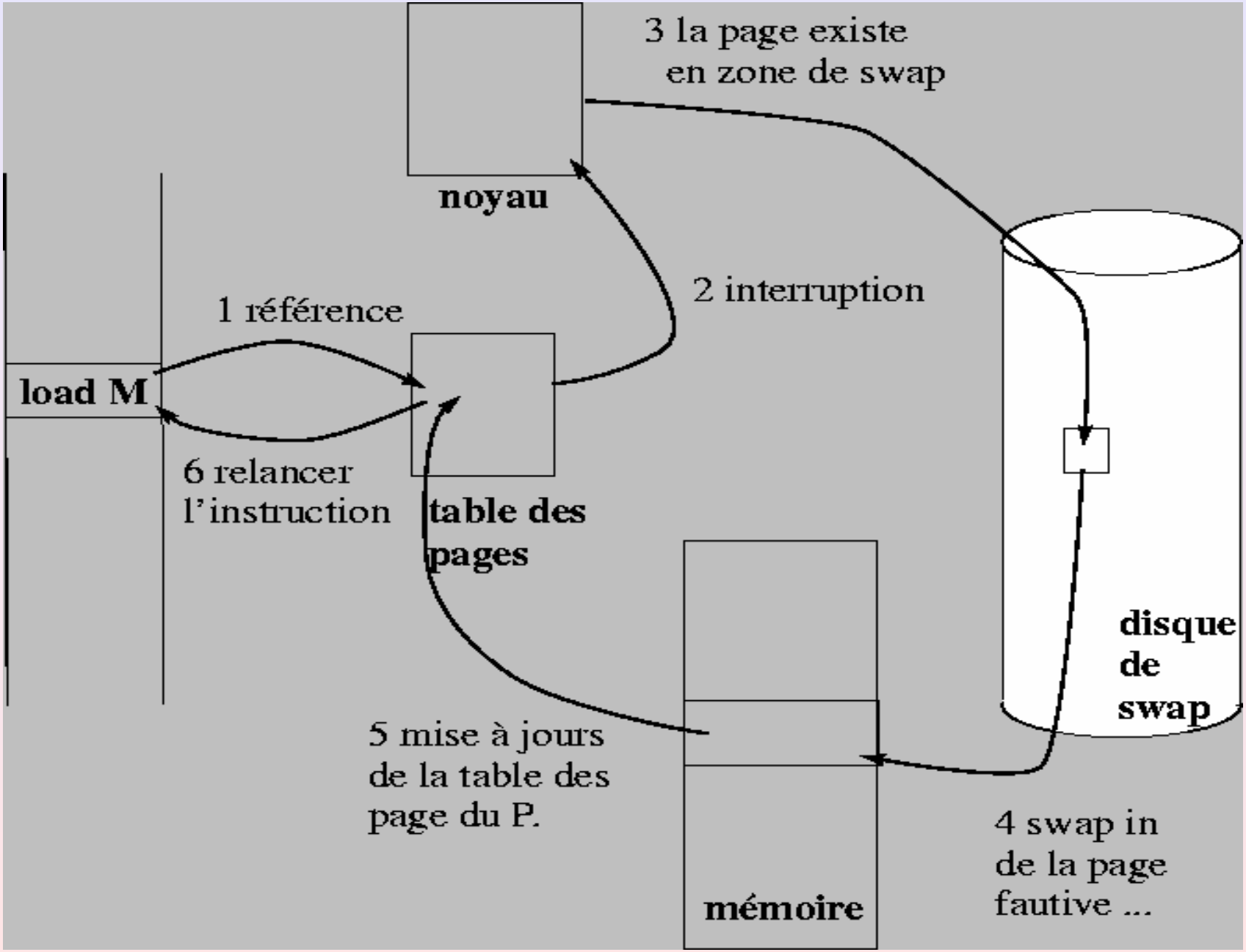
- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur, restauration du contexte du processus.

