La communication par tubes

Le système Linux propose deux types d'outils « tubes » : les *tubes anonymes* et les *tubes nommés*.

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.

Les tubes sont gérés par le système au niveau du système de gestion de fichiers et correspondent à un fichier au sein de celui-ci. Lors de la création d'un tube, deux descripteurs sont créés, permettant respectivement de lire et écrire dans le 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 », 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 <limits.h>. Un tube peut donc être plein et amener de ce fait les processus écrivains à s'endormir en attendant de pouvoir réaliser leur écriture.

Les tubes anonymes

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.

Création d'un tube anonyme

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

```
#include <unistd.h>
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 tandis que desc[1] correspond au descripteur pour l'écriture dans le tube.

Les deux descripteurs sont alloués dans la table des fichiers ouverts du processus et pointent respectivement sur un objet fichier en lecture et un objet fichier en écriture. 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() :

```
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 :

```
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 min(taille, 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.

L'opération de lecture sur le tube peut être rendue non bloquante en émettant un appel à la fonction fcntl(desc[0], F_SETFL, O_NONBLOCK). Dans ce cas, le retour est immédiat dans le cas où le tube est vide.

Écriture dans un tube anonyme

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

```
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.

De même que la lecture, l'écriture sur le tube peut être rendue non bloquante.

Les tubes nommés

Les *tubes nommés* sont également gérés par le système de gestion de fichiers, et correspondent au sein de celui-ci à un fichier avec un nom. De ce fait, ils sont accessibles par n'importe quel processus connaissant ce nom et disposant des droits d'accès au tube. Le tube nommé permet donc à des processus sans liens de parenté de communiquer selon un mode flots d'octets.

Les tubes nommés apparaissent lors de l'exécution d'une commande ls –l et sont caractérisés par le type p. Ce sont des fichiers constitués d'une inode à laquelle n'est associé aucun bloc de données. Tout comme pour les tubes anonymes, les données contenues dans les tubes sont placées dans un tampon, constitué par une seule case de la mémoire centrale.

Création d'un tube nommé

Un tube nommé est créé par l'intermédiaire de la fonction mkfifo() dont le prototype est :

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

Le paramètre nom correspond au chemin d'accès dans l'arborescence de fichiers pour le tube. Le paramètre mode correspond aux droits d'accès associés au tube, construits de la même manière que pour tout autre fichier La primitive renvoie 0 en cas de succès et – 1 dans le cas contraire.

Ouverture d'un tube nommé

L'ouverture d'un tube nommé par un processus s'effectue en utilisant la primitive open() déjà rencontrée au chapitre 3 :

```
int open (const char  *nom, int mode_ouverture);
```

Le processus effectuant l'ouverture doit posséder les droits correspondants sur le tube. La primitive renvoie un descripteur correspond au mode d'ouverture spécifié (lecture seule, écriture seule, lecture/écriture).

Par défaut, la primitive open() appliquée au tube nommé est bloquante. Ainsi, la demande d'ouverture en lecture est bloquante tant qu'il n'existe pas d'écrivain sur le tube. D'une manière similaire, la demande d'ouverture en écriture est bloquante tant qu'il n'existe pas de lecteur sur le tube. Ce mécanisme permet à deux processus de se synchroniser et d'établir un rendez-vous en un point particulier de leur exécution.

Lecture et écriture sur un tube nommé

La lecture et l'écriture sur le tube nommé s'effectuent en utilisant les primitives read() et write() vues précédemment.

Fermeture et destruction d'un tube nommé

La *fermeture* d'un tube nommé s'effectue en utilisant la primitive close(). La *destruction* d'un tube nommé s'effectue en utilisant la primitive unlink().

Les IPC : files de messages, mémoire partagée

Caractéristiques générales

Les IPC (*Inter Process Communication*) forment un groupe de trois outils de communication indépendants des tubes anonymes ou nommés, dans le sens où ils n'appartiennent pas au système de gestion de fichiers. Ces trois outils sont :

– les files de messages ou MSQ (*Messages Queues*) ;
– les régions de mémoire partagée ;
– les sémaphores que nous étudions au chapitre 8.

