TP1. Process

1.1 Quand on utilise la commande suivante *gcc -o exec.exe tp1.c* dans le terminal, rien ne s'affiche. Cependant un nouveau fichier « exec.exe » apparait dans le dossier où se trouve tp1.c. Nous venons de créer un executable de tp1.

1.2 Après voir exécuté la commande ci-dessus, si l'on entre la commande suivante: ./exec.exe (on lance l'executable de tp1.c) la console affiche : « Hello World ! » Le programme a correctement été exécuté.

```
adrito@ubuntu:~/Documents/ECE$ gcc -o hl tp1.c
adrito@ubuntu:~/Documents/ECE$ ./hl
Hello World!adrito@ubuntu:~/Documents/ECE$
```

1.3 On utilise l'option -g afin de faire apparaître les éventuelles érreurs de compilation du programme. C'est le mode débug de gcc. On peut désormais placer des points de breakpoint.

1.4 Il existe une commande d'aide « help » à laquelle il faut associer un type afin d'être aidé.

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands
```

2.2 Après l'utilisation de la fonction fork(), le process est cloné, ce dernier s'apelle process enfant. Le nombre de processus crée suit la formule suivante : $2^{n.}$ avec n le nombre de fork().
Pour différencier le proccessus parent de l'enfant, on peut regarder la valeure retournée par fork(). Si la valeur est égale à 0, c'est l'enfant, 1 le parent.

2.3

```
int main(){

        pid_t pid;
        pid = fork();

        if (pid == 0)
        {
                //printf("Child process id:      ");
                printf("I'm the child    ");
                printf("%d\n", getpid());
        }else{
                //printf("Parent process id:     ");
```

```
            printf("I'm the parent  ");
            printf("%d\n", getpid());
        }
        return 0;

}
```

```
adrito@ubuntu:~/Documents/ECE$  gcc -o exec.exe tp1fork.c
adrito@ubuntu:~/Documents/ECE$ ./exec.exe
I'm the parent  4226
adrito@ubuntu:~/Documents/ECE$ I'm the child    4227
```

2.4 Les deux processus parent et enfant ne partagent pas les données entre eux. Les données traitée de manière internes au processus n'en sortent pas.

I'm the parent 4954
5
adrito@ubuntu:~/Documents/ECE$ I'm the child   4955

En inversant :

I'm the parent 6627
adrito@ubuntu:~/Documents/ECE$ I'm the child   6628
5

2.5 Le code va permettre de créer deux processus enfant au premier parent puis un nouvel enfant à chacun des enfants (de la première génération).

```
adrito@ubuntu:~/Documents/ECE$ I'm the parent id: 2783 | Parent id : 1394

I'm the child id: 2784
Parent id: 1394

I'm the child id: 2785
Parent id: 1394

I'm the grandchild id: 2787
Parent id: 1394

I'm the grandchild id: 2786
Parent id: 1394
```

3.2 Quand on utilise la commande exec, on lance pas un nouveau process à proprement parler, on remplace dans le bash une commande à executer. En jetant à œil au proces id de l'exec, ce dernier reste inchangé.

```
int main(){

	if(fork()==0){
	execl("/usr/bin/firefox","--new window","www.eurosport.com",NULL);
	printf("I'm the child id: %d\n", getpid());
	}else{
	sleep(2);
	printf("I'm the parent id: %d", getpid());
	}
	return 0;
}
```

3.3 Lors du fork, le processus enfant crée possède les mêmes données que le processus parent. Une update du processus parent n'influe pas sur l'enfant car ils sont deux processus distincts.

3.4

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(){

	int i = 5;
	if(fork()==0){
		execl("/usr/bin/firefox","--new window","www.eurosport.com",NULL);
		i++;
		printf("I'm the child pid: %d\n, i: %d", getpid(), i);
	}else{
	printf("I'm the parent pid: %d", getpid());
	}
	sleep(3);
	return 0;
}
```

4. La fct system permet d’executer une commande dans le terminal.