Traitement du Page Fault

Erreur de page provoque :

- interruption,
- sauvegarde du contexte,
- reconnaissance erreur de page,
- déterminer où la page se trouve sur la mémoire secondaire,
- charger la page dans en endroit libre, ou remplacer une page,
- attente du transfert de la périphérique, et allocation la CPU à quelques autre processus,
- interruption périphérique,
- sauvegarde du contexte du processus,
- reconnaissance interruption périphérique,
- mise à jour table des pages,
- état : en attente de se dérouler,
- quand choisi par l'ordonnanceur,
  restauration du contexte du processus.

Traitement d'un Page Fault (Revuz 1998)



Coût du Demand Paging

Calcul du coût :

$$\text{temps effectif} = (1 - p) * ma + p * (\text{temps traitement})$$

où

$$p = \text{probabilité du Page Fault}$$
$$ma = \text{temps acces à la mémoire}$$

Exemple : si

$$ma = 1 \text{ microseconde}$$
$$\text{temps traitement} = 5000 \text{ microsecondes}$$
$$p = 1/1000 \quad \text{(un Page Fault chaque 1000 accès)}$$

alors

$$\text{temps effectif} = (1 - p) + p * 5000 = 5,999$$

Coût du Demand Paging

Calcul du coût :

$$\text{temps effectif} = (1 - p) * ma + p * (\text{temps traitement})$$

où

$$p = \text{probabilité du Page Fault}$$
$$ma = \text{temps acces à la mémoire}$$

Exemple : si

$$ma = 1 \text{ microseconde}$$
$$\text{temps traitement} = 5000 \text{ microsecondes}$$
$$p = 1/1000 \quad \text{(un Page Fault chaque 1000 accès)}$$

alors

$$\text{temps effectif} = (1 - p) + p * 5000 = 5,999$$

Coût du Demand Paging

Calcul du coût :

$$\text{temps effectif} = (1 - p) * ma + p * (\text{temps traitement})$$

où

$$p = \text{probabilité du Page Fault}$$
$$ma = \text{temps acces à la mémoire}$$

Exemple : si

$$ma = 1 \text{ microseconde}$$
$$\text{temps traitement} = 5000 \text{ microsecondes}$$
$$p = 1/1000 \quad \text{(un Page Fault chaque 1000 accès)}$$

alors

$$\text{temps effectif} = (1 - p) + p * 5000 = 5,999$$

Remplacement FIFO

La page la plus ancienne est remplacée.

État des pages en mémoire au début : xxx

Page démandée Pages en mémoire (aprés) No Page Faults

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6
3	230	6

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6
3	230	6
0	230	6

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6
3	230	6
0	230	6
4	430	7

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6
3	230	6
0	230	6
4	430	7
2	420	8

Remplacement FIFO

La page la plus ancienne est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	231	5
0	230	6
3	230	6
0	230	6
4	430	7
2	420	8
1	421	9

Remplacement LRU

La page moins recemment utilisé est remplacée.

État des pages en mémoire au début : xxx

Page démandée Pages en mémoire (aprés) No Page Faults

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5
3	203	5

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5
3	203	5
0	203	5

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5
3	203	5
0	203	5
4	403	6

Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5
3	203	5
0	203	5
4	403	6
2	402	7

