TP UNIX N°1
BUT - INFO 2°Année

1_ FIFO

En vous aidant du manuel (commande : man), créez deux programme C permettant d'échanger des messages contenus dans un fichier (donné en paramètre à l'un des deux processus) à travers une FIFO. Le premier processus créé la FIFO et ouvre le fichier en lecture pour lire les données et les écrire dans la FIFO. Le second, lit dans la FIFO et affiche à l'écran les données.

Writer.c :

Users > timothebelcour > BUT > Année 2 > S1 > Architecture des système > TP-20250903 > fifo_writer.c > main(void)

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

#define FIFO_PATH "/tmp/fifo"

int main(void) {
    if (mkfifo(FIFO_PATH, 0666) == -1 && errno != EEXIST) {
        perror("mkfifo");
        return 1;
    }
    int fd = open(FIFO_PATH, O_WRONLY);
    if (fd == -1) {
        perror("open ecriture FIFO");
        return 1;
    }
    char buf[512];
    ssize_t n;
    while ((n = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
        if (write(fd, buf, n) == -1) {
            perror("write FIFO");
            break;
        }
    }
    if (n == -1) perror("read stdin");

    close(fd);
    return 0;
}

```

Read.c :

Users > timothebelcour > BUT > Année 2 > S1 > Architecture des système > TP-20250903 > fifo_read.c > main(void)

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

#define FIFO_PATH "/tmp/fifo"

int main(void) {
    if (mkfifo(FIFO_PATH, 0666) == -1 && errno != EEXIST) {
        perror("mkfifo");
        return 1;
    }
    int fd = open(FIFO_PATH, O_RDONLY);
    if (fd == -1) {
        perror("open lecture FIFO");
        return 1;
    }
    char buf[512];
    ssize_t n;
    while ((n = read(fd, buf, sizeof(buf))) > 0) {
        if (write(STDOUT_FILENO, buf, n) == -1) {
            perror("write stdout");
            break;
        }
    }
    if (n == -1) perror("read FIFO");

    close(fd);
    return 0;
}

```

Tout d'abord on compile le fichier fifo_writer.c pour le rendre en exécutable nommé fifo_writer.
Puis on exécute le programme, le résultat est qu'il attend une entrée de l'utilisateur. Je tape « bonjour ». Le texte est envoyé dans la FIFO.

```
timothebelcour@MacBook-Air TP-20250903 % gcc fifo_writer.c -o fifo_writer
timothebelcour@MacBook-Air TP-20250903 % ./fifo_writer
bonjour
```

Tout d'abord on compile le fichier fifo_read.c pour le rendre en exécutable nommé fifo_read.
Puis on exécute le programme, le résultat est que le programme lit le contenu de la FIFO et affiche « bonjour » (message envoyé au préalable par fifo_writer).

```
timothebelcour@MacBook-Air TP-20250903 % gcc fifo_read.c -o fifo_read
timothebelcour@MacBook-Air TP-20250903 % ./fifo_read

bonjour
```

Pour conclure :
fifo_writer : transmet l'entrée utilisateur dans la FIFO.
fifo_read : lit et affiche ce qui a été écrit.

Je n'ai pas pu exécuter les commandes Linux des exercices 2 et 3, ayant récemment changé d'ordinateur et oublié de réinstaller Linux. Je pensais que le terminal macOS suffirait. Je m'en excuse.

2_ Les queues de message

Send.c

Users > timothebelcour > BUT > Année 2 > S1 > Architecture des système > TP-20250903 > send.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>

#define QUEUE_NAME "/ma_file"

int main() {
    mqd_t mq = mq_open(QUEUE_NAME, O_CREAT | O_WRONLY, 0666, NULL);
    if (mq == (mqd_t)-1) {
        perror("Erreur ouverture de la file de messages");
        exit(EXIT_FAILURE);
    }

    char message[64];
    for (int i = 1; i <= 10; ++i) {
        snprintf(message, sizeof(message), "Message %d", i);
        if (mq_send(mq, message, strlen(message) + 1, 0) == -1) {
            perror("Erreur envoi message");
            mq_close(mq);
            exit(EXIT_FAILURE);
        }
    }
    mq_close(mq);
    return 0;
}

```

Receive.c :

Users > timothebelcour > BUT > Année 2 > S1 > Architecture des système > TP-20250903 > receive.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>

#define QUEUE_NAME "/ma_file"

int main() {
    mqd_t mq = mq_open(QUEUE_NAME, O_CREAT | O_RDONLY, 0666, NULL);
    if (mq == (mqd_t)-1) {
        perror("Erreur ouverture de la file de messages");
        exit(EXIT_FAILURE);
    }
    char buffer[64];
    unsigned int prio;
    for (int i = 1; i <= 10; ++i) {
        ssize_t bytes_read = mq_receive(mq, buffer, sizeof(buffer), &prio);
        if (bytes_read >= 0) {
            buffer[bytes_read] = '\0';
            printf("Reçu: %s (prio=%u)\n", buffer, prio);
        } else {
            perror("Erreur réception message");
            mq_close(mq);
            mq_unlink(QUEUE_NAME);
            exit(EXIT_FAILURE);
        }
    }
    mq_close(mq);
    mq_unlink(QUEUE_NAME);
    return 0;
}

```

3_ Le système de fichiers /proc

cpuinfo → infos détaillées sur le processeur (modèle, fréquence, nombre de cœurs, flags supportés…).
devices → liste des périphériques reconnus par le noyau.
dma → canaux DMA utilisés par les périphériques.
filesystems → types de systèmes de fichiers supportés par le noyau.
interrupts → table des interruptions, montre quelles IRQ sont utilisées par quels périphériques.
partitions → liste des partitions détectées par le noyau.
meminfo → état mémoire : RAM totale, libre, buffers, swap…
modules → liste des modules noyau actuellement chargés.
uptime → temps écoulé depuis le démarrage du système.
version → version du noyau Linux en cours d'exécution.

Les répertoires dont le nom est un **numéro** correspondent aux **PID**
Chaque PID contient un sous-dossier avec des infos spécifiques au processus.

cmdline → la commande exacte utilisée pour lancer le processus.
cwd → lien symbolique vers le répertoire courant du processus.
environ → variables d'environnement du processus.
exe → lien vers l'exécutable du processus.
fd → répertoire listant tous les descripteurs de fichiers ouverts par le processus.
maps → carte mémoire du processus.