Initiation à l'environnement Unix
CM6 : Processus (2) environnement, fichiers ouverts, redirections

Pierre Rousselin

Université Sorbonne Paris Nord
L1 informatique et DL
novembre 2024

Environnement d'un processus

Processus et fichiers ouverts

Redirection des canaux standards

Ce qui se passe lors d'un *fork*

Sans entrer dans les détails (suite en L2 informatique), lors d'un *fork*, c'est-à-dire lorsqu'un processus crée un processus enfant :

le processus enfant hérite de (presque) tous les attributs du parent.

En particulier

- ils ont le même utilisateur propriétaire ;
- ils ont le même répertoire courant ;
- ils ont le même terminal de contrôle ;
- ...

Ce qui se passe lors d'un *fork*

Sans entrer dans les détails (suite en L2 informatique), lors d'un *fork*, c'est-à-dire lorsqu'un processus crée un processus enfant :

le processus enfant hérite de (presque) tous les attributs du parent.

En particulier

- ils ont le même utilisateur propriétaire ;
- ils ont le même répertoire courant ;
- ils ont le même terminal de contrôle ;
- ...
- ils ont le même environnement.

Le processus enfant peut toujours par la suite modifier certains de ces attributs...

Ce qui se passe lors d'un *fork*

Sans entrer dans les détails (suite en L2 informatique), lors d'un *fork*, c'est-à-dire lorsqu'un processus crée un processus enfant :

le processus enfant hérite de (presque) tous les attributs du parent.

En particulier

- ils ont le même utilisateur propriétaire ;
- ils ont le même répertoire courant ;
- ils ont le même terminal de contrôle ;
- ...
- ils ont le même environnement.

Le processus enfant peut toujours par la suite modifier certains de ces attributs... mais jamais ceux de son parent.

L'environnement

Mais qu'est-ce que l'environnement ?

Définition

L'environnement d'un processus est un tableau de chaînes de caractères du type `KEY=value`.

En shell, on peut visualiser l'environnement avec la commande standard `env`.

```
$ env
SHELL=/bin/bash
HISTSIZE=1000
EDITOR=vim
PWD=/home/alice/images
LANG=fr_FR.UTF-8
HOME=/home/alice
USER=alice
PATH=/home/alice/bin:/usr/bin:/bin
...
```

L'environnement (2)

L'environnement apporte du contexte qui peut être pris en compte par certaines commandes. Par exemple :

`LANG` contient les informations de localisation (langue, encodage des caractères)

`PATH` contient les chemins des répertoires où chercher les commandes externes

`EDITOR` contient le nom de l'éditeur par défaut de l'utilisateur

`PWD` contient le chemin absolu vers le répertoire courant

`HOME` contient le chemin absolu vers le répertoire courant de l'utilisateur

Lire ou modifier l'environnement

En shell :

- Pour une variable qui est déjà dans l'environnement : comme une variable habituelle :

```
$ echo $EDITOR
vim
$ EDITOR=emacs
$ echo $EDITOR
emacs
```

  Les commandes qui utilisent `EDITOR` (par exemple `fc`) ouvriront alors `emacs`.

Lire ou modifier l'environnement

En shell :

- Pour une variable qui est déjà dans l'environnement : comme une variable habituelle :

```
$ echo $EDITOR
vim
$ EDITOR=emacs
$ echo $EDITOR
emacs
```

  Les commandes qui utilisent `EDITOR` (par exemple `fc`) ouvriront alors `emacs`.

- Ajouter une variable à l'environnement :

```
$ export CC
$ CC=gcc # on modifie sa valeur comme d'habitude
```

  ou directement :

```
$ export CC=gcc
```

Lire ou modifier l'environnement

En shell :

- Pour une variable qui est déjà dans l'environnement : comme une variable habituelle :

```
$ echo $EDITOR
vim
$ EDITOR=emacs
$ echo $EDITOR
emacs
```

  Les commandes qui utilisent `EDITOR` (par exemple `fc`) ouvriront alors `emacs`.

