Redirections

Les redirections sont couramment utilisées dans les commandes Unix. Elles permettent de récupérer le résultat d'une ou de plusieurs commandes dans un fichier ou au contraire de faire lire un fichier à une commande. Cette partie expose de manière détaillée les différentes syntaxes possibles avec leur mécanisme interne associé.

Les redirections sont mises en place par le shell.

1. Entrée et sorties standard des processus

Les processus Unix ont, par défaut, leur fichier terminal ouvert trois fois, sous trois descripteurs de fichier différents.

a. Entrée standard

Le descripteur de fichier 0 est nommé également **Entrée standard du processus**. Les processus qui attendent des informations de la part de l'utilisateur déclenchent en fait une requête de lecture sur le descripteur 0. Si ce dernier est associé au terminal, ce qui est le cas par défaut, cela se matérialise pour l'utilisateur par une demande de saisie au clavier.

La majorité des commandes utilisent l'entrée standard pour déclencher une saisie. Il existe cependant des exceptions. Par exemple, la commande **`passwd`** ouvre le fichier terminal sous un autre descripteur.

---

b. Sortie standard

Le descripteur de fichier 1 est nommé également **Sortie standard du processus**. Par convention, un processus qui souhaite envoyer un message résultat à l'utilisateur doit le faire transiter via le descripteur 1. Si ce dernier est associé au terminal, ce qui est le cas par défaut, cela se matérialise pour l'utilisateur par un affichage à l'écran.

c. Sortie d'erreur standard

Le descripteur de fichier 2 est nommé également **Sortie d'erreur standard du processus**. Par convention, un processus qui souhaite envoyer un message d'erreur à l'utilisateur doit le faire transiter via le descripteur 2. Si ce dernier est associé au terminal, ce qui est le cas par défaut, cela se matérialise pour l'utilisateur par un affichage à l'écran.

2. Héritage

Les descripteurs de fichiers sont hérités lors de la duplication (fork()) et du remplacement (exec()). La figure 4 montre que le ksh enfant, puis la commande **`find`** héritent de la table des descripteurs de fichiers du shell parent. Cette commande envoie sur la sortie standard les résultats et sur la sortie d'erreur standard la liste des répertoires que l'utilisateur n'a pas le droit d'explorer. Comme les sorties 1 et 2 sont orientées sur le même terminal, les affichages s'entrelacent. L'un des intérêts des redirections est de pouvoir séparer les messages provenant de chaque descripteur.

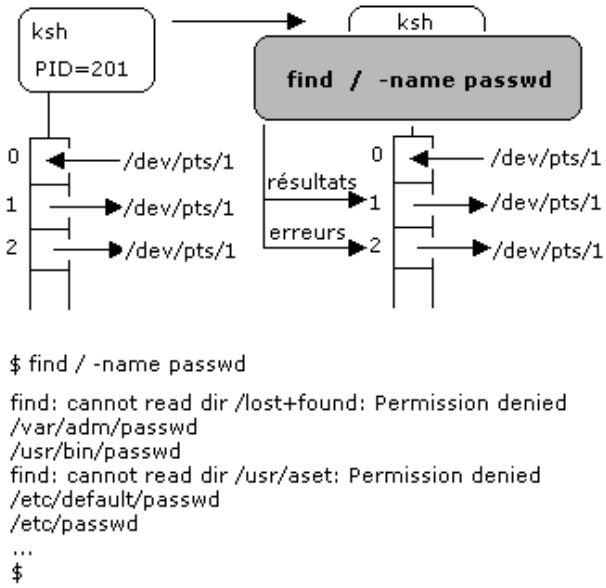

Figure 4 : Héritage de la table des descripteurs de fichiers

3. Redirection des sorties en écriture

La redirection en écriture permet d'envoyer les affichages liés à un descripteur particulier, non plus sur le terminal, mais dans un fichier.

a. Sortie standard

Simple redirection

Syntaxe

```
$ commande 1> fichier
```

équivalent à :

```
$ commande > fichier
```

Le nom du fichier est exprimé en relatif ou en absolu.

