R1.04 – TP 3

Table des matières

1 Processus

Un processus, du point du vu du noyau est un objet système qui regroupe l'ensemble des données relatives à un programme en cours d'exécution. La création d'un processus se fait par le lancement d'une commande externe (une commande interne est exécutée dans le processus shell courant). La commande est cherchée dans les répertoires indiqués par `PATH`. Si la commande est trouvée bash se clone, puis remplace le code à exécuter par celui de la commande. Le programme s'exécute alors et cette exécution est appelée **processus**. Le processus nouvellement créé est qualifié de **processus fils**, le shell à l'origine du fils est qualifié de **processus père**. Le processus fils se termine quand les actions à exécuter sont terminées. Bash reprend alors la main en acquittant la terminaison de son fils. En particulier, il récupère le statut de terminaison de son fils. Pendant son exécution, le contrôle d'un processus peut s'effectuer par des envois de signaux.

Question.

— Consultez le man de `ps`

— Utilisez `ps` sans argument

— À quoi correspondent les champs `PID`, `TTY`, `TIME` et `CMD` ?

— Utilisez `ps -l`

— À quoi correspondent les champs `UID`, `PPID` et `S` ?

— Affichez tous VOS processus en cours.

— Quelle est la différence entre `ps -l` et `ps l` ?

— Lancez `gnome-calculator &` depuis un terminal. Utilisez `ps` depuis le même terminal, puis depuis un autre, pour identifier le `PID` de ce processus.

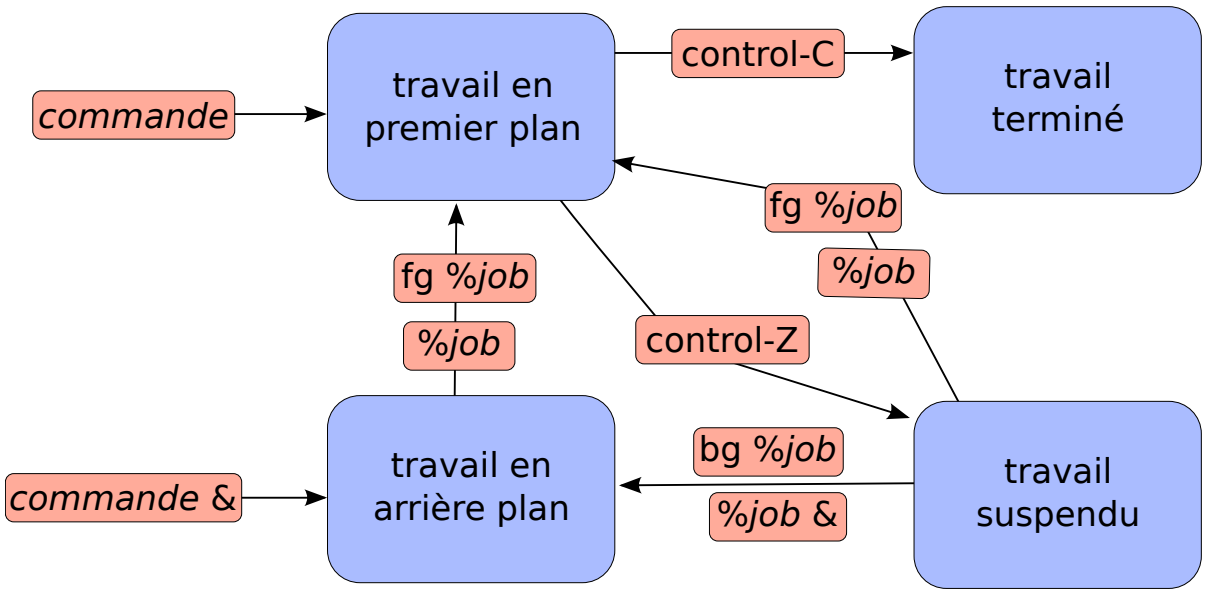
— Quel est le `PID` de `kthreadd` ?