- Ajouter une variable à l'environnement :

```
$ export CC
$ CC=gcc # on modifie sa valeur comme d'habitude
```

  ou directement :

```
$ export CC=gcc
```

- Retirer une variable de l'environnement pour une commande :

```
$ env -u CC commande
```

- Supprimer une variable (de l'environnement ou non) :

```
$ unset CC
```

Mais quelle différence avec les variables shell ?

Les variables qui ne sont pas dans l'environnement ne sont pas héritées par les processus enfants.

```
$ a=hop b=plop
$ export b
$ bash
$ # on est dans un shell enfant
$ printf 'a=%s\nb=%s\n' "$a" "$b"
a=
b=plop
```

Difficiles relations parent-enfant

Principe général : une fois né, le processus enfant est totalement indépendant du parent.

En particulier, l'enfant peut modifier son propre environnement mais jamais celui de son parent.

Difficiles relations parent-enfant

Principe général : une fois né, le processus enfant est totalement indépendant du parent.

En particulier, l'enfant peut modifier son propre environnement mais jamais celui de son parent.

```
$ echo $EDITOR
vim
$ bash
$ # on est dans un shell enfant
$ echo $EDITOR
vim
$ EDITOR=emacs
$ echo $EDITOR
emacs
$ exit
$ # retour au parent
$ echo $EDITOR
vim
```

Environnement d'un processus

Processus et fichiers ouverts

Redirection des canaux standards

Ouvrir, fermer, lire, écrire

Les 4 intéractions fondamentales entre processus sous Unix sont les suivantes :

`open` le processus demande au système l'ouverture d'un fichier dans un certain mode [1] :

- lecture (si permis)
- écriture (si permis) avec troncation ou création du fichier
- écriture (si permis) à la suite dans le fichier

`read` le processus lit des octets dans le fichier, le décalage dans le fichier *(file offset)* est mis à jour. Analogie : marque-page dans un fichier

`write` le processus écrit des octets dans le fichier, le décalage dans le fichier est mis à jour.

`close` le processus ferme le fichier.

1. on simplifie ici

Processus et fichiers

- Chaque processus doit donc garder trace de chaque fichier ouvert et pour chacun d'eux, savoir où il en est dans le fichier (décalage dans le fichier).
- Ceci est maintenu pour chaque processus dans un *tableau des fichiers ouverts*.
- L'indice du fichier ouvert dans ce tableau s'appelle *le descripteur de ce fichier* (*file descriptor*).

Canaux standards

Chaque processus a au départ 3 fichiers ouverts :

entrée standard de descripteur 0, par défaut associée au terminal de contrôle du processus

sortie standard de descripteur 1, par défaut asociée au terminal de contrôle du processus

erreur standard de descripteur 2, par défaut asociée au terminal de contrôle du processus

Environnement d'un processus

Processus et fichiers ouverts

Redirection des canaux standards

Redirection de la sortie standard

- Le caractère spécial `>` (chevron fermant, « supérieur à ») permet de *rediriger* la sortie standard vers un (autre) fichier.

Redirection de la sortie standard

- Le caractère spécial `>` (chevron fermant, « supérieur à ») permet de *rediriger* la sortie standard vers un (autre) fichier.

```
$ ls /usr/ >contenu_usr
$ cat contenu_usr
bin games include lib lib64 libexec local sbin share src tmp
$ printf '%s\n' "salut $USER !" > salut.txt
$ cat salut.txt
salut Alice !
$ cat /etc/passwd > ~/password_backup #copie du fichier
$ echo "Ah les cro-cro-cro" 1>~/crocrodiles
```

Redirection de la sortie standard

- Syntaxe : `[1]>[blanc...]nom_fichier`

Redirection de la sortie standard

- Syntaxe : `[1]>[blanc...]nom_fichier`
- Sémantique :
  - Le mot `nom_fichier` subit les développements du tilde, de variable, arithmétique, la substitution de commande et la suppression des caractères inhibiteurs.
  - Si le fichier correspondant au mot obtenu n'existe pas, il est créé (si permissions suffisantes) ;
  - mais s'il existe déjà, il est vidé (si permissions suffisantes) !
  - Le shell connecte la sortie standard de la (ou des commandes) concernées au fichier correspondant, avant de l'exécuter.