Si le fichier n'existe pas, il est créé. Si le fichier existe déjà, il est écrasé.

L'opération de création ou d'écrasement du fichier est réalisée par le shell.

Exemple

*Récupérer le résultat de la commande **ls** dans le fichier **resu** :*

```
$ ls > resu
$ cat resu
FIC
erreur
out1
resu
$
```

Double redirection

Elle permet de concaténer les messages résultat d'une commande au contenu d'un fichier déjà existant.

Syntaxe

```
$ commande 1>> fichier
```

équivalent à :

```
$ commande >> fichier
```

Si le fichier n'existe pas, il est créé. Si le fichier existe déjà, il est ouvert en mode ajout.

Exemple

*Ajouter le résultat de la commande **date** à la fin du fichier **resu** créé précédemment :*

```
$ date >> resu
$ cat resu
FIC
erreur
out1
resu
Tue Jan 21 18:31:56 WET 2003
$
```

b. Sortie d'erreur standard

Simple redirection

Syntaxe

```
$ commande 2> fichier
```

Exemple

*Redirection de la sortie d'erreur standard. Les messages d'erreur partent dans le fichier **erreur**, les résultats restent à l'écran :*

```
$ find / -name passwd 2> erreur
/var/adm/passwd
/usr/bin/passwd
/etc/default/passwd
/etc/passwd
$ cat erreur
find: cannot read dir /lost+found: Permission denied
find: cannot read dir /usr/aset: Permission denied
...
$
```

Double redirection

Elle permet de concaténer les messages d'erreur d'une commande au contenu d'un fichier existant.

Syntaxe

```
$ commande 2>> fichier
```

Exemple

*Concaténation des messages d'erreur de **ls-z** à la fin du fichier **erreur** :*

```
$ ls -z
ls: illegal option -- z
usage: ls -1RaAdCxmnlogrtucpFbqisfL [files]
$ ls -z 2>> erreur
$ cat erreur
find: cannot read dir /lost+found: Permission denied
find: cannot read dir /usr/aset: Permission denied
ls: illegal option -- z
usage: ls -1RaAdCxmnlogrtucpFbqisfL [files]
$
```

c. Sortie standard et sortie d'erreur standard

Il est possible de rediriger plusieurs descripteurs sur une même ligne de commande.

```
$ commande 1> fichier_a 2> fichier_b
```

ou

```
$ commande 2> fichier_b 1> fichier_a
```

Les redirections sont toujours traitées de gauche à droite. Dans le cas présent, l'ordre d'écriture des deux redirections n'a pas d'importance, ce qui ne sera pas toujours le cas (cf. le point Redirections - Redirections avancées).

Exemple

```
$ find / -name passwd 1> resu 2> erreur
$ cat resu
/var/adm/passwd
/usr/bin/passwd
/etc/default/passwd
/etc/passwd
$ cat erreur
find: cannot read dir /lost+found: Permission denied
find: cannot read dir /usr/aset: Permission denied
...
$
```

d. Éliminer les affichages

Toutes les plates-formes Unix possèdent un fichier spécial nommé **/dev/null** qui permet de faire disparaître les affichages. Ce fichier est géré comme un périphérique et n'a pas de notion de contenu. On peut donc considérer qu'il est toujours vide.

Exemple

```
$ find / -name passwd 1> resu 2> /dev/null
$ cat resu
/var/adm/passwd
/usr/bin/passwd
/etc/default/passwd
/etc/passwd
$ ls -lL /dev/null
crw-rw-rw-  1 root   sys    13, 2 Jan 21 17:22 /dev/null
$ cat /dev/null
$
```

e. Mécanisme interne

Cas de la commande externe

C'est le shell enfant qui s'occupe des redirections (cf. Figure 5 et Figure 6).