Ces trois outils sont gérés dans des tables du système, une par outils.

Un outil IPC est identifié de manière unique par un identifiant externe appelé la clé (qui a le même rôle que le chemin d'accès d'un fichier) et par un identifiant interne (qui joue le rôle de descripteur). Un outil IPC est accessible à tout processus connaissant l'identifiant interne de cet outil. La connaissance de cet identifiant s'obtient par héritage ou par une demande explicite au système au cours de laquelle le processus fournit l'identifiant externe de l'outil IPC.

La clé est une valeur numérique de type key_t. Les processus désirant utiliser un même outil IPC pour communiquer doivent se mettre d'accord sur la valeur de la clé référençant l'outil. Ceci peut être fait de deux manières :

– la valeur de la clé est figée dans le code de chacun des processus ;
– la valeur de la clé est calculée par le système à partir d'une référence commune à tous les processus. Cette référence est composée de deux parties, un nom de fichier et un entier. Le calcul de la valeur de la clé à partir cette référence est effectuée par la fonction ftok(), dont le prototype est :

```
#include <sys/ipc.h>
key_t ftok (const char *ref, int numero);
```

La commande ipcs permet de lister l'ensemble des outils IPC existants à un moment donné dans le système. Les informations fournies par cette commande sont notamment pour chaque outil IPC :

– le type (q pour messages queues, s pour sémaphores, m pour les régions de mémoire partagée) ;
– l'identification interne ;
– la valeur de la clé ;
– les droits d'accès définis ;
– le propriétaire ;
– le groupe propriétaire.

```
delacroix@vesuve:~> ipcs
------ Segments de mémoire partagée --------
touche     shmid  propriétaire perms    octets   nattch   statut
0x00000000 1015808   root      644      118784    3      dest
0x00000000 1114113   root      644      118784    2      dest
0x00000000 1146882   root      644      110592    2      dest
0x00000000 1179651   root      644      110592    2      dest
0x00000000 1212420   root      644      151552    4      dest
0x00000000 1245189   root      644      151552    4      dest

------ Tables de sémaphores --------
touche    semid   propriétaire perms      nsems
0x0000000c 0       delacroix     600       1

------ Files d'attente de messages --------
touche     msqid    propriétaire perms    octets utilisés messages
0x0000013a  0        delacroix    750     0          0
```

Les files de messages

Le noyau Linux gère au maximum MSGMNI files de messages (128 par défaut), pouvant contenir des messages dont la taille maximale est de 4 056 octets.

Accès à une file de message

L'accès à une file de message s'effectue par l'intermédiaire de la primitive msgget(). Cette primitive permet :

– la création d'une nouvelle file de messages ;
– l'accès à une file de messages déjà existante.

Le prototype de la fonction est :