Redirection de la sortie standard

- Syntaxe : `[1]>[blanc...]nom_fichier`
- Sémantique :
  - Le mot `nom_fichier` subit les développements du tilde, de variable, arithmétique, la substitution de commande et la suppression des caractères inhibiteurs.
  - Si le fichier correspondant au mot obtenu n'existe pas, il est créé (si permissions suffisantes) ;
  - mais s'il existe déjà, il est vidé (si permissions suffisantes) !
  - Le shell connecte la sortie standard de la (ou des commandes) concernées au fichier correspondant, avant de l'exécuter.

Remarque : le chiffre 1 optionnel qui précède le chevron `>` est le *descripteur de fichier* (*file descriptor*) associé à la sortie standard (en anglais *standard output* et en abrégé `stdout`).

Redirection de la sortie standard « à la suite »

Les doubles chevrons fermants permettent de rediriger une sortie *à la fin d'un fichier*.

```
$ echo "Les petits poissons dans l'eau" >comptine
$ cat comptine
Les petits poissons dans l'eau
$ echo "Nagent aussi bien que les gros" >>comptine
$ cat comptine
Les petits poissons dans l'eau
Nagent aussi bien que les gros
```

Redirection de la sortie standard « à la suite »

Si le fichier n'existe pas, il est créé (et il n'y a alors pas de différence avec la redirection >).

```
$ ls
comptine  shell_cm2.tex  shell_cm2.pdf
$ ls >> liste_fichiers; cat liste_fichiers
comptine  liste_fichiers  shell_cm2.tex  shell_cm2.pdf
```

Redirection de la sortie standard « à la suite »

Si le fichier n'existe pas, il est créé (et il n'y a alors pas de différence avec la redirection `>`).

```
$ ls
comptine  shell_cm2.tex  shell_cm2.pdf
$ ls >> liste_fichiers; cat liste_fichiers
comptine  liste_fichiers  shell_cm2.tex  shell_cm2.pdf
```

Question : pourquoi la chaîne `liste_fichiers` est présente dans le fichier `liste_fichiers` ?

Redirection de la sortie standard « à la suite »

Si le fichier n'existe pas, il est créé (et il n'y a alors pas de différence avec la redirection `>`).

```
$ ls
comptine  shell_cm2.tex  shell_cm2.pdf
$ ls >> liste_fichiers; cat liste_fichiers
comptine  liste_fichiers  shell_cm2.tex  shell_cm2.pdf
```

Question : pourquoi la chaîne `liste_fichiers` est présente dans le fichier `liste_fichiers` ?
Réponse (et rappel) : le fichier `liste_fichiers` est créé avant que la commande soit exécutée, donc la commande `ls` le liste (même si à ce moment-là, il est vide).

Redirection de la sortie standard « à la suite »

- Syntaxe : `[1]>>[blanc...]nom_fichier`
- Mêmes remarques que pour la redirection `>`
- sauf qu'il n'y a pas de risque de vider le fichier s'il existe.
- Le fichier dans lequel on redirige est dit ouvert en mode *écriture à la suite* (*append*).

Redirection de la sortie standard « à la suite »

- Syntaxe : `[1]>>[blanc...]nom_fichier`
- Mêmes remarques que pour la redirection `>`
- sauf qu'il n'y a pas de risque de vider le fichier s'il existe.
- Le fichier dans lequel on redirige est dit ouvert en mode *écriture à la suite* (*append*).

