# Algorithme du système de chat client-serveur en C

## Architecture générale

Le système est composé de :

- \*\*Serveur\*\* (`chatd.c`) : Accepte les connexions et gère la communication

- \*\*Client\*\* (`chatc.c`) : Se connecte au serveur et participe au chat

- \*\*Bibliothèque commune\*\* (`chats.c` + `chats.h`) : Fonctions partagées

---

## ALGORITHME DU SERVEUR (`chatd.c`)

### Fonction principale `main()`

```

DÉBUT main(argc, argv)

VARIABLES:

pcc, pce : entiers (descripteurs de socket)

Nport : entier global (numéro de port)

adr\_L : chaîne globale (adresse IP optionnelle)

SI (argc < 2 OU argc > 3) ALORS

AFFICHER "Usage: programme <port> [<adresse>]"

TERMINER programme

FIN SI

Nport ← atoi(argv[1])

SI (argc == 3) ALORS

adr\_L ← argv[2]

FIN SI

pce ← construirePointCommunication(Nport, adr\_L, argc)

SI (pce < 0) ALORS

AFFICHER "Problème de création de socket"

TERMINER programme

FIN SI

BOUCLE infinie

pcc ← accepterConnexion(pce)

SI (pcc > 0) ALORS

chater(pcc)

FERMER pcc

FIN SI

FIN BOUCLE

FIN main

```

---

## ALGORITHME DU CLIENT (`chatc.c`)

### Fonction `connecterAuServeur(port, ip)`

```

DÉBUT connecterAuServeur(port, ip)

VARIABLES:

sock : entier

serv\_addr : structure sockaddr\_in

sock ← socket(AF\_INET, SOCK\_STREAM, 0)

SI (sock < 0) ALORS

AFFICHER erreur

RETOURNER -1

FIN SI

serv\_addr.sin\_family ← AF\_INET

serv\_addr.sin\_port ← htons(port)

SI (inet\_pton(AF\_INET, ip, &serv\_addr.sin\_addr) <= 0) ALORS

AFFICHER "Adresse invalide"

FERMER sock

RETOURNER -1

FIN SI

SI (connect(sock, serv\_addr, taille) < 0) ALORS

AFFICHER "Connexion échouée"

FERMER sock

RETOURNER -1

FIN SI

AFFICHER "Connecté au serveur"

RETOURNER sock

FIN connecterAuServeur

```

### Fonction principale du client `main()`

```

DÉBUT main(argc, argv)

SI (argc < 3) ALORS

AFFICHER "Usage: programme <server\_ip> <server\_port>"

TERMINER

FIN SI

sock ← connecterAuServeur(atoi(argv[2]), argv[1])

SI (sock < 0) ALORS

TERMINER avec erreur

FIN SI

chatterAvecServeur(sock)

FERMER sock

AFFICHER "Connexion fermée"

FIN main

```

---

## ALGORITHMES COMMUNS (`chats.c`)

### Fonction `construirePointCommunication(port, ip, n)`

```

DÉBUT construirePointCommunication(port, ip, n)

VARIABLES:

adr : structure sockaddr\_in

desc, err : entiers

INITIALISER adr à zéro

desc ← socket(AF\_INET, SOCK\_STREAM, 0)

SI (desc < 0) ALORS

AFFICHER erreur

RETOURNER -1

FIN SI

adr.sin\_family ← AF\_INET

adr.sin\_port ← htons(port)

SI (n == 3) ALORS // Adresse IP fournie

adr.sin\_addr.s\_addr ← inet\_addr(ip)

SINON

adr.sin\_addr.s\_addr ← INADDR\_ANY // Toutes interfaces

FIN SI

err ← bind(desc, adr, taille)

SI (err != 0) ALORS

AFFICHER erreur de bind

FERMER desc

RETOURNER err

FIN SI

err ← listen(desc, 5) // File d'attente de 5 connexions

SI (err != 0) ALORS

AFFICHER erreur de listen

FERMER desc

RETOURNER err

FIN SI

RETOURNER desc

FIN construirePointCommunication

