OS Réseaux et Programmation Système - C3

Rabie Ben Atitallah

rabie.benatitallah@univ-valenciennes.fr

Communication entre processus

Système multi-programmé

- Chaque processus dispose d'un espace indépendant.
- Chaque espace est protégé par rapport aux autres processus

⇒ Mais les processus doivent pouvoir s'échanger des données

Communication entre processus

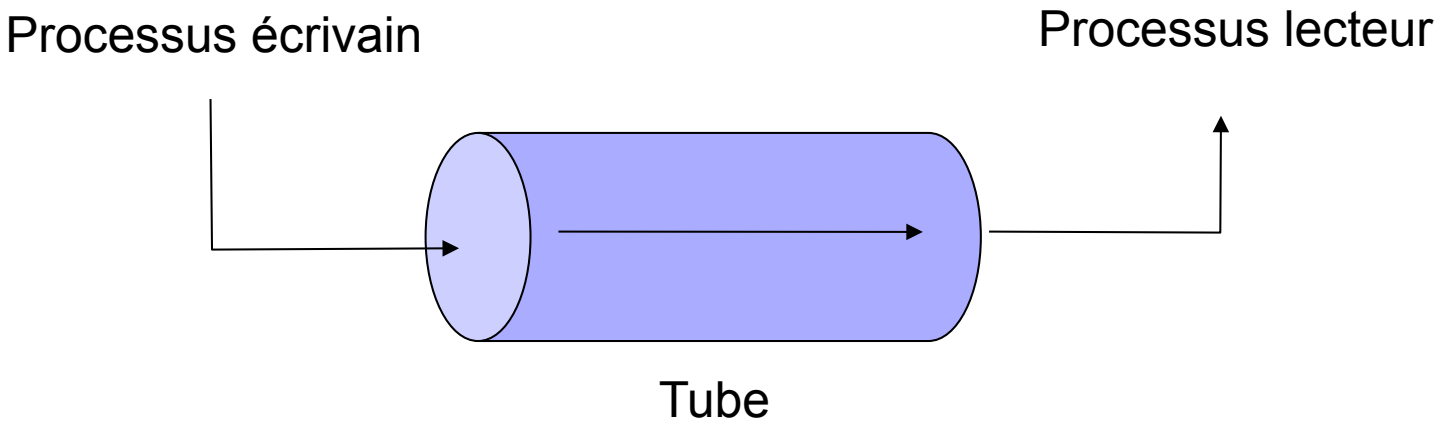
- Différentes solutions:
  - Par une zone mémoire
  - Par le biais d'un fichier
  - Par un outil fourni par le système d'exploitation

Outils dans Linux

- Appartenant au système de gestion de fichiers
  - Tubes anonymes
  - Tubes nommés
- Famille des IPC (Inter Processus Communication)
  - Files de messages (MSQ)
  - Mémoire partagée

Communication par tubes (1)

- Tuyau dans lequel un processus peut écrire des données qu'un autre processus peut lire.



Communication par tubes (2)

- Communication unidirectionnelle
  - Un lecteur ne peut devenir écrivain
- Les tubes sont gérés au niveau du système de fichier
- Gestion en flot d'octets (FIFO)
- Lecture destructive

Tubes anonymes

- Correspond à un fichier sans nom
- Nécessité de connaître les descripteurs lecture/écriture
  - Processus créateur du tube et descendant
  - Héritage des descripteurs

Création de tubes anonymes

- Primitive

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

- desc contient les descripteurs
  - desc[0] en lecture
  - desc[1] en écriture

Fermeture d'un tube anonyme

- Considéré fermé lorsque
  - Tous les descripteurs en lecture sont fermés
  - Tous les descripteurs en écriture sont fermés

- Primitive

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

Lecture dans un tube anonyme (1)

- Primitive

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

- Lecture de *nb* caractères depuis le tube *desc* dans le tampon *buf*
- Rendre la lecture non blocante

```
fcntl(desc[0], F_SETFL, O_NONBLOCK)
```

