Redirections

1. Table des descripteurs fichiers, des fichiers ouverts, des inodes libres

2. Redirections

Crédits

- Un cours sur les fichiers de Bernard Goossens

Crédits

- Un cours sur les fichiers de Bernard Goossens
- Une page de Wikipedia

TDF, TFO

- Dans POSIX, Un *descripteur de fichier* est un entier qui désigne une position dans une table appelée *table des descripteurs de fichiers* (TDF).

TDF, TFO

- Dans POSIX, Un *descripteur de fichier* est un entier qui désigne une position dans une table appelée *table des descripteurs de fichiers* (TDF).
- Chaque *processus* (c'est à dire un programme en cours d'exécution) est créé avec sa propre TDF : elle indique les fichiers utilisés par ce processus. Il y a donc une TDF par processus.

TDF, TFO

- Dans POSIX, Un *descripteur de fichier* est un entier qui désigne une position dans une table appelée *table des descripteurs de fichiers* (TDF).
- Chaque *processus* (c'est à dire un programme en cours d'exécution) est créé avec sa propre TDF : elle indique les fichiers utilisés par ce processus. Il y a donc une TDF par processus.
- Ce qui nous intéresse ici, ce sont les 3 premières entrées de la TDF : les 3 flux standards.

TDF, TFO

- Dans POSIX, Un *descripteur de fichier* est un entier qui désigne une position dans une table appelée *table des descripteurs de fichiers* (TDF).
- Chaque *processus* (c'est à dire un programme en cours d'exécution) est créé avec sa propre TDF : elle indique les fichiers utilisés par ce processus. Il y a donc une TDF par processus.
- Ce qui nous intéresse ici, ce sont les 3 premières entrées de la TDF : les 3 flux standards.
- A chaque entrée de la TDF on trouve une structure qui contient un champ d'attributs du descripteur (des informations sur le fichier correspondant) et un pointeur vers la *table des fichiers ouverts* (TFO), laquelle est unique pour le système.

TDF, TFO

- Dans POSIX, Un *descripteur de fichier* est un entier qui désigne une position dans une table appelée *table des descripteurs de fichiers* (TDF).
- Chaque *processus* (c'est à dire un programme en cours d'exécution) est créé avec sa propre TDF : elle indique les fichiers utilisés par ce processus. Il y a donc une TDF par processus.
- Ce qui nous intéresse ici, ce sont les 3 premières entrées de la TDF : les 3 flux standards.
- A chaque entrée de la TDF on trouve une structure qui contient un champ d'attributs du descripteur (des informations sur le fichier correspondant) et un pointeur vers la *table des fichiers ouverts* (TFO), laquelle est unique pour le système.
- Il y a donc dans le système une seule TFO et une multitude de TDF (une pour chaque processus).

TFO, TIM

- Chaque entrée de la TFO se compose

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent
  - des informations sur le nombre de pointeurs TFO vers cet inode,

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent
  - des informations sur le nombre de pointeurs TFO vers cet inode,
  - le device de l'inode (quel disque ou partition),

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent
  - des informations sur le nombre de pointeurs TFO vers cet inode,
  - le device de l'inode (quel disque ou partition),
  - son numéro,

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent
  - des informations sur le nombre de pointeurs TFO vers cet inode,
  - le device de l'inode (quel disque ou partition),
  - son numéro,
  - son statut (modifié ou non),

TFO, TIM

- Chaque entrée de la TFO se compose
  - d'un champ compteur de références (nombre de pointeurs tdf) sur cette entrée,
  - d'un champ mode d'ouverture du fichier (lecture, écriture, etc.),
  - d'un champ de position dans le fichier (déplacement en octet par rapport au début)
  - et d'un champ pointeur sur la *table des inodes en mémoire* (TIM)
- A leur tour les entrées de la TFO pointent vers une *table des inodes en mémoire*, laquelle est également unique.
- Les éléments de le TIM sont des structures qui contiennent
  - des informations sur le nombre de pointeurs TFO vers cet inode,
  - le device de l'inode (quel disque ou partition),
  - son numéro,
  - son statut (modifié ou non),
  - une copie de l'inode et notamment de sa table des blocs.

Les 3 premières entrées de la TDF (POSIX)

Le programme de MP2I ne mentionne que les 3 premières entrées de la TDF. Elles correspondent aux flux standards (voir figure 1) :

- l'*entrée standard* ou *point de lecture des processus en avant-plan*. C'est un *tuyau de données* (flux), que l'on branche sur un périphérique. Elle est représentée par la constante **STDIN_FILENO** qui vaut 0. Par défaut, ce flux est associé au clavier.

