



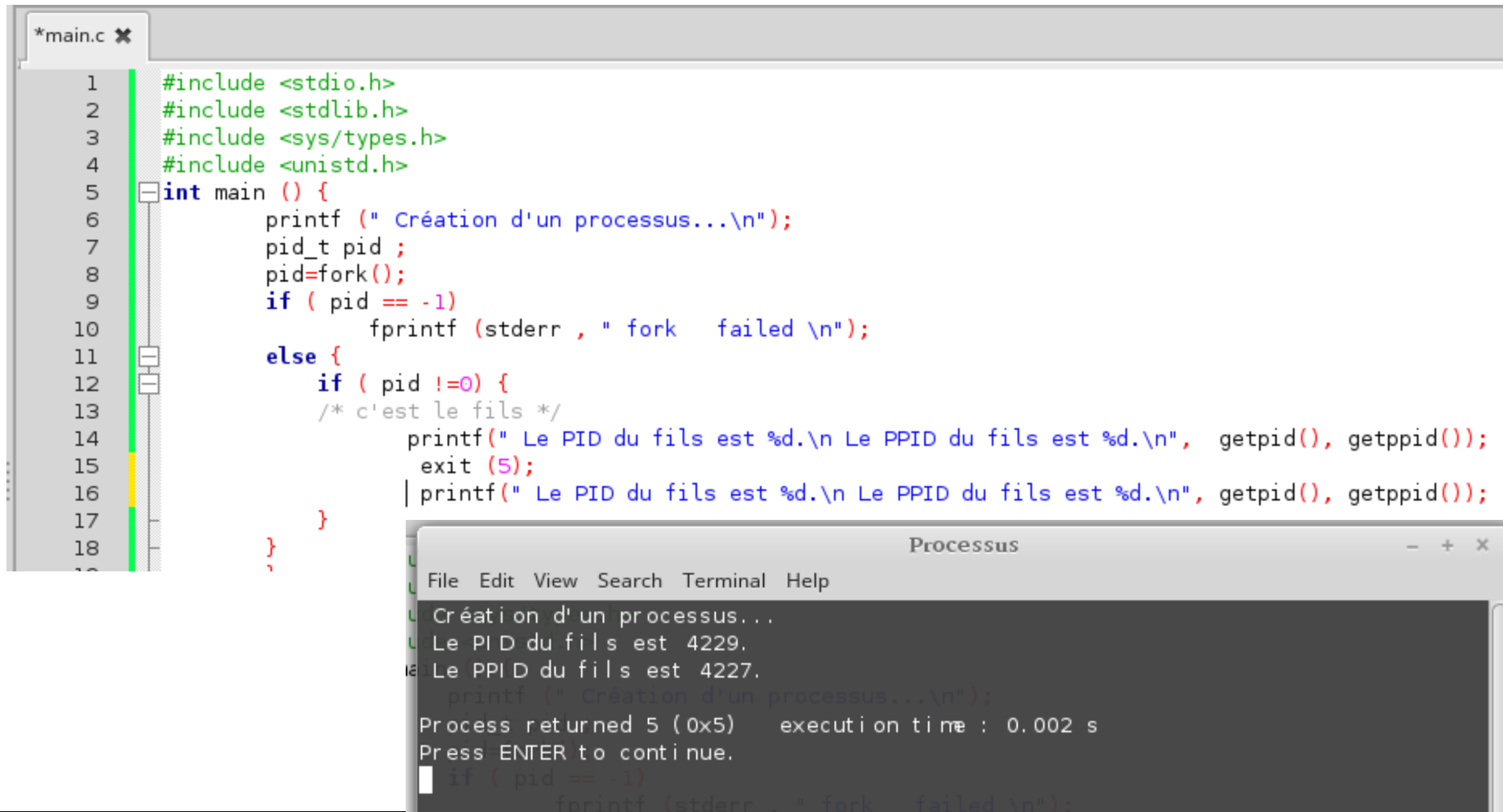
Cours  
Administration des Systèmes d'Exploitation  
Ecole Supérieure de Technologie – Guelmim  
Université Ibn Zohr –Agadir-

