TD 4	
Module : Programmation système - Section ICE3	
Enseignants : Chiheb Ameur ABID Islem DENDEN	Année universitaire : 2023-2024

I- Files de messages

Exercice 1.

- Écrire un programme lançant 2 processus : le premier (le client) génère et envoie des questions (ou requêtes) au second (le serveur), et le serveur répond aux questions et affiche les réponses. Les questions sont des opérations mathématiques simples à effectuer (par exemple « combien font 7 + 13 ? »). On commencera par définir la structure contenant les questions, c'est-à-dire une opération (+,-,* et /), et deux opérandes. La communication se fait en utilisant une file de messages.
- Modifier le programme précédent pour que (toujours en utilisant une seule file de messages) :
  - le serveur puisse répondre aux questions de différents clients,
  - le serveur renvoie la réponse au client qui a émis la question, par la file de messages,
  - et les clients récupèrent dans la file de messages les réponses qui leur sont adressées, et les affichent.
- On suppose que le traitement des questions par le serveur n'est pas immédiat (par exemple, il met 5 secondes pour traiter une question). Écrire une fonction affichant le nombre de requêtes en attente dans la file de messages.
- Modifier le programme pour qu'il y ait autant de serveur que de types de questions possibles, c'est-à-dire un serveur chargé de répondre aux questions +, un aux questions -, un aux * et un aux / (toujours en utilisant une seule file de messages).

Code source du client :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// Structure pour la question
struct question {
    long type;
    int operand1;
    int operand2;
    char operator;
};

struct response {
    long type;
    double result;
};

int main() {
    key_t key;
```



```
    int msgid;
    struct response resp;

    // Générer une clé unique pour la file de messages
    key = ftok("ex1.txt", 65);
    if (key == -1) {
        perror("Erreur lors de la création de la clé");
        exit(EXIT_FAILURE);
    }

    // Créer une file de messages
    msgid = msgget(key, 0666 | IPC_CREAT);
    if (msgid == -1) {
        perror("Erreur lors de la création de la file de messages");
        exit(EXIT_FAILURE);
    }

    // Exemple de 3 questions
    struct question questions;
    questions.type = 1;
    // Envoi des questions au serveur
    do {
        printf("Your question ?\n");
        char c;
        do {
            c = getchar();
          } while (c != '+' && c != '-' && c != '/' && c != '*' && c !=
'x');
        questions.operator = c;
        if (questions.operator != 'x') {
            printf("Enter 1st value :\n");
            scanf("%d", &questions.operand1);
            printf("Enter 2nd value :\n");
            scanf("%d", &questions.operand2);
                 if (msgsnd(msgid, &questions, sizeof(struct question) -
sizeof(long), 0) == -1) {
                perror("Erreur lors de l'envoi de la question");
                exit(EXIT_FAILURE);
            }
              printf("Question envoyée : %d %c %d\n", questions.operand1,
questions.operator, questions.operand2);
            printf("Attente de la reponse!\n");

            //while(1) {
            ssize_t taille;
```

```
            if ((taille = msgrcv(msgid, &resp, sizeof(struct response) -
sizeof(long), 2, 0)) == -1) {
                perror("msgrcv");
                exit(EXIT_FAILURE);
            }
            printf("Taille %ld\n", taille);
            printf("Reponse recu %f...\n", resp.result);
        }
    } while (questions.operator != 'x');
    printf("Toutes les questions ont été envoyées.\n");
    return 0;
}
```

Code source du serveur :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// Structure pour la réponse
struct response {
    long type;
    double result;
};

struct question {
    long type;
    int operand1;
    int operand2;
    char operator;
};

int main() {
    key_t key;
    int msgid;
    struct question q;
    struct response r;

    // Générer une clé unique pour la file de messages
    key = ftok("ex1.txt", 65);
    if (key == -1) {
        perror("Erreur lors de la création de la clé");
```



```
        exit(EXIT_FAILURE);
    }

    // Obtenir l'ID de la file de messages
    msgid = msgget(key, 0666 | IPC_CREAT);
    if (msgid == -1) {
        perror("Erreur lors de la récupération de la file de
messages");
        exit(EXIT_FAILURE);
    }

    // Réception et traitement des questions
    while (1) {
        ssize_t taille;
        if ((taille = msgrcv(msgid, &q, sizeof(struct question) -
sizeof(long), 1, 0)) == -1) {
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }
        //if (taille != 8) {
            printf("Taille : %ld \n", taille);
            printf("Question reçue : %d %c %d\n", q.operand1,
q.operator, q.operand2);

            // Calcul de la réponse
            switch (q.operator) {
                case '+':
                    r.result = q.operand1 + q.operand2;
                    break;
                case '-':
                    r.result = q.operand1 - q.operand2;
                    break;
                case '*':
                    r.result = q.operand1 * q.operand2;
                    break;
                case '/':
                    if (q.operand2 != 0)
                        r.result = (double) q.operand1 /
q.operand2;
                    else
```

```
                    r.result = 0.0;
                break;
            default:
                r.result = 0.0;
        }

        // Envoyer la réponse au client
        r.type = 2;
            if (msgsnd(msgid, &r, sizeof(struct response) -
sizeof(long), 0) == -1) {
            perror("Erreur lors de l'envoi de la réponse");
            exit(EXIT_FAILURE);
        }
        printf("Réponse envoyée : %lf\n", r.result);
    //}
}

    return 0;
}
```

II- Mémoire partagée

Exercice 2.

