Master 1 Informatique
Spécialité AIGLE

Projet : Enregistrement de rapports d'activité dans une entreprise

Réseaux et Communications

Responsable : Hinde BOUZIANE

Amayas ABBOUTE
Gilles ENTRINGER

SOMMAIRE

PARTIE1 ANALYSE

1.1 Introduction

Afin de mettre en application ce que nous avons vu en cours du module "Réseaux et Communications", à savoir la communication entre processus en utilisant des sockets, des threads ainsi que différents moyens de synchronisation, nous avons réalisé une application client/serveur multi-thread.

Voici les caractéristiques du serveur que nous avons programmé :

1. Serveur programmé en langage C.
2. Multi-thread : Le serveur peut accepter et gérer plusieurs clients simultanément.
3. Gestion de la réception de longs messages et de l'envoi de fichiers au format PDF.
4. Variables conditionnelles et mutex pour gérer les accès concurrents et la synchronisation entre threads.

1.2 Architecture de l'application

Processus Serveur

Le serveur est structuré comme suit :

1. Un thread principal qui tourne à l'infini pour pouvoir accepter plusieurs clients et ceci de façon simultanée. A chaque acceptation d'un client, le thread correspondant est lancé, ce qui permet une certaine autonomie entre les clients. Par exemple un contrôleur peut consulter un rapport pendant qu'un employé est entrain de saisir un nouveau rapport.
2. Le thread contrôleur est lancé dès que le processus contrôleur se connecte au serveur. Ainsi, le serveur peut recevoir une liste d'employés qui doivent rédiger un rapport ou envoyer au processus contrôleur un rapport au format PDF.
3. Le thread employé est lancé dès la connexion d'un processus employé après vérification de l'identifiant. Si ce dernier ne fait pas partie de la liste des employé, le thread n'est pas lancé et le processus employé est prévenu qu'il ne doit pas rédiger de rapport.

   Si l'identification est réussie, le thread reçoit les blocs du rapport saisi au fur et à mesure et les sauvegarde dans un fichier .tex et envoie le rapport dès que l'employé a terminé la saisie.

   Remarque : Le fait que les threads soient lancés seulement après vérification de l'identifiant permet d'optimiser l'utilisation du processeur et de la mémoire.

Processus Contrôleur

Le contrôleur est un processus à part entière qui permet de déclencher le lancement du thread contrôleur se trouvant au niveau du processus serveur. Le rôle de ce thread est d'envoyer la liste des employés qui doivent rédiger un rapport ainsi que la visualisation d'un rapport stocké sur le serveur.

Processus Employé

L'employé est également un processus à part entière, son rôle étant d'inviter un employé X à rédiger un rapport si ce dernier figure bien dans la liste des employés et de visualiser le rapport dès que l'employé a fini la saisie de son rapport.

