TD2 : Exo1

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


struct sigaction ActFils;
struct sigaction ActPere;
pid_t fils_pid;

void fonction_fils(){
    int val = rand() % 30 + 10;
    printf("%d", val);
    fflush(stdout);
}

void fonction_pere(){
    kill(fils_pid, SIGUSR1);
    alarm(5);
}

int main(){
    printf("INIT");
    fils_pid = fork();
    if (fils_pid == 0){
        // printf("Je suis le fils");
        ActFils.sa_handler = fonction_fils;
        sigaction(SIGUSR1, &ActFils, 0);
        while(1){
            pause();
        }
    } else if (fils_pid > 0){
        // printf("Je suis le pere");
        ActPere.sa_handler = fonction_pere;
        sigaction(SIGALRM, &ActPere, 0);
        alarm(5);
        while(1){
            sleep(1);
            printf("-");
            fflush(stdout); // Pour afficher le printf sans attendre le retour à la ligne (vide le buff
        }
    } else {
        printf("Erreur de création du fils");
    }

}
```

TD2 : Exo2

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


struct sigaction ActFils;
struct sigaction ActPere;
pid_t fils_pid;
int cpt = 0;
char cfils = 'a';
char cpere = 'A';

void fonction_fils(){
    cpt++;
    for (int i = 0; i < cpt && cfils <= 'z'; i++){
        printf("%c", cfils);
        fflush(stdout);
        cfils++;
    }
    kill(getppid(), SIGUSR1);
    if(cfils > 'z'){
        exit(0);
    }
}

void fonction_pere(){
    cpt++;
    for (int i = 0; i < cpt && cpere <= 'Z'; i++){
        printf("%c", cpere);
        fflush(stdout);
        cpere++;
    }
    if(cpere > 'Z'){
        exit(0);
    }
    kill(fils_pid, SIGUSR1);
}

int main(){
    fils_pid = fork();
    if (fils_pid == 0){
        ActFils.sa_handler = fonction_fils;
        sigaction(SIGUSR1, &ActFils, 0);
        while (1)
        {
```

```
            pause();
        }

    } else if (fils_pid > 0){
        ActPere.sa_handler = fonction_pere;
        sigaction(SIGUSR1, &ActPere, 0);
        sleep(1);
        kill(fils_pid, SIGUSR1);
        while (1)
        {
            pause();
        }
    } else {
        printf("Erreur de création du fils");
    }
    printf("\n");

}
```

TD2 : Exo3

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


//Variables globales
struct sigaction ActFils;
struct sigaction ActPere;
pid_t fils_pid;

//Compteurs de SIGINT
//Hypothèse : les compteurs de SIGINT sont indépendants
int cptPere = 0;
int cptFils = 0;


// Fonction fils
void captfils(){
    rectvert(5);
    cptFils++;

    // 3 SIGINT Recu pour le processus fils
    if(cptFils == 3){
        detruitrec();
        exit(0);
    }
```

```
}


//Fonction pere
void captpere(){
    cptPere++;
    printf("PERE %d : signal %d recu\n", getpid(),cptPere);
    fflush(stdout);

    //Si 3 SIGINT reçus pour le père
    if(cptPere == 3){
        printf("PERE : fin du pere , trois signals sont deja recu\n");
        fflush(stdout);
        exit(0);
    }
}


//-------------------------------------------------------------------------------------------

//Main

int main(){
    fils_pid = fork();

    //Partie Processus fils
    if (fils_pid == 0){
    int i =0;
        initrec();
        ActFils.sa_handler = captfils;
        sigaction(SIGINT, &ActFils, 0);

        //Tant qu'on ne clic pas sur le boutton de fin
        while (i != -1)
        {
            i = attendreclic();

            //On clic sur 0
            if(i==0){
                printf("FILS envoit au PERE\nFILS :  pid du mon pere est : %d\n", getppid());
                fflush (stdout);

            //Envoie du SIGINT au père
                kill(getppid(), SIGINT);
            }

            //On clic sur autre chose que 0
            else {
            printf("FILS :  pid de mon pere est : %d \n", getppid());
            fflush(stdout);
```