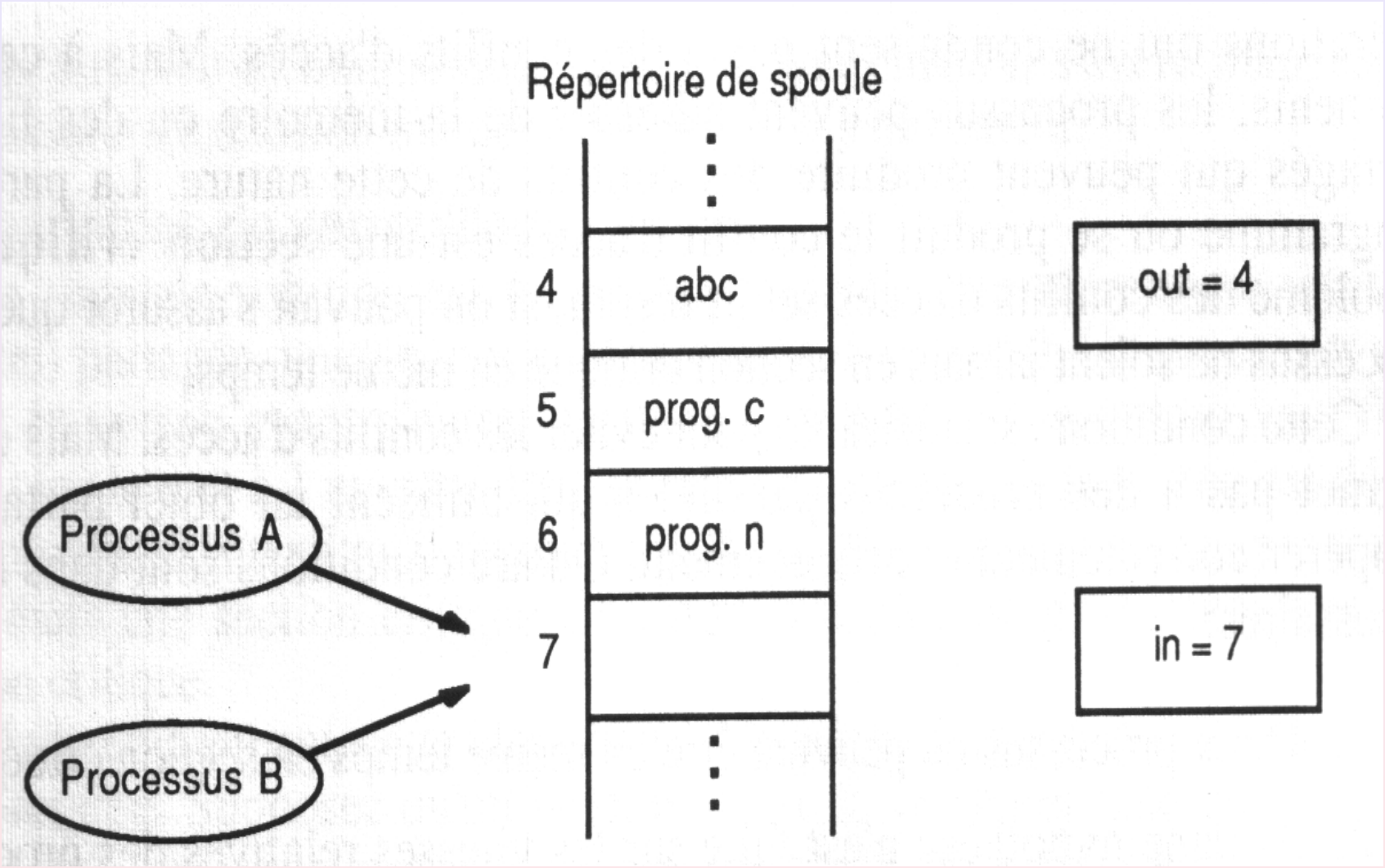
Remplacement LRU

La page moins recemment utilisé est remplacée.
État des pages en mémoire au début : xxx

Page démandée	Pages en mémoire (aprés)	No Page Faults
7	7xx	1
0	70x	2
1	701	3
2	201	4
0	201	4
3	203	5
0	203	5
3	203	5
0	203	5
4	403	6
2	402	7
1	412	8

Plan

Exemple : le spool d'impression (Tannenbaum)



Plan

Exclusion mutuelle et section critique

Section critique :
morceaux de code où l'on accède (souhaite accéder)
à une ressource de façon exclusive.

- Masquage des interruptions.
- Les variables de verrouillage :
  problème : atomicité des opérations.

Exclusion mutuelle et section critique

Section critique :
morceaux de code où l'on accède (souhaite accéder)
à une ressource de façon exclusive.

- Masquage des interruptions.
- Les variables de verrouillage :
  problème : atomicité des opérations.