1.3 Protocoles d'échanges

Serveur - Employé

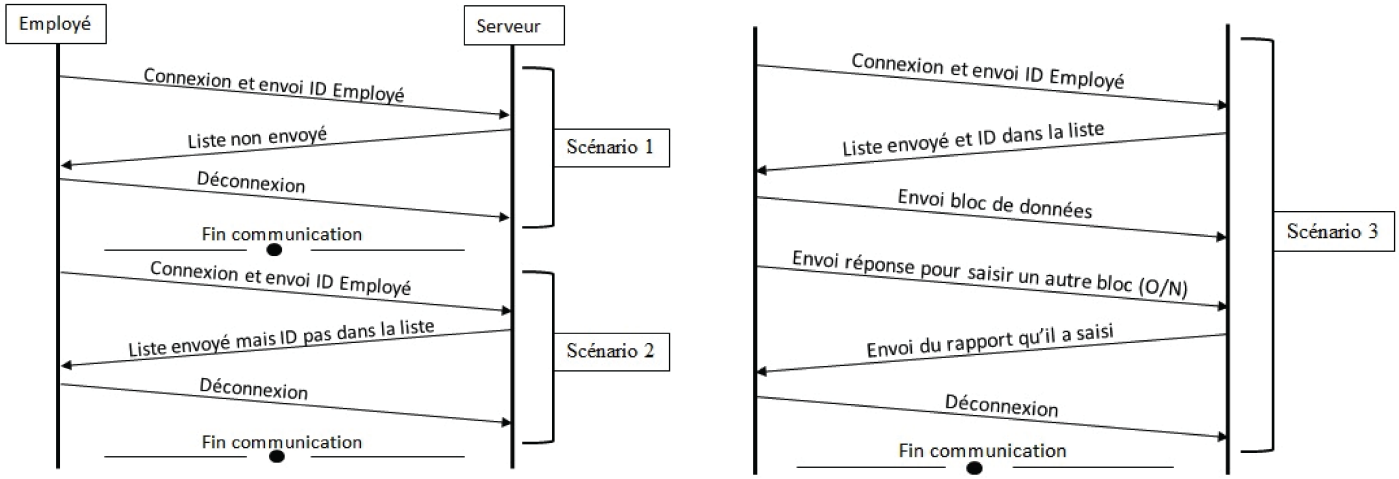

FIGURE 1.1 – Protocole d'échange Serveur - Employé

Serveur - Contrôleur

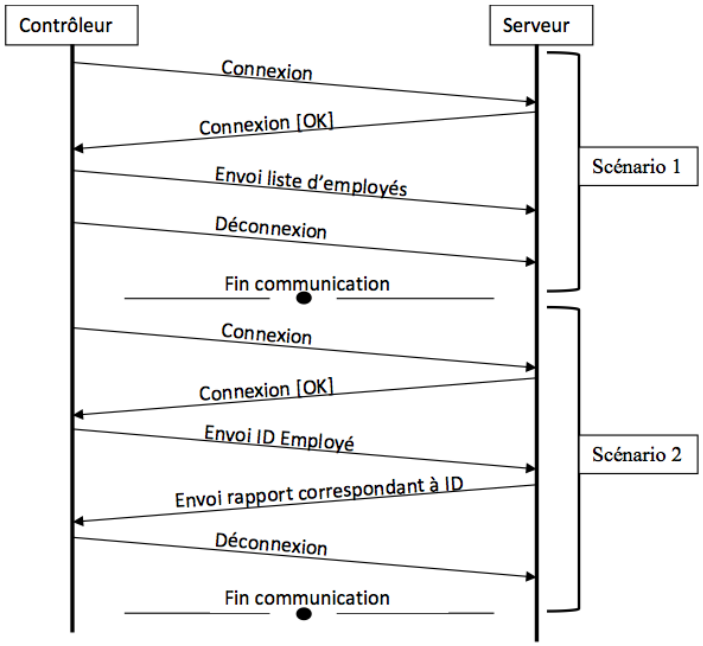

FIGURE 1.2 – Protocole d'échange Serveur - Contrôleur

1.4 Schémas algorithmiques

1.4.1 Serveur

Tant que (vrai)

Attente d'une connexion ;

Réception de l'ID d'un client ;

Lancer le thread correspondant à l'ID du client connecté (cf Thread Employé et Thread Controleur) ;

Thread Employé

Faire

Attente d'un bloc de données ;

Réception d'un bloc ;

Ecrire une section du rapport ;

Tant que l'employé n'a pas signalé la fin de la saisie ;

Enregistrement du rapport au format PDF ;

Prendre(verrou) ;

Sauvegarder l'employé ayant rédigé un rapport ;

Libérer(verrou) ;

Signaler au contrôleur la fin de saisie d'un rapport s'il souhaite envoyer une nouvelle liste ;

Envoyer le rapport à l'employé ;

Fermeture du thread ;

Thread Contrôleur

Attente si contrôleur veut saisir une liste ou consulter un rapport ;

Vérification s'il existe déjà une liste et envoi résultat au contrôleur ;

Vérification si au moins un rapport a été enregistré par un employé et envoi du résultat au contrôleur ;

Cas = Saisir une nouvelle liste

Prendre(verrou) ;

Attendre jusqu'à ce que tous les employés connectés aient terminé leurs saisies de rapports ;

Sauvegarder la liste et le nombre d'employés qui doivent saisir un rapport ;

Libérer (verrou) ;

Cas = consulter un rapport

Faire

Envoyer liste des identifiants des employés ayant rédigé un rapport ;

Recevoir l'identifiant du rapport à consulter ;

Prévenir contrôleur si rapport est disponible ;

Envoyer rapport au contrôleur ;

Tant que le contrôleur veut consulter un autre rapport ;

Fermeture du thread ;

1.4.2 Contrôleur

Connexion au serveur() ;
Saisir la liste des employés() ;
Envoyer la liste() ;
Déconnexion() ;
Connexion au serveur() ;
Choisir entre :
Saisir une nouvelle liste.
Consulter un rapport.
Cas choix = 1
    Saisir la liste des employés() ;
    Envoyer la liste() ;
    Déconnexion() ;
Cas choix = 2
    Faire
        Saisir nom employé() ;
        Recevoir le rapport() ;
    Tant que le controleur consulte rapports ;
Déconnexion() ;

1.4.3 Employé

Connexion au serveur() ;
Si liste non envoyé alors déconnexion() ;
Si liste envoyé mais ID ne correspond pas alors déconnexion () ;
Sinon
    Tant qu 'il y a des blocs de données à saisir
        Saisir bloc() ;
        Envoyer bloc() ;
    FinTq
Réception rapport du serveur() ;
Ouvrir et visualiser le rapport() ;
Déconnexion() ;