```
            }
        }

        // Après clic sur fin
    printf("FILS :fin du fils apres clic sur FIN\n");
    fflush(stdout);
    exit(0);
    }

    // Partie processus père
    else if (fils_pid > 0){
        ActPere.sa_handler = captpere;
        sigaction(SIGINT, &ActPere, 0);

        // Attente pour être sur que le sigaction
        // du fils a bien été pris en compte
        sleep(1);
        int n;
        while (1)
        {
            n = sleep(10);
            printf("temps restant : %d \n", n);
            fflush(stdout);
        }

    }

    // Si on a pas réussi à créer le fils
    else {
        printf("Erreur de création du fils");
        fflush(stdout);
    }
    printf("\n");
    fflush(stdout);
    return 0;
}
```

TD4 : Exo1

```
#include "sharemem.h"

#define BLKSIZE 1024

int copierfichier(int f1, int f2){
    char buf[BLKSIZE];
    int octets_lus, octets_ecrits, total = 0;
    for(;;){
        if((octets_lus = read(f1, buf, BLKSIZE)) <= 0){
            break;
        }
```

```
        if((octets_ecrits = write(f2, buf, octets_lus)) == -1){
            break;
        }
        total += octets_ecrits;
    }
    return total;
}
```

TD4 : Exo2

```
// suru
#include "sharemem.h"

#define BLKSIZE 1024

int copierfichier(int f1, int f2){
    char buf[BLKSIZE];
    int octets_lus, octets_ecrits, total = 0;
    for(;;){
        if((octets_lus = read(f1, buf, BLKSIZE)) <= 0){
            break;
        }
        if((octets_ecrits = write(f2, buf, octets_lus)) == -1){
            break;
        }
        total += octets_ecrits;
    }
    return total;
}

int main(int argc, char *argv[]){
    int octets_lus, childpid, fd, fd1, fd2;
    if(argc == 3){
        fprintf(stderr,"Usage: %s <fichier1> <fichier2> \n", argv[0]);
        return 1;
    }
    if(((fd1 = open(argv[1], O_RDONLY)) == -1) || (fd2 = open(argv[2], O_RDONLY) == -1)){ // Vérifie si
        perror("Echec");
        return 1;
    }
    if((childpid = fork()) == -1){
        perror("Echec");
        return 1;
    }
    if(childpid>0) // Parent code
        fd = fd1;
    else  // Child code
        fd = fd2;
    octets_lus = copierfichier(fd, STDOUT_FILENO);
    fprintf(stderr, "Octets Lus : %d \n", octets_lus);
```

```
    return 0;
}


// Compile : gcc -u prgm surv.c
// Run : ./prgm f1.data fé.data
// SURVSHM
#include "sharemem.h"

#define BLKSIZE 1024

#define PERM (S_IRUSR | S_IWUSR)
#define SHMZ 27

int copierfichier(int f1, int f2){
    char buf[BLKSIZE];
    int octets_lus, octets_ecrits, total = 0;
    for(;;){
        if((octets_lus = read(f1, buf, BLKSIZE)) <= 0){
            break;
        }
        if((octets_ecrits = write(f2, buf, octets_lus)) == -1){
            break;
        }
        total += octets_ecrits;
    }
    return total;
}

int statchAndRemove(int shmid, char *shmdt){
    int error = 0;
    if(shmdt(shmdt) == -1){
        error = errno;
    }
    if(shmdt(shmid, IPC_RMID, NULL) == -1 && !error){
        error = errno;
    }
    if(!error){
        return 0;
    }
}
```

TD4 : Exo3

```
// INIFIC
#include <stdio.h>
#include <fcntl.h>

void main(){
    int tab1[10]={11,22,33,44,55,66,77,88,99,1000};
```

```
    int fd = open("titi.dat", O_RDWR|O_CREAT|O_TRUNC, 0666);
    if(fd == -1){
        printf("Erreur lors de l'ouverture du fichier\n");
        perror("open");
        exit(-1);
    }
    write(fd, tab1, 10*sizeof(int));
    close(fd);
    printf("Fichier initialisé\n");
}
// LIREFIC
void main(){
    int tab2[10];
    int i;
    // Ouverture du fichier
    int fd = open("titi.dat", O_RDWR, 0666);
    if(fd == -1){
        printf("Erreur lors de l'ouverture du fichier\n");
        perror("open");
        exit(-1);
    }
    //Lecture du fichier
    read(fd, tab2, 10*sizeof(int));
    close(fd);
    for(i=0; i<10; i++)
        printf("%d\n",tab2[i]);
}