DUT-Informatique –S3

***Prof. ASIMI Younes***  
**asimi.younes@gmail.com**

**2020/2021**

## Création d'un Processus



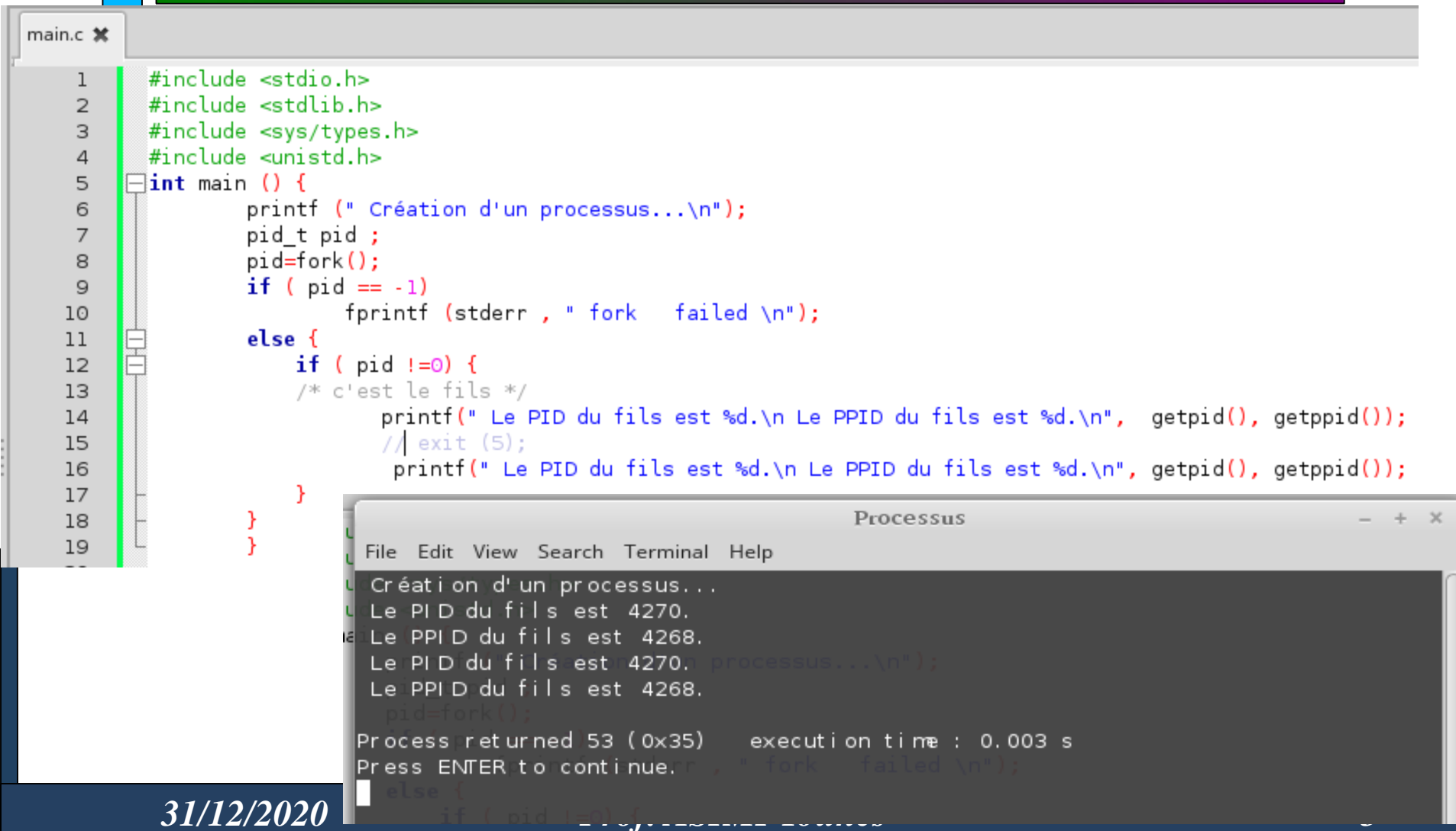
The image shows a C program in a text editor and its execution in a terminal window. The C program, named `main.c`, uses `fork()` to create a child process. It prints the parent's PID and the child's PID and PPID. The child process then prints its own PID and PPID and exits with a status of 5. The terminal window, titled "Processus", shows the output of the program, including the creation message, the PIDs, and the execution time.

```
*main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5  int main () {
6      printf (" Création d'un processus...\n");
7      pid_t pid ;
8      pid=fork();
9      if ( pid == -1)
10         fprintf (stderr , " fork   failed \n");
11     else {
12         if ( pid !=0) {
13             /* c'est le fils */
14             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
15             exit (5);
16             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
17         }
18     }
19 }
```

Processus

```
File Edit View Search Terminal Help
Création d'un processus...
Le PID du fils est 4229.
Le PPID du fils est 4227.
printf (" Création d'un processus...\n");
Process returned 5 (0x5)   execution time : 0.002 s
Press ENTER to continue.
if ( pid == -1)
    fprintf (stderr , " fork   failed \n");
```

## Création d'un Processus



The image shows a C program in a code editor and its execution output in a terminal window.

**Code Editor (main.c):**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5  int main () {
6      printf (" Création d'un processus...\n");
7      pid_t pid ;
8      pid=fork();
9      if ( pid == -1)
10         fprintf (stderr , " fork   failed \n");
11     else {
12         if ( pid !=0) {
13             /* c'est le fils */
14             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
15             // exit (5);
16             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
17         }
18     }
19 }
```

**Terminal Window (Processus):**

```
File Edit View Search Terminal Help
Création d'un processus...
Le PID du fils est 4270.
Le PPID du fils est 4268.
Le PID du fils est 4270. processus...\n");
Le PPID du fils est 4268.
pid=fork();
Process returned 53 (0x35)   execution time : 0.003 s
Press ENTER to continue
fork   failed \n");
else {
    if ( pid !=0) {
```

main.c [Processus] - Code::Blocks 13.12

File Edit View Search Project Build Debug Tools Plugins Settings Help

<global> main(): int

## Création d'un Processus

