# 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 :

```c
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc != 3){
        fprintf(stderr, "usage: %s <fifo_path> <input_file>\n", argv[0]);
        return 1;
    }
    const char *fifo = argv[1];
    const char *in   = argv[2];
    if (mkfifo(fifo, 0666) == -1 && errno != EEXIST){
        perror("mkfifo"); return 2;
    }
    FILE *fin = fopen(in, "r");
    if(!fin){ perror("fopen(input)"); return 3; }
    int fd = open(fifo, O_WRONLY);
    if(fd == -1){ perror("open(fifo, O_WRONLY)"); fclose(fin); return 4; }
    char buf[4096];
    ssize_t n;
    while ((n = fread(buf, 1, sizeof(buf), fin)) > 0){
        ssize_t w = write(fd, buf, (size_t)n);
        if(w < 0){ perror("write"); close(fd); fclose(fin); return 5; }
    }
    close(fd);
    fclose(fin);
    return 0;
}
```

Reader.c :

```c
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv){
    if(argc != 2){
        fprintf(stderr, "usage: %s <fifo_path>\n", argv[0]);
        return 1;
    }
    const char *fifo = argv[1];
    mkfifo(fifo, 0666);
    int fd = open(fifo, O_RDONLY);
    if(fd == -1){ perror("open(fifo, O_RDONLY)"); return 2; }
    char buf[4096];
    ssize_t n;
    while((n = read(fd, buf, sizeof(buf))) > 0){
        fwrite(buf, 1, (size_t)n, stdout);
    }
    close(fd);
    return 0;
}
```

```
[timothebelcour@mac tp-fifo % echo "Bonjour depuis le programme fifo_writer" > input.txt
[timothebelcour@mac tp-fifo % cat input.txt
 Bonjour depuis le programme fifo_writer
[timothebelcour@mac tp-fifo % ./writer /tmp/tp_fifo input.txt
 timothebelcour@mac tp-fifo %
```

Writer : crée la FIFO si besoin, ouvre le fichier en lecture, et écrit son contenu dans la FIFO.

```
[timothebelcour@mac tp-fifo % ./reader /tmp/tp_fifo
 Bonjour depuis le programme fifo_writer
[timothebelcour@mac tp-fifo % ls -l /tmp/tp_fifo
 prw-r--r--  1 timothebelcour  wheel  0 14 oct.  09:01 /tmp/tp_fifo
[timothebelcour@mac tp-fifo % rm /tmp/tp_fifo
 timothebelcour@mac tp-fifo %
```

Reader : ouvre la FIFO en lecture et affiche tout ce qu'il reçoit.
Le Terminal A affiche exactement le contenu de input.txt. La FIFO /tmp/tp_fifo est visible avec ls -l.

## 2_ Les queues de message

mq_send10.c :

```
  GNU nano 7.2                                          mq_send10.c *
#define _XOPEN_SOURCE 700
#include <mqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>

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

    const char *qname = argv[1];
    struct mq_attr attr = {0};
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = 256;
    mqd_t mq = mq_open(qname, O_CREAT | O_WRONLY, 0666, &attr);
    if (mq == (mqd_t)-1) {
        perror("mq_open");
        return 2;
    }
    char msg[256];
    for (unsigned i = 0; i < 10; ++i) {
        snprintf(msg, sizeof msg, "Message %u", i + 1);
```

```
    return 1;
    }

    const char *qname = argv[1];
    struct mq_attr attr = {0};
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = 256;
    mqd_t mq = mq_open(qname, O_CREAT | O_WRONLY, 0666, &attr);
    if (mq == (mqd_t)-1) {
        perror("mq_open");
        return 2;
    }
    char msg[256];
    for (unsigned i = 0; i < 10; ++i) {
        snprintf(msg, sizeof msg, "Message %u", i + 1);
        if (mq_send(mq, msg, strlen(msg) + 1, i) == -1) {
            perror("mq_send");
            mq_close(mq);
            return 3;
        }
    }

    mq_close(mq);
    return 0;
}
```

mq_recv10.c :

```
  GNU nano 7.2                                          mq_recv10.c *
#define _XOPEN_SOURCE 700
#include <mqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "usage: %s </queue_name>\n", argv[0]);
        return 1;
    }
    const char *qname = argv[1];
    mqd_t mq = mq_open(qname, O_RDONLY);
    if (mq == (mqd_t)-1) {
        perror("mq_open");
        return 2;
    }

    struct mq_attr attr;
    if (mq_getattr(mq, &attr) == -1) {
        perror("mq_getattr");
        mq_close(mq);
        return 3;
    }
```

```
 GNU nano 7.2                                                mq_recv10.c *
        mq_close(mq);
        return 3;
    }

    char *buf = malloc((size_t)attr.mq_msgsize);
    if (!buf) {
        perror("malloc");
        mq_close(mq);
        return 4;
    }
    for (int i = 0; i < 10; ++i) {
        unsigned prio = 0;
        ssize_t n = mq_receive(mq, buf, (size_t)attr.mq_msgsize, &prio);
        if (n == -1) {
            perror("mq_receive");
            free(buf);
            mq_close(mq);
            return 5;
        }
        printf("[prio=%u] %s\n", prio, buf);
    }

    free(buf);
    mq_close(mq);
    mq_unlink(qname);
    return 0;
}
```