— Faites afficher vos processus (associés au terminal courant) avec les champs `USER`, `S`, `%CPU`, `PID`, `PPID` et `CMD` (dans cet ordre).

2 Jobs

Question.

— Lancez `gnome-calculator` depuis un terminal. Vous n'avez



adapté de S. Krakowiak

job à remplacer par le numéro de la tâche

Figure 1 – États d'un processus

plus la main dans le terminal. La tâche s'exécute au premier plan. Terminer son exécution avec ctrl + c.

— Lancez `gnome-calculator &`. Vous conservez la main dans le terminal. La tâche s'exécute en arrière plan. Au moment du lancement, vous avez obtenu une ligne :

```
[1] 2997
```

qui vous indique le `PID` du processus ainsi que le numéro de tâche (entre crochets).

— Remettez `gnome-calculator` en avant plan.

— Interrompez l'exécution de `gnome-calculator` avec ctrl +

Ctrl + Z ; le processus est suspendu.

— Relancez par `bg` l'exécution de `gnome-calculator` en arrière plan.

— Lancez `eog &`.

— Remettez `gnome-calculator` en avant plan (`fg`).

3 Valeur de retour

Un processus (fils) se termine en restituant une valeur de retour. Cette valeur de retour permet à son père de connaitre les causes de sa terminaison. Par convention, une valeur 0 signifie une exécution et une terminaison sans problèmes. Une autre valeur signifie généralement une erreur. Cette valeur est consultable par la variable `$?`.

Question.

— Consultez le manuel de la commande `which`. Que fait cette commande ? Quelles sont les valeurs de retour possibles ?

— Utilisez cette commande pour avoir chacune de ces valeurs de retour. À chaque fois, faire afficher le code de retour.

— Pour une valeur de retour différente de 0, faites la afficher deux fois successivement. Le résultat vous surprend-il ? Pouviez-vous prévoir ce résultat ?

4 Signaux

Un signal est un événement asynchrone destiné à un processus. Les signaux sont l'équivalent pour les processus des interruptions matérielles pour un processeur. Certains signaux retranscrivent d'ailleurs au niveau du processus la survenue d'une interruption matérielle. Les signaux sont différenciés (à l'instar des interruptions) par un numéro (et un nom) de signal. Par exemple le signal de terminaison : 15 ou `SIGTERM`. À chaque signal est associé une action (traiteur ou *handler* de signal). Pour chaque signal, il existe une action par défaut, qui peut être d'ignorer le signal, de terminer le processus, ... Pour la plupart des signaux, il est possible de redéfinir cette action par défaut (dont `SIGUSR1` et `SIGUSR2` qui servent uniquement à ça). La commande shell permettant d'envoyer un signal à un processus est la commande `kill`. Le processus qui

reçoit le signal est usuellement désigné par son pid, mais `kill` accepte aussi la syntaxe `%jobs`.

Question.

- Consultez le manuel de la commande `kill`
- Lancez `gnome-calculator`. Utiliser `kill` sans arguments sur cette tâche. Que s'est-il passé ? Quel signal a-t-il été envoyé ?
- Quels sont les signaux disponibles ? À quels numéros correspondent `SIGKILL`, `SIGTERM`, `SIGSTOP` et `SIGCONT` ?
- À quoi servent les quatre signaux qui précèdent ? Les tester sur `gnome-calculator`.
- Refaites les mêmes actions qu'à l'exercice 2 à l'aide de la commande `kill`

5 Entrée sorties

Un processus utilise une entrée et plusieurs sorties (normale et pour les erreurs) pour prendre des données et restituer des ré-