```

1  #include <stdlib.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4
5  int main () {
6      printf (" Création d'un processus...\n");
7      pid_t pid ;
8      pid=fork();
9      printf(" La valeur de pid %d\n", pid);
10     if ( pid == -1)
11         fprintf (stderr , " fork   failed \n");
12     else {
13         if ( pid ==0) {
14             /* c'est le fils */
15             printf("\n");
16             printf(" Nous sommes dans le Fils!\n");
17             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
18             // exit (5);
19             printf(" Le PID du fils est %d.\n Le PPID du fils est %d.\n", getpid(), getppid());
20         } else{
21             /* c'est le père */
22             printf("\n");
23             printf(" Nous sommes dans le Père!\n");
24             printf(" Le PID du père est %d.\n Le PPID du père est %d.\n", getpid(), getppid());
25             // exit (5);
26             printf(" Le PID du père est %d.\n Le PPID du père est %d.\n", getpid(), getppid());
27         }
28     }
29 }

```

Management

Projects Symbols

Workspace

Processus

Sources

main.c

Logs & others

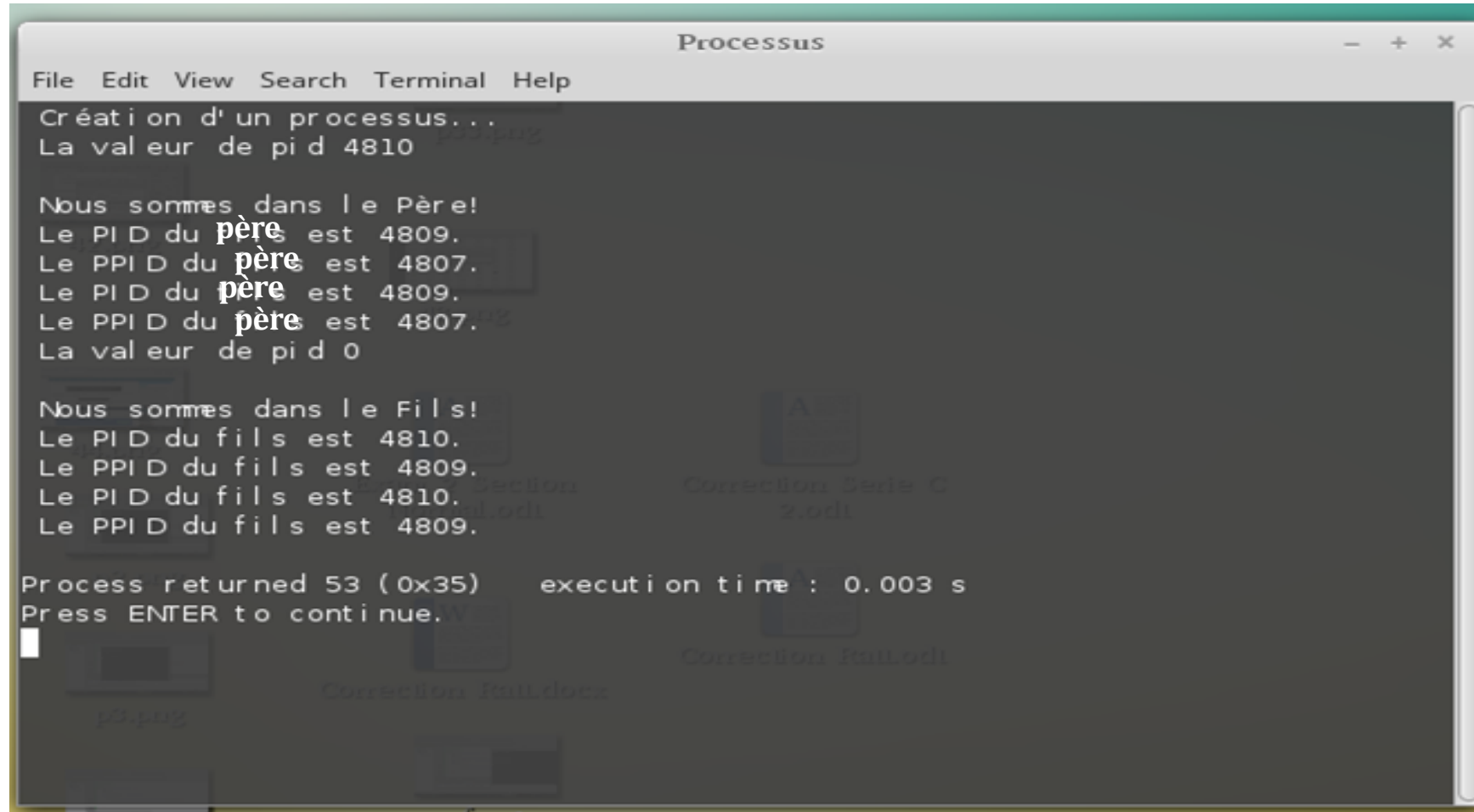
Code::Blocks Search results Build log Build messages Debugger

/home/asimi/Desktop/Processus/main.c

Unix (LF) UTF-8 Line 22, Column 27 Insert Read/Write default

Menu [Gestion des proces...] main.c [Processus] [Les processus - La... Untitled 1 - LibreOffi... Processus 16:45

## Processus



```
Processus
File Edit View Search Terminal Help
Création d'un processus...
La valeur de pid 4810

Nous sommes dans le Père!
Le PID du père est 4809.
Le PPID du père est 4807.
Le PID du père est 4809.
Le PPID du père est 4807.
La valeur de pid 0

Nous sommes dans le Fils!
Le PID du fils est 4810.
Le PPID du fils est 4809.
Le PID du fils est 4810.
Le PPID du fils est 4809.

Process returned 53 (0x35)   execution time : 0.003 s
Press ENTER to continue.
```

## Processus

### Résumé

- Un processus est un programme en cours d'exécution par un processeur;
- Un processus est forcément créé par un autre processus;
- Un processus est composé d'un code et des données : **variables d'environnement, tas et pile**;
- Le SE stocke les informations propres à un processus dans un block de contrôle: **Process Control Block**
- Le cycle de vie du processus : **prêt, en exécution, en attente**;
- Création de processus : **fork et exec ou createprocess**;
- Les **threads** partagent le **code, l'environnement et le tas**;
- Un **processus** peut créer **plusieurs threads**;

## Processus

### Les états d'un processus

Un processus peut avoir plusieurs états :

- **Exécution (R .. Running)** : le processus est en cours d'exécution ;
- **Endormi (S .. Sleeping)** : dans un multitâche coopératif, quand il rend la main ; ou dans un multitâche préemptif, quand il est interrompu au bout d'un quantum de temps ;
- **Arrêt (T .. sTopped)** : le processus a été temporairement arrêté par un signal. Il ne s'exécute plus et ne réagira qu'à un signal de redémarrage ;
- **Zombie (Z ..Zombie)** : le processus s'est terminé, mais son père n'a pas encore lu son code de retour.

## Commandes

La commande **ps** liste les processus d'un utilisateur. Selon les **Options** choisies, le résultat change:

- ✗ **l** affichage des informations détaillé;
- ✗ **u** informations sur les utilisateurs;
- ✗ **x** affiche les processus non attachés à un terminal;
- ✗ **a** affiche les processus des autres utilisateurs;
- ✗ **r** affiche seulement les processus en cours d'exécution.

```

Terminal
File Edit View Search Terminal Help

asi m@asi m -HP- 620 ~ $ ps axr
  PID TTY          STAT       TIME COMMAND
    4 ?           R           0:05 [kworker/0:0]
 3241 pts/1      R+          0:00 ps axr

asi m@asi m -HP- 620 ~ $ ps u
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
asi m        2270  0.0  0.1  22692  5444 pts/1    Ss   16:18   0:00 bash
asi m        3242  0.0  0.0  18464  2556 pts/1    R+   17:05   0:00 ps u

asi m@asi m -HP- 620 ~ $ ps l
 F  UID  PID  PPID PRI     VSZ    RSS WCHAN    STAT TTY          TIME COMMAND
 0 1000  2270  2262  20      0  22692  5444 wait     Ss   pts/1        0:00 bash
 0 1000  3243  2270  20      0  10036  2188 -        R+   pts/1        0:00 ps l

