Systèmes et C

Wieslaw Zielonka
zielonka@irif.fr
www.irif.fr/~zielonka

remarque sur la gestion d'erreur

les macro-fonctions PANIC et PANIC_EXIT affichent un message d'erreur, le nom de fichier source et le numéro de la ligne

```
#ifndef PANIC_H
#define PANIC_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define PANIC_EXIT( fin ) do{                \
    fprintf(stderr, "\n error in file %s in line %d : %s\n",\
      __FILE__,__LINE__,strerror(errno)); \
     exit(fin);\
   } while(0)
#define PANIC do{                            \
    fprintf(stderr,\
      "\n error in file %s in line %d : %s\n",\
      __FILE__,__LINE__,strerror(errno)); \
    abort();\
  } while(0)
#endif
```

pipes (tubes anonymes)

```
ls   |   wc -l
```

Le shell crée deux processus enfants, qui exécutent les programmes prog1 et prog2, la sortie standard de prog1 est envoyé vers l'entrée standard de prog2

création de tube anonyme

```
#include <unistd.h>

int pipe(int fd[2])
```

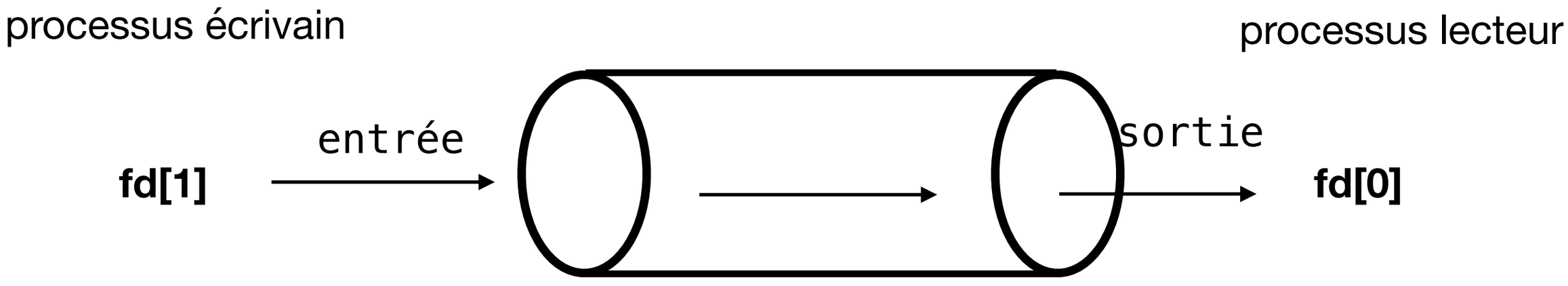
RETOURNE : 0 si OK, -1 si erreur

pipe() crée un tube anonyme,

`fd[0]` contient le descripteur ouvert de lecture

`fd[1]` contient le descripteur ouvert en écriture

pipe



ls *stdout*

wc *stdin*

écrivain : le processus qui écrit dans pipe
lecteur : le processus qui lit dans pipe

pipe

Caractéristiques d'un pipe :

- permet d'envoyer un flot d'octets
- pipe fonctionne en mode FIFO, les octets lus sont automatiquement supprimés, il n'y a pas de notion de position courante dans un pipe
- le pipe possède une capacité limité : il y a un limite de nombre d'octets qui se trouvent à un moment donnée dans le pipe
- les pipes sont unidirectionnels.

Après la création le deux descripteurs de pipe sont ouvert en mode bloquant.

pipe : la lecture en mode bloquant

Le pipe côté lecteur en **mode bloquant.**

Lecture de `n` octets dans un pipe qui contient `p` octets :

- la tentative de lecture bloque si `p==0` octets dans le pipe et il existe un processus écrivains
- si `p==0` octets dans le pipe et il n'y a pas d'écrivains alors `read()` retourne immédiatement `0` (comme pour la lecture à la fin d'un fichier).
- si `p>0` `read()` read lit `min(p,n)` octets.

`Pipe côté écrivain (mode bloquant):`

- `la tentative de write() dans un pipe qui n'a pas de lecteurs provoque le signal SIGPIPE qui tue l'écrivain (sauf si l'écrivain installe un gestionnaire d'interruption pour traiter le signal)`

pipe : l'écriture en mode bloquant

`Pipe côté écrivain (mode bloquant), l'`écriture de `n` octets. Le comportement dépend de la constante PIPE_BUF.