L'alternance

```
1 : while (true)
2 : {
3 :     /* Ligne suiv: attente */
4 :     while (tour != 0);
5 :     section_crirtique();
6 :     tour = 1;
7 :     section_noncrirtique();
8 : }
```

```
1 : while (true)
2 : {
3 :     /* Ligne suiv: attente */
4 :     while (tour != 1);
5 :     section_crirtique();
6 :     tour = 0;
7 :     section_noncrirtique();
8 : }
```

Problème :
si proc 0 a plus de taches (ou priorité majeure que) de proc 1.

L'alternance

```
1 : while (true)
2 : {
3 :     /* Ligne suiv: attente */
4 :     while (tour != 0);
5 :     section_crirtique();
6 :     tour = 1;
7 :     section_noncrirtique();
8 : }
```

```
1 : while (true)
2 : {
3 :     /* Ligne suiv: attente */
4 :     while (tour != 1);
5 :     section_crirtique();
6 :     tour = 0;
7 :     section_noncrirtique();
8 : }
```

Problème :
si `proc 0` a plus de taches (ou priorité majeure que) de `proc 1`.

La solution de Peterson

```
 1 : #define TRUE 1
 2 : #define FALSE 0
 3 : #define N 2                        /* Nombre processus */
 4 :
 5 : int tour;
 6 : int interesse[N];
 7 :
 8 : entre_region(int process)
 9 : {
10 :     int autre;
11 :     autre = 1 - process;
12 :     interesse[process] = TRUE;
13 :     tour = process;
14 :     /* Prochaine ligne : attente */
15 :     while (tour == process && interesse[autre] == TRUE);
16 : }
17 :
18 : quitter_region(int process)
19 : {
20 :     interesse[process] = FALSE;
21 : }
```

L'instruction TSL (Test, and set lock)

```
 1 : entrer_region:
 2 :         tsl registre, drapeau   | copie de drapeau dans registre,
 3 :                                 | mettre drapeau a 1
 4 :         cmp registre, #0        | est registre egal à 0
 5 :         jnz entrer_region       | non :  (verrous mis) on boucle
 6 :         ret                     | oui :  retour,
 7 :                                 | entree dans la section critique
 8 :
 9 : quitter_region:
10 :         mov drapeau, #0         | mettre 0 dans drapeau
11 :         ret                     | retourner
12 :
```

Plan

mkfifo

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * ref, mode_t mode);
```

ref : nom/chemin à donner au tube
mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-p] [-m] mode`
Équivalent à la primitive UNIX
`mknod(ref,mode|S_IFIFO,0)`

mkfifo

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * ref, mode_t mode);
```

ref : nom/chemin à donner au tube

mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-p] [-m] mode`

Équivalent à la primitive UNIX

`mknod(ref,mode|S_IFIFO,0)`

mkfifo

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * ref, mode_t mode);
```

ref : nom/chemin à donner au tube
mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-p] [-m] mode`
Équivalent à la primitive UNIX
`mknod(ref,mode|S_IFIFO,0)`

mkfifo

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * ref, mode_t mode);
```

ref : nom/chemin à donner au tube
mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-p] [-m] mode`
Équivalent à la primitive UNIX
`mknod(ref,mode|S_IFIFO,0)`

mkfifo

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * ref, mode_t mode);
```

ref : nom/chemin à donner au tube
mode : permissions : le tube est crée avec les permissions mode - umask

Retourne : 0/-1

Sommaire : Crée un fichier spécial de type FIFO (tube)

Remarques : cf. la commande `mkfifo [-p] [-m] mode`
Équivalent à la primitive UNIX
`mknod(ref,mode|S_IFIFO,0)`

Programme : premier exemple

