

Jeu de test

1- Compilation :

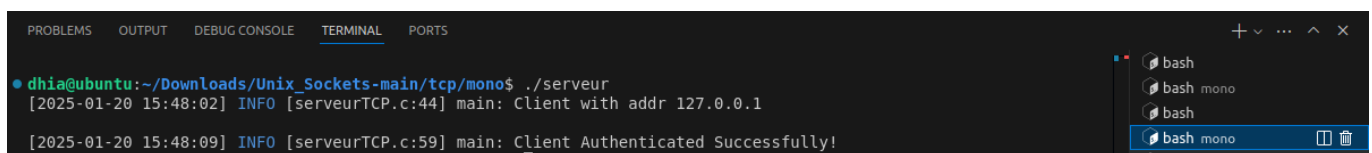
Pour chaque mode et pour chaque aspect du mode TCP, nous avons compilé les fichiers `make.sh` de la manière suivante :

```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$ chmod +x make.sh
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$ ./make.sh server
Target server built.
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$ ./make.sh client
Target client built.
```

2- Exécution (mode TCP) :

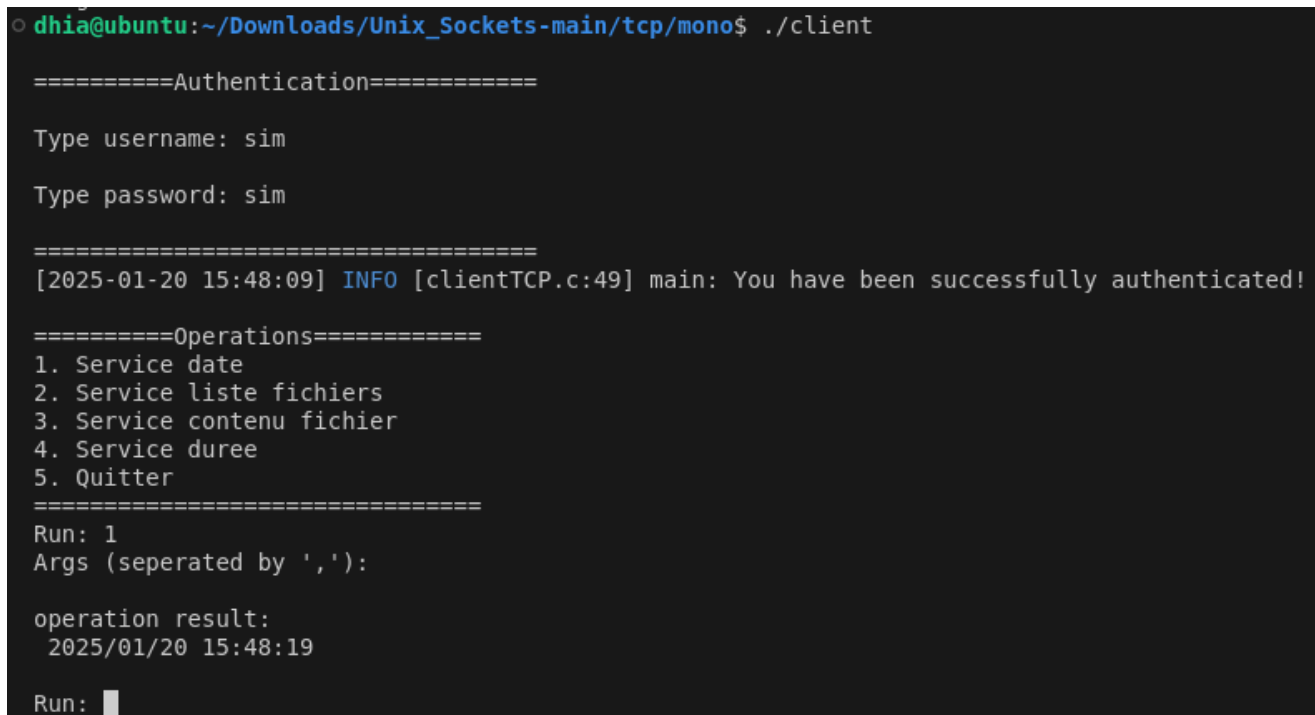
2.1- Monoclient/monoserveur :