Utilisation typique : écrire à la fin d'un fichier de configuration.

```
$ printf '%s\n' 'PATH=~/bin:$PATH' >>~/.bashrc
```

Parenthèse : la commande `tr`

La commande `tr` (pour *transliterate*) transpose ou supprime des caractères qui viennent de son *entrée standard* vers sa sortie standard. C'est une commande importante sur laquelle on reviendra plus en détail.

```
$ tr A-Z a-z #transforme les majuscules en minuscules
SALUT !
salut !
Alice et Bob.
alice et bob.
^D
$
```

Parenthèse : la commande `tr`

La commande `tr` (pour *transliterate*) transpose ou supprime des caractères qui viennent de son *entrée standard* vers sa sortie standard. C'est une commande importante sur laquelle on reviendra plus en détail.

```
$ tr A-Z a-z #transforme les majuscules en minuscules
SALUT !
salut !
Alice et Bob.
alice et bob.
^D
$
```

Remarque : ici les entrées de l'utilisateur sont `SALUT !` et `Alice et Bob.`, les autres lignes sont les sorties de `tr`. La combinaison de touches `Ctrl-d` signale une fin de fichier et donc la fin de la saisie de l'utilisateur.

Parenthèse : la commande `tr`

On peut évidemment rediriger la sortie de `tr` :

```
$ tr A-Z a-z >truc
SALUT, Alice et Bob !
^D
$ cat truc
salut, alice et bob !
```

Parenthèse : la commande `tr`

On peut évidemment rediriger la sortie de `tr` :

```
$ tr A-Z a-z >truc
SALUT, Alice et Bob !
^D
$ cat truc
salut, alice et bob !
```

Mais on peut aussi rediriger son entrée :

```
$ tr a-z A-Z <truc
SALUT, ALICE ET BOB !
```

Parenthèse : la commande `tr`

On peut évidemment rediriger la sortie de `tr` :

```
$ tr A-Z a-z >truc
SALUT, Alice et Bob !
^D
$ cat truc
salut, alice et bob !
```

Mais on peut aussi rediriger son entrée :

```
$ tr a-z A-Z <truc
SALUT, ALICE ET BOB !
```

Voire les deux à la fois :

```
$ tr sab SAB <truc >bidule
$ cat bidule
SAlut, Alice et BoB !
```

Redirection de l'entrée standard

- Certaines commandes *lisent* des informations sur le terminal (`tr`, `read`, ...).
- Beaucoup d'autres lisent sur le terminal à condition qu'aucun nom de fichier ne leur ait été donné en argument (`cat`, `sort`, `head`, `tail`, `cut`, `grep`, `sed`, `awk`, `iconv`, ...)
- On peut *rediriger l'entrée standard* à l'aide du *chevron ouvrant* `<` (« inférieur à ») suivi d'un nom de fichier.
- Syntaxe : `[0]<[blanc...]nom_fichier`
- Sémantique : même chose que pour la redirection de l'entrée standard sauf que `nom_fichier` doit correspondre à un fichier qui existe et que l'on a la permission de lire (`r`).
- Le chiffre optionnel 0 est le *descripteur de fichier* associé à l'entrée standard (en anglais `standard input`, en abrégé, `stdin`).

Redirection de l'entrée standard

```
$ cat >machin
Petit
Escargot
Porte
Sur
Son
Dos
^D
$ cat <machin
Petit
Escargot
Porte
Sur
Son
Dos
```

Redirection de l'entrée standard

```
$ sort <machin # pareil que 'sort machin'
Dos
Escargot
Petit
Porte
Son
Sur
$ tr '\n' ' ' 0< machin #change les fins de lignes en espaces
Petit Escargot Porte Sur Son Dos
$ cut -c 1,3 <machin # caractères 1 et 3 de chaque ligne
Pt
Ec
Pr
Sr
Sn
Ds
```

Redirection de l'entrée et de la sortie

Attention au piège suivant : je veux trier le fichier `machin`.

```
$ sort <machin >machin
$ cat machin
$ # Quoi ?? rien ??
```

Redirection de l'entrée et de la sortie

Attention au piège suivant : je veux trier le fichier `machin`.

```
$ sort <machin >machin
$ cat machin
$ # Quoi ?? rien ??
```

À cause de la redirection `>machin`, le fichier `machin` est vidé *avant* que la commande ne soit exécutée !
Ici, la commande `sort` trie un fichier vide (pas trop dur...)

