

Séance machine - redirection et tuyaux

S3 - M3101

2017-2018

Introduction

Ce qu'on a vu

Depuis un programme en C/C++

processus

- ▶ lancer un autre processus : `fork()`
- ▶ faire exécuter un programme : `exec()`
- ▶ attendre leur fin : `waitpid()`, `wait()`

signaux

- ▶ notifier des évènements : `kill()`, `raise()`
- ▶ les traiter : `signal()`

Ce qu'on va voir

rediriger l'entrée/la sortie d'un programme

Equivalent de

```
ls -l > sortie.txt
```

échanger des données entre programmes

Tuyau entre programmes

```
du -s . | sort -n
```

Redirections

Idée

Si on fait exécuter

```
int main()
{
    execl("/bin/ls", "ls", "-l", NULL);
}
```

la commande "ls" hérite de la table de descripteurs ouverts

| numéro | nom | fichier ouvert |
|--------|---------------|----------------|
| 0 | STDIN_FILENO | clavier |
| 1 | STDOUT_FILENO | fenêtre |
| 2 | STDERR_FILENO | fenêtre |

Idée : rediriger la sortie vers sortie.txt

Ce qu'on veut :

| numéro | nom | fichier ouvert |
|----------|----------------------|---|
| 0 | STDIN_FILENO | clavier |
| 1 | STDOUT_FILENO | fichier sortie.txt en écriture <= |
| 2 | STDERR_FILENO | fenêtre |

Étapes :

1. obtenir un descripteur fd en ouvrant le fichier
2. le copier dans celui de STDOUT_FILENO
3. fermer fd

Étape 1 : ouvrir le fichier de sortie (open)

```
int fd = open("sortie.txt",           // pathname
              O_CREAT | O_RDONLY,     // flags
              0644);                 // mode = permissions
```

Après :

| numéro | nom | fichier ouvert |
|--------|---------------|------------------------------|
| 0 | STDIN_FILENO | clavier |
| 1 | STDOUT_FILENO | fenêtre |
| 2 | STDERR_FILENO | fenêtre |
| => 3 | | fichier sortie.txt <= |

Étape 2 : dupliquer le descripteur (dup2)

```
dup2(fd, STDOUT_FILENO); // duplicate fd to STDOUT_FILENO
```

Après :

| numéro | nom | fichier ouvert |
|--------|---------------|------------------------------|
| 0 | STDIN_FILENO | clavier |
| => 1 | STDOUT_FILENO | fichier sortie.txt <= |
| 2 | STDERR_FILENO | fenêtre |
| 3 | | fichier sortie.txt |

Étape 3 : fermer le descripteur inutile (close)

```
close(fd);           // close file descriptor
```

Après :

| numéro | nom | fichier ouvert |
|--------|---------------|---------------------------|
| 0 | STDIN_FILENO | clavier |
| 1 | STDOUT_FILENO | fichier sortie.txt |
| 2 | STDERR_FILENO | fenêtre |
| — | — | — |

En résumé

```
int main()
{
    int fd = open("sortie.txt",
                  O_CREAT | O_RDONLY,
                  0644);
    dup2(fd, STDOUT_FILENO);
    close(fd);

    execl("/bin/ls",
          "ls", "-l", NULL);
}
```

Paramètre d'appel pour open()

appel

```
int fd = open(chemin_d_accès, indicateurs, mode);
```

Indicateurs : combinaison avec

- ▶ **obligatoirement** un des modes d'accès O_RDONLY, O_WRONLY, O_RDWR
- ▶ O_CREAT, créer le fichier si il n'existe pas
- ▶ O_APPEND, écrire à la fin du fichier existant
- ▶ O_TRUNC, vider avant d'y écrire
- ▶ ...

Mode = permissions

Permissions d'accès

- ▶ droits d'accès pour **utilisateur, groupe, autres**
- ▶ on les écrit souvent en **octal** (nombre commençant par **0**)
- ▶ constantes symboliques

```
S_IRUSR  00400 user has read permission  
S_IWUSR  00200 user has write permission  
S_IXUSR  00100 user has execute permission  
...
```

voir man 2 open

Exercice

Écrire un programme C++ : min2maj f1 f2

- ▶ prend un fichier texte en entrée et fabrique un fichier de sortie
- ▶ les minuscules ont été converties en majuscules.

