

# Commandes de base

---

## Question 3A

ps permet d'afficher une snapshot des processus courants

pstree affiche un arbre des processus

kill permet de fermer un processus (de le passer en mode T)

## Question 3B

l'argument -a selectionne tout les processus et -ux permet de les trier pour les traiter par PID

- USER indique l'utilisateur à qui appartient le processus
- PID est son identifiant unique
- %CPU est son utilisation sur le processeur en % d'utilisation
- %MEM et le % d'utilitacion de la memoire vive par le processus
- TTY est la session de shell depuis laquelle on lance le processus
- STAT est l'etat du processus
- START est sa date d'execution
- TIME est son temps d'execution

## Question 3C

Ces colonnes permettent d'etablir une priorité entre les processus

PR represente la priorité interne du kernel pour le processus donner et NI est la "nice value" plus elle est élevé plus elle laissera les autres processus prendre la priorité. C'est la gestion des priorités coté utilisateur

## Question 3D

```
ps -o pri -p <PID>
```

## Question 3E

la commande nice permet de demarrer un processus avec une priorité differente preciser en option

si le processus est deja en cours on peut utiliser la commande renice

## Question 3F

kill -3 correspond à SIGQUIT

kill -9 correspond à SIGKILL

dans la liste on retrouve :

SIGHUP	1	1	1	1
SIGINT	2	2	2	2
SIGQUIT	3	3	3	3
SIGILL	4	4	4	4
SIGTRAP	5	5	5	5
SIGABRT	6	6	6	6
SIGIOT	6	6	6	6
SIGBUS	7	10	10	10
SIGEMT	-	7	7	-
SIGFPE	8	8	8	8
SIGKILL	9	9	9	9
SIGSTOP	19	19	19	19

## Question 3G

Les signaux SIGKILL et SIGSTOP ont une particularité, ils ne sont pas envoyés au processus mais directement adressés au système qui se chargera de stopper ou d'arrêter le processus

## Question 3H

Nohup est une commande Unix permettant de lancer un processus qui restera actif même après une déconnexion de l'utilisateur ayant initialisé le processus

## Question 3I

On utilise CTRL+Z pour passer un processus en pause, la commande bg pour le faire exécuter en arrière plan et fg pour le repasser au premier plan

# Gestion des processus

---

## Question 4A

On peut utiliser la commande bg pour passer le processus en background ou utiliser l'esperluette quand on lance le programme

## Question 4B

le processus continue en arrière plan malgré le clear du terminal

## Question 4C

on peut utiliser CTRL+Z pour mettre en pause un processus, on peut aussi utiliser la commande kill -19

## Question 4D

on utilise la commande fg "UID"

## Question 4E

On peut utiliser la commande `kill -9` ou faire un `CTRL+C`

## Question 4F

numero de tache sert pour du multithreading car la variable est partager en toutes les taches dans le meme thread et numero de processus pour le multiprocessing et differencier les differents processus.

## Question 4G

le processus est lié au terminal car il s'interrompt quand on sort de notre session. On peut le verifier car lorsque qu'on se reconnecte au terminal le processus à disparu

## Question 4H

`nohup` permet de lancer un processus qui n'est pas liée à une session utilisateur

`disown` permet de modifier un processus deja actif pour le detacher de la session auquel il est rattacher

## Question 4I

```
nice ./randomgenerator -"numero de la priorité"
```

## Question 4J

```
renice -"numero de la priorité" "PID"
```

# Redirection de flux

---

## Question 5A

Crée un fichier `rand.txt` et ecrit le resultat de la commande à l'interieur. Si il existe deja un fichier, on l'ecrase et on reecrit dedans

## Question 5B

meme chose que la 5A mais n'ecrase pas le fichier si il est deja crée, on ecrit seulement à la suite.

## Question 5C

demande à l'utilisateur le nombre de nombres aleatoires qu'il souhaite generer

## Question 5D

permet de preciser le parametre dans un string qui sera directement appellé lors de la commande suivante

## Question 5E

On crée un fichier `response.txt` et on écrit 20 à l'intérieur

## Question 5F

lit le contenu du fichier `response.txt` et le précise directement à l'argument `i`

## Question 5G

Même chose que 5D mais trie les valeurs par ordre croissant et les affiche seulement lorsque le programme a fini de tourner

## Question 5H

`stdin` est l'entrée standard

`stdout` est la sortie standard

`stderr` est la sortie d'erreur standard

## Question 5I

Génère 50 nombres aléatoires et crée un fichier nommé `data.txt` pour écrire ces nombres à l'intérieur. On précise `&` car on lance le processus en background

## Question 5J

affiche le contenu du fichier `data.txt` trié par ordre croissant

## Question 5K

`atoi` permet de transformer une chaîne de caractères en entier

# Création d'un processus en C

---

## Question 6A

On utilise la commande `fork` pour démarrer plusieurs instances d'un processus. On peut par exemple retrouver son utilisation pour les serveurs web tels que Apache pour pouvoir lancer plusieurs serveurs simultanément.

## Question 6B

Son PID est différent du PID parent, c'est un processus à part entière qui partage les mêmes variables et le même code que le processus parent

## Question 6C

un processus Forké crée un nouveau PID est agis comme un nouveau processus alors que plusieurs thread peuvent appartenir au meme processus et partage la meme memoire virtuelle

## Question 6D

l'espace memoire disponible peut etre une limitation assez concrete de l'utilisation d'un fork

## Question 6E

# Utilisation de fork, wait, waitpid, sleep

---

## Question 7A

lors de l'execution du programme en arriere plan on execute la commande ps et on se rends compte que les childs créés par notre programme sont "defunct"

## Question 7B

Les processus créés sont des processus zombie, on va modifier le code du processus père pour corriger ce probleme

## Question 7C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/wait.h>

int main(int argc, char const *argv[])
{
    char *p;
    long nbChildProcess = strtol(argv[1], &p, 10);
    srand(time(NULL));
    int variable = 0;
    int tmp = 1;

    while(variable<1){
        variable = rand()%12;
    }

    for (int i = 0; i < nbChildProcess; ++i) {
        sleep(variable);
        if (fork() == 0) {
            printf("%d Je suis un fils ---- PID : %ld\n", tmp, getpid());
        }
        while (waitpid(-1, NULL, WNOHANG) != 0);
        tmp++;
    }
}
```

```
    }  
    return (0);  
}
```

## Question 7D

```
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <time.h>  
#include <sys/wait.h>  
  
int main(int argc, char const *argv[])  
{  
    char *p;  
    //long nbChildProcess = strtol(argv[1], &p, 10);  
    srand(time(NULL));  
    int variable = 0;  
    int tmp = 1;  
    int const constante = 3;  
  
    while(variable<1){  
        variable = rand()%12;  
    }  
  
    for (int i = 0; i < constante; ++i) {  
        sleep(variable);  
        if (fork() == 0) {  
            printf("%d Je suis un fils ---- PID : %ld\n",tmp,getpid());  
        }  
        while (waitpid(-1,NULL, WNOHANG) != 0);  
        tmp++;  
    }  
    return (0);  
}
```

## Question 7E