Timothé Belcour
TP1

TP Système Exploitation IPC/SHM-signaux

Exercice 1 :
shm_create.c :

```
  GNU nano 7.2                                          shm_create.c *
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

#define SHM_SIZE 256

int main(void) {
    key_t key = ftok("shm.key", 'T');
    if (key == -1) { perror("ftok"); return 1; }

    int shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) { perror("shmget"); return 1; }

    printf("SHM créée ok (key=%d, shmid=%d, size=%d)\n", (int)key, shmid, SHM_SIZE);
    return 0;
}
```

Timothé Belcour
TP1

shm_write.c :

```
  GNU nano 7.2                                                shm_wr
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#define SHM_SIZE 256

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "usage: %s \"message\"\n", argv[0]);
        return 1;
    }

    key_t key = ftok("shm.key", 'T');
    if (key == -1) { perror("ftok"); return 1; }

    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) { perror("shmget"); return 1; }

    char *p = (char*)shmat(shmid, NULL, 0);
    if (p == (void*)-1) { perror("shmat"); return 1; }

    snprintf(p, SHM_SIZE, "%s", argv[1]);

    if (shmdt(p) == -1) { perror("shmdt"); return 1; }
    puts("Écriture OK.");
    return 0;
}
```

Timothé Belcour
TP1

shm_read.c :

```
  GNU nano 7.2                                                shm_read.c *
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

#define SHM_SIZE 256

int main(void) {
    key_t key = ftok("shm.key", 'T');
    if (key == -1) { perror("ftok"); return 1; }

    int shmid = shmget(key, SHM_SIZE, 0666);
    if (shmid == -1) { perror("shmget"); return 1; }

    char *p = (char*)shmat(shmid, NULL, 0);
    if (p == (void*)-1) { perror("shmat"); return 1; }

    printf("Contenu SHM : \"%s\"\n", p);

    if (shmdt(p) == -1) { perror("shmdt"); return 1; }
    return 0;
}
```

```
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ gcc shm_create.c   -o shm_create
gcc shm_write.c    -o shm_write
gcc shm_read.c     -o shm_read
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ ls
shm_create  shm_create.c  shm.key  shm_read  shm_read.c  shm_write  shm_write.c
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ ./shm_create
SHM créée ok (key=1409419760, shmid=10, size=256)
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ ./shm_write "Bonjour"
Écriture OK.
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ ./shm_read
Contenu SHM : "Bonjour"
```

Ce résultat montre que la mémoire partagée permet à plusieurs processus distincts d'échanger des données de manière directe et efficace, sans passer par des fichiers ni par des canaux anonymes.

La chaîne « Bonjour » écrite par le programme d'écriture « write » a bien été retrouvée par le programme de lecture « read », prouvant le bon fonctionnement de la zone mémoire commune.

Timothé Belcour
TP1

Exercice 2 :
p1_p2_shm_signaux.c :

```
  GNU nano 7.2                                          p1_p2_shm_signaux.c
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 15

static int shmid;
static char *shm = NULL;
static pid_t pid_pere, pid_fils;

void hdl_sigusr1(int sig) {
    (void)sig;
    char recu[SHM_SIZE];
    snprintf(recu, SHM_SIZE, "%s", shm);
    printf("[FILS %d] Reçu A=\"%s\"\n", (int)getpid(), recu);
    fflush(stdout);

    const char *B = "123456789";
    snprintf(shm, SHM_SIZE, "%s", B);
    kill(pid_pere, SIGUSR2);
}

void hdl_sigusr2(int sig) {
```

```
  GNU nano 7.2                                  p1_p2_shm_signaux.c
void hdl_sigusr2(int sig) {
    (void)sig;
    char recu[SHM_SIZE];
    snprintf(recu, SHM_SIZE, "%s", shm);
    printf("[PERE %d] Reçu B=\"%s\"\n", (int)getpid(), recu);
    fflush(stdout);

    shmdt(shm);
    shmctl(shmid, IPC_RMID, NULL);
    exit(0);
}

int main(void) {
    key_t key = ftok("shm.key", 'T');
    if (key == -1) { perror("ftok"); return 1; }

    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) { perror("shmget"); return 1; }

    shm = (char*)shmat(shmid, NULL, 0);
    if (shm == (void*)-1) { perror("shmat"); return 1; }

    pid_pere = getpid();
    pid_fils = fork();
    if (pid_fils < 0) { perror("fork"); return 1; }
    if (pid_fils == 0) {
        pid_pere = getppid();
        signal(SIGUSR1, hdl_sigusr1);
```

Timothé Belcour
TP1

```
 GNU nano 7.2                                          p1_p2_shm_signaux.c
    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) { perror("shmget"); return 1; }

    shm = (char*)shmat(shmid, NULL, 0);
    if (shm == (void*)-1) { perror("shmat"); return 1; }

    pid_pere = getpid();
    pid_fils = fork();
    if (pid_fils < 0) { perror("fork"); return 1; }
    if (pid_fils == 0) {
        pid_pere = getppid();
        signal(SIGUSR1, hdl_sigusr1);
        pause();
        shmdt(shm);
        return 0;
    } else {
        signal(SIGUSR2, hdl_sigusr2);

        const char *A = "ABCDEFGHI";
        snprintf(shm, SHM_SIZE, "%s", A);

        sleep(1);
        kill(pid_fils, SIGUSR1);
        pause();
    }
    return 0;
}
```

```
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ nano p1_p2_shm_signaux.c
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ gcc p1_p2_shm_signaux.c -o p1p2
belcour@belcour-QEMU-Virtual-Machine:~/TP_IPC$ ./p1p2
[FILS 6752] Reçu A="ABCDEFGHI"
[PERE 6751] Reçu B="123456789"
```

Cet exercice permet de voir la communication entre un processus père et son fils à l'aide d'une mémoire partagée et de signaux.

Les signaux permettent de synchroniser précisément les échanges et d'éviter tout accès simultané, tandis que la mémoire partagée sert de zone de transfert rapide et directe, sans passer par des fichiers ou des canaux.

L'exécution montre que la chaîne écrite par le père a bien été lue puis remplacée par le fils, confirmant le bon fonctionnement et l'usage correct des primitives shmget, shmat, shmdt, signal, kill et pause.