Tout d'abord, nous avons exécuté le serveur :



```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$ ./serveur
[2025-01-20 15:48:02] INFO [serveurTCP.c:44] main: Client with addr 127.0.0.1
[2025-01-20 15:48:09] INFO [serveurTCP.c:59] main: Client Authenticated Successfully!
```

Après avoir exécuter le serveur avec succès, nous passons à exécuter le client qui doit etre s'authentifier, puis lancer la demande selon un choix souhaité.



```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$ ./client

=====Authentication=====

Type username: sim
Type password: sim

=====
[2025-01-20 15:48:09] INFO [clientTCP.c:49] main: You have been successfully authenticated!

=====Operations=====
1. Service date
2. Service liste fichiers
3. Service contenu fichier
4. Service duree
5. Quitter
=====
Run: 1
Args (seperated by ','):

operation result:
2025/01/20 15:48:19

Run: █
```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: 2
Args (seperated by ','): utils

operation result:
utils.c
utils.h

Run: 3
Ln 3, Col 26 Spaces: 4 UTF-8 LF Markdown

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: 3
Args (seperated by ','): utils/utils.h

operation result:
#ifndef UTILS_H
#define UTILS_H
#include "common.h"

size_t get_args_from_buff(int service_fd, int sock, char *buff, char *args[], size_t args_size);
int safe_socket(void);
void safe_listen(int sock, int max_clients);
void safe_bind(int sock, struct sockaddr_in* server_addr);

int safe_accept(int sock, struct sockaddr_in *client_addr, socklen_t *client_addr_len);
void safe_setsockopt(int sock);

void safe_send(int service_fd, int sock, msg *message);
void safe_rcv(int service_fd, int sock, msg *message);
void concat_args(char *buff, char *args[], size_t args_size);
#endif
Ln 3, Col 26 Spaces: 4 UTF-8 LF Markdown

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Run: 4
Args (seperated by ','): 284.665794s

operation result:
284.665794s
Run: 5
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/mono$
Ln 3, Col 26 Spaces: 4 UTF-8 LF Markdown

```

2.2- Multiclients/monoserveur :

Après avoir exécuter le serveur et 2 clients qui sont authentifiés avec succès, le serveur indique les clients qui sont connectés :

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi_mono$ ./serveur
[2025-01-20 16:02:43] INFO [serveurTCP.c:116] main: Active clients: 1
[2025-01-20 16:02:43] INFO [serveurTCP.c:119] main: Client with addr 127.0.0.1 connected

[2025-01-20 16:03:22] INFO [serveurTCP.c:56] connection_handler: Client Authenticated Successfully!
[2025-01-20 16:07:16] INFO [serveurTCP.c:116] main: Active clients: 2
[2025-01-20 16:07:16] INFO [serveurTCP.c:119] main: Client with addr 127.0.0.1 connected

[2025-01-20 16:07:20] INFO [serveurTCP.c:56] connection_handler: Client Authenticated Successfully!
Ln 19, Col 3 Spaces: 4 UTF-8 LF Shell Script

```

Chaque client peut alors demander le service à traiter

2.3- Multiclients/multiserveurs :

Le modèle multiclients-multiserveurs repose sur la communication entre plusieurs clients et plusieurs serveurs via un serveur intermédiaire, souvent appelé proxy;

Nous avons alors exécuter tout d'abord le proxy puis les serveurs qui permet chacun d'entre eux de traiter un service(auth,duree,etc..),enfin nous avons exécuté les clients.