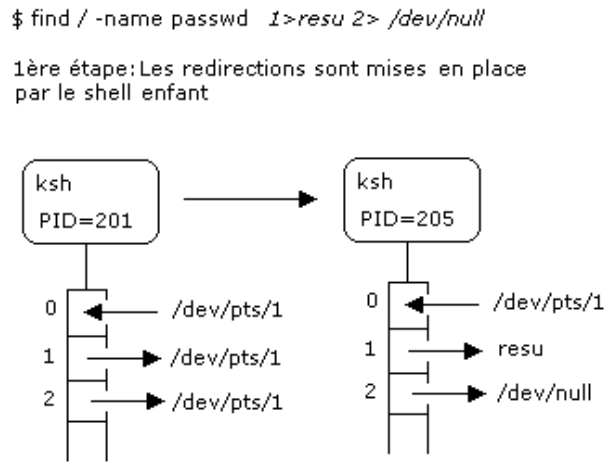

Figure 5 : Mécanisme interne des redirections en écriture - Première étape

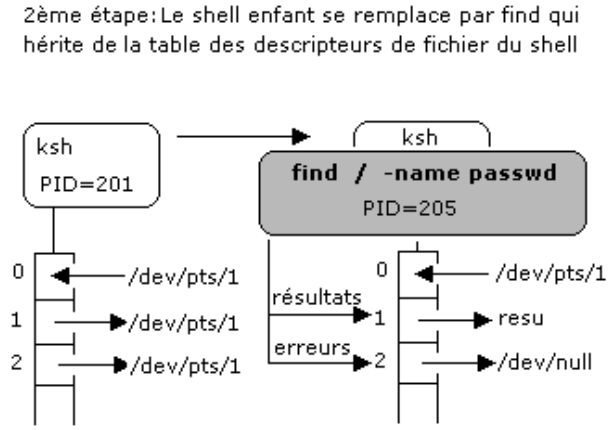

Figure 6 : Mécanisme interne des redirections en écriture - Deuxième étape

Cas de la commande interne

Une commande interne est exécutée par le shell courant. Ce dernier gère lui-même les redirections. Pour cela, il sauvegarde les associations **descripteur-fichier** courantes, connecte les descripteurs sur les fichiers demandés, exécute la commande, puis restaure l'environnement **descripteur-fichier** précédent.

4. Redirection de l'entrée standard

La redirection de l'entrée standard concerne les commandes qui utilisent le descripteur 0, autrement dit celles qui déclenchent une saisie au clavier.

Exemple

```
$ mail olive
RV a 13 heures au restaurant    (Entrée standard)
Christine                       (Entrée standard)
^d                              (Entrée standard)
$
```

La commande **mail** lit l'entrée standard jusqu'à réception d'une fin de fichier (touches ^d). Les données saisies seront envoyées dans la boîte aux lettres de l'utilisateur **olive**.

Si l'on souhaite faire lire à la commande **mail**, non plus le clavier, mais le contenu d'un fichier, il suffit de connecter le descripteur 0 sur le fichier désiré.

Syntaxe

```
$ commande 0< fichier_message
```

équivalent à

```
$ commande < fichier_message
```

Exemple

```
$ cat message
RV a 13 heures au restaurant
Christine
$ mail olive < message
$
```

Les figures 7 et 8 représentent le mécanisme interne associé.