Lecture dans un tube anonyme (2)

- Tube non vide
  - Extrait du tube *nb* caractères ou moins
- Tube vide et nombre d'écrivain > 0
  - Lecture bloquante en attente écriture
- Tube vide et nombre d'écrivain = 0
  - Fin de fichier atteinte, aucun caractères rendus

Ecriture dans un tube anonyme (1)

- Primitive

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

- Ecriture de *nb* caractères dans le tube *desc* depuis le tampon *buf*
- Rendre l'écriture non blocante

```
fcntl(desc[1], F_SETFL, O_NONBLOCK)
```

Ecriture dans un tube anonyme (2)

- Nombre de lecteur = 0
  - Génération d'erreur et terminaison du processus
- Nombre de lecteur > 0
  - Ecriture bloquante de *nb* caractères dans le tube
  - Si *nb* > PIPE_BUF, la chaine est arbitrairement découpée

Ex: Communication unidirectionnelle

```
main() {
  int pip[2], status, pid;
  char chaine[5];

  pipe(pip);
  pid=fork();

  if(pid==0) {
    close (pip[0]);
    write (pip[1], "hello", 5);
    close (pip[1]);
    exit(0);
  }
  else {
    close (pip[1]);
    read (pip[0], chaine, 5);
    printf ("chaine recue : %s", chaine);
    close (pip[0]);
    wait(&status);
  }
}
```

Ex: Communication bidirectionnelle

```
main() {
  int pip1[2];
  int pip2[2];
  int status;
  int pid;
  char mesg[5], rep[7];

  pipe(pip1);
  pipe(pip2);

  pid=fork();

  if(pid==0) {
    close (pip1[1]);
    close (pip2[0]);
    read (pip1[0], mesg, 5);
    close(pip1[0]);
    printf ("chaine recue par le fils : %s", mesg);
    write (pip2[1], "bonjour", 7);
    close (pip2[1]);
    exit(0);
  }
```

Ex: Communication bidirectionnelle

```
  else {
    close (pip1[0]);
    close (pip2[1]);
    write (pip1[1], "hello", 5);
    close(pip1[1]);
    read (pip2[0], rep, 7);
    close (pip2[0]);
    printf ("réponse reçue par le père : %s", rep);
    exit(0);
  }
}
```

Tubes nommés

- Gérés par le système de fichiers
  - Correspond à un fichier avec un nom
- Accessible par tout processus
  - Connaissant le nom du fichier
  - Disposant des droits
- Affiché par ls -l
  - Caractérisés par le type P

Création d'un tube nommé

- Primitive

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

- *nom* correspond au nom du fichier
- *mode* correspond aux droits d'accès associés au tube

Ouverture d’un tube nommé

- Primitive

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

- Le processus appelant doit disposer des droits correspondants
- Renvoie un descripteur correspondant au mode d’ouverture spécifié
- Ouverture en lecture/écriture bloquante tant qu’il n’existe pas d’écrivain/lecteur

Lecture, écriture et fermeture d'un tube nommé

- Lecture / écriture
  - Primitives read() et write()
- Fermeture
  - Primitive close()
- Destruction
  - Primitive unlink()

Ex: Communication unidirectionelle (1)

```
main() {

  int tub;
  mode_t mode;
  mode = S_IRUSR | S_IWUSR;

  mkfifo ("fictub", mode);

  tub = open ("fictub", O_WRONLY);

  write (tub, "hello", 5);

  close(tub);

}
```

Ex: Communication unidirectionelle (2)

```
main() {

  char lec[6];
  int tub;


  tub = open ("fictub", O_RDONLY);

  read (tub, lec, 6);

  printf("message reçu : %s", lec);

  close(tub);

}
```