Les 3 premières entrées de la TDF (POSIX)

Le programme de MP2I ne mentionne que les 3 premières entrées de la TDF. Elles correspondent aux flux standards (voir figure 1) :

- l'*entrée standard* ou *point de lecture des processus en avant-plan*. C'est un *tuyau de données* (flux), que l'on branche sur un périphérique. Elle est représentée par la constante **STDIN_FILENO** qui vaut 0. Par défaut, ce flux est associé au clavier.
- La *sortie standard* ou *point d'écriture des processus en avant-plan*. c'est un flux qui est représenté par la constante **STDOUT_FILENO**, laquelle vaut 1. Par défaut, ce flux est associé à la console d'où l'application à été lancée.

Les 3 premières entrées de la TDF (POSIX)

Le programme de MP2I ne mentionne que les 3 premières entrées de la TDF. Elles correspondent aux flux standards (voir figure 1) :

- l'*entrée standard* ou *point de lecture des processus en avant-plan*. C'est un *tuyau de données* (flux), que l'on branche sur un périphérique. Elle est représentée par la constante **STDIN_FILENO** qui vaut 0. Par défaut, ce flux est associé au clavier.
- La *sortie standard* ou *point d'écriture des processus en avant-plan*. c'est un flux qui est représenté par la constante **STDOUT_FILENO**, laquelle vaut 1. Par défaut, ce flux est associé à la console d'où l'application à été lancée.
- la *sortie standard d'erreur* ou *point d'écriture des erreurs des processus en avant-plan*. C'est un flux représenté par la constante **STDERR_FILENO** qui vaut 2. Par défaut, ce flux est associé à la console d'où l'application à été lancée

Les 3 premières entrées de la TDF

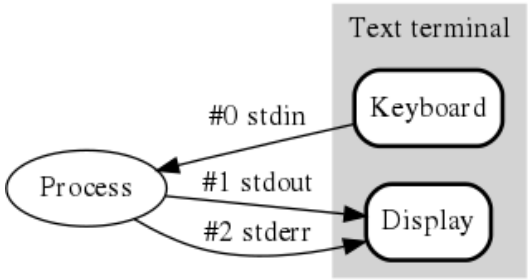

FIGURE 1 – Les descripteurs de fichiers pour l'entrée, la sortie et les erreurs (d'après Wikipedia).Par défaut, quand un programme fonctionne dans un terminal, l'entrée standard correspond au clavier, et la sortie standard ainsi que l'erreur standard sont affichées dans ce terminal.

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,
  - **stdout** pour la sortie standard,

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,
  - **stdout** pour la sortie standard,
  - **stderr** pour l'erreur standard.

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,
  - **stdout** pour la sortie standard,
  - **stderr** pour l'erreur standard.
- Ces pointeurs peuvent être utilisés directement avec la majorité des fonctions qui agissent sur les fichiers. Par exemple :

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,
  - **stdout** pour la sortie standard,
  - **stderr** pour l'erreur standard.
- Ces pointeurs peuvent être utilisés directement avec la majorité des fonctions qui agissent sur les fichiers. Par exemple :
  - La fonction **scanf** utilise le flux **stdin**.

Lien avec le C

- Le fichier d'en-tête **stdio.h** de la bibliothèque standard du C définit trois pointeurs de type **FILE*** qui représentent les flux standard :
  - **stdin** pour l'entrée standard,
  - **stdout** pour la sortie standard,
  - **stderr** pour l'erreur standard.
- Ces pointeurs peuvent être utilisés directement avec la majorité des fonctions qui agissent sur les fichiers. Par exemple :
  - La fonction **scanf** utilise le flux **stdin**.
  - La fonction **printf** utilise le flux **stdout**.

Rappel

Pour dialoguer avec son ordinateur, on utilise un terminal. Par défaut les commandes du terminal récupèrent les données tapées par l'utilisateur au clavier. Le résultat de leur exécution s'affiche à l'écran. En cas d'erreur à l'exécution, les messages d'erreur apparaissent aussi à l'écran.

Redirections d'entrées/sorties

Il est possible d'indiquer au terminal que des données en entrée ou sortie doivent être récupérées/stockées dans un fichier plutôt qu'au clavier ou à l'écran. Voici la syntaxe des commandes de redirection :

- $<$ **entree** signifie que les données de la commande seront lues dans le fichier de nom **entree** plutôt qu'au clavier. On veut qu'un fichier soit l'entrée d'une commande qui, normalement, n'accepte pas un fichier comme option.

