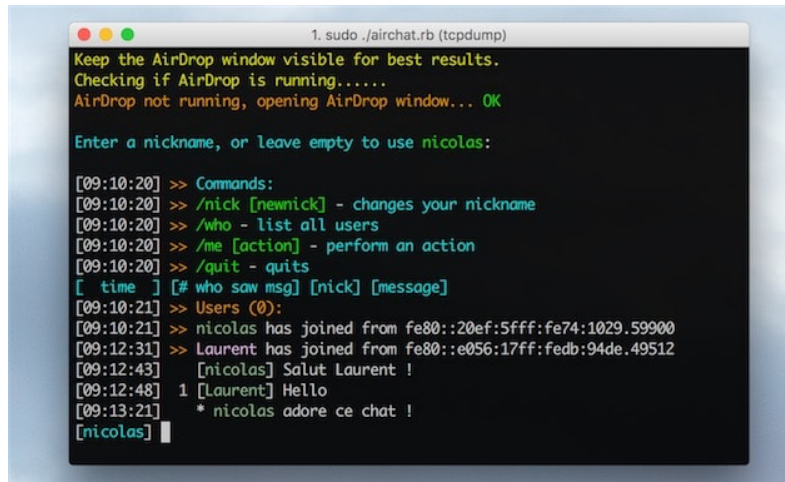


# MI207 - Projet



```
1. sudo ./airchat.rb (tcpdump)
Keep the AirDrop window visible for best results.
Checking if AirDrop is running.....
AirDrop not running, opening AirDrop window... OK

Enter a nickname, or leave empty to use nicolas:

[09:10:20] >> Commands:
[09:10:20] >> /nick [newnick] - changes your nickname
[09:10:20] >> /who - list all users
[09:10:20] >> /me [action] - perform an action
[09:10:20] >> /quit - quits
[ time ] [# who saw msg] [nick] [message]
[09:10:21] >> Users (0):
[09:10:21] >> nicolas has joined from fe80::20ef:5fff:fe74:1029.59900
[09:12:31] >> Laurent has joined from fe80::e056:17ff:fedb:94de.49512
[09:12:43] [nicolas] Salut Laurent !
[09:12:48] 1 [Laurent] Hello
[09:13:21] * nicolas adore ce chat !
[nicolas]
```

## Introduction

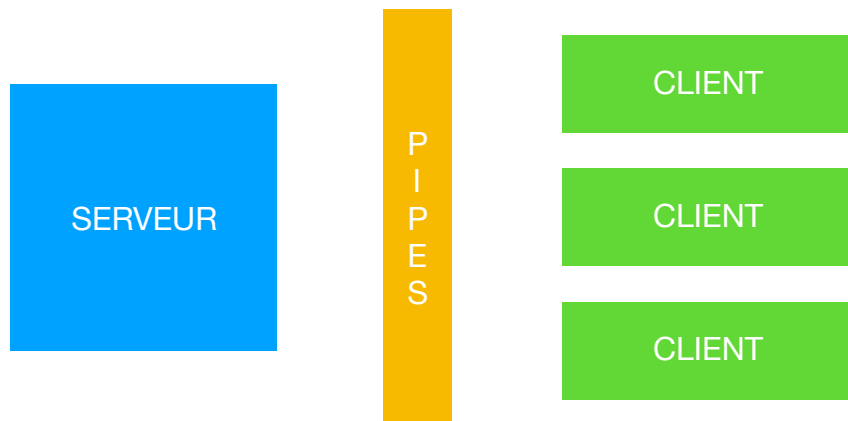
L'objectif de ce projet est d'implémenter un **chat**. Le projet sera basé sur le principe du client/"serveur". Il y aura ainsi :

- un processus par **client** souhaitant utiliser le chat,
- un processus comme contrôleur/répartiteur (baptisé par abus de langage **serveur**) pour rediriger les communications entre les différents clients.

Pour cela, vous mettrez en oeuvre les différentes solutions de communication vues en cours afin de faire dialoguer ensemble client(s) et serveur. Ainsi, l'objectif principal de ce projet est de mettre en évidence les difficultés d'écriture d'un code système correct/propre.

La propreté du comportement (appels système testés, fichiers nettoyés, etc.) de votre projet sera prise en compte dans la notation du projet.

# Architecture



L'architecture proposée est la suivante :

- Un tube nommé (0), crée par le serveur et partagé par tous les clients, pour envoyer un message. Le format des messages est spécifié dans la partie protocole (ci-dessous),
- Un tube nommé, crée par chaque client, pour recevoir les messages du serveur (au final des autres clients).

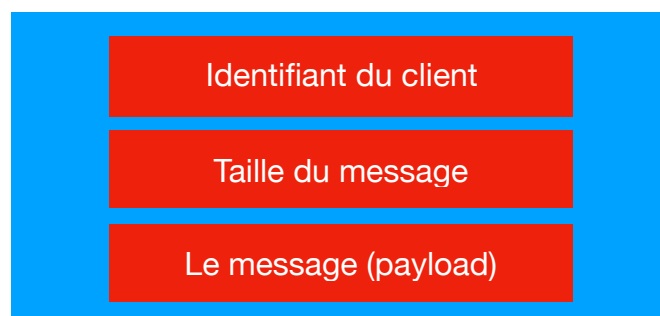
Chaque client lors de son démarrage se choisira un identifiant parmi les non utilisés et créera son tube. Il notifie le serveur de son existence en envoyant un message `HELLO`. Le serveur vient ensuite se connecter sur le tube du client.

Tous les fichiers seront créés dans un dossier (éventuellement configurable via la ligne de commande), par exemple `/tmp/chat`.

Les questions à vous poser sont (par exemple) les suivantes :

- Comment garantir que l'identifiant choisi est réellement non utilisé ? (atomicité)
- Dans quels cas y a-t-il (risque) inter-blocage.
- Comment terminer proprement un client.
- ...

## Protocole



Nous définissons un message spécial : `HELLO`. Si un message a une taille nulle, on considère qu'il s'agit d'un message spécial de bienvenue. Le contrôleur essaiera de se connecter au tube correspondant.

# Paliers intermédiaires

## A

Un exécutable pour le contrôleur, un exécutable pour le client. Il ne doit pas y avoir d'inter-leaving dans les messages reçus.

## B

Le client déclenchera le contrôleur, s'il n'existe pas déjà. Le faire fonctionner dans tous les cas peut s'avérer compliqué...

## C

Pour terminer le contrôleur, on se propose de lui envoyer le signal `SIGINT`.

## D

Le contrôleur en tâche de fond : Essayer de lancer le contrôleur en tâche de fond (daemon).

## E

Des threads : On pourra utiliser des `*threads*` à la place des processus en interne dans le client.

## F

Un protocole un peu plus complexe : Vous pouvez étendre le protocole afin qu'il fasse ce qui vous paraît bien. Probablement l'idée la plus sympa serait de rajouter la notion de canal, seuls les membres d'un canal voient les messages sur ce canal. Le canal zéro peut représenter le `*broadcast*`.

## G

Poste-restante : Utilisation de files de messages pour faire un système de poste restante ...

# Rendu

- Votre code (via votre dépôt `git`). Le code devra obligatoirement contenir un `Makefile` valide qui construira **TOUS** les exécutables de votre projet,
- Un README qui présente brièvement comment compiler, lancer et utiliser votre projet,
- Un rapport PDF de quelques pages sur les fonctionnalités gérées par votre implémentation, les fonctionnalités non gérées (les cas pathologiques) et le pourquoi vous n'avez pas réussi (le cas échéant).

`guillaume.delbergue@bordeaux-inp.fr`