Une image contenant texte

Description générée automatiquement

2021/2022

Rapport de Mini-projet

Master Spécialisé en Big Data & Cloud Computing

Partage de fichiers entre client et serveur en utilisant le langage C sous linux

Réalisé par : Encadré par :

Tachrimant Saad Ouadoudi Zytoune

Kemmach Asmaa

Toumlal Kaoutar

Salma Er-rouidi

Table des matières

[Introduction 3](#_Toc106577301)

[I- Client-serveur 4](#_Toc106577302)

[1-Définition 4](#_Toc106577303)

[2-Caractéristiques 4](#_Toc106577304)

[3-Fonctionnement 5](#_Toc106577305)

[4-Avantages des architectures centralisées 5](#_Toc106577306)

[5-Inconvénients des architectures centralisées 5](#_Toc106577307)

[6-Exemples client-serveur 5](#_Toc106577308)

[II- Les sockets 6](#_Toc106577309)

[1-Définition 6](#_Toc106577310)

[2-Fonctionnement 6](#_Toc106577311)

[3-Position dans le modèle OSI 7](#_Toc106577312)

[III- Mise en place côté serveur 7](#_Toc106577313)

[1-Définition 7](#_Toc106577314)

[2-Les services offerts par un serveur informatique 8](#_Toc106577316)

[2- Quels sont les types de serveurs informatiques existants ? 8](#_Toc106577317)

[3-Choix du système d’exploitation du serveur 8](#_Toc106577318)

[4-Choix du format de serveur 9](#_Toc106577319)

[IV- Mise en place côté client 9](#_Toc106577320)

[1-définition 9](#_Toc106577321)

[2-Le mode client 9](#_Toc106577322)

[V-Echange de données 10](#_Toc106577323)

[VI- Généralité : 10](#_Toc106577324)

[VII- Partie pratique 12](#_Toc106577325)

[1- Cote client 12](#_Toc106577326)

[2- Cote serveur 15](#_Toc106577327)

[Conclusion: 21](#_Toc106577328)

**Liste des figures**

[Figure 1: Diagramme du modèle OSI. 7](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577094)

[Figure 2:Le serveur et les clients. 8](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577095)

[Figure 3:le schéma de communication entre client serveur 12](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577096)

[Figure 4:l’importation des bibliothèque 12](#_Toc106577097)

[Figure 5:fichier.txt 12](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577098)

[Figure 6:la finction send\_file 13](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577099)

[Figure 7:La fonction recieve\_html\_file 13](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577100)

[Figure 8:La création de socket 13](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577101)

[Figure 9:La création de connexion 14](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577102)

[Figure 10:La lecteur de fichier .txt 14](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577103)

[Figure 11:L’envoi de fichier .txt 14](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577104)

[Figure 12:créer un socket et une connection 14](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577105)

[Figure 13:La réception de fichier html 15](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577106)

[Figure 14:l’importation des bibliothèque 15](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577107)

[Figure 15: la fonction write\_file 15](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577108)

[Figure 16:la fonction send\_html\_file 16](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577109)

[Figure 17: la création de socket et de connexion avec le client 16](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577110)

[Figure 18:: le serveur est en listening 17](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577111)

[Figure 19:acceptation de la connexion 17](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577112)

[Figure 20: la création de fichier html 17](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577113)

[Figure 21:l'envoie du fichier .html 17](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577114)

[Figure 22:l'a compilation de serveur.c 18](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577115)

[Figure 23:l’exécution de server.c 18](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577116)

[Figure 24:compilation de client.c 18](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577117)

[Figure 25:l’exécution de client.c 18](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577118)

[Figure 26:le réception de fichier .txt 18](#_Toc106577119)

[Figure 27:l création de fichier .html 19](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577120)

[Figure 28:l'apparition de fichier recv.html 19](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577121)

[Figure 29:le fichier recv.html 19](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577122)

[Figure 30:la réception de fichier .html 20](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577123)

[Figure 31:l'apparition de fichier recvfromservwe.html 20](#_Toc106577124)

[Figure 32:Le contenue de fichier recvfromserver.html 20](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577125)

[Figure 33:L’affichage de contenue dans le browser Firefox 20](file:///C:\Users\kawtar\Downloads\Rapport%20de%20SR.docx#_Toc106577126)

Introduction

Dans le cadre du mini projet de notre étude en Master Big Data et Cloud Computing, et pour appliquer les concepts du module de Systèmes réparties, notre Professeur **Ouadoudi ZYTOUNE** nous a donné l’opportunité de choisir un sujet parmi un ensemble de sujets, où nous avons choisir de travailler sur le sujet numéro cinq qui consiste à écrire un programme Client/serveur en C permettant à un client d’envoyer au serveur un fichier texte de le formater en format html et de l’afficher sur l’écran du client. Les programmes serveur et le client sont lancés sur des machines différentes.