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t cle, int option);
```

Le paramètre cle correspond à l'identification externe de la file de messages. Le paramètres options est une combinaison des constantes IPC_CREAT, IPC_EXCL et de droits d'accès définis comme dans le cadre des fichiers. La fonction renvoie l'identifiant interne de la file de messages en cas de succès et la valeur – 1 sinon.

Création d'une file de messages

La création d'une file de messages est demandée en positionnant les constantes IPC_CREAT et IPC_EXCL. Une nouvelle file est alors créée avec les droits d'accès définis dans le paramètre option. Le processus propriétaire de la file est le processus créateur tandis que le groupe propriétaire de la file

est le groupe du processus créateur. Si ces deux constantes sont positionnées et qu'une file d'identifiant externe cle existe déjà, alors une erreur est générée. Si seule la constante IPC_CREAT est positionnée et qu'une file d'identifiant externe égal à cle existe déjà, alors l'accès à cette file est retourné au processus.

Ainsi l'exécution de msgget (cle, IPC_CREAT | IPC_EXCL | 0660) crée une nouvelle file avec des droits en lecture et écriture pour le processus propriétaire de la file et pour les processus du groupe.

Accès à une file déjà existante

Un processus désirant accéder à une file déjà existante effectue un appel à la primitive msgget() en positionnant à 0 le paramètre option.

Le cas particulier clé = IPC_PRIVATE

Un processus peut demander l'accès à une file de messages en positionnant le paramètre cle à la valeur IPC_PRIVATE. Dans ce cas, la file créée est seulement accessible par ce processus et ses descendants.

Envoi et réception de message

La communication au travers d'une file de messages peut être bidirectionnelle, c'est-à-dire qu'un processus consommateur de messages dans la file peut devenir producteur de messages pour cette même file. La communication mise en œuvre est une communication de type boite aux lettres, préservant les structures des messages.

Chaque message comporte les données en elles-mêmes ainsi qu'un type qui permet de faire du multiplexage dans la file de messages et de désigner le destinataire d'un message.

‰ Format des messages

Un message est toujours composé de deux parties :

– la première partie constitue le type du message. C'est un entier long positif ;
– la seconde partie est composée des données proprement dites.

Toutes les données composant le message doivent être contiguës en mémoire centrale. De ce fait, le type pointeur est interdit.

Un exemple de structure de messages est par exemple :

```
struct message {
        long mtype;
        int n1;
        char[4];
        float fl1; };
```

Envoi d'un message

L'envoi d'un message dans une file de messages s'effectue par le biais de la primitive msgsnd().

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd (int idint, const void *msg, int longueur, int option);
```

Le paramètre idint correspond à l'identifiant interne de la file. Le paramètre *msg est l'adresse du message en mémoire centrale tandis que le paramètre longueur correspond à la taille des données seules dans le message msg envoyé. Par défaut, la primitive msgsnd() est bloquante c'est-à-dire qu'un processus est suspendu lors d'un dépôt de messages si la file est pleine. En positionnant le paramètre option à la valeur IPC_NOWAIT, la primitive de dépôt devient non bloquante. La primitive renvoie 0 en cas de succès, – 1 sinon.

Réception d'un message

Un processus désirant prélever un message depuis une file de messages utilise la primitive msgrcv().

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv (int idint, const void *msg, int longueur, long letype, int option);
```

Le paramètre idint correspond à l'identifiant interne de la file. Le paramètre *msg est l'adresse d'une zone en mémoire centrale pour recevoir le message tandis que le paramètre longueur correspond à la taille des données seules dans le message msg envoyé.

Le paramètre letype permet de désigner un message à extraire, en fonction du champ type de celui-ci. Plus précisément :

– si letype est strictement positif, alors le message le plus ancien dont le type est égal à letype est extrait de la file ;
– si letype est nul, alors le message le plus ancien est extrait de la file. La file est alors gérée en FIFO ;
– si letype est négatif, alors le message le plus ancien dont le type est le plus petit inférieur ou égal à |letype| est extrait de la file. Ce mécanisme instaure des priorités entre les messages.

Par défaut, la primitive msgrcv() est bloquante c'est-à-dire qu'un processus est suspendu lors d'un retrait de messages si la file ne contient pas de messages correspondant au type attendu. En positionnant le paramètre option à la valeur IPC_NOWAIT, la primitive de retrait devient non bloquante. Le paramètre option peut également prendre la valeur MSG_EXCEPT. Dans ce cas, un message de n'importe quel type sauf celui spécifié dans letype est prélevé. Destruction d'une file de message
La destruction d'une file de messages s'effectue en utilisant la primitive msgctl() dont le paramètre operation est positionné à la valeur IPC_RMID. La valeur renvoyée est 0 en cas de succès et – 1 sinon.

```
msgctl (int idint, IPC_RMID, NULL);
```

La suppression d'une file de messages peut également être réalisée depuis le prompt du shell par la commande ipcrm – q identifiant ou ipcrm –Q cle.