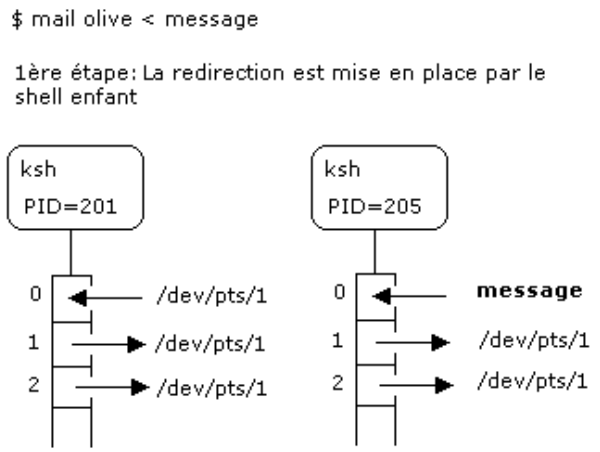

Figure 7 : Mécanisme interne de la redirection en lecture - Première étape

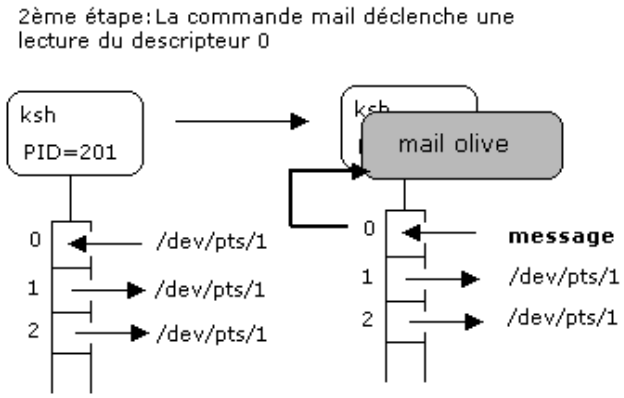

Figure 8 : Mécanisme interne de la redirection en lecture - Deuxième étape

5. Redirections avancées

a. Rediriger les descripteurs 1 et 2 vers le même fichier

Pour envoyer la sortie standard et la sortie d'erreur standard dans le même fichier, il faut employer une syntaxe particulière. Voici ce qu'il ne faut pas écrire et la raison pour laquelle cela ne fonctionne pas.

Syntaxes incorrectes

```
$ commande 1> fichier 2> fichier
$ commande 1> fichier 2>> fichier
```

Le problème ne réside pas dans le fait que l'on ouvre deux fois le même fichier (ce qui est parfaitement légal au sein d'un même processus), mais qu'il y a un offset (position courante dans le fichier) associé à chaque ouverture. Quelles sont les conséquences ?

- La séquence des résultats dans le fichier ne sera pas forcément représentative de l'ordre dans lequel se sont déroulés les évènements.
- Les résultats émis à travers les descripteurs 1 et 2 risquent de se chevaucher.

Les figures 9 et 10 présentent le mécanisme interne associé à la commande suivante :

```
$ find / -name passwd 1> resu 2> resu
```

Première étape (cf. Figure 9)

Traitement de la redirection 1> resu :

Le shell ouvre le fichier **resu** (le fichier est créé avec une taille égale à 0) et l'associe au descripteur 1 **(1, 2, 3, 4, 5)**. Lorsqu'un processus ouvre un fichier, il y a toujours un enregistrement alloué dans la table des fichiers ouverts du noyau **(2)** (celle-ci regroupe toutes les ouvertures de fichier du système à un instant donné). L'enregistrement contient le mode d'ouverture du fichier (ici écriture), ainsi que l'emplacement courant dans le fichier (ici, l'écriture sur le descripteur 1 commencera en début de fichier). Le poste 1 **(1)** de la table des descripteurs de fichiers du processus contient un pointeur (p2) vers l'enregistrement de la table des fichiers ouverts.

Traitement de la redirection 2> resu :

Le shell ouvre de nouveau le fichier **resu** en écriture **(6, 7, 3, 4, 5)** (le fichier est écrasé, ce qui n'est pas grave puisque la taille était à 0) et l'associe au descripteur 2. Un nouvel enregistrement est créé dans la table des fichiers ouverts (p3) **(7)**. Les opérations d'écriture ultérieures sur le descripteur 2 se feront à partir du début de fichier.

Dans le cas du **2>> resu**, le fichier aurait été ouvert en mode ajout : positionnement en fin de fichier (donc 0 octet). Les écritures ultérieures sur le descripteur 2 garantissent l'écritures en fin de fichier. Cette solution peut produire un résultat convenable dans certains cas, mais reste néanmoins incorrecte (2 offsets différents).