asi m@asi m -HP- 620 ~ $
  
```



## Processus

- Infos retournées par **ps**:

**temps CPU utilisé**

**numéro de processus**

**terminal associé**

**état du processus:**

**commande exécutée**

PID	TT	STAT	TIME	COMMAND
3899	p1	S	0:00.08	-zsh
4743	p1	S	0:00.14	emacs
4180	std	S	0:00.04	-zsh

R	actif
T	bloqué
P	en attente
S	endormi
Z	Zombie

## Principaux signaux

### Gestion d'un processus

- Un signal permet de prévenir un processus qu'un événement particulier va produire dans le système.
- Envoi de signal à un processus Syntaxe:
  - **kill -<S> pid**
    - Envoie le signal <s> au processus identifié par son pid.
    - <s> doit être remplacé par un nombre de 1 à 64 dont:
      - 1: **SIGHUP** ⇒ Fin du shell;
      - 3: **SIGQUIT** ⇒ Terminaison brutale;
      - 2: **SIGINT** ⇒ Interruption du programme;

## Priorité (nice et renice value)

- Elle définit l'importance du processus et donc le fait qu'il doit avoir un accès prioritaire au processeur.
- Chaque processus actif sur le système s'exécute avec une priorité donnée. Cette valeur, sous Linux, varie entre -20 à +19 :
  - **-20: priorité maximale;**
  - **De -19 à -1: priorité miximale;**
  - **De 0 à 19: priorité minimale;**
  - **19 la plus basse.**
- Un processus hérite du nice de son père par défaut (en général 0 car init est à 0).



## Priorité (nice et renice value)

- Lancer un programme avec une certaine priorité :
- `$ nice -n priorité commande`
- la priorité d'un processus déjà lancé :
- `$ renice -n priorité PID`
- La commande :
  - `renice -n <Priorité> {-p <PID> | -g <GID> | -u <User>}`
  - `nice -n +<priorite> <PID>`
  - `nice -n -<priorite> <PID>`
- Exemples:
  - `nice -n 12 asimi` ⇒ Définir une priorité de 12 pour le processus asimi;
  - `nice -n - 12 1134` ⇒ Donner une priorité négative;
  - `renice -n 17 -p 1134` ⇒ Redéfinir la priorité pour le processus 1134 .

## Processus

### Types d'ordonnancement

- Il existe deux types d'ordonnancement:
  - **Ordonnancement préemptif:** L'Ordonnanceur peut interrompre un processus en cours d'exécution si un nouveau processus de priorité plus élevée est inséré **dans la file des Prêts.**
  - **Ordonnancement coopératif :** Le processus élu garde le contrôle jusqu'à épuisement du temps qui lui a été alloué même si **des processus plus prioritaires** ont atteint **la liste des Prêts**

## Processus

### Algorithme d'ordonnancement

- Le but d'**ordonnanceur** est de **sélectionner le prochain processus** pour le CPU.
- ✧ Les **anciens algorithmes** étaient tels qu'une fois un processus sélectionné, **il s'exécutait jusqu'à la fin**.
  - ↪ **Le choix était de type: First-in-First-out (FIFO)**, ou d'abord les petits programmes (**SJF**) ou encore sur base de **priorité assigné à chacun des programmes**.
- ✧ Les **algorithmes plus récents** sont plus **flexibles**. Le **round robin** alloue à chaque processus un certain **quantum de temps** et **tous les processus tournent**.

## Processus

### Algorithme: First In, First Out

Cet algorithme reste le plus Simple à implémenter, également, tous les processus accèdent au processeur dans l'ordre d'arrivée à la file d'attente.

- Règle:

Premier arrive, premier servi;

Proc.	Date	Durée
P1	0	10
P2	1	2
P3	1	2

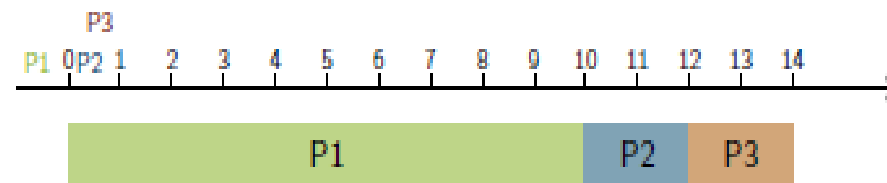


Diagramme de Gantt

## Processus

### Algorithme: Plus court d'abord

Cet algorithme favorise le processus court. Il réagit à base du temps restant estimé. La priorité ici se fonde sur:

- le temps restant;
- En cas d'égalité, on prend le processus actif afin d'évite la commutation;
- En cas d'égalité de temps restant et de date, on prend l'ordre de la file d'attente (**Premier arrive, premier servi**) ;

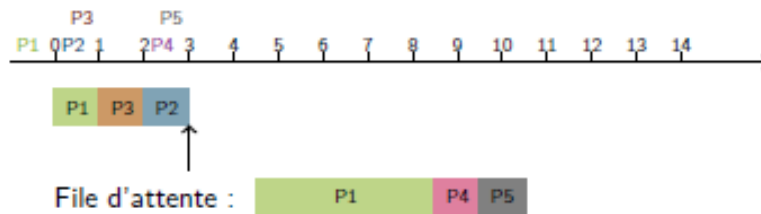
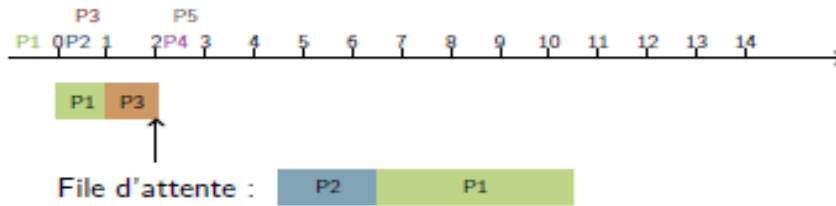
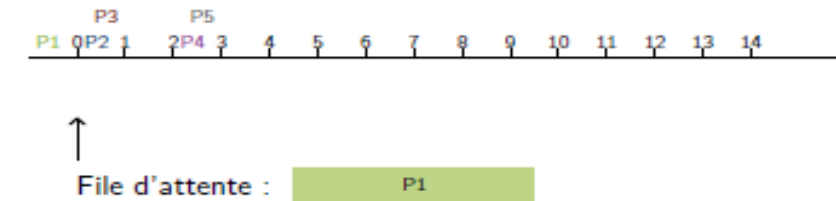
■ Règle:

**Plus court d'abord;**

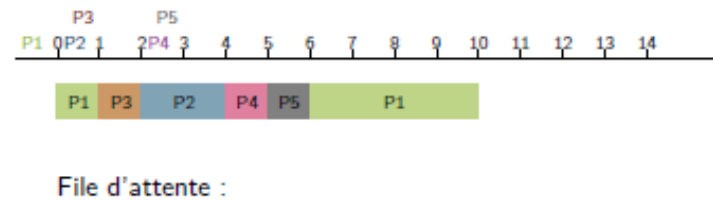
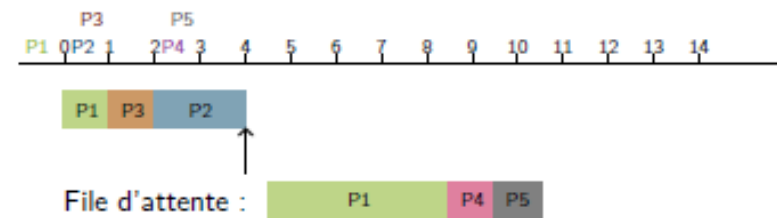


# Processus

## Algorithme: Plus court d'abord



Proc.	Date	Durée
P1	0	5
P2	1	2
P3	1	1
P4	3	1
P5	3	1



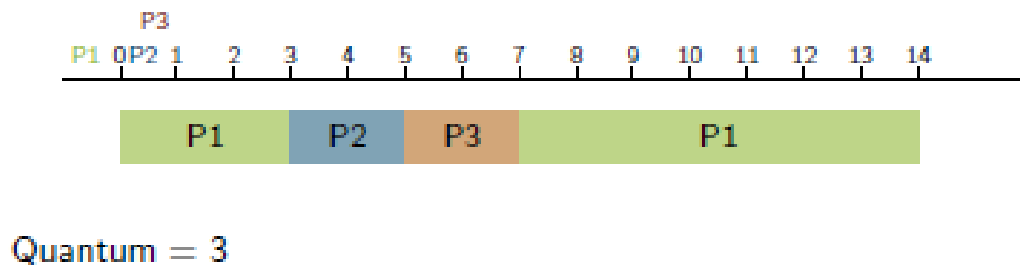
