Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

COMPTE RENDU

Administration Réseaux et Programmation Système - TP1 - Processus, Alias, Complétion et Historique
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant : HATHOUTI Mohammed Taha

Filière : Cybersecurité

Année : 2025/2026

Enseignante : M.Noufel & M.ElAmrani

Date : 13 février 2026

Table des matières

1 Exercice 1 : Processus

Un processus est un programme qui s'exécute. Un processus a un propriétaire (l'utilisateur qui l'a lancé) et est identifié par un numéro : son pid (process identity). On peut visualiser les processus en cours grâce à la commande ps.

1.1

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$ man ps
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$ ps axu
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  24292 15420 ?        Ss   févr.09   0:01 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    févr.09   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    févr.09   0:00 [pool_workqueue_release]
root           4  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-rcu_gp]
root           5  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-sync_wq]
root           6  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-kvfree_rcu_reclaim]
root           7  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-slub_flushwq]
root           8  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-netns]
root          10  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/0:0H-events_highpri]
root          13  0.0  0.0      0     0 ?        I<   févr.09   0:00 [kworker/R-mm_percpu_wq]
root          14  0.0  0.0      0     0 ?        I    févr.09   0:00 [rcu_tasks_kthread]
root          15  0.0  0.0      0     0 ?        I    févr.09   0:00 [rcu_tasks_rude_kthread]
root          16  0.0  0.0      0     0 ?        I    févr.09   0:00 [rcu_tasks_trace_kthread]
root          17  0.0  0.0      0     0 ?        S    févr.09   0:00 [ksoftirqd/0]
root          18  0.0  0.0      0     0 ?        I    févr.09   0:05 [rcu_preempt]
root          19  0.0  0.0      0     0 ?        S    févr.09   0:00 [rcu_exp_par_gp_kthread_worker/0]
root          20  0.0  0.0      0     0 ?        S    févr.09   0:00 [rcu_exp_gp_kthread_worker]
root          21  0.0  0.0      0     0 ?        S    févr.09   0:00 [migration/0]
```

FIGURE 1 – Liste des processus système avec utilisateurs propriétaires (`ps aux`)

1.2

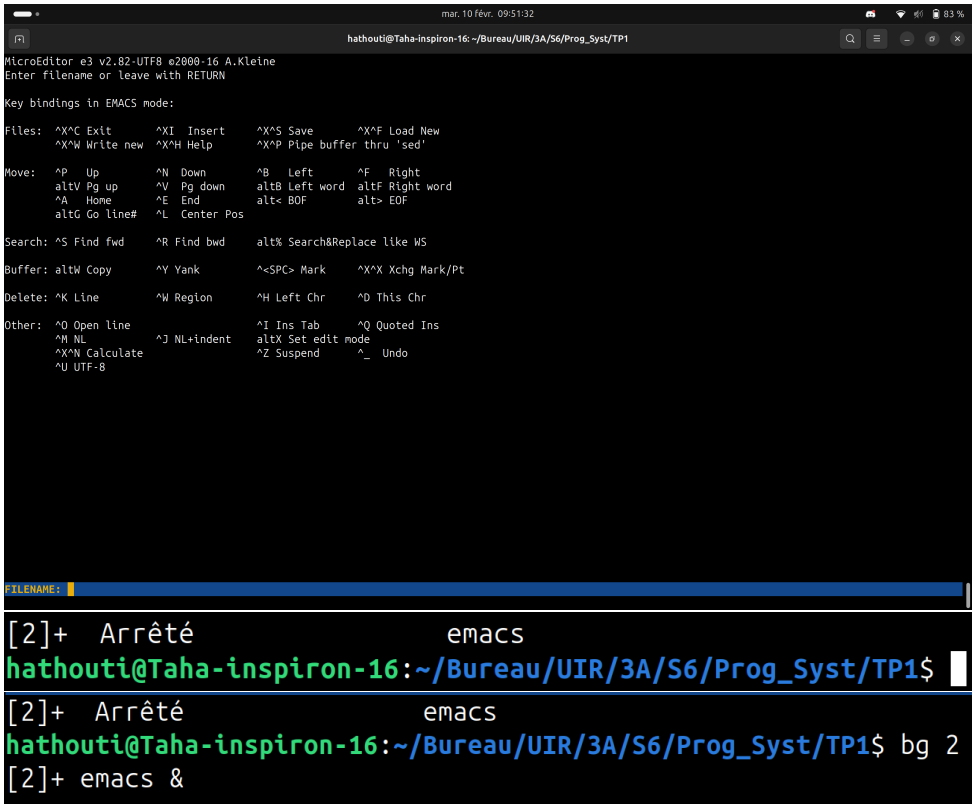

FIGURE 2 – Gestion d'un processus emacs : interruption (Ctrl-Z), vérification du job numéro 2, et relance en arrière-plan avec `bg 2`

1.3

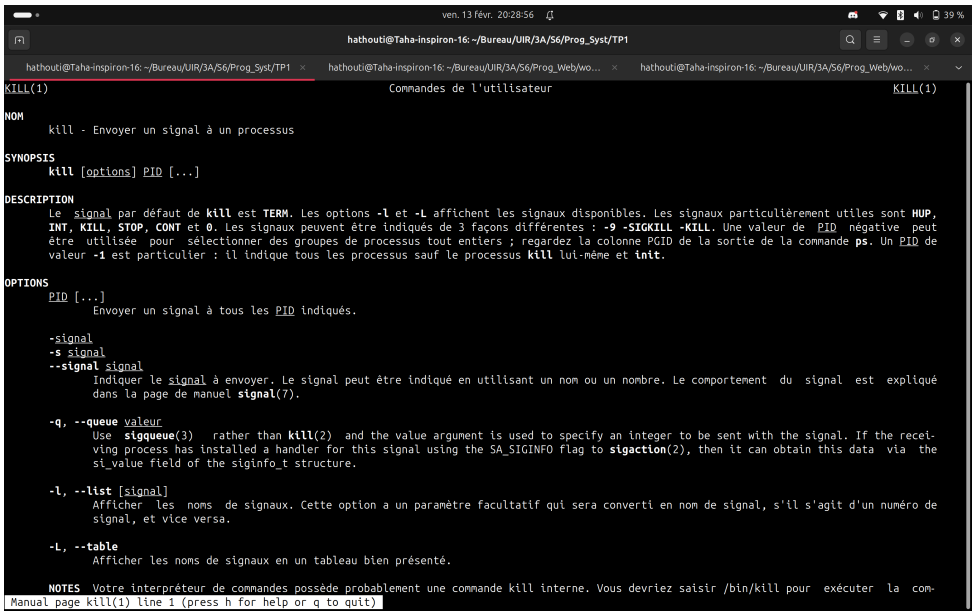

FIGURE 3 – Manuel de la commande `kill`

1.4

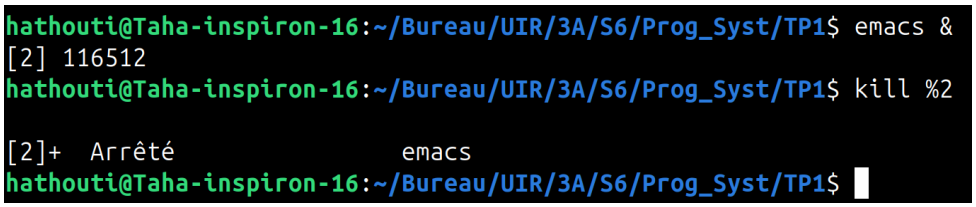

FIGURE 4 – Arrêt du processus emacs via son numéro de job avec `kill`

1.5

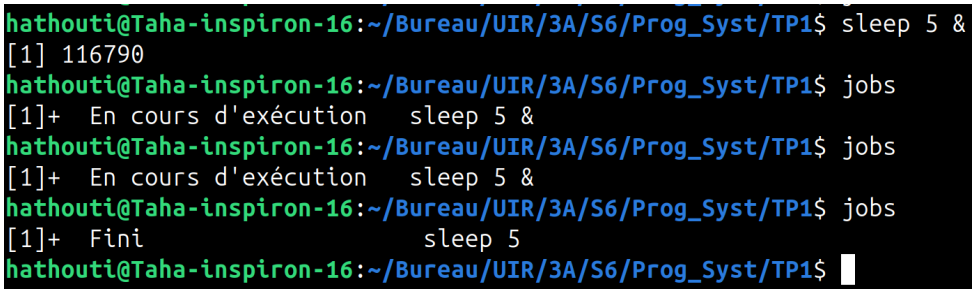

FIGURE 5 – Arrêt du processus emacs via son numéro de job avec `kill`

Différence entre `sleep 5` et `sleep 5 &` :

— `sleep 5` :
  — Bloque le terminal pendant 5s ;
  — Il faut attendre la fin des 5s avant de pouvoir réutiliser le terminal ;

— `sleep 5 &` :
  — **NE BLOQUE PAS** le terminal ;
  — Récupération immédiate du prompt ;
  — Utile pour utiliser des commandes en parallèle ;

1.6

Pour invoquer textuellement une autre fenêtre terminal il suffit de taper :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$ gnome-terminal
```