Si nous créons une connexion entre le client et le serveur à l'aide de TCP, il a peu de fonctionnalités comme TCP est adapté aux applications qui nécessitent une grande fiabilité et le temps de transmission est relativement moins critique. Il est utilisé par d'autres protocoles comme HTTP, HTTPs, FTP, SMTP, Telnet. TCP réorganise les paquets de données dans l'ordre spécifié. Il y a une garantie absolue que les données transférées restent intactes et arrivent dans le même ordre dans lequel elles ont été envoyées. TCP effectue le contrôle de flux et nécessite trois paquets pour configurer une connexion socket, avant que des données utilisateur puissent être envoyées. TCP gère la fiabilité et le contrôle de la congestion. Il effectue également la vérification des erreurs et la récupération des erreurs. Les paquets erronés sont retransmis de la source vers la destination.

# I- Client-serveur

# 1-Définition

Le protocole ou environnement client–serveur désigne un mode de transaction (souvent à travers un réseau) entre plusieurs programmes ou processus : l'un, qualifié de client, envoie des requêtes ; l'autre, qualifié de serveur, attend les requêtes des clients et y répond. Le serveur offre ici un service au client. Par extension, le client désigne souvent l'ordinateur sur lequel est exécuté le logiciel client, et le serveur, l'ordinateur sur lequel est exécuté le logiciel serveur. Les machines serveurs sont généralement dotées de capacités supérieures à celles des ordinateurs personnels en ce qui concerne la puissance de calcul, les entrées-sorties et les connexions réseau, afin de pouvoir répondre de manière efficace à un grand nombre de clients. Les clients sont souvent des ordinateurs personnels ou terminaux individuels (téléphone, tablette), mais pas systématiquement. Un serveur peut répondre aux requêtes de plusieurs clients. Parfois le client et le serveur peuvent être sur la même machine.

Il existe une grande variété de serveurs et de clients en fonction des besoins ou services à fournir : un serveur Web publie des pages Web demandées par des navigateurs Web ; un serveur de messagerie électronique transmet les courriels à des clients de messagerie ; un serveur de fichiers permet de partager des fichiers sur un réseau aux machines qui le sollicitent ; un serveur de base de données permet aux clients de récupérer des données stockées dans une base de données, etc.

Le client et le serveur doivent bien sûr utiliser le même protocole de communication au niveau de la couche transport du modèle OSI. On parle souvent d'un service pour désigner la fonctionnalité offerte par un processus serveur. Attention, bien que souvent confondues, les notions de programme ou processus client et de programme ou processus serveur sont toutefois différents de celles de machine. En effet un ordinateur unique peut exécuter à la fois un programme serveur et un programme client. Cette configuration est plutôt utilisée à des fins de test, ou assez fréquente dans le monde Linux.

# 2-Caractéristiques

Caractéristiques d'un programme serveur :

* il attend une connexion entrante sur un ou plusieurs ports réseaux locaux ;
* à la connexion d'un client sur le port en écoute, il ouvre un socket local au système d'exploitation ;
* à la suite de la connexion, le processus serveur communique avec le client suivant le protocole prévu par la couche application du modèle OSI.
* l'action réalisée par le serveur en réponse à la requête client est souvent appelée service.

Caractéristiques d'un programme client :

* il établit la connexion au serveur à destination d'un ou plusieurs ports réseaux ;
* lorsque la connexion est acceptée par le serveur, il communique comme le prévoit la couche application du modèle OSI.