Les clients veulent connaitre leur durées de connexion :

Client 1:

```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./client

=====Authentication=====

Type username: sim

Type password: sim

=====
[2025-01-20 16:15:58] INFO [clientTCP.c:53] main: You have been successfully authenticated!

=====Operations=====
1. Service date
2. Service liste fichiers
3. Service contenu fichier
4. Service duree
5. Quitter
=====
Run: 4
Args (seperated by ','):

operation result:
251.446545s
Run: 
```

Client 2:

```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./client

=====Authentication=====

Type username: sim

Type password: sim

=====
[2025-01-20 16:19:14] INFO [clientTCP.c:53] main: You have been successfully authenticated!

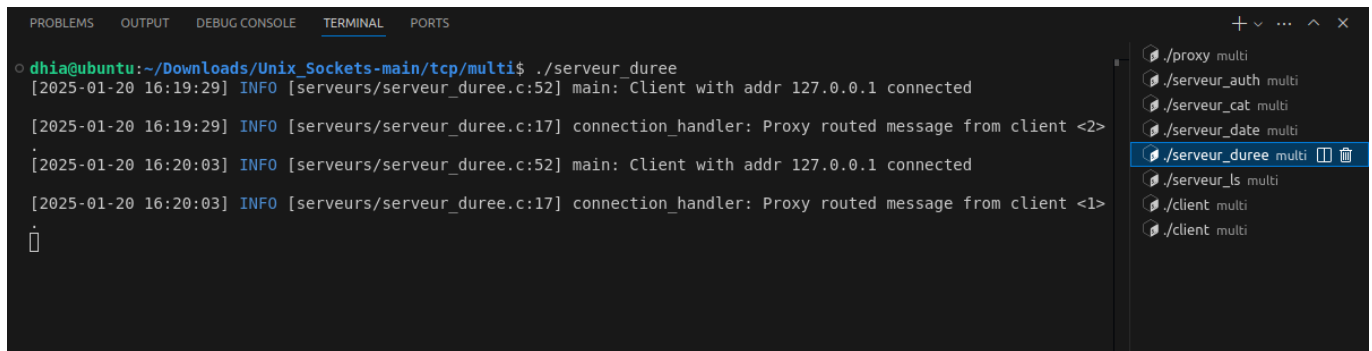
=====Operations=====
1. Service date
2. Service liste fichiers
3. Service contenu fichier
4. Service duree
5. Quitter
=====
Run: 4
Args (seperated by ','):

operation result:
18.448219s
Run: 
```

Proxy:

```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ chmod +x make.sh
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./make.sh server
Target server built.
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./make.sh client
Target client built.
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./proxy
[2025-01-20 16:15:52] INFO [proxy.c:76] connection handler: Client with id <1> connected.
[2025-01-20 16:15:58] INFO [proxy.c:22] route connection: Routing to Auth server
[2025-01-20 16:19:10] INFO [proxy.c:76] connection handler: Client with id <2> connected.
[2025-01-20 16:19:14] INFO [proxy.c:22] route connection: Routing to Auth server
[2025-01-20 16:19:29] INFO [proxy.c:51] route connection: Routing to duree server
[2025-01-20 16:20:03] INFO [proxy.c:51] route connection: Routing to duree server
```

serveur_duree:



The image shows a terminal window with the following output:

