TP 1: Fork

```
% man 3 printf
```

> May the manual be strong with you!

Exercice 1. Say Hello!

Ecrire un simple programme **hello.c** permettant d'afficher le message "*Hello World!*" grâce à la fonction `printf` et compilez le.

La syntaxe de la fonction main sera la suivante :

```c
int main(void){
  // Votre code ici
}
```

Fonction utile: `printf`

Exercice 2. Quels sont mes arguments? `argc`, `argv`

Écrire un programme **compte_arg.c** qui prend en argument une liste d'entier et qui en affiche la somme.

```
% ./compte_arg 1 2 3
6
```

La syntaxe de la fonction main sera la suivante :

```c
int main(int argc, char** argv){
  // Votre code ici
}
```

N'oubliez pas que le type de *argv* est `char**`, la conversion en entier doit donc être faite soit :

- par *atoi* : `n = atoi(argv[i]);`
- par *sscanf* : `sscanf(argv[i], "%d", &n);`

Fonctions utiles : `atoi` *ou* `sscanf`

Include	Fonction
`#include <stdlib.h>`	`int atoi(const char* str)`
`#include <unistd.h>`	`int sscanf(const char *restrict s, const char *restrict format)`

Exercice 3. Fork

Écrire un programme *fork1.c* qui utilise la primitive `fork()` pour créer un processus fils :

- Dans le processus père vérifier que le `fork` s'est bien passé (code de retour != -1).
- Le processus père devra afficher son *PID*, le *PID* de son propre père et celui de son fils, attendre la terminaison du fils et quitter. Vous utiliserez pour cela la fonction `wait`.
- Le processus fils devra afficher son *PID*, le *PID* de son propre père.

Écrire un programme *fork2.c*, modification de *fork1.c*, où la fonction `wait` est remplacée par la fonction `waitpid`.

Modifiez le code pour que le père affiche le code de retour du fils en utilisant les macros : `WIFEXITED` et `WEXITSTATUS`.

Fonctions utile:

Include	Fonction
`#include <unistd.h>`	`pid_t fork(void);`
`#include <unistd.h>`	`pid_t getppid(void);`
`#include <unistd.h>`	`pid_t getpid(void);`
`#include <sys/wait.h>`	`pid_t wait(int *stat_loc);`
`#include <sys/wait.h>`	`pid_t waitpid(pid_t pid, int * stat_loc, int options);`

Exercice 4. Fork et For (mise en oeuvre de l'exercice 1 qui sera repris au TD 1)

Écrivez un programme **fork_loop.c** qui réalise une boucle `for` variant de *0* à *n-1* et qui, à chaque itération, effectue un `fork`. *n* est un paramètre du programme lu sur la ligne de commande (*i.e.* en utilisant `argv` et `argc`).

D'après vous, pour *n* fixé, combien de processus fils sont créés par ce programme ? Pour vérifier, après le `fork`, placez l'affichage suivant :

```
1    printf("pid = %8d, pid_fils = %8d, ppid = %8d, i=%8d\n", getpid(),
        pid_f, getppid(), i);
```

où `pid_f` est la valeur du retour de `fork`.

Observez la valeur du *ppid*. Que remarquez-vous ? Explications ?