Indication : en langage de commande, on utiliserait `/usr/bin/tr`

```
tr [a-z] [A-Z] < entree > sortie
```

Amélioration

- ▶ sortie standard si f2 absent ou égal à “-”
- ▶ entrée standard si f1 absent ou égal à “-”

Remarque : indicateur O_CLOEXEC (*close on exec*)

- ▶ un descripteur marqué O_CLOEXEC n'est pas transmis par exec()
- ▶ l'indicateur O_CLOEXEC n'est pas copié par dup2()

```
int main()
{
    int fd = open("sortie.txt",
                  O_CREAT | O_RDONLY | O_CLOEXEC,
                  0644);
    dup2(fd, STDOUT_FILENO);

    // close(fd);

    execl("/bin/ls",
          "ls", "-l", NULL);
}
```

Travail : préparer l'intégration dans le shell

Ecrire une fonction

```
void executer_programme(const vector<string> & mots);
```

qui lance un programme en tenant compte des redirections

Tests :

```
executer_programme({ "ls" , "-l" });  
executer_programme({ "ls" , "-l", ">", "r.txt"});  
executer_programme({ "<", "f.cc", "wc", "-l",  
                    ">", "r.txt"});
```


Tuyaux

Tuyau = mécanisme de communication

L'appel système `pipe()` retourne deux descripteurs de fichiers, connectés à un tampon de données.

- ▶ on peut écrire par un descripteur
- ▶ on peut lire par l'autre



Figure 1 – tuyau

suite

- ▶ La **capacité** du tampon est **limitée**
 - ▶ POSIX : au moins 512 octets
 - ▶ Linux : de 4 à 64 Ko
- ▶ La lecture et l'écriture sont
 - ▶ **atomiques** : deux écritures simultanées ne peuvent pas se mélanger
 - ▶ **bloquantes** : écriture dans tampon trop plein, lecture tampon vide

Exemple

- ▶ Un processus produit des messages (nombres)
- ▶ Le processus père les lit, et les affiche

```
function main
  creer pipe
  lancer produire
  consommer
```

```
fonction produire
  pour i de 1 à 10
    | envoyer i
    | attendre 1 sec
```

```
fonction consommer
  tant que c'est possible
    | lire un nombre
    | l'afficher
```

Code du producteur

```
struct Message {    # choix : transmission de structures
    int data;        # de taille fixe
};

void produire(int sortie)
{
    for (int i = 1; i<=10; i++) {
        Message m;
        m.data = i;
        write(sortie, & m, sizeof m);
    }
    close(sortie);
}
```

Code du consommateur

```
void consommer(int entree)
{
    while (true) {
        Message m;
        int lus = read(entree, & m, sizeof m);
        if (lus != sizeof m) {
            break;
        }
        cout << m.data << endl;
    }
    close(entree);
}
```

Le programme (main)

```
int fds[2];
pipe(fds);
int entree = fds[0], sortie = fds[1];

if (fork() == 0) { // processus fils
    close(entree);
    produire(sortie);
    exit(EXIT_SUCCESS);
}
close(sortie); // père
consommer(entree);
wait (nullptr);
exit(EXIT_SUCCESS);
```

Question

```
if (fork() == 0) { // processus fils
    close(entree);
    produire(sortie);
    exit(EXIT_SUCCESS);
}
close(sortie);      // père
consommer(entree);
```

Que se passe-t'il si le processus père ne fait pas `close(sortie)` ?

Synthèse

Tuyaux

- ▶ appel `pipe(fds)` pour créer un tuyau
- ▶ retourne deux descripteurs dans tableau `int fds[2]` ;
- ▶ lecture dans `fds[0]`, écriture dans `fds[1]`

Fin de fichier

La lecture détecte une fin de fichier si

- ▶ il n'y a plus de données à lire,
- ▶ toutes les copies du descripteur `fds[1]` sont fermées

Synthèse (suite)

écriture

- ▶ `write(fd, adresse, nombre d'octets)`

Paramètres :

- ▶ descripteur
- ▶ adresse du premier octet à transmettre
- ▶ nombre d'octets à transmettre

Synthèse (suite)

lecture

- ▶ `n = read(fd, adresse, nombre d'octets)`

Paramètres

- ▶ descripteur
- ▶ adresse du tampon de réception
- ▶ taille maximum

retourne le nombre d'octets *effectivement* lus

- ▶ 0 en fin de fichier
- ▶ -1 si erreur.

Travail en groupe : pipeline

Sujet

Ecrire un “pipe-line” enchainant plusieurs actions simples

- ▶ produire les entiers de 1 à 10
- ▶ sélectionner ceux qui ne sont pas multiples de 3
- ▶ les multiplier par 100
- ▶ faire afficher les résultats

Chaque action aura en paramètre un descripteur d'entrée ou un descripteur de sortie, ou les deux.

Le résultat devrait être 100, 200, 400, 500, 700, 800, 1000.

Répartition du travail

Objectif A la fin, chacun est capable de présenter le programme au nom du groupe (de 3)

1. chacun réalise le programme réduit “produire les entiers de 1 à 10, les afficher”
2. coordination : aidez-vous pour que **tout le monde** comprenne
3. chacun ajoute un “filtre” : sélection, ou transformation
4. coordination : mettre en commun les filtres
5. intégrer l'autre filtre.
6. coordination : tout le monde a un programme qui marche.