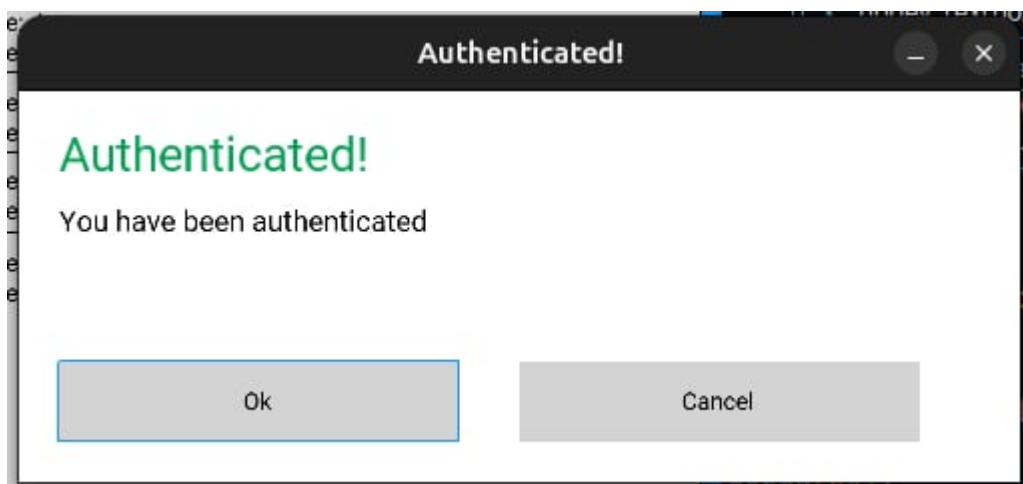
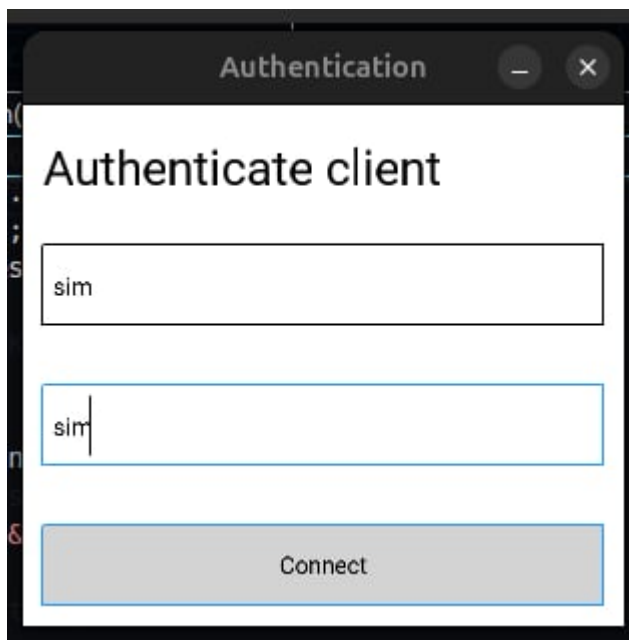
```
dhia@ubuntu:~/Downloads/Unix_Sockets-main/tcp/multi$ ./serveur_duree
[2025-01-20 16:19:29] INFO [serveurs/serveur_duree.c:52] main: Client with addr 127.0.0.1 connected
[2025-01-20 16:19:29] INFO [serveurs/serveur_duree.c:17] connection_handler: Proxy routed message from client <2>
[2025-01-20 16:20:03] INFO [serveurs/serveur_duree.c:52] main: Client with addr 127.0.0.1 connected
[2025-01-20 16:20:03] INFO [serveurs/serveur_duree.c:17] connection_handler: Proxy routed message from client <1>
.
```

To the right of the terminal, a list of running processes is shown:

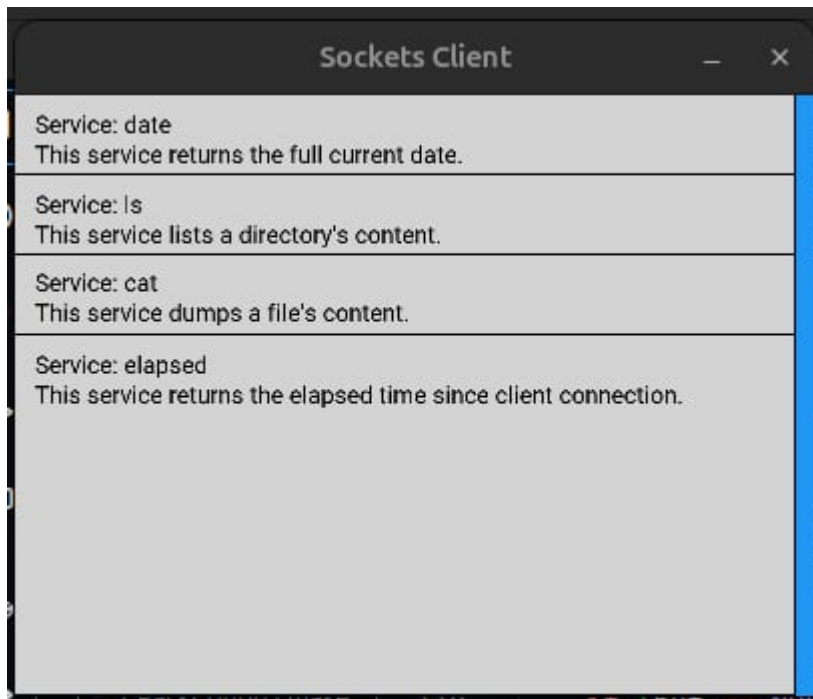
- ./proxy multi
- ./serveur_auth multi
- ./serveur_cat multi
- ./serveur_date multi
- ./serveur_duree multi** (highlighted)
- ./serveur_ls multi
- ./client multi
- ./client multi