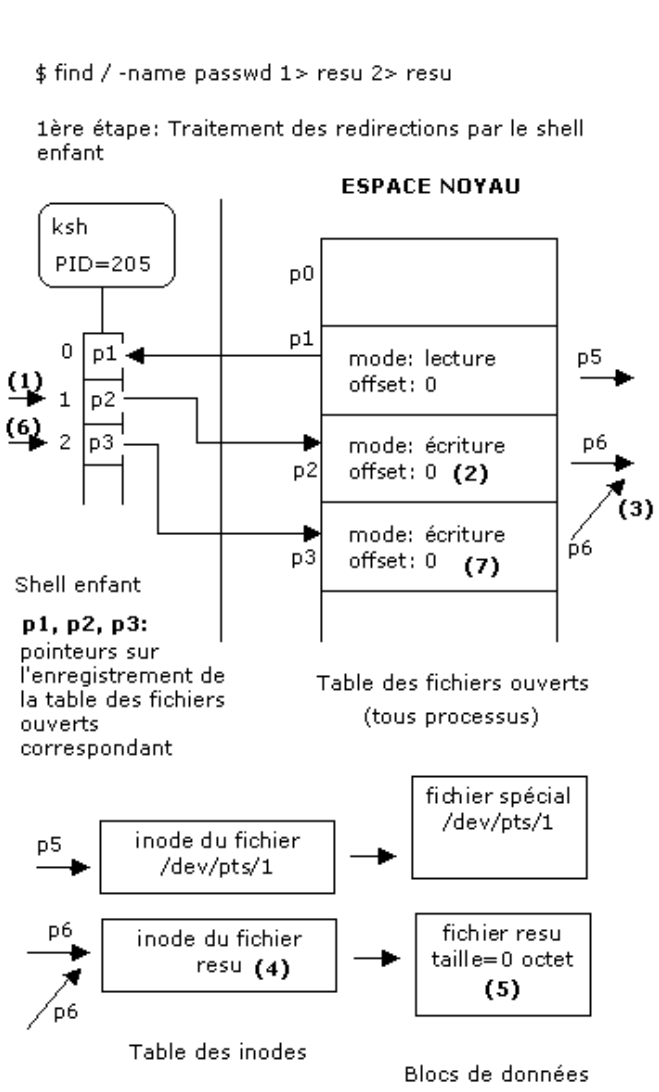

Figure 9 : Gestion des offsets dans les redirections - Première étape

Deuxième étape (cf. Figure 10)

Le shell se remplace par la commande **`find`** qui hérite de la table des descripteurs de fichiers. Simulation du fonctionnement de la commande **`find`** :

- find envoie sur le descripteur 1 un message résultat de 20 octets **(1, 2, 3...)** ;
- ces 20 octets sont écrits à partir de l'offset 0 **(2)**. Puis, l'offset prend la valeur 20 **(3)** qui sera utilisée lors de la prochaine écriture sur la sortie standard ;
- le fichier grossit de 20 octets **(4)** ;
- find envoie sur le descripteur 2 un message d'erreur de 10 octets **(5)** ;
- ceux-ci seront écrits en début de fichier (l'offset du descripteur 2 vaut 0 **(7)**), et vont donc écraser le message écrit précédemment **(8)**. L'offset de la sortie d'erreur passera ensuite à 10 **(7)**.

Conclusion : le fichier est inexploitable puisque les messages d'erreur et de résultat se chevauchent.