## Processus

### Algorithme: Round-Robin (tourniquet)

Cet algorithme cherche à équilibrer le temps d'attente entre les processus dans une file d'attente. Il minimise le temps d'attente pour **les petits processus**. La priorité ici se fonde sur:

- le temps de **quantum équitable** entre les processus prêt;
- Algorithme **FIFO** ;
- Règle:

### FIFO + Quantum de Temps:



Proc.	Date	Durée
P1	0	10
P2	1	2
P3	1	2

## Processus

### Application

On considère le tableau des processus suivant:

Proc.	Arrivée	Durée
P1	0	7
P2	1	4
P3	1	2
P4	2	2
P5	3	1

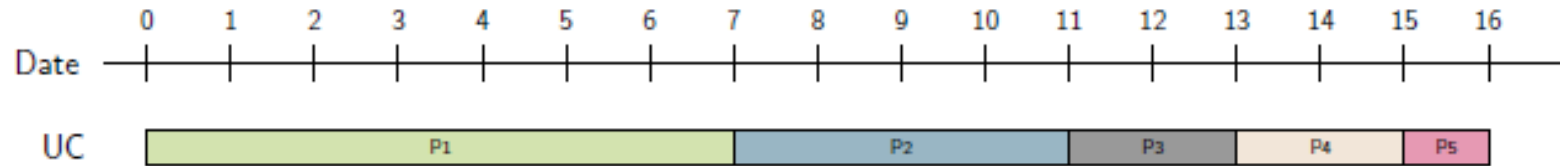
Indiquez dans un diagramme de Gantt le résultat d'un ordonnancement de type:

- **FIFO;**
- **Plus court d'abord;**
- **Round Robin, Q=3;**

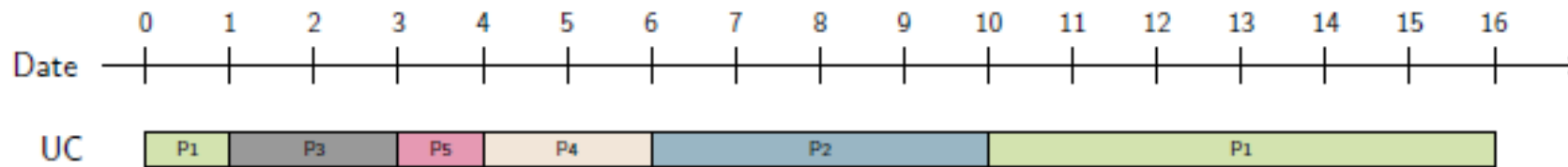
## Processus

### Solutions

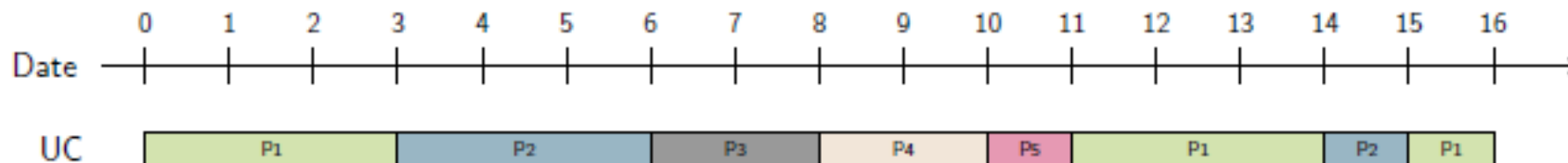
- FIFO;



- Plus court d'abord;



- Round Robin;

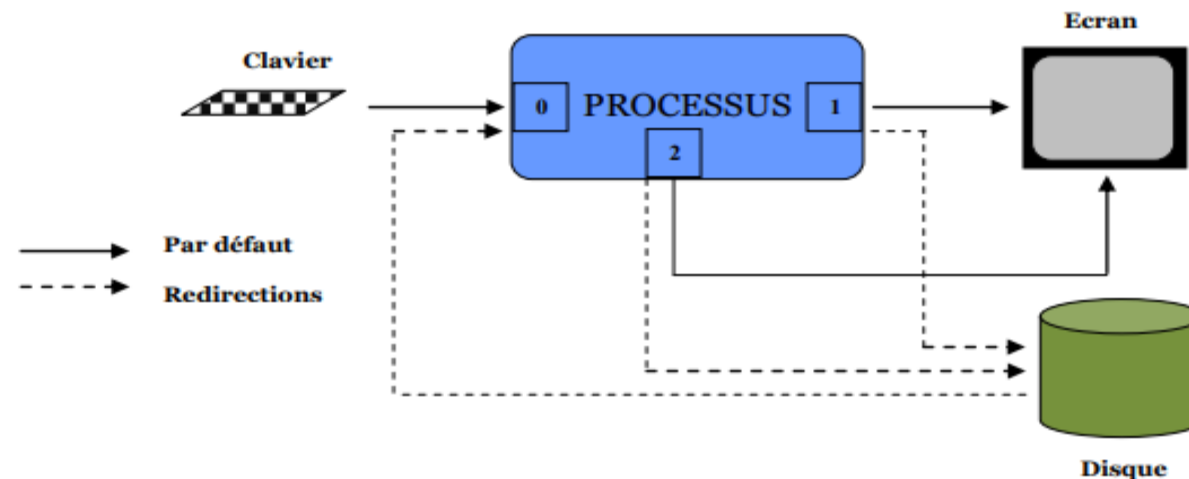


## Redirections

Les redirections facilitent la gestion du flux d'informations ; elles dirigent la sortie d'un programme vers l'entrée d'un autre programme.

Lorsqu'un processus démarre de façon standard, 3 fichiers spéciaux sont ouverts automatiquement :

- Entrée standard (**STDIN**): descripteur 0, par défaut équivalent au clavier.
- Sortie standard (**TDNOUT**): descripteur 1, par défaut équivalent à l'écran.
- Sortie erreur standard (**STDERR**): descripteur 2, par défaut équivalent à l'écran;



## Redirections

Ces 3 entrées/sorties permettent le redirection des résultats à partir/vers d'autres fichiers:

- Pour rediriger la sortie d'un processus vers un fichier: **>**
- Pour lire l'entrée d'un processus à partir d'un fichier: **<**
- Pour rediriger la sortie d'un processus vers la fin d'un fichier sans écraser son contenu: **>>**

La redirection de la sortie indique à la commande d'envoyer ses résultats **vers un fichier et non pas vers l'écran.**

Exemple:

- `.....~$ ls-l // vers l'écran.`
- `.....~$ ls -l > sortie // vers un fichier`

## Redirections

### La redirection des entrées sorties

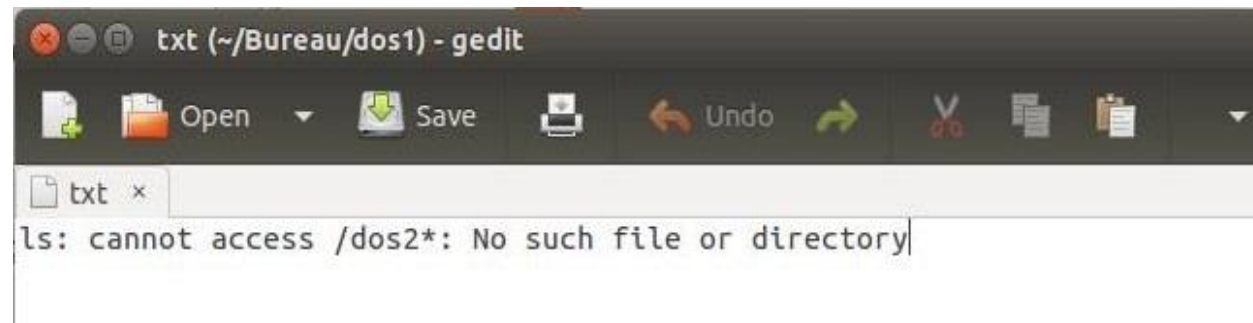
- ★ **Le caractère <** suivi du nom d'un fichier indique la redirection de **l'entrée** standard à partir de ce fichier.
- ★ **Le caractère >** suivi du nom d'un fichier indique la redirection de la sortie standard vers ce fichier.
- ★ **Les caractères >>** suivis du nom indiquent que l'information ou la redirection sera **ajoutée** au fichier.
- ★ **Les caractères 2>** suivis du nom d'un fichier indiquent la redirection de la **sortie d'erreur** standard vers ce fichier.

**Exemples :**

★ **ls >temp ; cat f1 >f2 ; cat <f1 >f2 ; ls /toto \* >temp 2>ertemp**

## Commandes

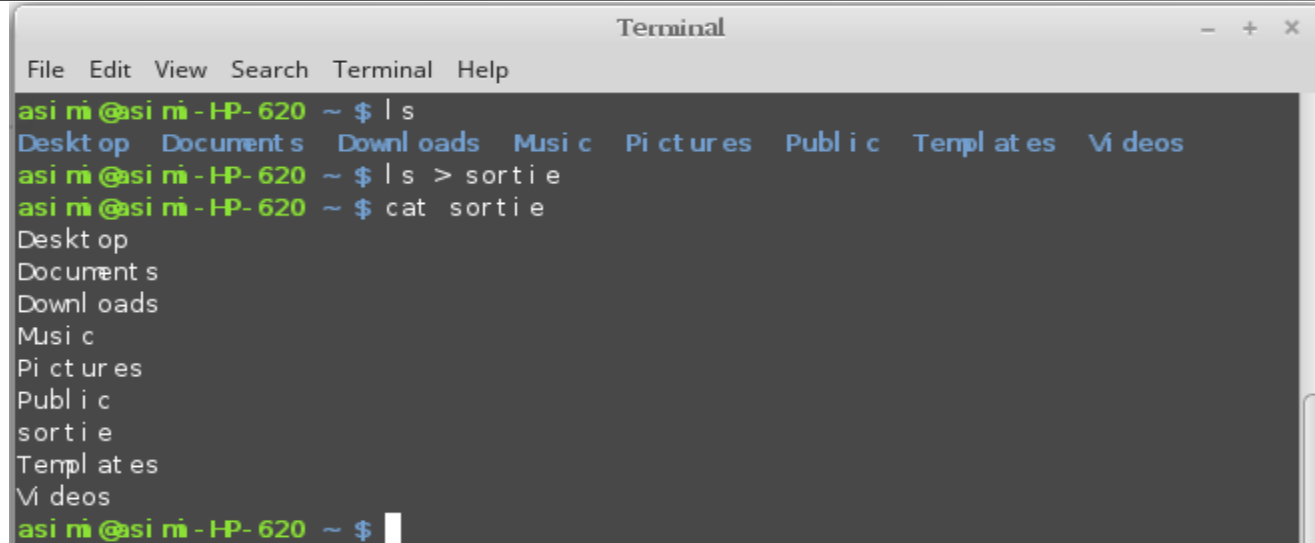
- **cat f1 > f4** : copier le contenu de f1 dans un nouveau fichier f4.
- **cat < f1 > f5** : copier le contenu de f1 dans un nouveau fichier f5.
- **ls /dos2\* 2>txt** : créer un fichier 'txt' et écrire dans ce fichier le message erreur.



The screenshot shows a gedit text editor window titled 'txt (~/Bureau/dos1) - gedit'. The window has a menu bar with 'Open', 'Save', 'Undo', and other options. Below the menu bar, there is a tab labeled 'txt'. The main text area contains the error message: 'ls: cannot access /dos2\*: No such file or directory'.

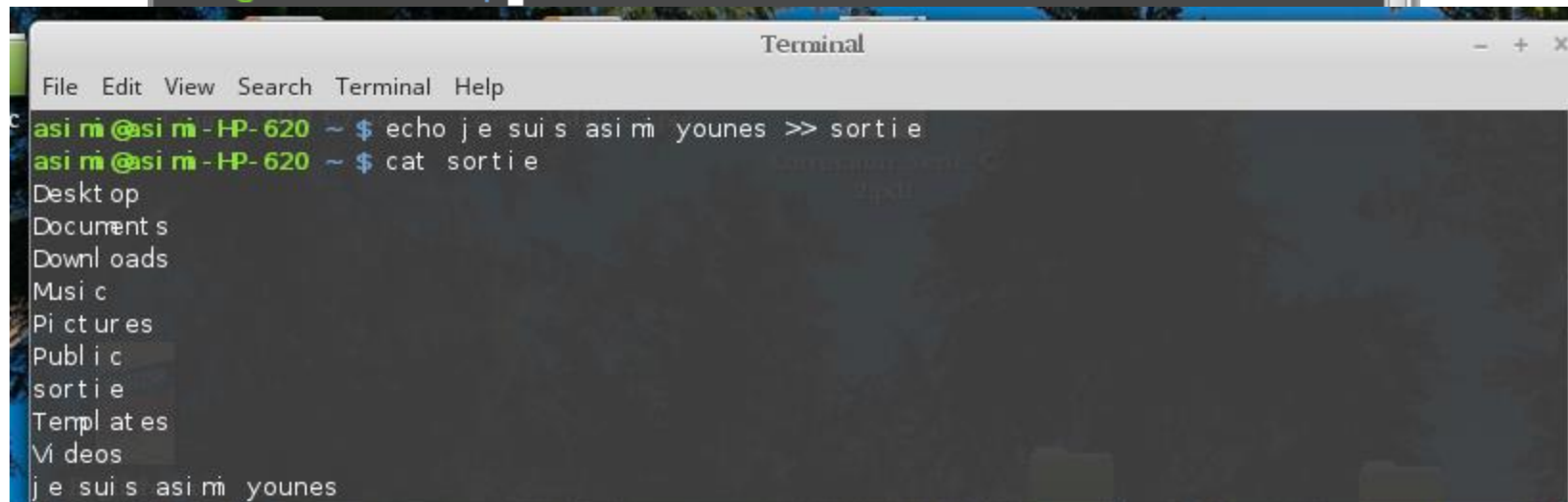


## Redirections



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "asi ni@asi ni -HP- 620 ~ \$". The user enters "ls", showing a directory listing. Then they enter "ls > sortie", creating a file named "sortie". Finally, they enter "cat sortie", which displays the directory listing contents in the terminal.

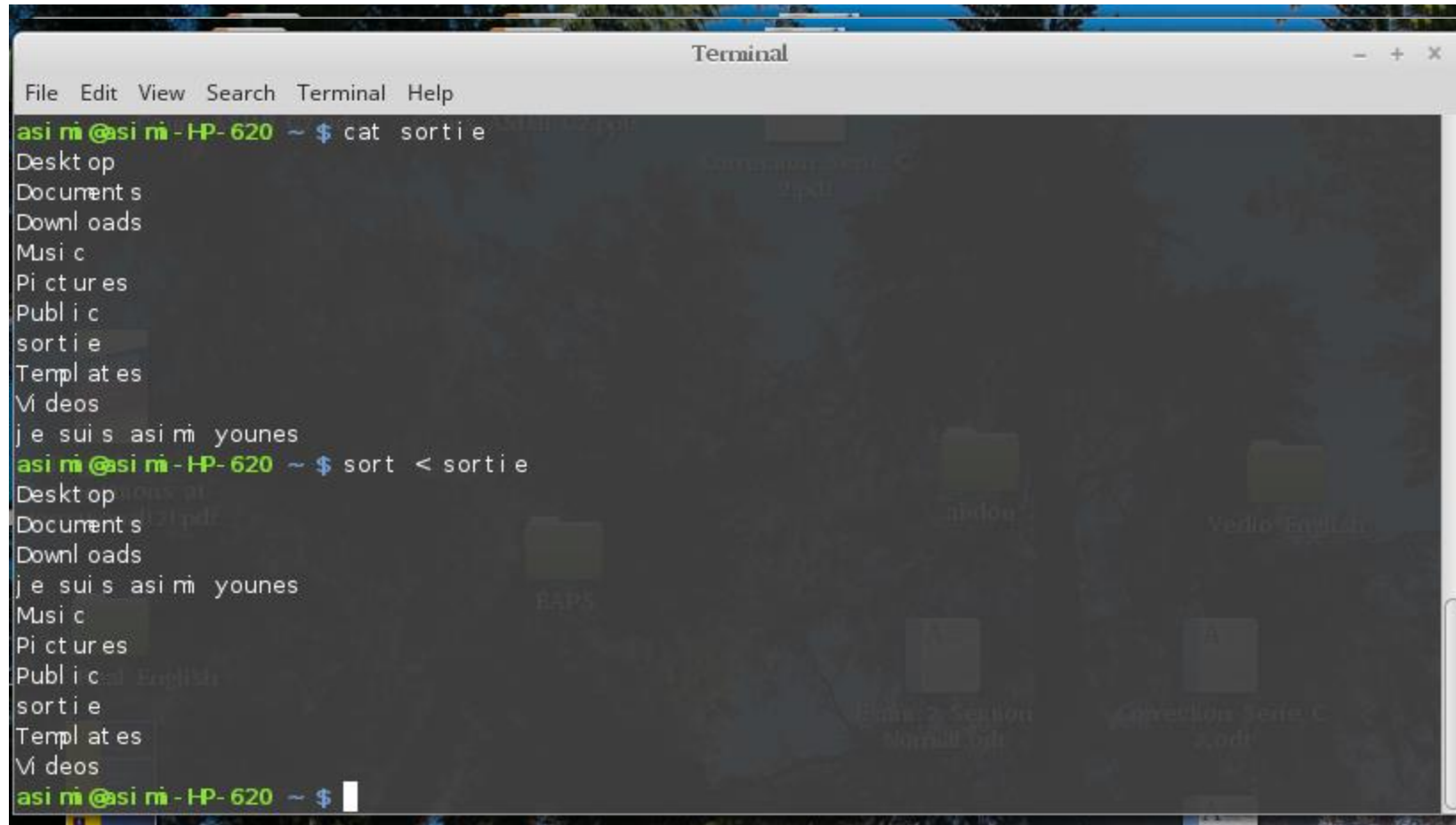
```
asi ni@asi ni -HP- 620 ~ $ ls
Desk top  Document s  Downl oads  Musi c  Pi ct ures  Publ i c  Tem pl at es  Vi deos
asi ni@asi ni -HP- 620 ~ $ ls > sortie
asi ni@asi ni -HP- 620 ~ $ cat sortie
Desk top
Document s
Downl oads
Musi c
Pi ct ures
Publ i c
sortie
Tem pl at es
Vi deos
asi ni@asi ni -HP- 620 ~ $
```



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "asi ni@asi ni -HP- 620 ~ \$". The user enters "echo je suis asi ni younes >> sortie", which appends the text "je suis asi ni younes" to the "sortie" file. Then they enter "cat sortie", which displays the output of the echo command.