Caractéristiques de leur protocole d'échange:

* le client et le serveur doivent bien sûr utiliser le même protocole de communication au niveau de la couche transport du modèle OSI.
* les échanges peuvent se faire à travers un réseau, ou parfois en local
* ce protocole doit être défini, connu et compris des clients et des serveurs

# 3-Fonctionnement

Avant que n'apparaisse l'environnement client–serveur, les réseaux informatiques étaient configurés autour d'un ordinateur central (mainframe en anglais) auquel étaient connectés des terminaux passifs (écran adjoint d'un clavier sans unité centrale et n'effectuant aucun traitement). Tous les utilisateurs étaient alors connectés sur la même unité centrale.

# 4-Avantages des architectures centralisées

* Toutes les données sont centralisées sur un seul serveur, physique ou virtuel, ce qui simplifie les contrôles de sécurité, l'administration, la mise à jour des données et des logiciels.
* La complexité du traitement et la puissance de calculs sont à la charge du ou des serveurs, les utilisateurs utilisant simplement un client léger sur un ordinateur terminal qui peut être simplifié au maximum.
* Recherche d'information : les serveurs étant centralisés, cette architecture est particulièrement adaptée et véloce pour retrouver et comparer de vastes quantités d'informations (moteur de recherche sur le Web), par rapport à l'architecture distribuée beaucoup plus lente, à l'image de Freenet.
* Maintenance matériel minime.
* Grande vélocité sur des grands volumes de données et de traitements

# 5-Inconvénients des architectures centralisées

* Si trop de clients veulent communiquer avec l'ordinateur central au même moment, ce dernier risque de ne pas supporter la charge (alors que les architectures distribuées peuvent répartir la charge si les serveurs sont redondés).
* Si l'ordinateur central n'est plus disponible, plus aucun des clients ne fonctionne (les architectures distribuées peuvent continuent à fonctionner, si les serveurs utilisés sont redondés).
* Les coûts de mise en place et de maintenance peuvent être élevés.
* Les clients ne peuvent communiquer directement entre eux, entrainant une concentration des flux sur l'ordinateur central.
* interface homme-machine minimaliste.
* Utilisation de langages de programmation anciens.
* Calcul scientifique complexe impossible.

# 6-Exemples client-serveur

La consultation de pages sur un site Web fonctionne sur une architecture client–serveur. Un internaute connecté au réseau via son ordinateur et un navigateur Web est le client, le serveur est constitué par le ou les ordinateurs contenant les applications qui fournissent les pages demandées. C'est le protocole de communication HTTP ou XML socket qui est utilisé.

Les courriels sont envoyés et reçus par des clients et gérés par un serveur de messagerie. C'est le protocole de communication SMTP, POP ou IMAP qui est utilisé.

Le système X Window fonctionne sur une architecture client–serveur. En général le client (une application graphique, xeyes par exemple) tourne sur la même machine que le serveur mais peut être aussi bien lancé sur un autre ordinateur faisant partie du réseau.

L'organisation en client léger, façon terminal-serveur, a donné naissance à des projets innovants comme le projet LTSP ou la technologie NX.

# II- Les sockets

# 1-Définition

Est un terme informatique qui peut avoir plusieurs significations suivant qu’il est utilisé dans le cadre logiciel ou matériel.

Matériel : dans un contexte matériel, le socket, ou réceptacle de processeur, désigne le connecteur utilisé pour interfacer un processeur avec sa carte mère.

Contrairement aux processeurs ayant la forme d’une carte et fixés dans un slot assimilable à un slot d’extension, la plupart des sockets et des processeurs actuels sont construits autour de l’architecture matricielle Pin Grid Array (PGA), dans laquelle les broches en dessous du processeur s'insèrent dans le socket.

Logiciel : dans le contexte des logiciels, on peut le traduire par « connecteur réseau » ou « interface de connexion »

Apparu dans les systèmes UNIX, un socket est un élément logiciel qui est aujourd’hui répandu dans la plupart des systèmes d’exploitation. Il s’agit d’une interface logicielle avec les services du système d’exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d’un protocole réseau.

Il lui sera ainsi par exemple aisé d’établir une session TCP, puis de recevoir et d’expédier des données grâce à elle. Cela simplifie sa tâche car cette couche logicielle, de laquelle il requiert des services en appelant des fonctions, masque le travail nécessaire de gestion du réseau, pris en charge par le système. Le terme socket désigne en pratique chaque variable employée dans un programme afin de gérer l’une des sessions.