```
 1 : #include <stdio.h>
 2 : #include <sys/types.h>
 3 : #include <sys/stat.h>
 4 : #include <stdlib.h>
 5 :
 6 : int main(void)
 7 : {
 8 :     mode_t mode = S_IRWXU | S_IRWXG | S_IRWXO;
 9 :
10 :     if (mkfifo("tube", mode) == -1)
11 :     {
12 :         perror("mkfifo");
13 :         exit(EXIT_FAILURE);
14 :     }
15 :     /* nécessaire à cause de la masque de creation */
16 :     if (chmod("tube", mode) == -1)
17 :     {
18 :         perror("chmod");
19 :         exit(EXIT_FAILURE);
20 :     }
21 :     exit(EXIT_SUCCESS);
22 : }
```

Ouverture d'un tube

Bloquante :

- `open("tube",O_RDONLY)` :
  bloque si #écrivains = 0
- `open("tube",O_WRONLY)` :
  bloque si #lecteurs = 0

Ouverture d'un tube

Bloquante :

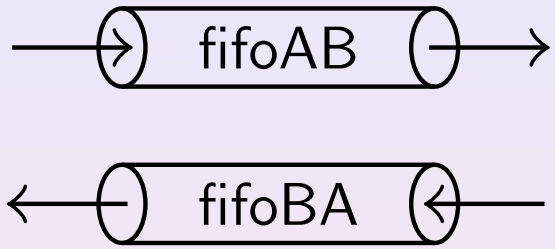
- `open("tube",O_RDONLY)` :
  bloque si #écrivains = 0
- `open("tube",O_WRONLY)` :
  bloque si #lecteurs = 0

Ouverture d'un tube

Bloquante :

- `open("tube",O_RDONLY)` :
  bloque si #écrivains = 0
- `open("tube",O_WRONLY)` :
  bloque si #lecteurs = 0

Interbloquage



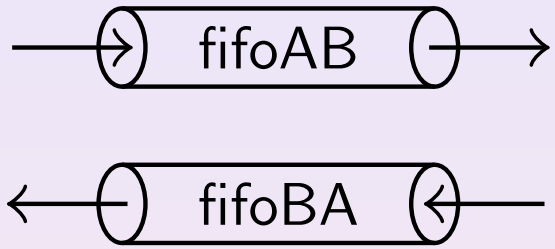
Code bloquant :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_AB = open("tubeAB",O_RDONLY);
d_AB = open("tubeAB",O_WRONLY);   d_BA = open("tubeBA",O_WRONLY);
```

Correction :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_BA = open("tubeBA",O_WRONLY);
d_AB = open("tubeAB",O_WRONLY);   d_AB = open("tubeAB",O_RDONLY);
```

Interbloquage



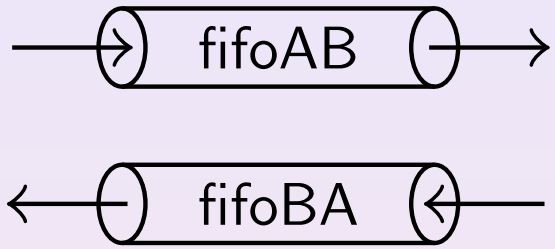
Code bloquant :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_AB = open("tubeAB",O_RDONLY);
d_AB = open("tubeAB",O_WRONLY);   d_BA = open("tubeBA",O_WRONLY);
```

Correction :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_BA = open("tubeBA",O_WRONLY);
d_AB = open("tubeAB",O_WRONLY);   d_AB = open("tubeAB",O_RDONLY);
```

Interbloquage



Code bloquant :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_AB = open("tubeAB",O_RDONLY);
d_AB = open("tubeAB",O_WRONLY);   d_BA = open("tubeBA",O_WRONLY);
```

Correction :

```
/* proc A */                          /* proc B */
    ...                                   ...
d_BA = open("tubeBA",O_RDONLY);   d_BA = open("tubeBA",O_WRONLY);
d_AB = open("tubeAB",O_WRONLY);   d_AB = open("tubeAB",O_RDONLY);
```