- la tentative de `write()` dans un pipe qui n'a pas de lecteurs provoque le signal `SIGPIPE qui tue` l'écrivain (sauf si l'écrivain installe un gestionnaire d'interruption pour traiter le signal).

Supposons qu'il y a au moins un lecteur.

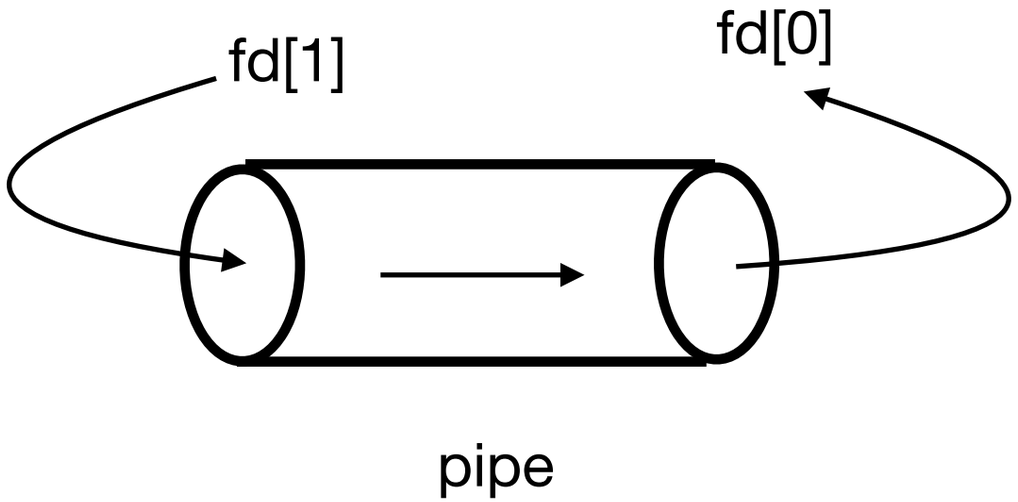
- si n<= PIPE_BUF alors n octets sont écrits de façon atomique dans pipe (ou write() bloque jusqu'à ce que il y a de la place pour n octets).

- si n>PIPE_BUF l'écriture de n octets, non-atomique. Un possible chevauchement avec les octets écrits par d'autre processus. write() termine quand tous les octets sont écrits.

communication enfant--> père par un tube

- Le pipe doit être créé avant *`fork()`*
- Le fils possède une copie de tableau de descripteur du père, donc il peut utiliser les descripteurs du pipe pour communiquer avec le père
- Le père et le fils ferment les descripteurs qu'ils n'utilisent pas
- S'il faut une communication dans le deux directions alors **le père devra créer deux pipes**, une pour la communication père --> fils et une autre pour la communication fils -->père
- si le fils exécute un autre programme (le fils fait exec) alors duplication de descripteur chez le fils