# 2-Fonctionnement

La communication par socket est souvent comparée aux communications humaines. On distingue ainsi deux modes de communication :

\* le mode connecté (comparable à une communication téléphonique), utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l’adresse de destination n’est pas nécessaire à chaque envoi de données le mode non connecté (analogue à une communication par courrier), utilisant le protocole UDP.

\* Ce mode nécessite l’adresse de destination à chaque envoi, et aucun accusé de réception n’est donné.

Les sockets sont implémentés dans différents langages (C, Java, ...) . En langage C, ils utilisent des fonctions et des structures disponibles dans la bibliothèque

# 3-Position dans le modèle OSI

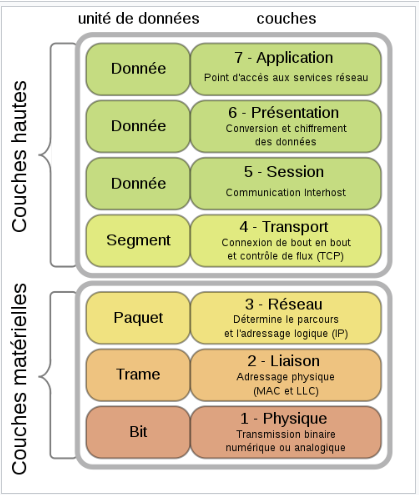
Les sockets se situent entre la couche réseau (incluse) et les couches applicatives du modèle OSI (protocoles UDP ou TCP utilisant IP / ARP).

Figure 1: Diagramme du modèle OSI.

# III- Mise en place côté serveur

# 1-Définition

Un serveur informatique offre des services accessibles via un réseau. Il peut être matériel ou logiciel, c’est un ordinateur qui exécute des opérations suivant les requêtes effectuées par un autre ordinateur appelé « client ». C’est pourquoi on entend souvent parler de relation « client/serveur ». Par exemple, un utilisateur (côté client) va rechercher un site internet en utilisant un navigateur web, pour que ce dernier puisse l’afficher il va effectuer une requête au serveur HTTP qui est un serveur web.

# 

Figure 2:Le serveur et les clients.

# 2-Les services offerts par un serveur informatique

Deux serveurs clients (deux ordinateurs reliés) peuvent se partager électriquement des données. Un serveur informatique permet de fournir un accès à un site internet via un navigateur web, collecter et d’envoyer des mails, de stocker et de consulter des bases de données, de gérer les sites e-commerce, de créer un réseau de partage d’imprimantes, de gérer l’authentification et les accès à des sites internet ou encore d’héberger des logiciels en tant que service (SaaS).

Enfin, un logiciel de serveur permet de protéger ses données contre les hackers, de centraliser la gestion de son système d’information, et de partager ses ressources.

# 2- Quels sont les types de serveurs informatiques existants ?

Un serveur offre plusieurs services, ainsi il existe plusieurs types de serveurs informatiques :

Web : c’est ce qui permet à un navigateur web d’afficher un site internet

Fichiers : il permet de stocker tous les documents et de les rendre accessibles depuis n’importe quel ordinateur

Applicatifs métiers : il permet de stocker toutes les données commerciales, clients, comptabilité et d’en extraire que les données nécessaires

Messagerie : il permet de gérer la distribution des mails et d’y avoir accès à distance

# 3-Choix du système d’exploitation du serveur

Il existe plusieurs systèmes d’exploitation dédiés aux serveurs : Linux, Windows Server, Redhat Enterprise, Gentoo, Fedora, etc. à vous de choisir le système qui vous convient le mieux.

# 4-Choix du format de serveur

3 serveurs physiques existent :

Serveur Tour : similaire aux unités centrales d’un PC fixe. Ils peuvent être installés n’importe où, mais nécessitent tout de même une arrivée électrique adaptée.

Serveur Rack : ils ont un format aplati et sont destinés à être installés dans une baie. Les configurations sont plus performantes que sur les serveurs Tour. L’accès est facilité car ils sont généralement rangés dans des armoires dans des salles dédiées.

Serveur Lame (ou serveur Blade) : c’est le format le plus récent, ces serveurs ne peuvent fonctionner seuls, ils se glissent dans un châssis intégrant d’autres serveurs lames. La différence avec ce format de serveur, c’est que c’est le châssis qui est relié aux connexions réseau et alimentations électriques.