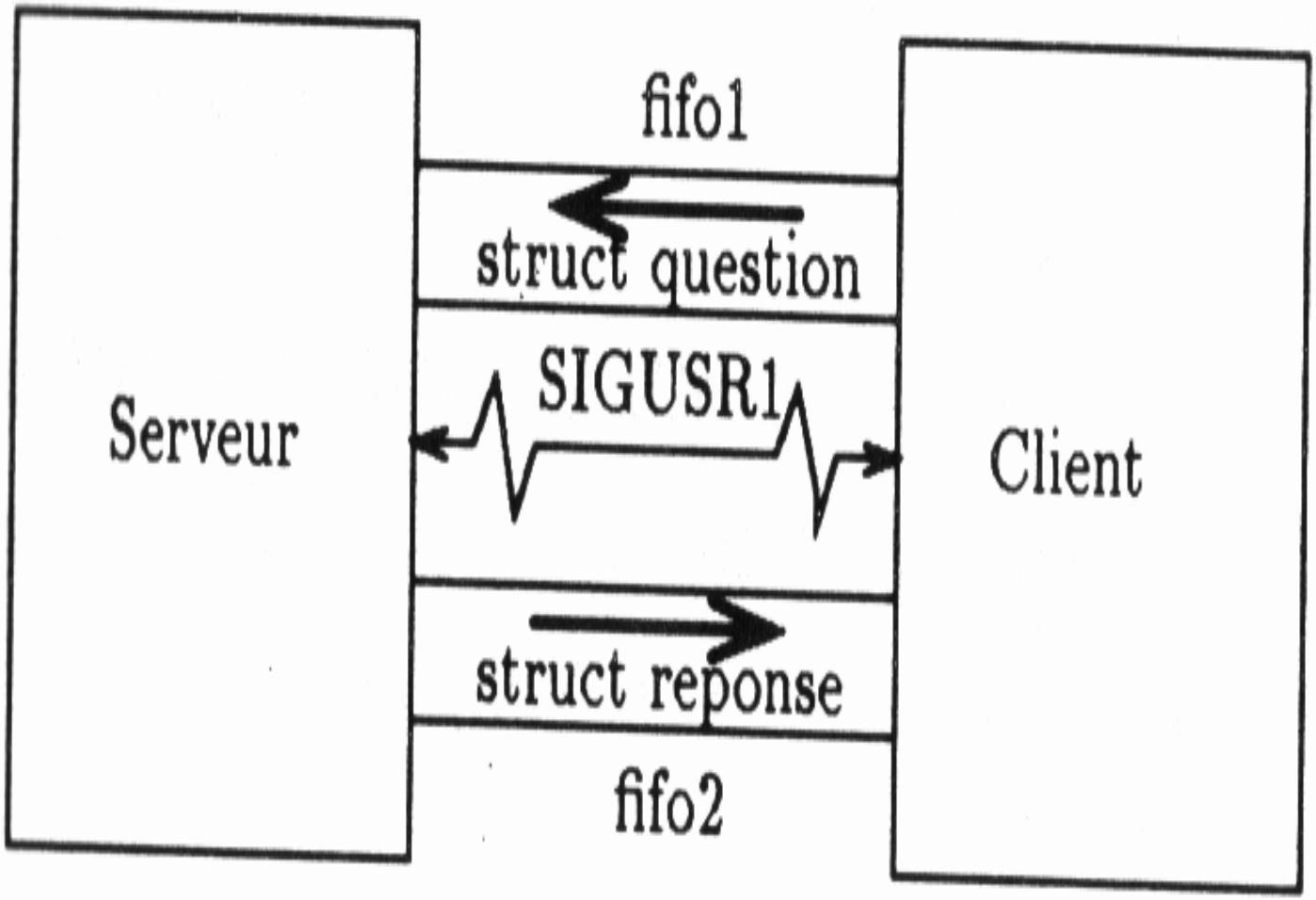
Ouverture non bloquante

Obtenue à l'aide des indicateurs `O_NONBLOCK`, `O_NDELAY` :

- `open("tube", O_RDONLY | O_NONBLOCK)` :
  reussit toujours meme si #ecrivains = 0
- `open("tube", O_WRONLY | O_NONBLOCK)` :
  renovoit -1 si #lecteurs = 0