Ex: Communication bidirectionelle (1)

```
main() {

  int tub1, tub2;
  int nb1, nb2, res;
  char ent[2];
  mode_t mode;
  mode = S_IRUSR | S_IWUSR;

  mkfifo ("tube1", mode);
  mkfifo ("tube2", mode);

  tub1 = open ("tube1", O_RDONLY);

  tub2 = open ("tube2", O_WRONLY);

  read (tub1, ent, 2);
  nb1 = atoi(ent);
  read (tub1, ent, 2);
  nb2 = atoi(ent);

  res = nb1 + nb2;
  sprintf(ent", "%d", res);

  write (tub2, ent, 2);
  close (tub1); close (tub2);
}
```

Ex: Communication bidirectionelle (2)

```
main() {

  int tub1, tub2;
  int nb1, nb2, res;
  char ent[2];

  tub1 = open ("tube1", O_WRONLY);

  tub2 = open ("tube2", O_RDONLY);

  printf("entrez deux nombres à additionner :");
  scanf("%d %d", &nb1, &nb2);

  sprintf(ent, "%d", nb1);
  write (tub1, ent, 2);

  sprintf(ent, "%d", nb2);
  write (tub1, ent, 2);

  read (tub2, ent, 2);
  res = atoi(ent);

  printf("résultat : %d", res);

  close(tub1); close(tub2);
  unlink(tub1); unlink(tub2);
}
```

Les IPC

- Caractéristiques générales
  - Files de messages MSQ
  - Mémoire partagée
  - Sémaphores

- Gérés par les tables du système

Accessibilité

- Identifié de manière unique
  - Identifiant externe appelé "clé"
  - Identifiant interne (descripteur)
- Outil IPC accessible à tout processus connaissant le descripteur
  - Obtenu par héritage
  - Obtenu par demande explicite au système

Clé d'identification

- Valeur numérique
- Etre d'accord sur la valeur de la clé
  - Valeur figée dans le code
  - Calculée par le système à partir d'une référence commune à tous les processus
    - Composée de deux parties, nom de fichier + entier

```
key_t ftok (const char *ref, int numero);
```

Commande ipcs

```
bash-3.2$ ipcs
IPC status from <running system> as of Sun Oct  5 22:30:14 CEST 2008
T    ID    KEY        MODE       OWNER    GROUP
Message Queues:

T    ID    KEY        MODE       OWNER    GROUP
Shared Memory:

T    ID    KEY        MODE       OWNER    GROUP
Semaphores:
```

Files de messages

- Primitive

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

- *clé* identification externe
- *options* combinaison de constantes IPC_CREAT, IPC_EXCL et de droits d'accès

Accès à une file de message (1)

- Création en positionnant IPC_CREATE et IPC_EXCL
- Si une file existe déjà
  - IPC_EXCL positionné --> erreur
  - IPC_EXCL non positionné --> retourne la file
- Accès à une file existante
  - Valeur d'*option* à 0

Accès à une file de message (2)

- Cas particulier clé = IPC_PRIVATE
  - File uniquement accessible aux descendants
- Communication bidirectionnelle

Format des messages

- Composé de deux parties
  - Le type du message (entier long positif)
  - Les données proprement dites

- Type pointeur interdit car les données doivent être contiguës

Envoi de messages

- Primitive

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

- *idint* identification interne de la file
- *msg* le message
- *longueur* taille des données seules dans le message envoyé
- *options* exemple IPC_NOWAIT rend la primitive de dépôt non bloquante

Réception de messages

- Primitive

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv (idint, msg, l, type, option);
```

- *idint* identification interne de la file
- *msg* zone de réception du message
- *longueur* taille des données seules dans le message envoyé
- *type* désigne le message à extraire
- *options* exemple MSG_EXCEPT permet le retrait des messages de type différent de *type*

Type de message

- En fonction du type
  - >0 --> message le plus ancien de ce type est extrait de la file
  - null --> message le plus ancien est extrait de la file
  - <0 --> message le plus ancien dont le type est le plus petit <= à |*type*| est extrait

Destruction d'une file de message

- Primitive

  int msgctl (int idint, IPC_RMID, NULL);

- *Commande shell*
  - *ipcrm -q id*
  - *ipcrm -Q cle*

Ex: Format de message

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define CLEF_REQUETES          0x00012345
#define CLEF_REPONSES          0x00012346
#define LG_MAX                 512


struct msgform {
    long mtype;
    char mtext[ LG_MAX ];
}
msg;
```