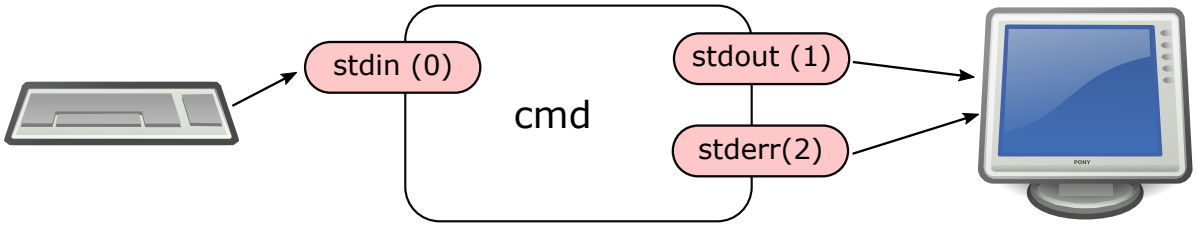


Figure 2 – Entrée sorties d'un processus

sultats. Ces entrées-sorties sont vues par le processus comme des fichiers, identifiés pat un numéro. Par défaut, l'entrée, qualifiée d'**entrée standard** (`stdin` *standard input*) de numéro 0, est associée au clavier, alors que les sorties, qualifiées respectivement de sortie standard (`stdout` *standard output*) de numéro 1 et d'erreur standard (`stderr` *standard error*) de numéro 2 sont associées à l'écran du terminal.

Question.

- Utilisez la commande `cat`, sans argument.
- Tapez au clavier (entrée standard).
- Où (sortie standard) est restitué le résultat de la commande ?
- Saisissez ctrl + d pour finir.

Il est possible de rediriger cette entrée et ces sorties depuis et vers des fichiers. Pour l'entrée cela signifie que les données sont lues

depuis un fichier. Pour les sorties cela veut dire que les résultats d'une commande sont envoyés dans un fichier.

redirection de la sortie

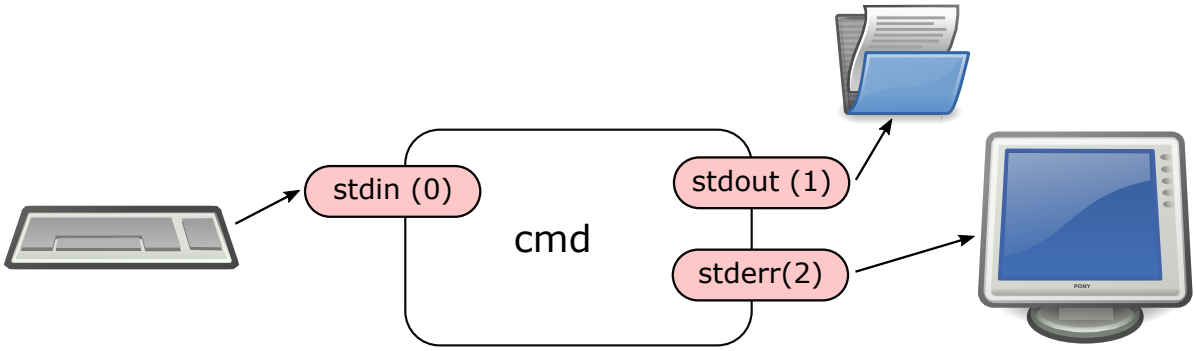


Figure 3 – `cmd > fichier`

Deux types de redirection existent :

1. `> fichier` (ou `1>`) la sortie de la commande est placée dans `fichier` (dont le contenu est écrasé préalablement s'il existait)
2. `>> fichier` la sortie de la commande est ajoutée à la fin de fichier (qui est créé si besoin)

Question.