Redirections d'entrées/sorties

Il est possible d'indiquer au terminal que des données en entrée ou sortie doivent être récupérées/stockées dans un fichier plutôt qu'au clavier ou à l'écran. Voici la syntaxe des commandes de redirection :

- $<$ **entree** signifie que les données de la commande seront lues dans le fichier de nom **entree** plutôt qu'au clavier. On veut qu'un fichier soit l'entrée d'une commande qui, normalement, n'accepte pas un fichier comme option.
- $>$ **sortie** signifie que les données générées par la commande seront écrites dans le fichier de nom **sortie** plutôt qu'à l'écran. Si le fichier sortie existait déjà, son ancien contenu est effacé, sinon ce fichier est créé au lancement de la commande.

Redirections d'entrées/sorties

Il est possible d'indiquer au terminal que des données en entrée ou sortie doivent être récupérées/stockées dans un fichier plutôt qu'au clavier ou à l'écran. Voici la syntaxe des commandes de redirection :

- $<$ **entree** signifie que les données de la commande seront lues dans le fichier de nom **entree** plutôt qu'au clavier. On veut qu'un fichier soit l'entrée d'une commande qui, normalement, n'accepte pas un fichier comme option.
- $>$ **sortie** signifie que les données générées par la commande seront écrites dans le fichier de nom **sortie** plutôt qu'à l'écran. Si le fichier sortie existait déjà, son ancien contenu est effacé, sinon ce fichier est créé au lancement de la commande.
- $>>$ **sortie** semblable au cas précédent sauf que si le fichier **sortie** existait déjà, son ancien contenu est conservé et les nouvelles données sont copiées à la suite.

Redirections d'erreurs

2> erreur signifie que les messages d'erreur générés par la commande seront écrits dans le fichier de nom erreur plutôt qu'à l'écran. Si le fichier **erreur** existait déjà, son ancien contenu est effacé, sinon ce fichier est créé au lancement de la commande.

Redirections d'erreurs

2> erreur signifie que les messages d'erreur générés par la commande seront écrits dans le fichier de nom erreur plutôt qu'à l'écran. Si le fichier **erreur** existait déjà, son ancien contenu est effacé, sinon ce fichier est créé au lancement de la commande.

2>> erreur semblable au cas précédent sauf que si le fichier **erreur** existait déjà, son ancien contenu est conservé et les nouveaux messages d'erreur sont copiés à la suite.

Exemple de redirection d'entrée

```
ivan@fixe:~/.../UFS$ touch file.txt
ivan@fixe:~/.../UFS$ echo "Hello world" > file.txt
ivan@fixe:~/.../UFS$ cat file.txt
Hello world
ivan@fixe:~/.../UFS$ wc -l file.txt
1 file.txt
ivan@fixe:~/.../UFS$ wc -l <file.txt
1
```

- On crée un fichier vide (**touch file.txt**) dont on écrit une ligne avec **echo "Hello world" > file.txt**

Exemple de redirection d'entrée

```
ivan@fixe:~/.../UFS$ touch file.txt
ivan@fixe:~/.../UFS$ echo "Hello world" > file.txt
ivan@fixe:~/.../UFS$ cat file.txt
Hello world
ivan@fixe:~/.../UFS$ wc -l file.txt
1 file.txt
ivan@fixe:~/.../UFS$ wc -l <file.txt
1
```

- On crée un fichier vide (**touch file.txt**) dont on écrit une ligne avec **echo "Hello world" > file.txt**
- La commande **wc -l file.txt** compte le nombre de ligne d'un fichier et affiche le nom de ce fichier.

Exemple de redirection d'entrée

```
ivan@fixe:~/.../UFS$ touch file.txt
ivan@fixe:~/.../UFS$ echo "Hello world" > file.txt
ivan@fixe:~/.../UFS$ cat file.txt
Hello world
ivan@fixe:~/.../UFS$ wc -l file.txt
1 file.txt
ivan@fixe:~/.../UFS$ wc -l <file.txt
1
```

- On crée un fichier vide (**touch file.txt**) dont on écrit une ligne avec **echo "Hello world" > file.txt**
- La commande **wc -l file.txt** compte le nombre de ligne d'un fichier et affiche le nom de ce fichier.
- Avecd la redirection <, on redirige l'entrée standard vers **file.txt**. On obtient `1` comme sortie puisque la commande supppose qu'elle prend son entrée dans `stdin` plutôt que dans un fichier. Le nom de fichier a disparu !