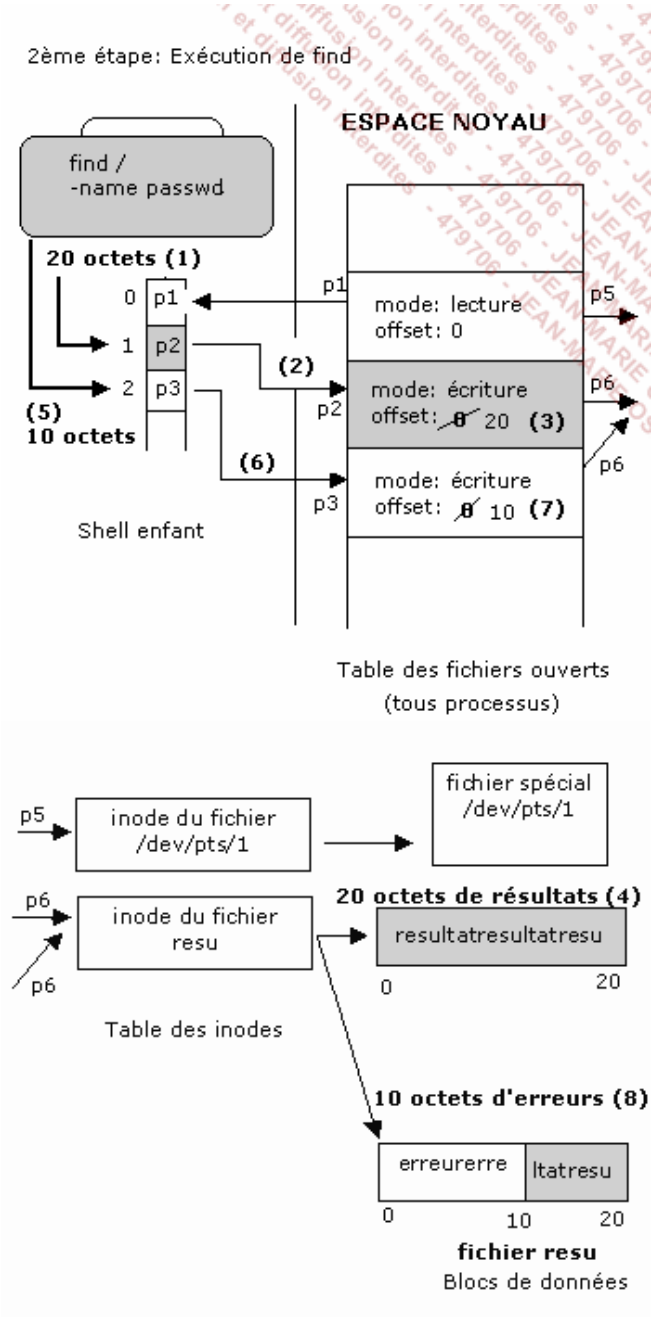

Figure 10 : Gestion des offsets dans les redirections - Deuxième étape

Syntaxes correctes

Pour obtenir un résultat correct, il faut utiliser l'une des deux syntaxes suivantes :

```
$ commande 1> fichier 2>&1
```

ou

```
$ commande 2> fichier 1>&2
```

La figure 11 représente le mécanisme interne engendré par la première syntaxe :

Traitement de la redirection 1> resu :

Même mécanisme que précédemment : création du fichier **resu**, allocation d'un enregistrement dans la table des fichiers ouverts, offset à 0 **(1)**.

Traitement de la redirection 2>&1 :

Pour traduire cette expression en français, on peut dire que le descripteur 2 est redirigé sur le descripteur 1

(représenté par &1). La chose importante à comprendre est que le shell, grâce à cette syntaxe, duplique **(2)** simplement l'adresse du descripteur 1 dans le descripteur 2. Les deux descripteurs pointent alors **(p2)** sur le même enregistrement de la table des fichiers ouverts, et par conséquent, partagent le même offset. Il n'y a pas d'allocation d'un nouvel enregistrement dans la table des fichiers ouverts du noyau. Les écritures ultérieures, qu'elles soient émises par la sortie ou la sortie d'erreur standard, se serviront du même offset.