pipe enfant --> parent

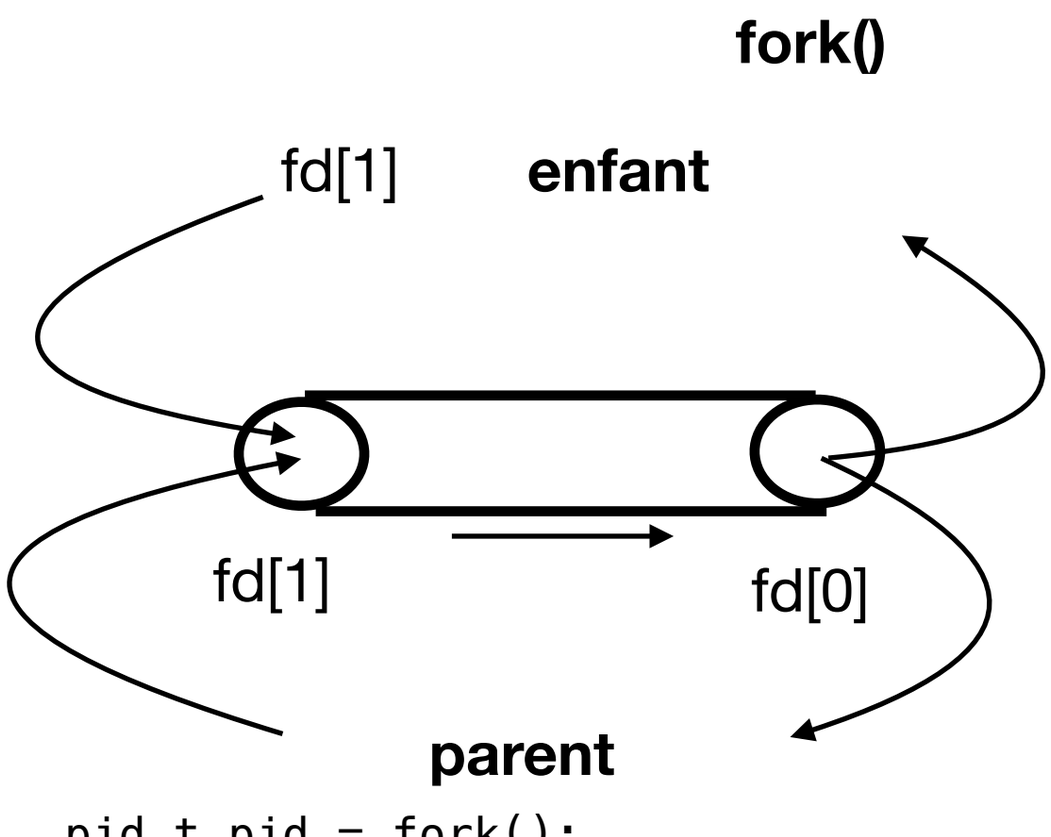


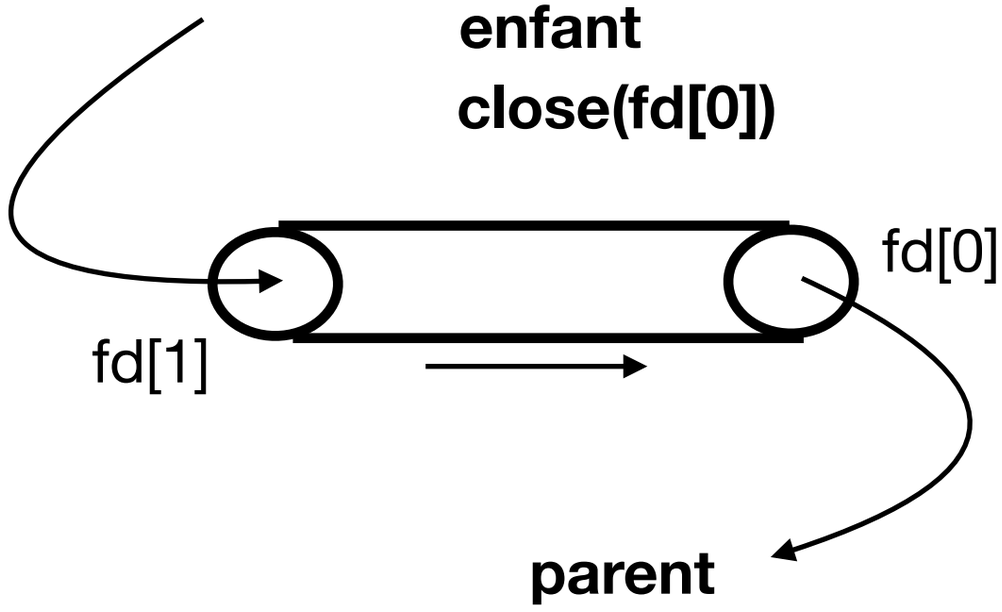
int fd[2];
pipe(fd);

```
int fd[2];

if( pipe( fd ) == -1 )
    PANIC(1);
```

pipe enfant--> parent






```
pid_t pid = fork();
#define LEN 1024
char buf[ LEN ];
switch( pid ){
case -1: PANIC(1);
case 0: close( fd[0] ); /* enfant ne lit pas */
        /* preparer un string à écrire dans buf */
        write( fd[1], buf, strlen(buf) );/*écrire le contenu de  buf dans le pipe*/
        close( fd[1] );
        exit( 0 );
```

