TP N°: 2 : Correction

Exercice 1

Écrire un programme en C qui crée un tube anonyme, puis crée deux processus enfants. Le premier processus doit lire un message à partir du tube et l'afficher. Le deuxième processus doit envoyer un message au tube. Le processus parent doit attendre que les processus enfants se terminent avant de se terminer lui-même.

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

int main() {
    int fd[2];
    pid_t pid1, pid2;
    char message[20];
    pipe(fd) ;
    if (pipe(fd) == -1) {
        printf("Erreur lors de la création du tube");
        exit(1);
    }

    // Création du premier processus enfant
    pid1 = fork();
    if (pid1 == -1) {
        printf("Erreur lors de la création du premier processus enfant");
        exit(1);
    } else if (pid1 == 0) {
        // Code pour le premier processus enfant (lecteur)
        close(fd[1]); // Fermeture du côté écriture du tube
        read(fd[0], message, 20);
        printf("Message reçu : %s\n", message);
        exit(0);
    }

    // Création du deuxième processus enfant
    pid2 = fork();
    if (pid2 == -1) {
        perror("Erreur lors de la création du deuxième processus enfant");
        exit(1);
    } else if (pid2 == 0) {
        // Code pour le deuxième processus enfant (écrivain)
        close(fd[0]); // Fermeture du côté lecture du tube
        sprintf(message, "Bonjour !");
        write(fd[1], message, 20);
       // write(fd[1], "Bonjour", 20);
        exit(0);
    }

    // Attente de la terminaison des processus enfants
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);
     exit(0);

    return 0;
}
```

Exercice 2

Écrire un programme en C qui crée un tube (pipe) et deux processus enfants.

- → Le processus parent doit écrire une chaîne de caractères dans le tube, et les deux processus enfants doivent lire cette chaîne de caractères.
- → Le **premier** processus enfant doit convertir tous les caractères en **minuscules** et le **deuxième** processus enfant doit convertir tous les caractères en **majuscules**.
- → Les deux processus enfants doivent afficher la chaîne de caractères convertie et se terminer.

Note : Vous pouvez utiliser les fonctions prédéfinies dans la bibliothèque **ctype.h** suivante :

`int tolower(int c);`	pour convertir un caractère en minuscule.
`int toupper(int c);`	pour convertir un caractère en majuscule.

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

int main() {
    int fd[2];
    pid_t pid1, pid2;
    char msg[] = "BoNjouR MONDE";
    char buffer[50];
    pipe(fd) ;
    if (pipe(fd) == -1) {
        printf("Erreur lors de la création du tube");
        exit(1);
    }
    pid1 = fork();
    if (pid1 == -1) {
        printf("Erreur lors de la création du processus enfant 1");
        exit(1);
    }
    else if (pid1 == 0) { // Code du premier processus enfant
        close(fd[1]); // Fermer le descripteur d'écriture
        read(fd[0], buffer, sizeof(buffer)); // Lire depuis le tube
        for (int i = 0; i < strlen(buffer); i++) {
            buffer[i] = tolower(buffer[i]); // Convertir en minuscules
        }
        printf("Premier processus enfant : %s \n", buffer);
        close(fd[0]); // Fermer le descripteur de lecture
        exit(0);
    }else{
    pid2 = fork();
    if (pid2 == -1) {
        printf("error");
        exit(1);
    }
    else if (pid2 == 0) { // Code du deuxième processus enfant
        close(fd[1]); // Fermer le descripteur d'écriture
        read(fd[0], buffer, sizeof(buffer)); // Lire depuis le tube
        for (int i = 0; i < strlen(buffer); i++) {
            buffer[i] = toupper(buffer[i]); // Convertir en majuscules
        }
        printf("Deuxième processus enfant : %s\n", buffer);
        close(fd[0]); // Fermer le descripteur de lecture
        exit(0);
    }
   }
    close(fd[0]); // Fermer le descripteur de lecture
    write(fd[1], msg, strlen(msg) + 1); // Écrire dans le tube
    close(fd[1]); // Fermer le descripteur d'écriture
    waitpid(pid1, NULL, 0); // Attendre la fin des processus enfants
    waitpid(pid2, NULL, 0);
     exit(0);
    return 0;
}
```

Exercice 3

Écrire un programme en C qui crée deux processus enfants à l'aide de la fonction ***fork***.

→ Le processus parent doit envoyer un signal **SIGTERM** aux processus enfants après 5 secondes à l'aide de la fonction ***kill***.

→ Le processus enfant qui reçoit le signal doit afficher "Signal SIGTERM reçu" et se terminer.

→ Le processus parent doit attendre que les deux processus enfants se terminent avant de se terminer lui-même.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>

void gestionnaire_signal(int sig) {
    if (sig == SIGTERM) {
        printf("Signal SIGTERM reçu\n");
        sleep(10);
        exit(0); }
    }
int main() {
    pid_t pid1, pid2;
    pid1 = fork();
    if (pid1 == 0) {
        printf("Erreur lors de la création du processus enfant 1");
        exit(1);}
    else if (pid1 == 0) {
        printf("Je suis le processus enfant 1 avec PID %d\n", getpid());
        signal(SIGTERM, gestionnaire_signal);
        while(1){}
    }else {
    pid2 = fork();
    if (pid2 == 0) {
        printf("Erreur lors de la création du processus enfant 2");
        exit(1);}
    if (pid2 == 0) {
        printf("Je suis le processus enfant 2 avec PID %d\n", getpid());
        signal(SIGTERM, gestionnaire_signal);
        while(1){}
        }
    }

    sleep(5);
    kill(pid1, SIGTERM);
    kill(pid2, SIGTERM);
    printf("Signal SIGTERM envoyé aux processus enfants\n");
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);
    printf("Les processus enfants sont terminés\n");
    return 0;
}
```