Il y a aussi une autre alternative plus pratique :

`Ctrl-Alt-T`

1.7

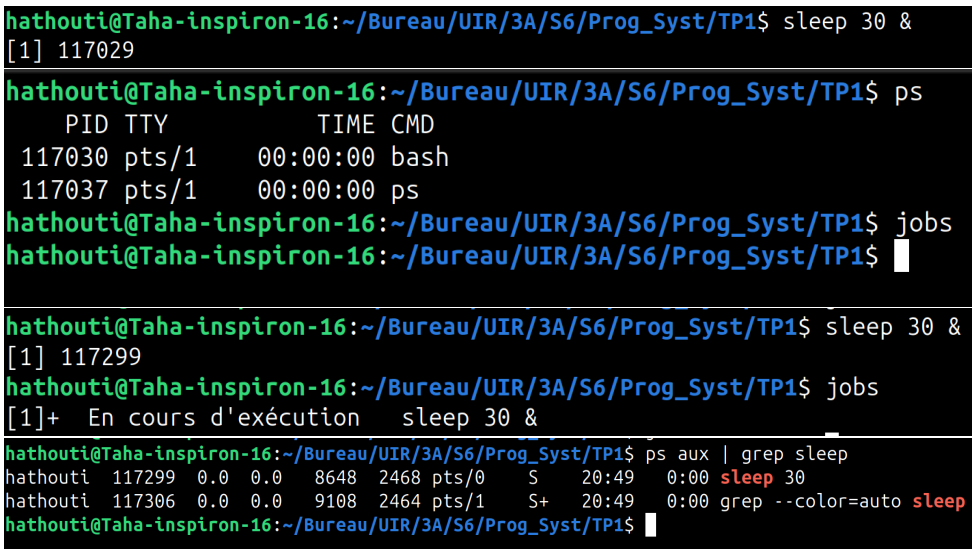

FIGURE 6 – Démonstration de la portée des jobs : `sleep 30` visible avec `ps aux` mais absent de `jobs` dans le Terminal 2

1.8

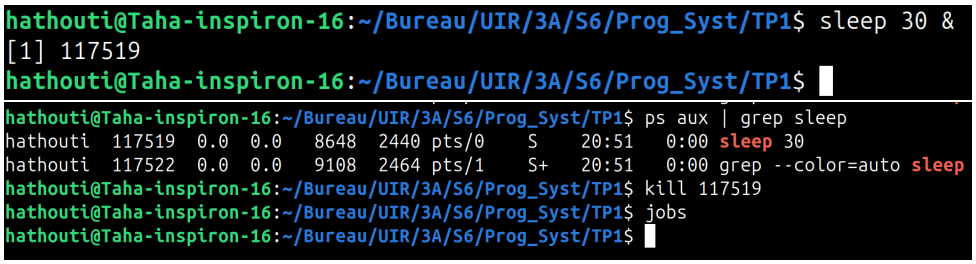

FIGURE 7 – Arrêt du processus `sleep` via son PID depuis un autre terminal

2 Exercice 2 : Historique

Le *C-shell* dispose d'un mécanisme d'historique qui permet de retrouver les dernières commandes tapées. On peut en fixer la taille en fixant la valeur de la variable history. Par exemple, `set history =` 100 permet d'enregistrer les 100 dernières commandes tapées dans le *shell*. On peut utiliser les *flèches* pour se déplacer dans l'historique ou bien utiliser le métacaractère `!`. La commande `history` fournit l'historique des commandes tapées. La commande `!n` relance la commande numéro n et `!chaine` la dernière commande qui commence par la chaîne de caractères fournie.