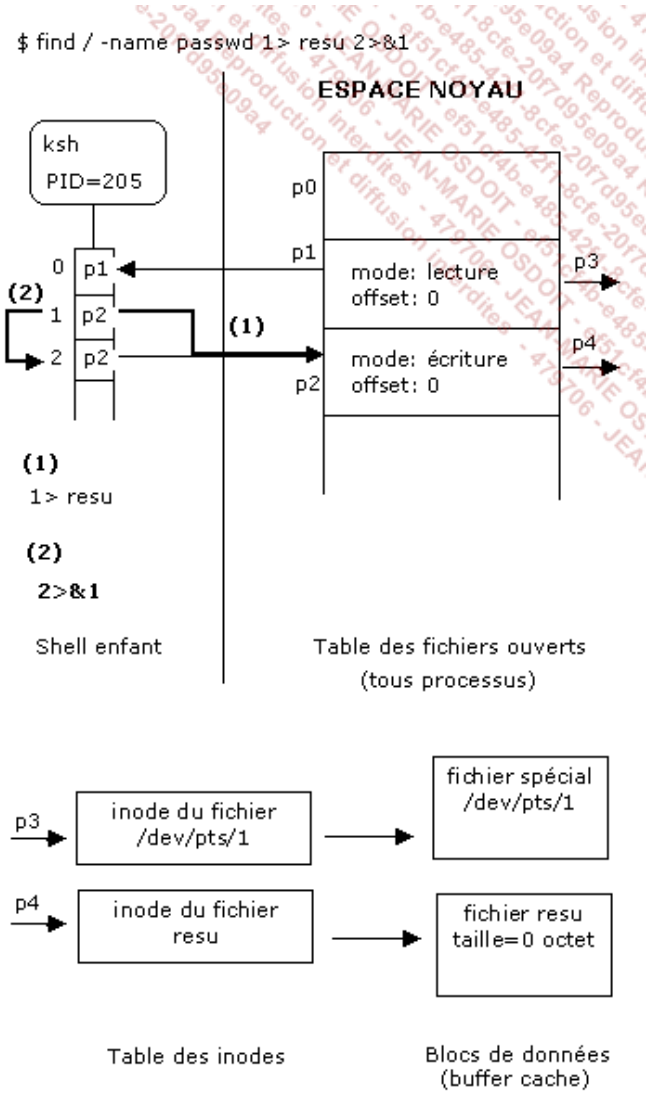

Figure 11 : Redirection de deux descripteurs dans le même fichier

La syntaxe **1> resu 2>&1** n'est pas équivalente à **2>&1 1>resu**. Dans le second cas, la sortie d'erreur standard est redirigée vers la sortie standard, c'est-à-dire vers le terminal. Puis la sortie standard est associée au fichier **resu**. Conclusion : les messages d'erreur sont dirigés vers le terminal et les messages résultat dans le fichier **resu**.

b. La double redirection en lecture

Elle est principalement utilisée dans les scripts shell. Elle permet de connecter l'entrée standard d'une commande sur une portion du script.

Première syntaxe

```
commande <<ETIQUETTE
données
données
données
ETIQUETTE
```

Deuxième syntaxe

```
commande <<-ETIQUETTE
données
données
données
  ETIQUETTE
```

Le symbole placé à la suite des caractères **<** est une déclaration d'étiquette. Cette dernière sera utilisée pour marquer la fin des données à lire. Les lignes insérées entre les deux mots **ETIQUETTE** seront envoyées sur l'entrée standard de la commande.

Exemple

```
$ mail olive <<FIN
> RV a 13 heures au restaurant
> Christine
> FIN
```

Quelques informations syntaxiques

- dans la première syntaxe, l'étiquette est obligatoirement collée en marge gauche ;
- dans la deuxième syntaxe, le fait de placer un caractère **-** devant l'étiquette autorise l'utilisateur à décaler cette dernière d'une ou plusieurs tabulations ;
- les étiquettes doivent être immédiatement suivies d'un retour à la ligne.

c. Fermeture d'un descripteur

Descripteur fermé	Syntaxe	Conséquence
0	commande <&-	L'entrée standard de la commande est fermée, cette dernière ne pourra donc pas s'en servir pour recevoir des données.
1	commande >&-	La sortie standard de la commande est fermée, cette dernière ne pourra donc pas s'en servir. Aucun message résultat ne pourra s'afficher à l'écran.
2	commande 2>&-	La sortie d'erreur standard de la commande est fermée, cette dernière ne pourra donc pas s'en servir. Aucun message d'erreur ne pourra s'afficher à l'écran.