```

### Fonction `accepterConnexion(sd)`

```

DÉBUT accepterConnexion(sd)

VARIABLES:

adresse\_client : structure sockaddr\_in

client : entier

ls : taille de sockaddr\_in

INITIALISER adresse\_client à zéro

client ← accept(sd, adresse\_client, ls)

SI (client < 0) ALORS

AFFICHER erreur d'acceptation

RETOURNER -1

FIN SI

AFFICHER "Connexion acceptée de [IP:PORT]"

RETOURNER client

FIN accepterConnexion

```

### Fonction `chater(pcc)` - Côté serveur

```

DÉBUT chater(pcc)

VARIABLES:

msge[80], msgr[80] : tableaux de caractères (messages)

ps\_name\_local[15], ps\_name\_remote[15] : pseudonymes

INITIALISER tous les tableaux à zéro

AFFICHER "Entrez votre pseudonyme: "

LIRE ps\_name\_local depuis stdin

ENVOYER ps\_name\_local via pcc

RECEVOIR ps\_name\_remote via pcc

AFFICHER "Pseudonyme de l'autre utilisateur: " + ps\_name\_remote

RÉPÉTER

INITIALISER msge et msgr à zéro

AFFICHER ps\_name\_local + ": "

LIRE msge depuis stdin

ENVOYER msge via pcc

RECEVOIR msgr via pcc

AFFICHER ps\_name\_remote + ": " + msgr

JUSQU'À (msge == "exit" OU msgr == "exit")

FIN chater

```

### Fonction `chatterAvecServeur(sock)` - Côté client

```

DÉBUT chatterAvecServeur(sock)

VARIABLES:

ps\_name\_local[15], ps\_name\_remote[15] : pseudonymes

msge[80], msgr[80] : messages

INITIALISER tous les tableaux à zéro

AFFICHER "Entrez votre pseudonyme: "

LIRE ps\_name\_local depuis stdin

ENVOYER ps\_name\_local via sock

RECEVOIR ps\_name\_remote via sock

AFFICHER "Pseudonyme de l'autre utilisateur: " + ps\_name\_remote

RÉPÉTER

// Le client reçoit d'abord

INITIALISER msgr à zéro

RECEVOIR msgr via sock

AFFICHER ps\_name\_remote + ": " + msgr

SI (msgr == "exit\n") ALORS

SORTIR de la boucle

FIN SI

// Puis le client répond

INITIALISER msge à zéro

AFFICHER ps\_name\_local + ": "

LIRE msge depuis stdin

ENVOYER msge via sock

JUSQU'À (msge == "exit\n")

FIN chatterAvecServeur

```

---

## PROTOCOLE DE COMMUNICATION

### Séquence d'établissement de connexion :

1. \*\*Serveur\*\* : Crée socket → Bind → Listen → Accept

2. \*\*Client\*\* : Crée socket → Connect vers serveur

3. \*\*Échange de pseudonymes\*\* :

- Serveur et Client saisissent leurs pseudonymes

- Échange simultané des pseudonymes

### Séquence de chat :

1. \*\*Serveur\*\* envoie un message

2. \*\*Client\*\* reçoit et affiche le message

3. \*\*Client\*\* envoie une réponse

4. \*\*Serveur\*\* reçoit et affiche la réponse

5. Répétition jusqu'à qu'un des deux tape "exit"

### Terminaison :

- Connexion fermée quand "exit" est tapé

- Le serveur retourne en attente d'une nouvelle connexion

- Le client se termine complètement

---

## CARACTÉRISTIQUES TECHNIQUES

- \*\*Type de socket\*\* : TCP (SOCK\_STREAM)

- \*\*Famille d'adresse\*\* : IPv4 (AF\_INET)

- \*\*Taille des buffers\*\* : 80 caractères pour messages, 15 pour pseudonymes

- \*\*Mode de fonctionnement\*\* : Séquentiel (un client à la fois)

- \*\*Gestion d'erreurs\*\* : Vérification des retours de fonctions système