2.1

FIGURE 8 – Visualisation de l'historique des commandes avec `history`

2.2

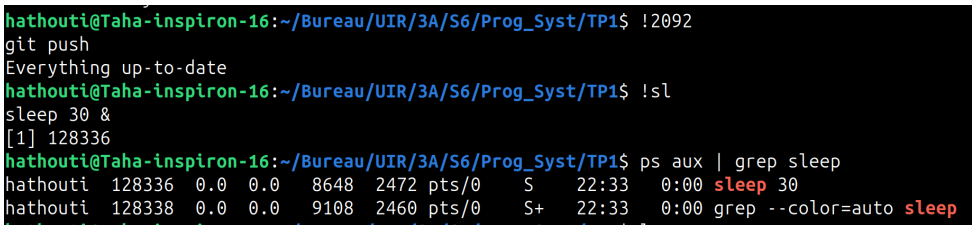

FIGURE 9 – Relance de commandes avec `!` : par numéro (`!2092`) et par chaîne (`!sl`)

2.3

FIGURE 10 – Fixation de la taille d'historique : `HISTSIZE=20`

2.4

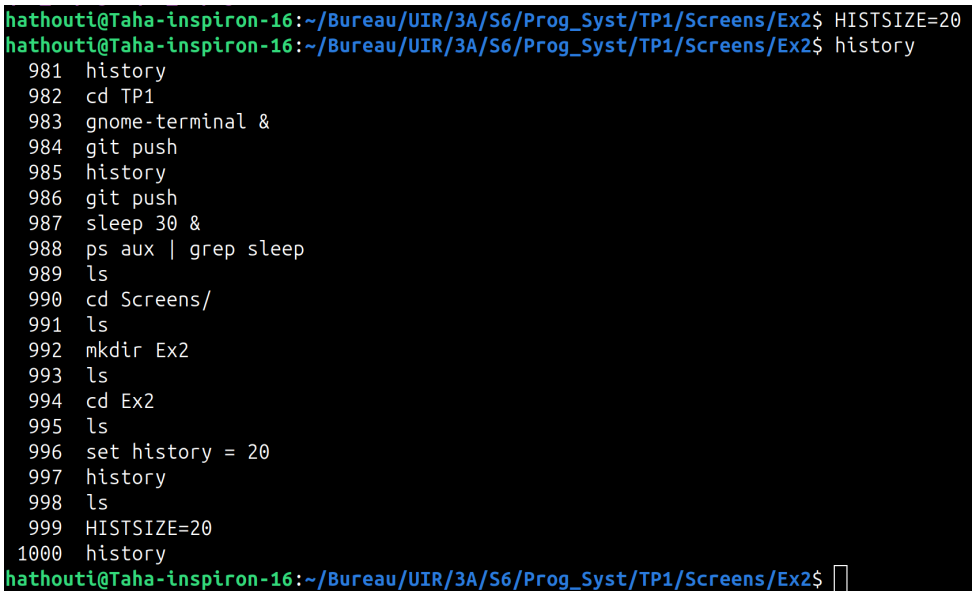

FIGURE 11 – Vérification de l'historique après limitation à 20 commandes

Après avoir exécuté la commande `history`, on constate que l'historique est désormais limité aux 20 dernières commandes (numérotées de 981 à 1000). Les commandes antérieures ont été supprimées de l'historique suite à la modification de la variable `HISTSIZE=20`.

3 Exercice 3 : Complétion et Alias

Le *csh* est capable de compléter un mot à partir des premiers caractères. Pour cela, il faut entrer le début du mot (par exemple, `ls pr` puis taper sur la touche `tab`. Le shell essaie alors de compléter `pr`. Si plusieurs mots différents peuvent compléter ce que vous avez entré, le shell n'affichera rien. Si vous tapez alors `Ctrl-D`, toutes les possibilités seront affichées. Vous pourrez alors entrer les caractères supplémentaires nécessaires qui permettront de choisir entre les diverses possibilités.

Le *C-shell* permet de créer des alias des commandes. Le but est soit de simplifier une commande ou de forcer une option de commande. Un *alias* est créé grâce à la commande `alias nom commande`. On le supprime avec `unalias nom`.

3.1

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$ ls /usr/bin/ch
chacl                  chardetect             chcon                  chfn                   chmod                  chrt
chage                  chartex                checkgid               chgrp                  choom                  chsh
chardet                chattr                 check-language-support chkdvifont             chown                  chvt
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$
```

FIGURE 12 – Complétion automatique : liste des commandes `/usr/bin/ch`

3.2

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$ alias m='more'
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$ alias del='rm -i'
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$ alias lr='ls -R'
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$
```

FIGURE 13 – Définition des alias : `m='more'`, `del='rm -i'` et `lr='ls -R'`

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1/Screens/Ex3$ lr ../../Screens
../../Screens:
Ex1  Ex2  Ex3