Ex: Processus serveur

```
int main(void) {
    int res, i;
    int frequete, freponse;

    frequete = msgget(CLEF_REQUETES, 0700 | IPC_CREAT);
    if (frequete == -1) { perror("msgget"); return (EXIT_FAILURE); }

    freponse = msgget(CLEF_REPONSES, 0700 | IPC_CREAT);
    if (freponse == -1) { perror("msgget"); return (EXIT_FAILURE); }

    printf("Waiting a request...\n");
    res = msgrcv(frequete, & msg, LG_MAX, 0, 0);
    if (res == -1) { perror("msgrcv"); exit(0); }

    for(i=0; i < strlen(msg.mtext); i++)
        msg.mtext[i] = toupper(msg.mtext[i]);
    res = msgsnd(freponse, & msg, strlen(msg.mtext) + 1, 0);
    if (res == -1) { perror("msgsnd"); exit(0); }

    return (EXIT_SUCCESS);
}
```

Ex: Processus client

```
int main(void) {
    int res;
    int frequete, freponse;

    frequete = msgget(CLEF_REQUETES, 0700 | IPC_CREAT);
    if (frequete == -1) { perror("msgget"); return (EXIT_FAILURE); }

    freponse = msgget(CLEF_REPONSES, 0700 | IPC_CREAT);
    if (freponse == -1) { perror("msgget"); return (EXIT_FAILURE); }

    msg.mtype = getpid();
    strcpy(msg.mtext, "Hello");
    res = msgsnd(frequete, & msg, strlen(msg.mtext) + 1, 0);
    if (res == -1) { perror("msgsnd"); exit(0); }

    res = msgrcv(freponse, & msg, LG_MAX, getpid(), 0);
    if (res == -1) { perror("msgrcv"); exit(0); }

    printf("result : %s\n", msg.mtext);
    return (EXIT_SUCCESS);
}
```

Mémoire partagée

- Chaque processus dispose d'un espace d'adressage privé
- Extension de cet espace en mémoire partagée
- Etapes
  - Création de la région
  - Rattachement à son espace
- Difficulté: synchronisation

Création et accès

- Primitive

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget (key_t cle, int taille, int option);
```

- *cle* identification externe
- *taille* taille de la région mémoire
- *options* combinaison de constantes IPC_CREAT, IPC_EXCL et de droits d'accès

Attachement d'une région de mémoire partagée

- Primitive

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
shmat ( shmid, shmadd, option);
```

- *shmid* identification interne
- *shmadd* adresse de la mémoire partagée
  - =0 --> rattachement à la première adresse disponible
  - sinon rattachement à shmadd
- *options* droits d'accès à la région, read/write par défaut (ex: SMH_RONLY,…)

Détachement d'une région de mémoire partagée

- Primitive

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
shmdt (shmadd);
```

- La mémoire détachée devient inaccessible pour le processus appelant
- La terminaison d'un processus entraîne le détachement de toutes les régions

Destruction d'une région de mémoire partagée

- Primitive

  int shmctl (int shmid, IPC_RMID, NULL);

- *Commande shell*
  - *ipcrm -q id*
  - *ipcrm -Q cle*

Ex: Créateur et écrivain

```
#define CLE 256


main() {

  int shmid;
  char *mem;


  shmid = shmget((key_t)CLE, 1000, 0750|IPC_CREAT|IPC_EXCL);

  mem = shmat (shmid, NULL, 0);

  strcpy (mem, "hello");

  exit (0);

}
```

Ex: Lecteur et destructeur

```
#define CLE 256


main() {

  int shmid;
  char *mem;


  shmid = shmget((key_t)CLE, 0, 0);

  mem = shmat (shmid, NULL, 0);

  printf ("message : %s", mem);

  smhdt (mem);

  shmctl(shmid, IPC_RMID, NULL);

  exit (0);

}
```