Redirection de l'entrée et de la sortie

Attention au piège suivant : je veux trier le fichier `machin`.

```
$ sort <machin >machin
$ cat machin
$ # Quoi ?? rien ??
```

À cause de la redirection `>machin`, le fichier `machin` est vidé *avant* que la commande ne soit exécutée !
Ici, la commande `sort` trie un fichier vide (pas trop dur...)

```
$ sort <machin >tmp_machin; mv tmp_machin machin
$ cat machin
Dos
Escargot
Petit
Porte
Son
Sur
```

L'erreur standard

Exemple étonnant de redirection de la sortie standard :

```
$ ls -l comptine chanson /boot/efi
ls: impossible d'accéder à 'chanson':
    Aucun fichier ou dossier de ce type
-rw-rw-r--. 1 pierre pierre   63 27 avril 22:15 comptine
ls: impossible d'ouvrir le répertoire '/boot/efi':
    Permission non accordée
$ ls -l comptine chanson /boot/efi > ls_out
ls: impossible d'accéder à 'chanson':
    Aucun fichier ou dossier de ce type
ls: impossible d'ouvrir le répertoire '/boot/efi':
    Permission non accordée
$ cat ls_out
-rw-rw-r--. 1 pierre pierre   63 27 avril 22:15 comptine
```

L'erreur standard

- Les messages d'erreurs apparaissent bien sur le terminal...
- mais ne sont pas écrits sur la sortie standard.
- Ils sont écrits sur *l'erreur standard* appelée aussi *sortie standard des erreurs* (*standard error*, `stderr`).
- L'erreur standard est, comme les deux autres canaux standards, associée au départ au terminal.
- Son descripteur de fichier est l'entier 2.
- On peut la rediriger !

Redirection de l'erreur standard

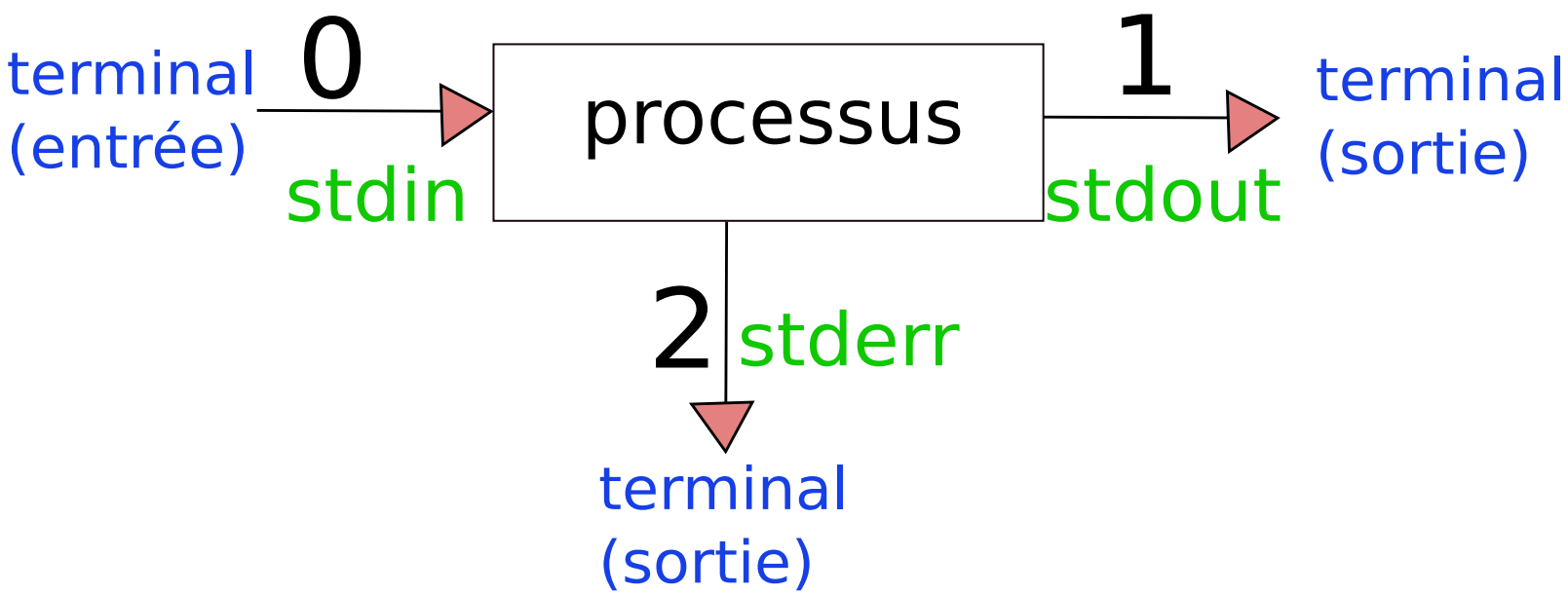
```
$ ls -l comptine chanson /boot/efi >ls_out 2>ls_err
$ cat ls_err
ls: impossible d'accéder à 'chanson':
    Aucun fichier ou dossier de ce type
ls: impossible d'ouvrir le répertoire '/boot/efi':
    Permission non accordée
$ ls /ust/
ls: impossible d'accéder à 'ust':
    Aucun fichier ou dossier de ce type
$ ls /ust/ 2>>ls_err # append
$ cat ls_err
ls: impossible d'accéder à 'chanson':
    Aucun fichier ou dossier de ce type
ls: impossible d'ouvrir le répertoire '/boot/efi':
    Permission non accordée
ls: impossible d'accéder à 'ust':
    Aucun fichier ou dossier de ce type
```