2.4- Interface graphique :

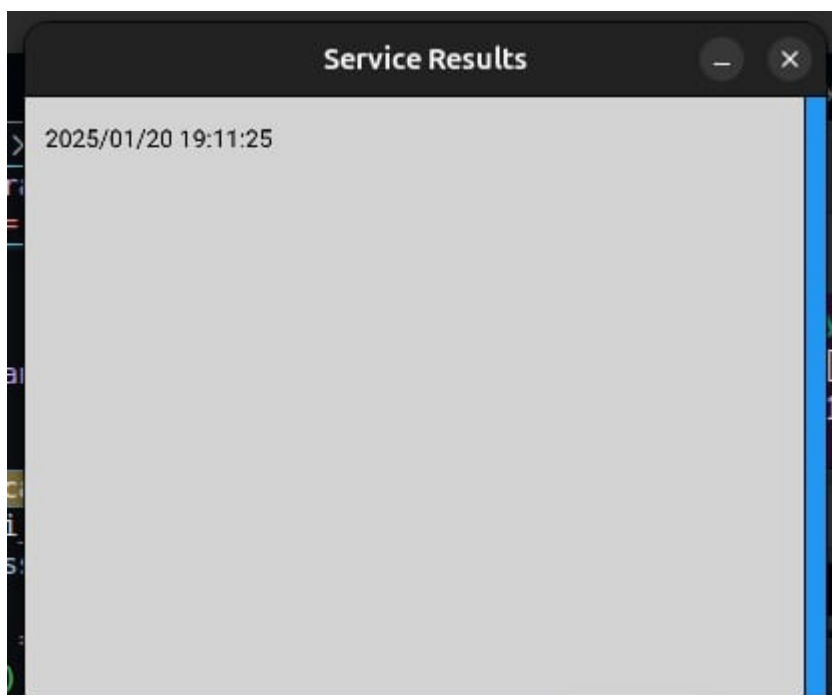
Authentication:



Services :

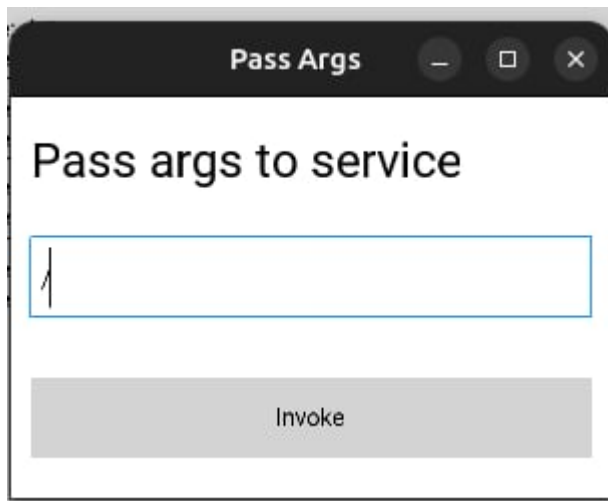


Service_date:



Service_liste_fichiers:

Le client passe en arguments le chemin dans-lequel se trouvent les fichiers



A dialog box titled "Pass Args" with a dark header bar containing standard window controls. The main area has the title "Pass args to service" and a text input field with a cursor. Below the input field is a grey button labeled "Invoke".