— Créez dans le répertoire courant, en utilisant la commande `echo` et une redirection, un fichier `fichier1.txt` contenant :

```
 $ cat fichier1.txt
Bonjour
le
monde!
```

— Par redirection, fabriquez un fichier constitué d'une première ligne comportant des =, puis l'aide de la commande `help`, puis à nouveau une ligne de =. Résultat attendu :

```
$ cat resultat.txt
==========================
help : help [-dms] [motif ...]
    Affiche des informations sur les commandes intégrées.
...
==========================
```

redirection de la sortie erreur

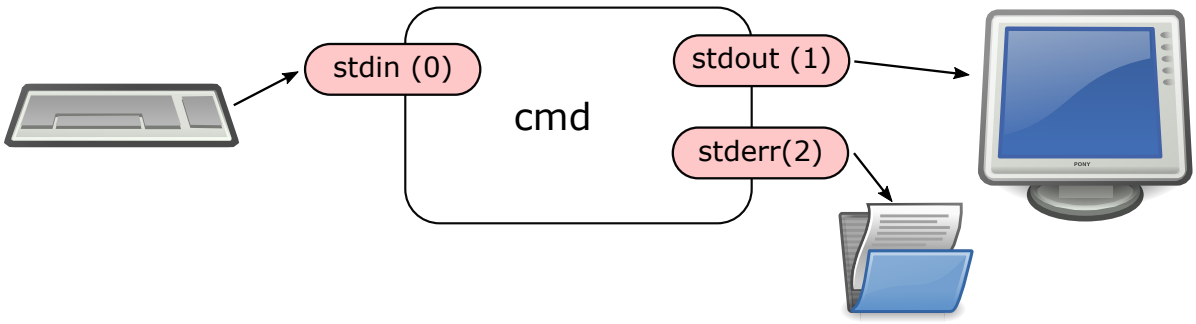


Figure 4 – `cmd 2> fichier`

Deux types de redirection existent :

1. `2> fichier` la sortie erreur de la commande est placée dans `fichier` (dont le contenu est écrasé préalablement s'il existait)
2. `2>> fichier` la sortie erreur de la commande est ajoutée à la fin de fichier (qui est créé si besoin)

Question.

— Vérifiez que le répertoire courant ne contient pas de fichier `fichier2.txt`.

— Testez la commande `ls -l fichier1.txt fichier2.txt` et constatez qu'elle produit une erreur. Comment expliquez-vous cette erreur ?

— Redirigez la sortie standard de la commande précédente sur un fichier `sortie.txt`.

— Redirigez ensuite l'erreur standard de la commande précédente sur un fichier `erreur.txt`.

Il est possible de rediriger la sortie et la sortie erreur simultanément. Si on souhaite que les redirections se fassent dans le même fichier, on utilise `>&` (il existe une autre syntaxe plus complexe).

Question.

— Redirigez à la fois la sortie standard et l'erreur standard de la commande précédente sur un fichier `tout.txt`.

— Comparez le contenu de ces trois fichiers.

— Concaténez (par une redirection, bien sûr) les fichiers `sortie.txt` et `erreur.txt` en un seul fichier `autreTout.txt`.

— Comparez la taille, le nombre de lignes et le contenu des fichiers `tout.txt` et `autreTout.txt`.

redirection de l'entrée

Certaines commandes lisent du texte entrée au clavier, on dit

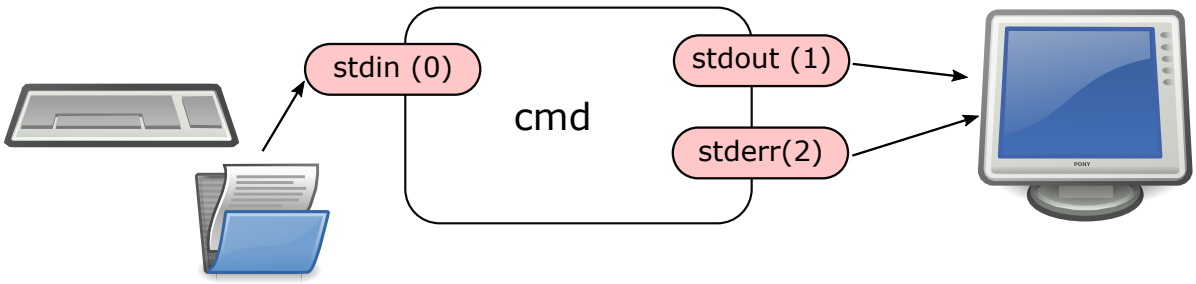


Figure 5 – `cmd < fichier`

qu'elles lisent l'entrée standard. C'est le cas, par exemple de la commande `wc` lorsqu'elle est utilisée sans argument.