pipe enfant --> parent

```
default : /* parent */
    close ( fd[ 1 ]); /* parent n'écrit pas*/
    ssize_t n ;
    while( ( n = read( fd[0], buf , LEN ) ) > 0 ){
        if( n < 0 ) PANIC(1);
        if( n == 0 ) break ; /* fin de lecture */
        /* traiter n octets lus */
}/*fin switch */
close( fd[0] );
```

communication enfant--> parent par le tube

Si le processus enfant doit exécuter un programme on changer le code de case 0:

```
case 0:

   close( fd[0]);

   char *arg={ ..... }; /* paramètres de main */

   dup2( fd[1], 1); /* dupliquer le descripteur */

   close( fd[1] );

   execvp(arg[0], arg);

   PANIC(1); /* terminer si exec echoue*/
```

read() write() dans le pipe

Il est essentiel de fermer les descripteurs non-utilisés.

Supposons que le lecteur ne ferme pas df[1] (le descripteur d'écriture).

Dans ce cas il restera bloqué sur un read() même quand l'écrivain ferme son descripteur d'écriture. (pourquoi?)

Supposons que l'écrivain ne ferme pas df[0] (le descripteur de lecture). Quel problème cela peut engendrer?

L'écrivain pourra écrire dans le pipe même si tous les lecteurs ferment le descripteur de lecture (sauf lui-même). Quand le pipe est plein le père sera bloqué sur write().

L'écrivain peut ignorer le signal SIGPIPE (reçu à une tentative d'écriture quand il n'y a plus de lecteurs). Dans ce cas write() retourne -1 et errno==EPIPE. De cette façon l'écrivain peut détecter l'absence de lecteurs.

écriture dans le tube mode bloquant en détail

Si plusieurs processus écrivent dans le tube alors l'écriture de PIPE_BUF ou moins d'octets ne sera mélangée avec d'autres écritures.

la valeur `PIPE_BUF,` définie dans `limits.h,`

varie d'un système à l'autre (valeur minimale 512 dans FreeBSD et MacOS).

Si le processus écrit plus que PIPE_BUF octets alors le noyau peut diviser l'écriture en plusieurs écritures (qui peuvent être de taille inférieure à PIPE_BUF), effectuées au fur et à mesure que le tube se vide. Ces écritures peuvent être mélangées avec les écritures effectuées par d'autres processus.

write() reste bloqué jusqu'à ce que tous les octets soient écrits ou le processus écrivain reçoit un signal.

En pratique, d'habitude on a un seul écrivain et un seul lecteur.

read() write() dans le tube anonyme

Après ouverture les opération read()/write() sur le pipe sont effectuées en mode bloquant.

Est-ce qu'on peut modifier ce mode et passer en mode non-bloquant?

Oui, activer le flag O_NONBLOCK.

Pour **activer le flag** *O_NONBLOCK* sur un descripteur `fd` ouvert (passer en mode non-bloquant):

```
#include <fcntl.h>

int flags;

/* récupérer les flags de fichier ouvert */

flags = fcntl( fd , F_GETFL );

/* positionner le bit O_NONBLOCK */

flags |= O_NONBLOCK;

/* mettre à jour les flags */

fcntl(fd, F_SETFL, flags );
```

Cette méthode s'applique sur les descripteurs de tubes nommés, tubes anonymes, les sockets etc.

Pour **désactiver** *O_NONBLOCK* sur un descripteur `fd` ouvert (passer en mode bloquant):

```
#include <fcntl.h>

int flags;

/* récupérer les flags de fichier ouvert */

flags = fcntl( fd , F_GETFL );

/* annuler le bit O_NONBLOCK */

flags &= ~O_NONBLOCK;

/* mettre à jour les flags */

fcntl(fd, F_SETFL, flags );
```

Cette méthode s'applique sur les descripteurs de tubes nommés, tubes anonymes, les sockets etc.