# IV- Mise en place côté client

# 1-définition

Dans un réseau informatique, un client est le logiciel qui envoie des demandes à un serveur. Il peut s'agir d'un logiciel manipulé par une personne, ou d'un bot. Est appelé client aussi bien l'ordinateur depuis lequel les demandes sont envoyées que le logiciel qui contient les instructions relatives à la formulation des demandes et la personne qui opère les demandes.

L'ordinateur client est généralement un ordinateur personnel ordinaire, équipés de logiciels relatifs aux différents types de demandes qui vont être envoyées, comme un navigateur web, un logiciel client pour le World web.

# 2-Le mode client

Une application distribuée est une application informatique dans laquelle les traitements sont effectués de concert par plusieurs ordinateurs d'un réseau informatique. Un protocole de communication établit les règles selon lesquelles les ordinateurs communiquent, dans le cadre de l'application en question.

Dans une application informatique en mode client/serveur les traitements sont effectués par l'exécution conjointe de deux logiciels différents et complémentaires placés sur des ordinateurs différents: le client et le serveur.

Un client léger est un client qui ne fait que formuler des demandes et présenter les résultats à l'utilisateur. Les traitements sont alors effectués entièrement par le serveur. Par opposition, un client lourd effectue une partie des traitements sans faire intervenir le serveur.

Client riche: les premiers navigateur web possédaient une maniabilité inférieure aux clients lourds. Apparue en 2002, l'évolution appelée client riche consiste en un ensemble de technologies telles que Adobe Flash ou Ajax, destinées à donner au navigateur web une maniabilité équivalente voire supérieure aux clients lourds.

# V-Echange de données

Lorsqu’on parle de webservices pour les données, on pense aujourd’hui naturellement au modèle d’API REST ou éventuellement à SOAP, qui permettent de parcourir de façon normée les informations dont dispose un serveur à partir d’un client. Si ce modèle a fait ses preuves et est largement utilisé aujourd’hui, il ne vient cependant pas sans défauts.

Ses inconvénients sont l’absence de typage de la donnée, la nécessité de faire plusieurs (au moins deux) aller-retour pour obtenir une information de détail, un surplus de données renvoyé par le serveur par rapport au besoin du client (over-fetching) et le couplage fort entre les versions du serveur et du client.

Je vais détailler ces points dans les paragraphes suivants, si vous êtes déjà convaincus par ces limitations de REST vous pouvez avancer au titre

Dans un monde idéal, on voudrait une application qui puisse grossir sans remettre en cause l’existant :

\*un même serveur qui puisse évoluer en restant compatible avec toutes les versions du client ;

\*un serveur qui puisse retourner toutes les informations nécessaires en un minimum d’aller-retour ;

\*un serveur qui ne retourne que les informations strictement nécessaires ;

\*une référence concernant le typage des données échangées, qui soit partagée entre le serveur et le client.

Comme nous sommes exigeants, et que nous n’avons pas envie de refaire tout l’existant parce que c’est la mode, ce système d’échange de données devra pouvoir s’intégrer dans une application existante en parallèle de l’API REST déjà en place.

On dirait une liste au Père Noël, mais c’est plutôt vers Netflix et Facebook que nous allons nous tourner

# VI- Généralité :

On veut pouvoir échanger des données : de tous les types, de toutes les tailles, de manière sécurisée et synchronisée.

Pour la communication entre processus :

Les signaux : peu adapté à l’échange de données

fichiers spéciaux :

* entrées et sorties standards des processus,
* tubes anonymes (processus sont créés dans un même programme),
* tubes nommés (qui ont un nom dans le système de fichiers).

Ces modes fonctionnent entre processus d’un même système. . . Il faut aussi pourvoir échanger des données entres processus via le réseau : c’est le rôle des sockets

Les systèmes Unix/Lunix utilisent le même système de descripteur de fichiers

Cela permet d’utiliser les mêmes primitives d’accès, que ce soit sur des canaux de communication ou de vrais fichiers :

On peut toujours utiliser read() et write() ;

Pour les sockets, ils existe des adaptations comme recv() et send().