Plan

Le modèle Serveur-Client (Rifflet 1999)



unlink

```
#include <unistd.h>
int unlink(char * lien);
```

lien : le lien à effacer

Retourne : 0/-1

Remarques : si

- le nombre de lien à l'i-noeud est 0,
- aucun processus utilise cet i-noeud,

alors l'espace sur disque du fichier est libéré.

unlink

```
#include <unistd.h>
int unlink(char * lien);
```

lien : le lien à effacer

Retourne : 0/-1

Remarques : si

- le nombre de lien à l'i-noeud est 0,
- aucun processus utilise cet i-noeud,

alors l'espace sur disque du fichier est libéré.

unlink

```
#include <unistd.h>
int unlink(char * lien);
```

lien : le lien à effacer

Retourne : 0/-1

Remarques : si

- le nombre de lien à l'i-noeud est 0,
- aucun processus utilise cet i-noeud,

alors l'espace sur disque du fichier est libéré.

unlink

```
#include <unistd.h>
int unlink(char * lien);
```

lien : le lien à effacer

Retourne : 0/-1

Remarques : si

- le nombre de lien à l'i-noeud est 0,
- aucun processus utilise cet i-noeud,

alors l'espace sur disque du fichier est libéré.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

sigsuspend

```
#include <signal.h>
int sigsuspend(const sigset_t * ens);
```

ens : masque à installer jusqu'au retour de la primitive.

Remarques : Cette opération a pour effet de réaliser de *façon atomique* :

- l'installation de la masque ens jusqu'au retour de la primitive,
- la mise en sommeil jusqu'à l'arrivée d'un signal non masqué, et qui soit est capté, soit termine le processus.

Programme : serv_cli_fifo.h

```
 1 : #ifndef S_C_FIFO
 2 : #define S_C_FIFO
 3 :
 4 : #include <stdio.h>
 5 : #include <stdlib.h>
 6 : #include <unistd.h>
 7 : #include <sys/types.h>
 8 : #include <sys/stat.h>
 9 : #include <sys/fcntl.h>
10 : #include <signal.h>
11 :
12 : #define NMAX 20
13 : #define QUESTION "fifo1"
14 : #define REPONSE "fifo2"
15 :
16 : struct question{
17 :   int pid_client;
18 :   int question;
19 : };
20 :
21 : struct reponse{
22 :   int pid_serveur;
23 :   int reponse[NMAX];
24 : };
25 :
26 : extern void hand_reveil(int sig);
27 : extern void init_action(struct sigaction *action,void (*handler)(int));
28 :
29 : #endif /* S_C_FIFO */
```

Programme : serveur_fifo.c

```
 1 : #include "serv_cli_fifo.h"
 2 :
 3 : void fin_serveur(int sig)
 4 : {
 5 :     unlink(QUESTION);
 6 :     unlink(REPONSE);
 7 :     exit(2);
 8 : }
 9 :
10 : int main(void)
11 : {
12 :     struct sigaction action;
13 :     int d_question, d_reponse;  /* Descripteurs sur les tubes */
14 :     int ind, sig;
15 :     sigset_t ens, ens_vide;
16 :     struct question question;
17 :     struct reponse reponse;
18 :     mode_t mode =
19 :         S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP | S_IWOTH;
20 :
21 :     if (mkfifo(QUESTION, mode) == -1 || mkfifo(REPONSE, mode) == -1)
22 :     {
23 :         fprintf(stderr, "Creation des tubes impossible\n");
24 :         exit(2);
25 :     };
26 :     d_question = open(QUESTION, O_RDONLY);
27 :     d_reponse = open(REPONSE, O_WRONLY);
```