```
asi ni@asi ni -HP- 620 ~ $ echo je suis asi ni younes >> sortie
asi ni@asi ni -HP- 620 ~ $ cat sortie
je suis asi ni younes
```

## Redirections



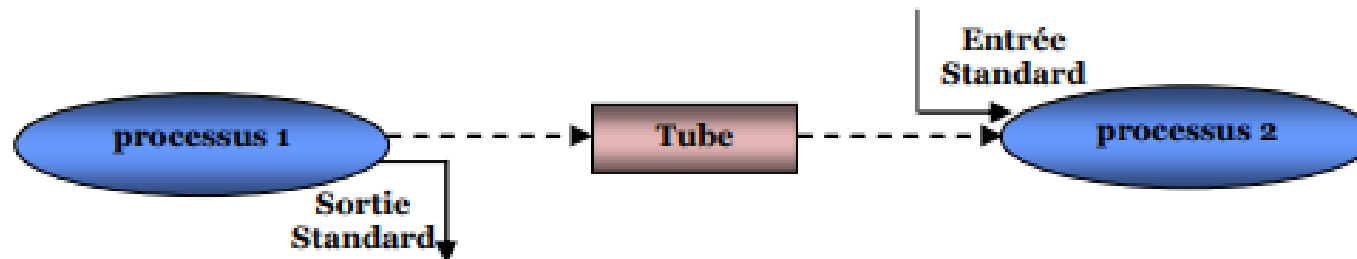
A screenshot of a Linux terminal window titled "Terminal". The window shows the execution of two commands. The first command is `cat sortie`, which outputs the contents of the file `sortie`. The second command is `sort < sortie`, which sorts the contents of `sortie` and outputs the result to the terminal. The background of the terminal window shows a desktop environment with various icons and folders.