On peut utiliser des bibliothèques de hauts niveaux pour les fichiers la bibliothèque C avec stdio.h : FILE ∗, fread(), fwrite(),quelques notions sur les réseaux :

Un dialogue via le réseau suppose l’existence de deux processus sur deux ordinateurs qui sont capables de se reconnaître et se transmettre des informations.

Dans les modèles les plus souvent utilisés (TCP/IP et UDP/IP) les données sont adressées :

* un ordinateur particulier via une adresse IP destination ;
* un processus particulier via un numéro de port destination ;

De la même manière, l’origine des données est connue via l’adresse IP source et le numéro de port source.

Deux modèles de transport sont couramment utilisés :

le modèle déconnecte (analogies : courrier papier), les données sont envoyées et reçues sous forme de paquets :

* d’outil pour savoir si une donnée est perdue ;
* d’outil pour assurer l’ordre dans lequel elles arrivent.

protocole **UDP** (User Datagramme Protocol) :le modèle connecté (analogie : téléphone) une connexion est mise en place qui permet de gérer les échanges paquets :

si des paquets disparaissent ou arrivent dans le d´désordre, cela est automatiquement corrigé la connexion est maintenue si elle se coupe de manière irréversible, une erreur est générée (Broken Pipe) pas d’outil pour délimiter les messages ou les ensembles de données (page web, fichier. .

protocole **TCP** (Transmission Control Protocol) on n’utiliserons que le mode connecté, TCP/IP il faut établir puis terminer la connexion, deux acteurs

* Le serveur qui attend une demande connexion et accepte le client.
* Le client qui est à l’origine de la demande de connexion.

Modèle client-serveur chacun a des actions à faire le client doit contacter le processus serveur dont il connait le nom de machine (ou son adresse IP) et le port le serveur doit faire deux choses se mettre en attente sur un certain port et certaines adresses IP quand un client se connecte, dialoguer avec lui. Un serveur peut accepter des connexions de plusieurs clients. Par exemple, un serveur web écoute sur le port 80 et répond a plein de clients.

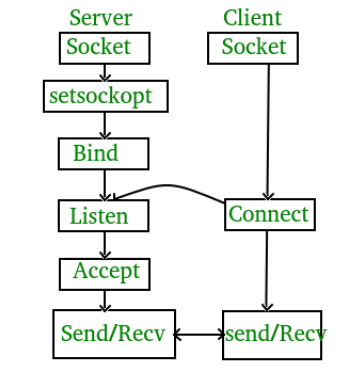


Figure 3:le schéma de communication entre client serveur

# VII- Partie pratique

# Cote client

on commence par l’ importation des bibliothèque necessaire pour le programme

Une image contenant texte

Description générée automatiquement

Figure 4:l’importation des bibliothèque

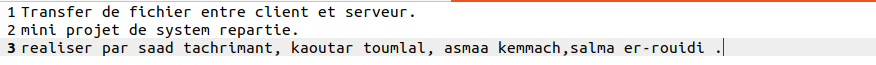
le fichier .txt que le client veut l’envoyer au serveur

Figure 5:fichier.txt

la creation de la fonction send\_file qui prend en parametre le fichier et le socket.

le but de cette fonction est de envoyer le fichier text au serveur.

Une image contenant texte

Description générée automatiquement

Figure 6:la finction send\_file

La creation de la fonction recieve\_html\_file qui prend en prametre le socket

Le but de cette fonction est de recevoir le fichier .html qui viennent de la part de serveur

Une image contenant texte

Description générée automatiquement

Figure 7:La fonction recieve\_html\_file

Dans le main on définie l’addresse ip de serveur et le port et on crée soket et on affiche le message: ‘’serveer soket created successfuly.’’

Une image contenant texte

Description générée automatiquement

Figure 8:La création de socket

la création de connection avec le serveur

Une image contenant texte

Description générée automatiquement

Figure 9:La création de connexion

Une image contenant texte

Description générée automatiquement la lecture du fichier .txt avec la fonction fopen avec le mood ‘’read’’

Figure 10:La lecteur de fichier .txt

l’envoi du fiechier .txt l’aide de la fonction send\_file qu’on a déjà créer avec l’affichage de message :‘’file data sent successfully’’

Une image contenant texte

Description générée automatiquement

Figure 11:L’envoi de fichier .txt

Une image contenant texte

Description générée automatiquementla creation d’un soket de serveur et créer une connection avec le, et l’affichage des message de sucée .