Programme : serveur_fifo.c

```
28 :
29 :     /* Tous les signaux mettront fin au serveur,
30 :        ... de façon gentile  */
31 :     for (sig = 1; sig < NSIG; sig++)
32 :         signal(sig, fin_serveur);
33 :     /* Sauf SIGUSR1 qui nous reveillerai */
34 :     init_action(&action, hand_reveil);
35 :     sigaction(SIGUSR1, &action, NULL);
36 :     /* On bloque SIGUSR1 */
37 :     sigemptyset(&ens);
38 :     sigaddset(&ens, SIGUSR1);
39 :     sigprocmask(SIG_SETMASK, &ens, NULL);
40 :     sigemptyset(&ens_vide);     /* Ensemble vide de signaux */
41 :
42 :     srand(getpid());
43 :     reponse.pid_serveur = getpid();
```

Programme : serveur_fifo.c

```
44 :     /* Boucle principale */
45 :     while (1)
46 :     {
47 :         if (read(d_question, &question, sizeof(struct question)) <= 0)
48 :         {
49 :             /* Aucune attente attive :
50 :                on s'endort en attente de quelque écrivain */
51 :             close(d_question);
52 :             d_question = open(QUESTION, O_RDONLY);
53 :             continue;
54 :         }
55 :
56 :         for (ind = 0; ind < question.question; ind++)
57 :             reponse.reponse[ind] = rand() % 10;
58 :
59 :         if (write(d_reponse, &reponse, sizeof(struct reponse)) == -1)
60 :         {
61 :             perror("write");
62 :             fin_serveur(SIGUSR2);
63 :         }
64 :         kill(question.pid_client, SIGUSR1);
65 :         /* En attente de tous les signaux */
66 :         sigsuspend(&ens_vide);
67 :     }
68 : }
```

Programme : client_fifo.c

```
 1 : #include "serv_cli_fifo.h"
 2 :
 3 : int main(void)
 4 : {
 5 :     struct sigaction action;
 6 :     int d_question, d_reponse;
 7 :     int ind;
 8 :     sigset_t ens, ens_vide;
 9 :     struct question question;
10 :     struct reponse reponse;
11 :
12 :     d_question = open(QUESTION, O_WRONLY);
13 :     d_reponse = open(REPONSE, O_RDONLY);
14 :     if (d_reponse == -1 || d_question == -1)
15 :     {
16 :         fprintf(stderr, "Ouverture des tubes impossible\n");
17 :         exit(2);
18 :     };
19 :
20 :     /* Signaux */
21 :     init_action(&action, hand_reveil);
22 :     sigaction(SIGUSR1, &action, NULL);
23 :     sigemptyset(&ens_vide);
24 :     sigemptyset(&ens);
25 :     sigaddset(&ens, SIGUSR1);
26 :     sigprocmask(SIG_SETMASK, &ens, NULL);
27 :
```

Programme : client_fifo.c

```
28 :     /*  Initialisation */
29 :     srand(getpid());
30 :     question.pid_client = getpid();
31 :     question.question = 1 + rand() % NMAX;
32 :
33 :     /* Envoyer message */
34 :     if (write(d_question, &question, sizeof(struct question)) == -1)
35 :     {
36 :         perror("write");
37 :         exit(2);
38 :     }
39 :     sigsuspend(&ens_vide);
40 :
41 :     /* Lire reponse */
42 :     if (read(d_reponse, &reponse, sizeof(struct reponse)) <= 0)
43 :     {
44 :         fprintf(stderr, "Problème su read\n");
45 :         exit(2);
46 :     };
47 :     /* ACK */
48 :     kill(reponse.pid_serveur, SIGUSR1);
49 :
50 :     printf("Client : %d nombres reçus :\n", question.question);
51 :     for (ind = 0; ind < question.question; ind++)
52 :         printf("%d ", reponse.reponse[ind]);
53 :     printf("\n");
54 :     exit(0);
55 : }
```

Programme : outils.c

```
 1 : #include "serv_cli_fifo.h"
 2 :
 3 : void hand_reveil(int sig)
 4 : {
 5 :     return;
 6 : }
 7 :
 8 : void init_action(struct sigaction *action, void (*handler) (int))
 9 : {
10 :     sigemptyset(&action->sa_mask);
11 :     action->sa_flags = 0;
12 :     action->sa_handler = handler;
13 : }
```