Écrire un programme qui créé une mémoire partagée (un int) et qui créé un fils.
Le père écrit dans cette variable 0 et attend que la valeur soit égale à 1 Quand c'est le cas, il affiche « le père dit 0 ! » et écrit de nouveau zéro… Cette série se répète à l'infini.
Le fils écrit dans cette variable 1 et attend que la valeur soit égale à 0 Quand c'est le cas, il affiche « le fils dit 1 ! » et écrit de nouveau 1… Cette série se répète à l'infini.

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

int main() {
    // Créer une clé unique pour la mémoire partagée
    key_t key = ftok("/etc/passwd", 'A');
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Créer la mémoire partagée
    int shmid = shmget(key, sizeof(int), 0666 | IPC_CREAT);
```


كلية العلوم بتونس
FACULTE DES SCIENCES DE TUNIS

```
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attacher la mémoire partagée
    int *shared_memory = (int *)shmat(shmid, NULL, 0);
    *shared_memory = 0;  // Initialise la mémoire partagée à 0

    // Créer un processus fils
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork");
        exit(1);
    }

    if (pid == 0) {
        // Code du fils
        while (1) {
            while (*shared_memory != 0) {
                // Attendre que la valeur soit 0
                usleep(100);
            }
            printf("Le fils dit 1 !\n");
            *shared_memory = 1;
        }
    } else {
        // Code du père
        while (1) {
            while (*shared_memory != 1) {
                // Attendre que la valeur soit 1
                usleep(100);
            }
            printf("Le père dit 0 !\n");
            *shared_memory = 0;
        }
    }

    // Détacher et supprimer la mémoire partagée (normalement, cela ne se
produira jamais)
    shmdt(shared_memory);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

Exercice 3.
Écrire un programme qui créé une mémoire partagée (un int) et qui incrémente cette valeur toutes les secondes.
Écrire un second programme qui affiche la valeur de cette mémoire.

Premier programme :

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

int main() {
    // Créer une clé unique pour la mémoire partagée
    key_t key = ftok("shmfile", 65);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Créer la mémoire partagée
    int shmid = shmget(key, sizeof(int), 0666 | IPC_CREAT);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attacher la mémoire partagée
    int *shared_memory = (int *)shmat(shmid, NULL, 0);

    // Incrémenter la mémoire partagée toutes les secondes
    while (1) {
        sleep(1);
        (*shared_memory)++;
    }


    shmdt(shared_memory);
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}
```

Second Programme :

```
#include <stdio.h>
#include <stdlib.h>
```



```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main() {
    // Créer une clé unique pour la mémoire partagée
    key_t key = ftok("shmfile", 65);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Obtenir l'ID de la mémoire partagée
    int shmid = shmget(key, sizeof(int), 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attacher la mémoire partagée
    int *shared_memory = (int *)shmat(shmid, NULL, 0);

    // Afficher la valeur de la mémoire partagée en boucle
    while (1) {
  sleep(3);
        printf("Valeur de la mémoire partagée : %d\n", *shared_memory);
    }


    shmdt(shared_memory);

    return 0;
}
```

Exercice 4.
Écrire deux programmes : l'un créé une zone mémoire partagée (un bool) , l'initialise à false, et attend qu'elle soit égale à true. L'autre programme utilise la zone de mémoire partagée et écrit la valeur true. Cela doit avoir pour effet de débloquer le premier processus.

Le premier programme :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```c
int main() {
    key_t key = ftok("/etc/passwd", 66);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Obtenir l'ID de la mémoire partagée
    int shmid = shmget(key, sizeof(bool), 0666 | IPC_CREAT);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attacher la mémoire partagée
    bool *shared_memory = (bool *)shmat(shmid, NULL, 0);


    *shared_memory=false;
    while (!(*shared_memory)) {
        usleep(10);
    }

    shmdt(shared_memory);
    shmctl(shmid, IPC_RMID, NULL);
    printf("Le premier programme a terminé.\n");
    return 0;
}
```

Le deuxième programme :

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>



int main() {
    key_t key = ftok("/etc/passwd", 66);
    if (key == -1) {
        perror("ftok");
```



```c
        exit(1);
    }

    // Obtenir l'ID de la mémoire partagée
    int shmid = shmget(key, sizeof(bool), 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

    // Attacher la mémoire partagée
    bool *shared_memory = (bool *)shmat(shmid, NULL, 0);

    *shared_memory=true;


    shmdt(shared_memory);

    printf("Le deuxième programme a terminé.\n");
    return 0;
}
```