Figure 12:créer un socket et une connection

le dernier étape est de recevoir le fichier html de puis le serveur est afficher des message de sucée et a la fin la fermenteur de la connexion

Une image contenant texte

Description générée automatiquement

Figure 13:La réception de fichier html

# Cote serveur

on commence par l’ importation des bibliothèque necessaire pour le programme

Une image contenant texte

Description générée automatiquement

Figure 14:l’importation des bibliothèque

La création de la fonction write\_file qui va ajouter les balises HTML au fichier text envoyer par le client

Une image contenant texte

Description générée automatiquement

Figure 15: la fonction write\_file

La création de la fonction send\_html\_file qui va envoyer le nouveau fichier .html qui contient le message et les balise au client

Une image contenant texte

Description générée automatiquement

Figure 16:la fonction send\_html\_file

La fonction main on crée un socket pour assurer la connexion avec le client

Une image contenant texte

Description générée automatiquement

Figure 17: la création de socket et de connexion avec le client

Dans cette étape le serveur est en listening en attend de l’envoie de fichier txt de la part de client

Une image contenant texte

Description générée automatiquement

Figure 18:: le serveur est en listening

Dans cette étape le serveur accepte la connexion avec le client

Une image contenant texte

Description générée automatiquement

Figure 19:acceptation de la connexion

Cette étape est pour le but d’écrire le contenu du fichier .txt envoyer par le client dans un autre fichier .html



Figure 20: la création de fichier html

La dernière étape pour le serveur c’est d’envoyer le fichier html au client

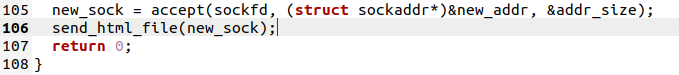


Figure 21:l'envoie du fichier .html

D 'après avoir les deux programmes client.c et server.c maintenant on va présenter l’exécution des ceux derniers

On commence avec la compilation de serveur

Une image contenant texte

Description générée automatiquement

Figure 22:l'a compilation de serveur.c

Une fois on exécute le server.c il est en lestening en attend que le client lui envoie le fichier .txt