../../Screens/Ex1:
 qst_1.png      qst_2_exec_emacs.png   qst_3.png   qst_5.png         'qst_6_sleep_30_&.png'       qst_6_terminal2.png  'qst_7_sleep_30_&.png'
 qst_2_bg.png   qst_2_num_job.png      qst_4.png   qst_6_ps_aux.png   qst_6_sleep_tentative2.png   qst_7_kill.png

../../Screens/Ex2:
qst_1.png  qst_2.png  qst_3.png  qst_4.png

../../Screens/Ex3:
qst_1.png
```

FIGURE 14 – Test de l'alias `lr` pour un listage récursif

Je n'ai pas créé d'alias `ll` car un alias du même nom existe déjà dans mon système. L'utilisation de la commande `alias nom='commande'` crée un alias temporaire propre à la session actuelle. Cet alias apparaît dans la liste affichée par la commande `alias`, mais il n'est pas enregistré dans le fichier `~/.bashrc`. Par conséquent, il sera perdu à la fermeture du terminal. Pour rendre un alias permanent, il faut l'ajouter manuellement dans le fichier `~/.bashrc`. Seuls les alias présents dans ce fichier persistent d'une session à l'autre.

4 Exercice 4 : Métacarctères

Les métacaractères sont un mécanisme très puissant des systèmes Unix. Les principaux métacaractères sont rappelés ici :

— *** :** remplace zéro fois ou autant de fois que voulu n'importe quel caractère ;

— **? :** remplace exactement un caractère ;

— **ˆ :** désigne un début de ligne ;

— **$:** désigne une fin de ligne ;

— **\ :** indique que caractère suivant doit être interprété normalement ;

— **& :** permet de lancer une commande en arrière-plan ;

— **; :** permet de séparer des commandes écrites sur une seule ligne ;

— **[] :** permettent d'identifier une liste ou un intervalle de caractères. Par exemple `ls [a-p]*` liste tous les fichiers du répertoire courant qui commencent par une minuscule comprise entre **'a'** et **'p'** ;

4.1

Écrire les commandes permettant de lister :

1. Toutes les sources de programme *C++* :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls -R *.cc
```

2. Les fichiers commençant par *fic* :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls fic*
```

3. Les programmes *C++* commençant par *tp* :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls tp*.cc
```

4. Les fichiers de la forme *tpx.c* ou `x` est un chiffre (de 0 à 9) :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls tp[0-9].c
```

5. Les fichiers qui commencent par une majuscule comprise entre **'B'** et **'I'** :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls [B-I]*
```

6. Les fichiers dont le nom comporte au moins un caractère compris entre **'p'** et **'w'** :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls *[p-w]*
```

7. Les fichiers dont les noms se terminent par une minuscule comprise entre **'l'** et **'p'** :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls *[p-l]
```

8. Les fichiers dont le nom commence par une minuscule comprise entre **'b'** et **'f'** et ceux donc le nom commence par une minuscule comprise entre **'r'** et **'u'** :

```
hathouti@Taha-inspiron-16:~/Bureau/UIR/3A/S6/Prog_Syst/TP1$
   ls *[b-fr-u]*
```

5 Conclusion

Ce TP1 a permis de consolider les bases essentielles de l'administration système sous Linux, en se concentrant sur la gestion des processus, l'historique des commandes et la personnalisation de l'environnement shell.

Ce rapport détaille la mise en pratique de plusieurs commandes et concepts, structurés autour des points suivants :

— **Gestion des processus :** L'utilisation des commandes *ps*, *jobs*, *bg*, *fg* et *kill* pour visualiser, contrôler et terminer les processus en avant-plan et en arrière-plan ;

— **Mécanisme d'historique :** La manipulation de l'historique des commandes avec *history*, les rappels via *!n* et *!chaine*, ainsi que la configuration de la taille de l'historique avec *HISTSIZE* ;

— **Complétion et alias :** L'utilisation de la complétion automatique pour accélérer la saisie et la création d'alias temporaires et permanents pour simplifier l'exécution de commandes fréquentes ;

— **Métacaractères :** L'application des métacaractères (`*`, `?`, `[]`, `&`) pour effectuer des recherches avancées et manipuler efficacement les fichiers selon des motifs spécifiques ;

La maîtrise de ces outils est cruciale car ils constituent le fondement de l'administration système et de l'automatisation des tâches sous environnement Linux, permettant une gestion efficace des processus et une productivité dans l'utilisation du shell.