Question.

— Que fait `wc` ?

— Utilisez la commande `wc -l` puis tapez quelques lignes au clavier, puis appuyez ctrl + d pour terminer la saisie.

La redirection de l'entrée standard vous sera probablement moins utile que les redirections des sorties (la plupart des commandes prennent en argument un fichier au lieu de l'entrée standard) mais pourra vous servir lorsque vous écrirez des programmes qui lisent l'entrée standard et que vous voudrez les tester.

Question.

— Utilisez `wc -l` pour déterminer le nombre de ligne d'un fichier en utilisant la redirection de l'entrée standard

— la commande interne `read` lit une ligne depuis l'entrée standard et affecte ce qui a été lu à une ou des variables. Utilisez `read` pour lire et affecter la variable `OK` depuis le fichier `un.txt`.

```
$ cat un.txt
1
$ read ...
$ echo $OK
1
```

6 Communication par tube

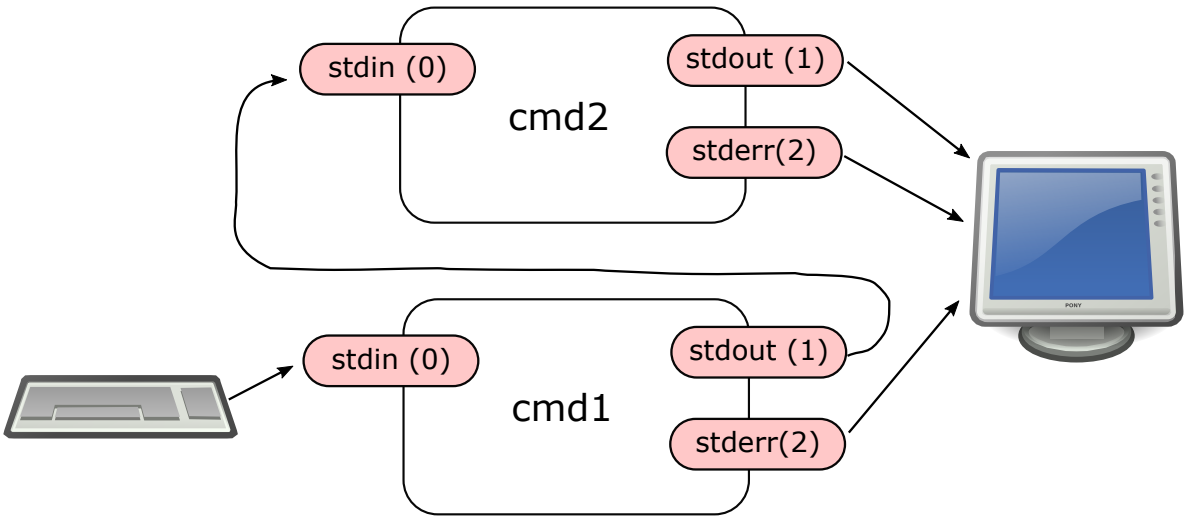


Figure 6 – `cmd1 | cmd2`

Il est possible de rediriger la sortie d'une commande à l'entrée d'une autre. On parle alors de communication par tube (*pipe* en anglais). Ceci est **très** utilisé pour composer les commandes pour obtenir un comportement plus élaboré. La syntaxe utilise le caractère `|`. Et, on peut étendre le principe : `cmd1 | cmd2 | cmd3 | cmd4` ....

Question.

- Comptez le nombre de dossier (non cachés) figurant directement dans votre répertoire personnel
- Reprenez le fichier `personnes.csv` du Tp1.
- Faites afficher les 20 premières personnes du fichier (première ligne non comprise)
- Consultez le man de la commande `sort`
- Faites afficher les personnes (première ligne non comprise), triées par nom, puis prénom.