// MODIFIC
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>


void main(int argc, char* argv[]){
    int i;
    struct stat file_stat;
    char* addr;
    int file;
    int* tab1;
     // Ouverture du fichier
    file = open("titi.dat",O_RDWR, 0666);
    if(file == -1){
        printf("Erreur lors de l'ouverture du fichier\n");
        perror("open");
        exit(-1);
```

```
    }

    // Récupération des informations du fichier
    fstat(file, &file_stat);

    //Ouverture de la mémoire partagé
    tab1 = mmap(NULL,file_stat.st_size,PROT_READ | PROT_WRITE,MAP_SHARED,file,0);
    close(file);
    //Boucle sur i
    while (1){
        printf("Rentrez i \n");
        scanf(" %d",&i);
        if (i == 99){
            // Sortie du programme
            munmap(tab1, file_stat.st_size);
            exit(0);
        }
        // Incrémentation du nombre de 1 à l'indice [i] dans le segment de la mémoire partagée
        else if( i >= 0 && i <= 9){
            tab1[i]++;
        }
    }

    // Libération de la mémoire partagée
    munmap(tab1, file_stat.st_size);
    return 0;
}

// SHOWFIC
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>


void main(int argc, char* argv[]){
    int i;
    struct stat file_stat;
    char* addr;
    int file;
    int* tab1;

    // Ouverture du fichier
    file = open("titi.dat",O_RDWR, 0666);
    if(file == -1){
        printf("Erreur lors de l'ouverture du fichier\n");
        perror("open");
```

```
        exit(-1);
    }

    // Récupération des informations du fichier
    fstat(file, &file_stat);

    //Ouverture de la mémoire partagé
    tab1 = mmap(NULL,file_stat.st_size,PROT_READ | PROT_WRITE,MAP_SHARED,file,0);
    close(file);
    //Boucle sur i
    while (1){
        printf("Rentrez i \n");
        scanf(" %d",&i);
        if (i == 99){
            // Sortie du programme
            munmap(tab1, file_stat.st_size);
            exit(0);
        }
        // Affichage du contenu de la mémoire partagée
        else if( i >= 0 && i <= 9){
            for (int ctp = 0 ; ctp <= 9 ; ctp++)
                printf("%d\n", tab1[ctp]);
        }
    }

    // Libération de la mémoire partagée
    munmap(tab1, file_stat.st_size);
    return 0;
}
```

TD4 : Exo4

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    if(argc != 2) {
        printf("Syntaxe: %s fichier_à_inverser\n", argv[0]);
        exit(-1);
    }

    int fd = open(argv[1], O_RDWR);
    if(fd == -1) {
        printf("Erreur lors de l'ouverture du fichier\n");
        perror("open");
        exit(-1);
```

```
    }

    struct stat sb;
    fstat(fd, &sb); // Cette fonction nous permet de récupérer les informations sur le fichier
(dans notre cas la taille)
    int *fichier = mmap(NULL, sb.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    int *fichier_inverse = malloc(sb.st_size); // nouvelle zone de mémoire pr le contenu inversé
    int i = sb.st_size / sizeof(int) - 1; // On commence à la fin du premier fichier
    int j = 0; // On écrit au début du fichier à inverser
    while (i >= 0) {
        fichier_inverse[j] = fichier[i]; // copie des caractères inversés
        i--;
        j++;
    }
    memcpy(fichier, fichier_inverse, sb.st_size); // copie de la nouvelle zone de mémoire dans le fichi
    printf("Le contenu du fichier a été inversé avec succès\n");
    munmap(fichier, sb.st_size); // libération de la mémoire
    free(fichier_inverse); // libération de la mémoire alouée dynamiquement
    close(fd);
    return 0;
}
```