Redirection de l'erreur standard

- Syntaxes :
  `2>[blanc...]nom_fichier`
  `2>>[blanc...]nom_fichier` (*append*).
- Mêmes remarques que pour `>` et `>>` :
- sauf que le descripteur de fichier 2 est à préciser (sinon, par défaut ces redirections s'appliquent à la sortie standard).

Redirections : premier bilan

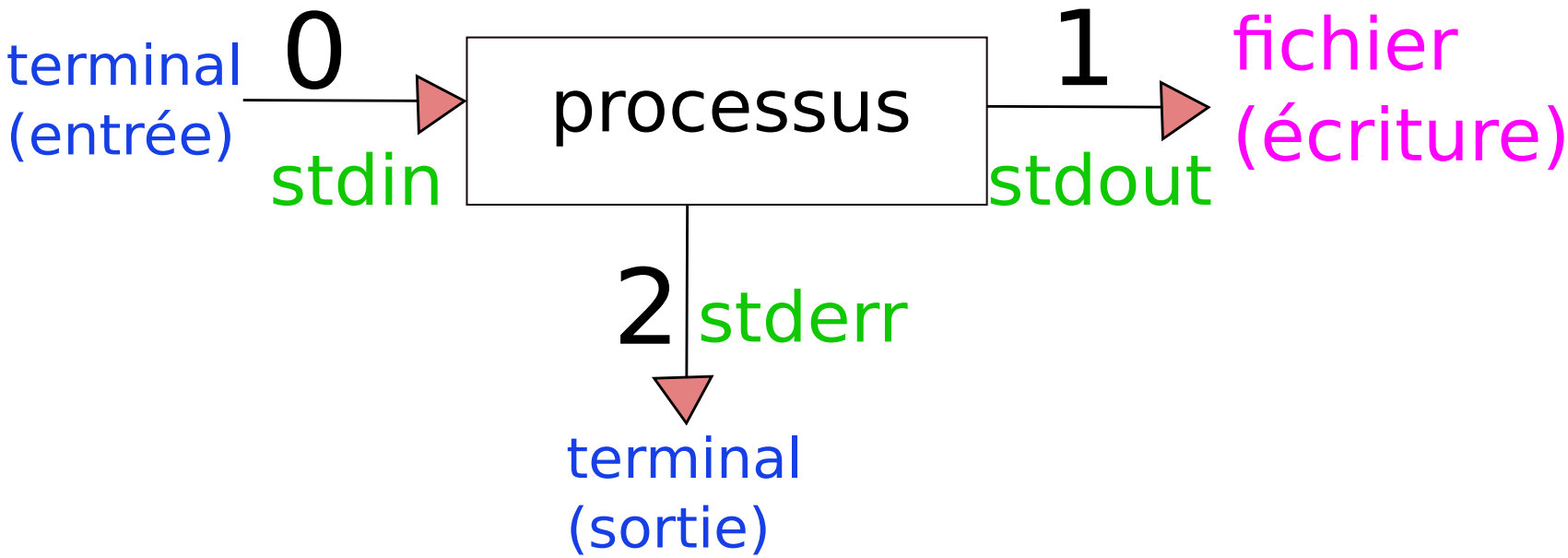
Sans aucune redirection :