lecture dans un pipe en détail

lecture de n octets sur un tube:

ssize_t nb = read(df, buffer, n) df - descripteur de tube ouvert en lecture

p : le nombre d'octets dans le tube

O_NONBLOCK activé ?	il y a p octets dans le tube			
read	p == 0 il y a des écrivains	p==0 il n'y a pas d'écritvains	p < n	p >= n
NON	bloqué	lire 0 octets (EOF)	lire p octets	lire n octets
OUI	read retourne -1 errno == EAGAIN	lire 0 octets (EOF)	lire p octets	lire n octets

l'écriture dans un pipe en détail

l'écriture de n octets dans un tube:

ssize_t nb = write(df, buffer, n) df - descripteur de tube ouvert en écriture

O_NONBLOCK activé ?	il y a au moins un lecteur	
write	n <= PIPE_BUF	n > PIPE_BUF
NON	Ecriture atomique de n octets. Bloquer s'il n'y a pas assez de place.	écrire n octets. Bloquer jusqu'à ce que tous les octets écrits. Les octets peuvent être mélangés avec les octets écrits par d'autres processus
OUI	S'il y a assez de place alors écrire de façon atomique n octets. Sinon rien écrire, retourner -1 et errno == EAGAIN	S'il y a de la place écrire entre 1 et n octets (peut-être mélangés avec les octets écrits par d'autres processus). Sinon write retourne -1 et errno == EAGAIN

écriture dans le tube

S'il n'y a pas de lecteurs alors

- write() dans le tube échoue (write() retourne -1),
- errno == EPIPE et
- le système envoie le signal SIGPIPE (qui terminera le processus s'il n'est pas traité).

Tout cela indépendamment de O_NONBLOCK activé ou désactivé.

les tubes nommés - fifo

les fifo ont les comportement identique que les pipes mais ils permettent un communication entre n'importe quels couple de processus, même sans lien de parenté.

Une tube nommé reste de façon permanente dans un système de fichiers même quand il n'y a pas de lecteurs et d'écrivain.

les tubes nommés - fifo

La création d'un tube nommé en ligne de commande

mkfifo nom_de_fifo

mkfifo -m droits nom_de_fifo

Création de fifo dans un programme C :

```
int  mkfifo(const char *path, mode_t mode)
```

(retourne -1 en cas d'échec, 0 si OK)

ouverture d'un fifo

pour le lecteur :

```
int d = open(nom_de_fifo, O_RDONLY);
```

pour l'écrivain :

```
int d = open(nom_de_fifo, O_WRONLY);
```

entrées/sorties non-bloquant

On spécifie que la communication sur un descripteur de tube est non-bloquante avec le drapeaux O_NONBLOCK

Pour un fifo on peut spécifier le flag O_NONBLOCK pendant `open()` :

```
int d = open("tube", O_RDONLY | O_NONBLOCK);
```

ouverture (open) de tube nommé

open() en	drapeau	l'autre bout du tube est ouvert	l'autre bout du tube est fermé
lecture (O_RDONLY)	O_NONBLOCK désactivé (bloquant)	open() réussi immédiatement	open() bloque
lecture (O_RDONLY)	O_NONBLOCK dans open	open() réussi immédiatement	open() réussi immédiatement
écriture (O_WRONLY)	O_NONBLOCK désactivé (bloquant)	open() réussi immédiatement	open() bloque
écriture (O_WRONLY)	O_NONBLOCK dans open	open() réussit immédiatement	signal ENXIO (termine le processus)

ouverture (open) de tube nommé

- si le tube ouvert par un écrivain alors `open()` en lecture réussit
- si le tube est ouvert par un lecteur alors `open()` en écriture réussit
- en mode bloquant, open() en lecture (écriture) bloqué s'il n'y a pas d'écrivain (lecteur)
- en mode non-bloquant, open en lecture réussit tout de suite (même s'il n'y a pas d'écrivain)
- en mode non-bloquant open() en écriture, erreur est signal ENXIO

Ouverture non-bloquante permet d'éviter un dead-lock.
Pour asymétrie de O_NONBLOCK en lecture/écriture :

- lire sur un tube vide retourne tout de suite avec 0 octets lus.
- écrire dans un tube vide provoque le signal SIGPIPE qui peut tuer l'écrivain.

question

Comment supprimer un fichier fifo ?

Comme un fichier ordinaire c'est-à-dire avec unlink