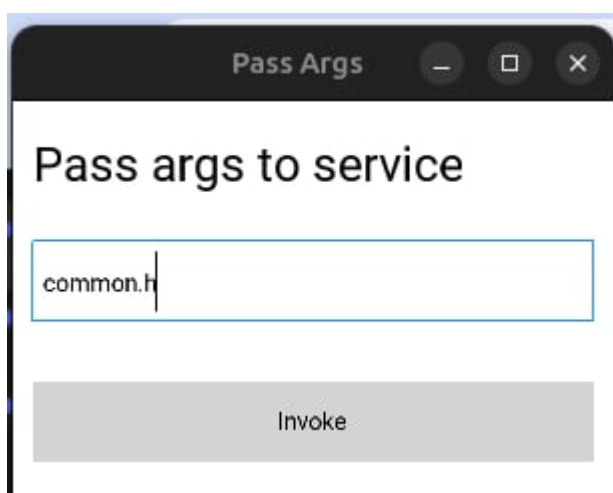


A window titled "Service Results" with a dark header bar. It displays a list of file system paths in a table-like structure. A blue vertical bar is on the right side.

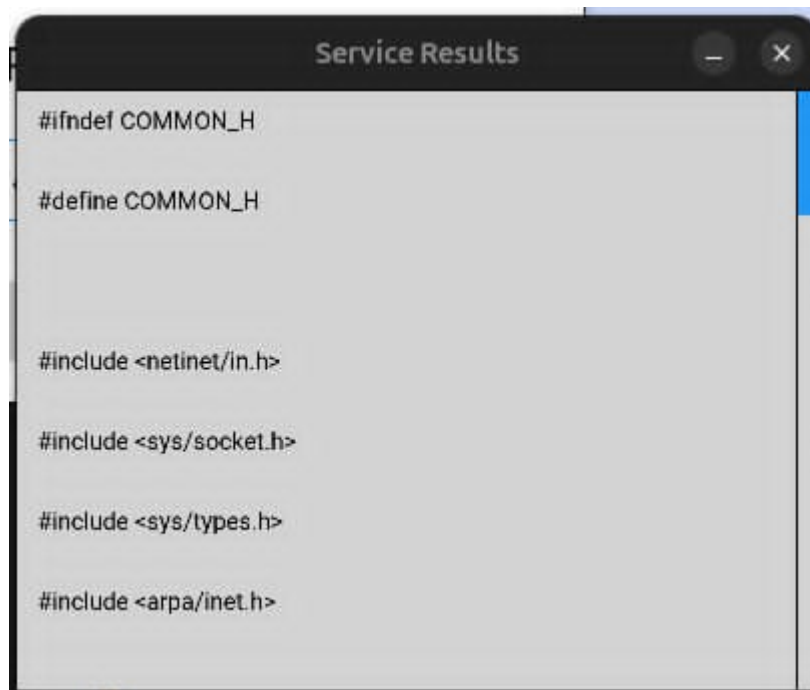
bin
bin usr-is-merged
boot
cdrom
dev
etc
home

Service_contenu_fichier :

Le client passe en arguments le fichier afin d'afficher son contenu



The same "Pass Args" dialog box as above, but the text input field now contains the text "common.h". The "Invoke" button remains below it.



A screenshot of a window titled "Service Results". The window has a dark header bar with the title and standard window controls (minimize, maximize, close). The main content area is light gray and contains the following C preprocessor directives:

```
#ifndef COMMON_H  
  
#define COMMON_H  
  
#include <netinet/in.h>  
  
#include <sys/socket.h>  
  
#include <sys/types.h>  
  
#include <arpa/inet.h>
```

Service_durée_connexion:



A screenshot of a window titled "Service Results". The window has a dark header bar with the title and standard window controls (minimize, maximize, close). The main content area is light gray and displays the following text:

```
444.410830s
```