1er Terminal :

```
belcour@belcour-QEMU-Virtual-Machine:~$ mkdir -p ~/tp-mqueue && cd ~/tp-mqueue
belcour@belcour-QEMU-Virtual-Machine:~/tp-mqueue$ nano mq_send10.c
belcour@belcour-QEMU-Virtual-Machine:~/tp-mqueue$ nano mq_recv10.c
belcour@belcour-QEMU-Virtual-Machine:~/tp-mqueue$ gcc -Wall -Wextra -O2 mq_send10.c -o mq_send10 -lrt
gcc -Wall -Wextra -O2 mq_recv10.c -o mq_recv10 -lrt
belcour@belcour-QEMU-Virtual-Machine:~/tp-mqueue$ ./mq_send10 /tp_mq
```

mq_send10.c crée la file de messages /tp_mq et envoie 10 messages de priorités différentes (0 à 9).

2ème Terminal :

```
belcour@belcour-QEMU-Virtual-Machine:~/tp-mqueue$ ./mq_recv10 /tp_mq
[prio=9] Message 10
[prio=8] Message 9
[prio=7] Message 8
[prio=6] Message 7
[prio=5] Message 6
[prio=4] Message 5
[prio=3] Message 4
[prio=2] Message 3
[prio=1] Message 2
[prio=0] Message 1
```

mq_recv10.c ouvre la file, lit les 10 messages et les affiche avec leur priorité.

La file est ensuite supprimée automatiquement (mq_unlink) à la fin du programme.

L'affichage montre que la réception se fait selon la priorité décroissante, prouvant le bon fonctionnement de l'API POSIX mqueue.

## 3_ Le système de fichiers /proc

```
belcour@belcour-QEMU-Virtual-Machine:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1937208k,nr_inodes=484302,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=399316k,mode=755,inode64)
/dev/vda2 on / type ext4 (rw,relatime)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
efivarfs on /sys/firmware/efi/efivars type efivarfs (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=32,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe
_ino=7224)
belcour@belcour-QEMU-Virtual-Machine:~$ mount | grep " on /proc "
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

La commande mount montre que le système de fichiers proc est monté sur /proc. Cela prouve que /proc est un pseudo-système géré directement par le noyau.

J'ai consulté ces fichiers à l'aide des commandes cat et head pour en afficher les premières lignes.

```
belcour@belcour-QEMU-Virtual-Machine:~$ head -n 10 /proc/cpuinfo
cat /proc/filesystems
head -n 10 /proc/meminfo
cat /proc/uptime
cat /proc/version
processor       : 0
BogoMIPS        : 48.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid asimdrdm jscvt fcma lrcpc dcpop
 sha3 asimddp sha512 asimdfhm dit uscat ilrcpc flagm sb paca pacg dcpodp flagm2 frint bf16 afp
CPU implementer : 0x61
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0x000
CPU revision    : 0