Une image contenant texte

Description générée automatiquement

Figure 23:l’exécution de server.c

On passe pour compiler le client et l’exécuter

Une image contenant texte

Description générée automatiquement

Figure 24:compilation de client.c

Apres l’exécution de client.c il envoie le fichier .txt serveur

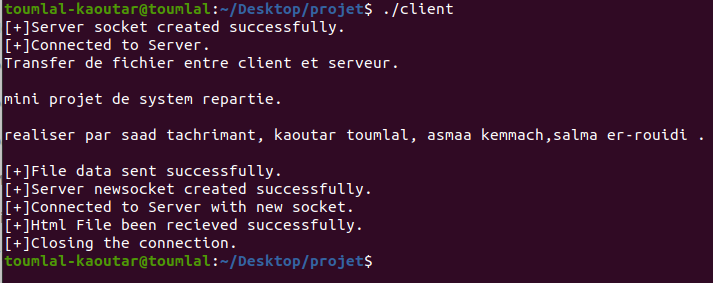


Figure 25:l’exécution de client.c

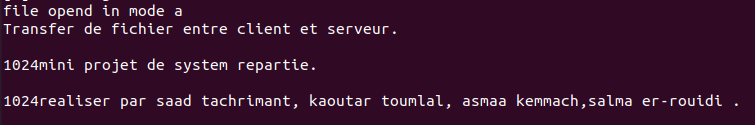
Le serveur reçoi le fichier .txt

Figure 26:le réception de fichier .txt

Le serveur écrire le fichier .html et l’envoie au client

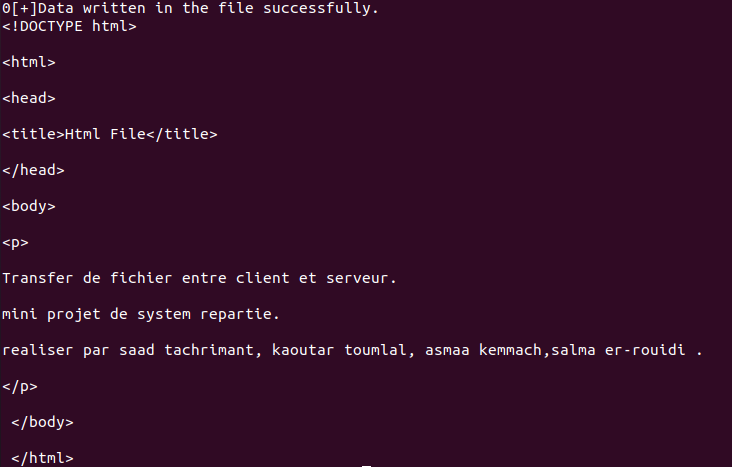


Figure 27:l création de fichier .html

Le fichier créer est recv.html

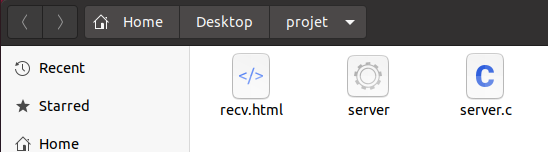


Figure 28:l'apparition de fichier recv.html

Le contenue du fichier recv.html qui contient le message de client avec les balises HTML

Une image contenant texte

Description générée automatiquement

Figure 29:le fichier recv.html

Le client reçoi le fichier .html avec et affiche le message ‘’html file been recieved successfully’’

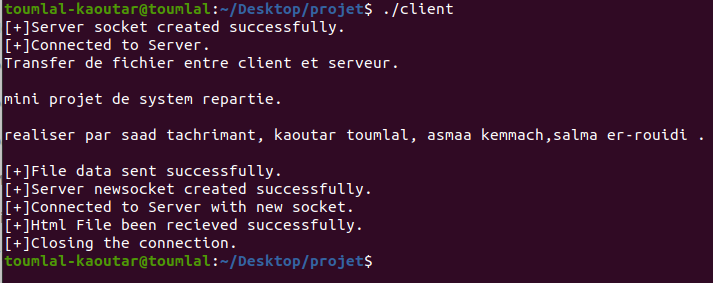


Figure 30:la réception de fichier .html

Apparition de fichier .html qui continent le message et les balises html

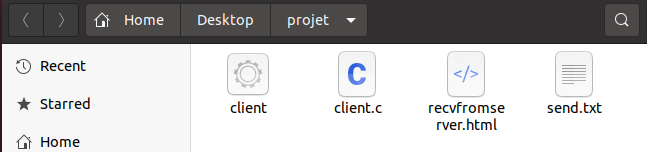


Figure 31:l'apparition de fichier recvfromservwe.html

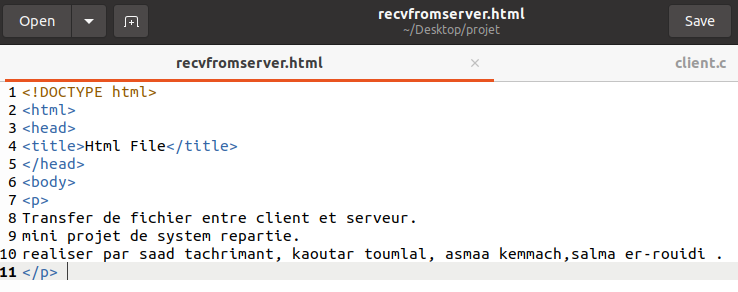
Le contenue de fichier recvfromserver.html

Figure 32:Le contenue de fichier recvfromserver.html

L’affichage de contenue dans le browser Firefox:

Figure 33:L’affichage de contenue dans le browser Firefox

# Conclusion:

Pour conclure, ce sujet nous a aidés à développer nos connaissances théoriques sur les Systèmes réparties en générale, et spécifiquement sur la communication inter- processus, où nous avons travaillé

De plus ceci a été une bonne expérience dans la mesure où j’ai pu découvrir le monde du système reparties et le fait que nous avons appréciait celui-ci.

Pour terminer, nous renouvellerons ce type d’expérience avec grand plaisir, c’est pourquoi toujours en restant dans l’optique de choisir le projet qui nous correspondront le mieux

Références

<https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal>

<https://www.syloe.com/glossaire/serveur-informatique/>

<https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/amp/>