```
File Edit View Search Terminal Help
asi ni@asi ni -HP- 620 ~ $ cat sortie
Desk top
Document s
Downl oads
Musi c
Pi ct ures
Publ i c
sortie
Templ at es
Vi deos
je suis asi ni younes
asi ni@asi ni -HP- 620 ~ $ sort < sortie
Desk top
Document s
Downl oads
je suis asi ni younes
Musi c
Pi ct ures
Publ i c
sortie
Templ at es
Vi deos
asi ni@asi ni -HP- 620 ~ $
```

## Communication Interprocessus

Les **tubes (Pipes)** permettent la **redirection** de la sortie standard d'un processus vers d'autre processus. Ainsi, la sortie de 1<sup>er</sup> processus définit l'entrée d'un tube dont la sortie est dirigée vers l'entrée du processus suivant.

Le but est d'inter-changer les données entre les processus sans passer par un fichier intermédiaire.



La commande qui permet la création des tubes est la suivante:

**commande1 | commande2 | ... | commandeN**



## Communication Interprocessus

Voila un exemple de communication entre les processus sans l'utilisation d'un tube:

- Commande qui calcule le nombre des lignes dans un fichier:

`who > sortie ; wc -l < sortie`

Ecrire cette commande avec l'utilisation d'un tube:

`who | wc -l`

Ecrire cette commande qui affiche et compte le nombre des processus lancés dans votre système :

`ps -l | wc -l`

Ecrire cette commande qui affiche et compte le nombre des fichiers et des répertoires existes dans le répertoire courant :

`ls | wc -l`

Ecrire cette commande qui permet de trier alphabétiquement la liste des utilisateurs connectés.

`ls -l /home | more`

## Communication Interprocessus