processor       : 1
nodev   sysfs
nodev   tmpfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cgroup2
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   bpf
nodev   pipefs
nodev   ramfs
nodev   hugetlbfs
```

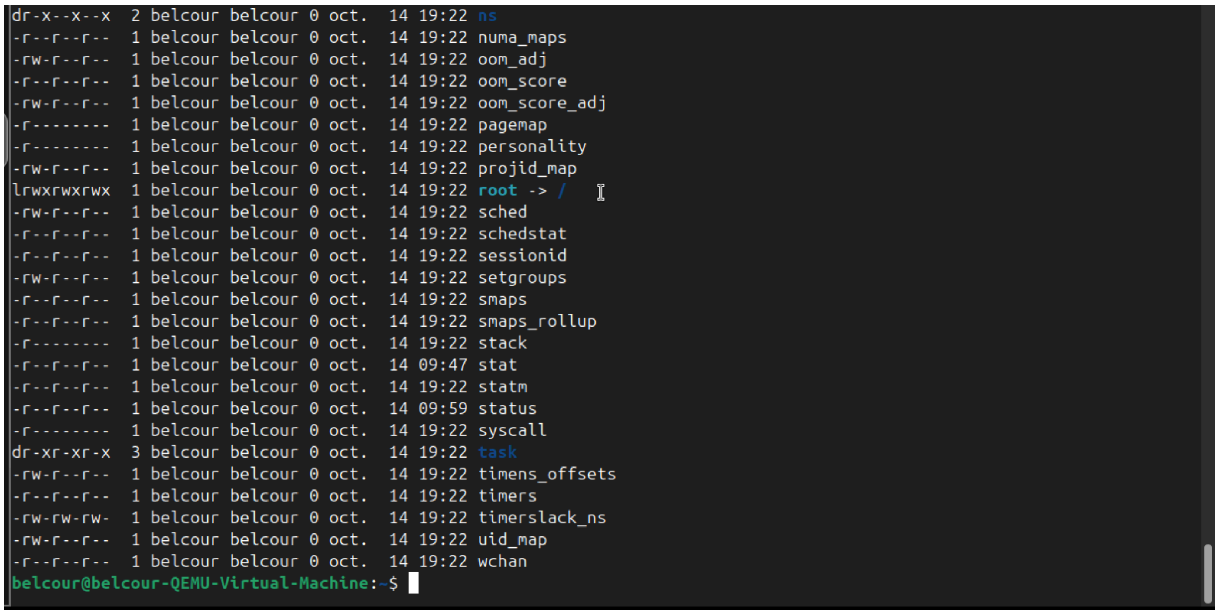
```
nodev   hugetlbfs
nodev   devpts
        ext3
        ext2
        ext4
        squashfs
        vfat
nodev   ecryptfs
        fuseblk
nodev   fuse
nodev   fusectl
nodev   efivarfs
nodev   mqueue
nodev   pstore
nodev   autofs
nodev   binfmt_misc
MemTotal:        3993144 kB
MemFree:          808612 kB
MemAvailable:    2908792 kB
Buffers:           68152 kB
Cached:          2214236 kB
SwapCached:            0 kB
Active:          1597532 kB
Inactive:        1256932 kB
Active(anon):     662656 kB
Inactive(anon):        0 kB
13148.96 51890.57
Linux version 6.14.0-33-generic (buildd@bos03-arm64-083) (aarch64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.
0, GNU ld (GNU Binutils for Ubuntu) 2.42) #33~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Sep 19 16:19:58 UTC 2
belcour@belcour-QEMU-Virtual-Machine:~$
```

Ces fichiers m'ont permis de constater que les informations qu'ils contiennent (CPU, mémoire, modules, uptime, version du noyau) changent en temps réel et ne sont pas stockées sur le disque.

Pour analyser un processus précis, j'ai lancé mon programme mq_recv10 puis recherché son PID avec :

```
belcour@belcour-QEMU-Virtual-Machine:~$ ps -efa | grep mq_recv10
belcour      7168    2571  0 18:06 pts/0    00:00:00 grep --color=auto mq_recv10
```

```
belcour@belcour-QEMU-Virtual-Machine:~$ ls -l /proc/2571
total 0
dr-xr-xr-x  2 belcour belcour 0 oct.  14 19:22 attr
-rw-r--r--  1 belcour belcour 0 oct.  14 19:22 autogroup
-r--------  1 belcour belcour 0 oct.  14 19:22 auxv
-r--r--r--  1 belcour belcour 0 oct.  14 09:59 cgroup
--w-------  1 belcour belcour 0 oct.  14 19:22 clear_refs
-r--r--r--  1 belcour belcour 0 oct.  14 09:59 cmdline
-rw-r--r--  1 belcour belcour 0 oct.  14 19:22 comm
-rw-r--r--  1 belcour belcour 0 oct.  14 19:22 coredump_filter
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 cpuset
lrwxrwxrwx  1 belcour belcour 0 oct.  14 19:22 cwd -> /home/belcour
-r--------  1 belcour belcour 0 oct.  14 18:06 environ
lrwxrwxrwx  1 belcour belcour 0 oct.  14 19:22 exe -> /usr/bin/bash
dr-x------  2 belcour belcour 4 oct.  14 19:22 fd
dr-xr-xr-x  2 belcour belcour 0 oct.  14 19:22 fdinfo
-rw-r--r--  1 belcour belcour 0 oct.  14 19:22 gid_map
-r--------  1 belcour belcour 0 oct.  14 19:22 io
-r--------  1 belcour belcour 0 oct.  14 19:22 ksm_merging_pages
-r--------  1 belcour belcour 0 oct.  14 19:22 ksm_stat
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 latency
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 limits
-rw-r--r--  1 belcour belcour 0 oct.  14 19:22 loginuid
dr-x------  2 belcour belcour 0 oct.  14 19:22 map_files
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 maps
-rw-------  1 belcour belcour 0 oct.  14 19:22 mem
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 mountinfo
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 mounts
-r--------  1 belcour belcour 0 oct.  14 19:22 mountstats
dr-xr-xr-x 54 belcour belcour 0 oct.  14 19:22 net
dr-x--x--x  2 belcour belcour 0 oct.  14 19:22 ns
-r--r--r--  1 belcour belcour 0 oct.  14 19:22 numa_maps
```




Ce dossier contient plusieurs fichiers et liens symboliques :
**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.
Cela montre que /proc permet d'explorer en direct tous les détails internes d'un processus actif.

**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.