Redirections : premier bilan

Redirection de la sortie standard avec au choix

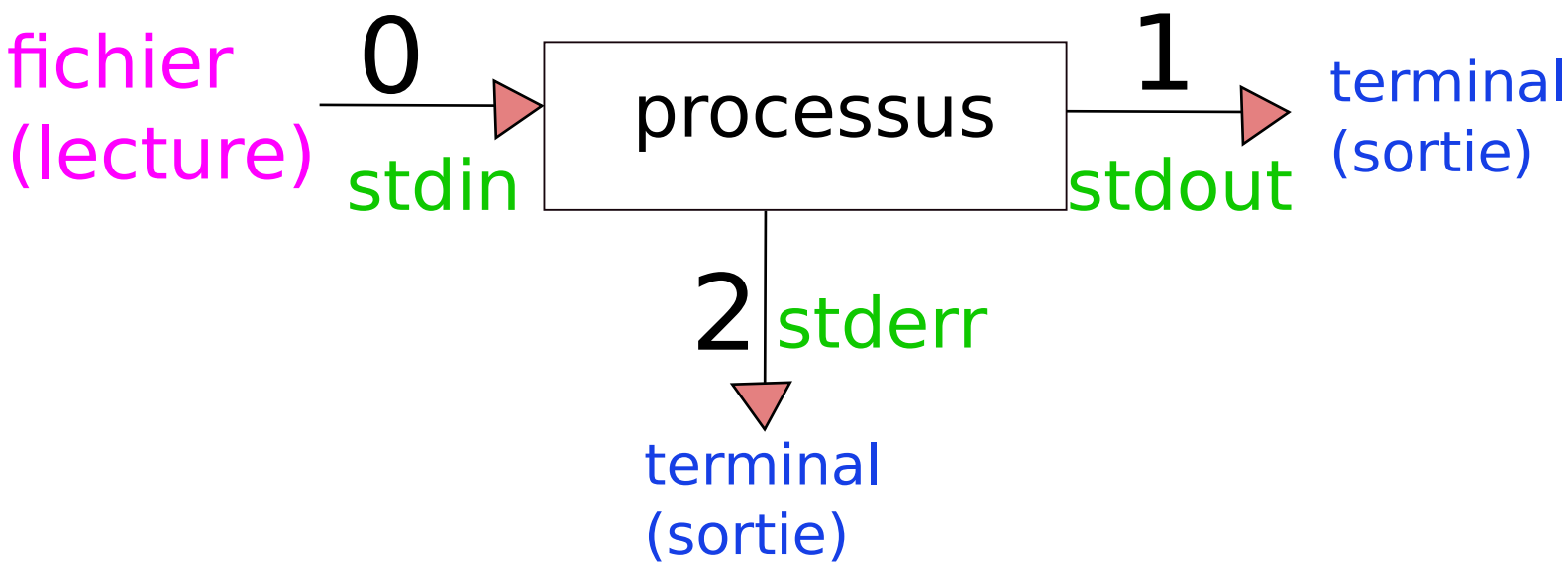
- `>fichier` ou `1>fichier`
- `>>fichier` ou `1>>fichier` (*append*)



Redirections : premier bilan

Redirection de l'entrée standard avec

- `<fichier` ou `0<fichier`



Redirections : premier bilan

Redirection de l'erreur standard avec au choix

- `2>fichier`
- `2>>fichier` (*append*)