```
Terminal
File Edit View Search Terminal Help
asi ni@asi ni -HP- 620 ~ $ who
asi ni      tty8          2018-11-29 22:12 (:0)
asi ni      pts/1        2018-11-29 22:18 (:0)
asi ni@asi ni -HP- 620 ~ $ who | wc -l
2
asi ni@asi ni -HP- 620 ~ $ ps -l
F S   UID   PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000   2352   2345  0  80   0 -  5673 wait  pts/1        00:00:00 bash
0 R   1000   2372   2352  0  80   0 -  2509 -      pts/1        00:00:00 ps
asi ni@asi ni -HP- 620 ~ $ ps -l | wc -l
4
asi ni@asi ni -HP- 620 ~ $ ls
Desktop  Downloads  erreur  f2      Pi ct ures  sortie  Vi deos
Document s  entre      f1      Musi c  Publ i c    Templ at es
asi ni@asi ni -HP- 620 ~ $ ls | wc -l
13
asi ni@asi ni -HP- 620 ~ $ ls -l /home
total 4
drwxr-xr-x 30 asi ni asi ni 4096 Nov 29 22:12 asi ni
asi ni@asi ni -HP- 620 ~ $ ls -l /home | wc -l
2
asi ni@asi ni -HP- 620 ~ $
```