Fusion des sorties et des erreurs

Supposons que dans un répertoire on dispose d'un fichier **asup** mais pas de **asip**.

- La commande **ls -l asup asip** va afficher avec succès les attributs de **asup** mais va exprimer une erreur pour **asip**.

```
$ ls -l asup asip
ls: impossible d'accéder à'asip': Aucun fichier ou dossier de ce type
-rw-rw-r-- 1 ivan ivan 0 août 26 2021 asup
```

Fusion des sorties et des erreurs

Supposons que dans un répertoire on dispose d'un fichier **asup** mais pas de **asip**.

- La commande **ls -l asup asip** va afficher avec succès les attributs de **asup** mais va exprimer une erreur pour **asip**.

```
$ ls -l asup asip
ls: impossible d'accéder à'asip': Aucun fichier ou dossier de ce type
-rw-rw-r-- 1 ivan ivan 0 août 26 2021 asup
```

- On veut écrire dans un même fichier ces deux sorties. On utilise pour cela l'opérateur de redirection **2>1&** :

```
$ ls -l asup asip > foo 2>&1 # fusion des sorties et erreurs dans foo
$ cat foo # affichage du contenu de foo
ls: impossible d'accéder à'asip': Aucun fichier ou dossier de ce type
-rw-rw-r-- 1 ivan ivan 0 août 26 11:15 asup
```

Fusion des sorties et des erreurs

Supposons que dans un répertoire on dispose d'un fichier **asup** mais pas de **asip**.

- La commande **ls -l asup asip** va afficher avec succès les attributs de **asup** mais va exprimer une erreur pour **asip**.

```
$ ls -l asup asip
ls: impossible d'accéder à'asip': Aucun fichier ou dossier de ce type
-rw-rw-r-- 1 ivan ivan 0 août 26 2021 asup
```

- On veut écrire dans un même fichier ces deux sorties. On utilise pour cela l'opérateur de redirection **2>1&** :

```
$ ls -l asup asip > foo 2>&1 # fusion des sorties et erreurs dans foo
$ cat foo # affichage du contenu de foo
ls: impossible d'accéder à'asip': Aucun fichier ou dossier de ce type
-rw-rw-r-- 1 ivan ivan 0 août 26 11:15 asup
```

- L'instruction **ls asup asip 2> foo 1>&2** aurait le même effet.

Exercice

Exercice

Écrire un programme **simple.c** dont le **main** fait un affichage mais sans aucune inclusion de bibliothèque.
Le compiler avec **gcc** et rediriger les erreurs vers un fichier **erreursimple.txt**

Exercice

Exercice

1. Consulter le manuel de **tr**
2. Supprimer la lettre « o » de tous les mots du fichier **myfile**

Exercice

Exercice

Avec la commande **ls** lister les droits du contenu du répertoire courant et d'un fichier **t_oufoukoa** qui n'existe pas.
La sortie de **ls** doit être redirigée vers un fichier **liste_fichiers** tandis que les erreurs sont envoyées vers **erreurls**.

Les pipes

- Un *pipe* (mot anglais signifiant "tuyau") permet de passer le résultat d'une commande shell à une autre commande shell.

Les pipes

- Un *pipe* (mot anglais signifiant "tuyau") permet de passer le résultat d'une commande shell à une autre commande shell.
- Les données en sortie d'une commande sont utilisées en entrée de la commande suivante sans qu'il soit nécessaire de recourir à des fichiers intermédiaires.

Syntaxe des pipes

- La syntaxe générale pour les pipes est

```
commande1 options arguments | commande2 options | commande3...
```

Cette barre verticale, le pipe, s'obtient avec la combinaison de touche ALT GR + 6 (touche ALT GR maintenue enfoncée pendant qu'on presse 6).

Syntaxe des pipes

- La syntaxe générale pour les pipes est

```
commande1 options arguments | commande2 options | commande3...
```

Cette barre verticale, le pipe, s'obtient avec la combinaison de touche ALT GR + 6 (touche ALT GR maintenue enfoncée pendant qu'on presse 6).

- Par exemple, on liste le contenu du répertoire courant et on envoie le résultat à la commande **sort** avec l'option **-r** (pour reverse). L'affichage du contenu se fera dans l'ordre alphabétique inverse (à cause du **r**) :

```
$ ls # affichage habituel de ls
bla bli foo zut
$ ls | sort -r # trier le résultat dans l'ordre alphabétique inverse
zut
foo
bli
bla
```