Partie2 Mise en oeuvre

2.1 Diagramme de déploiement

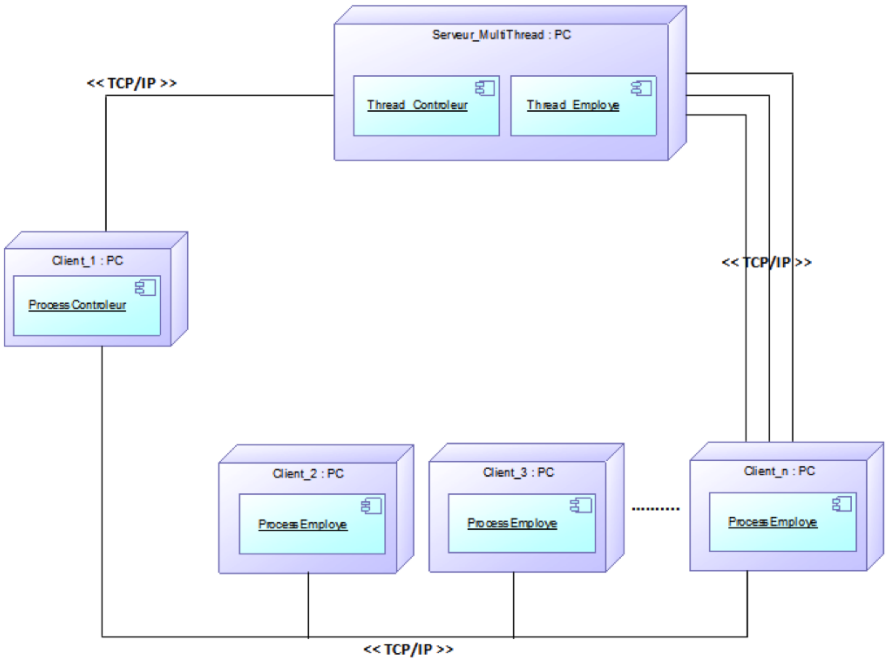

Figure 2.1 – Diagramme de déploiement

2.2 Mode d'emploi

2.2.1 Compilation à partir du makefile

Dans le répertoire principal se trouve un makefile. Il suffit de saisir dans un terminal la commande "make". Le "makefile" va créer les exécutables Controleur, Employe et Serveur respectivement dans les répertoires Controleur, Employe et Serveur

2.2.2 Lancement du serveur

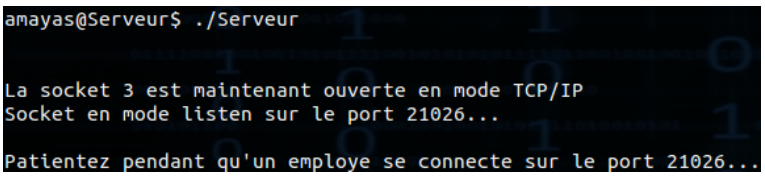

Figure 2.2 – Lancement du serveur

Le serveur est lancé dans le terminal à partir de la commande "./Serveur" et se met en attente de connexion d'un client.

2.2.3 Lancement du contrôleur

FIGURE 2.3 – Lancement du controleur

Le contrôleur est lancé avec en paramètre l'adresse IP du serveur (ici en local). Il se connecte au serveur et les étapes suivantes peuvent être exécutées :

- Pour la première connexion, le contrôleur est directement invité à saisir la liste des employés qui doivent rédiger un rapport.
- Si au moins un employé a saisi un rapport, l'option 2 qui est de visualiser un rapport est affichée (cf. figure 2.3)

2.2.4 Lancement de l'employé

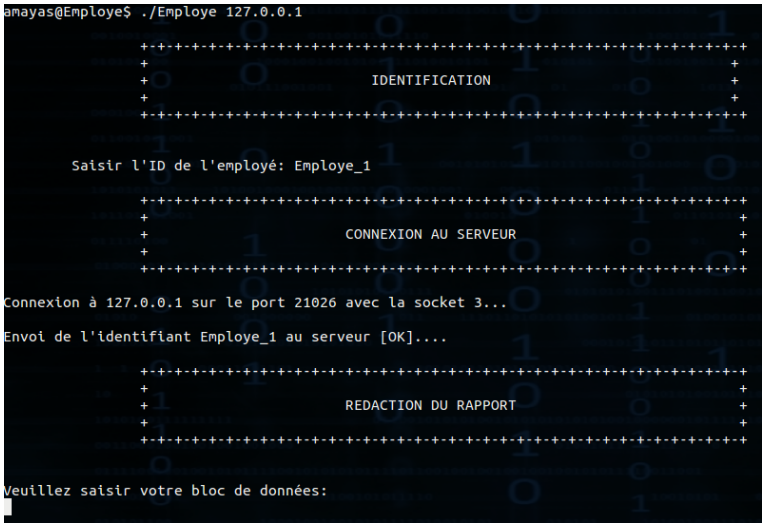

FIGURE 2.4 – Lancement de l'employé

L'employé est également lancé avec en paramètre l'adresse IP du serveur (ici en local). L'employé se connecte au serveur et saisit son identifiant. Plusieurs scénarios ont été considérés dans l'implémentation du processus "Employé" :

- Le serveur n'est pas lancé et un message d'erreur est retourné à l'employé.
- La liste n'a pas encore été envoyé, l'employé se déconnecte.
- L'employé se déconnecte car son identifiant ne figure pas dans la liste envoyée par le contrôleur
- L'employé est invité à saisir son rapport (cf . 2.4).

PARTIE3 DIFFICULTÉS RENCONTRÉES ET SOLUTIONS APPORTÉES

Tout au long du développement de cette application client-serveur, nous avons été confrontés à de nombreuses difficultés. Dans cette partie, nous expliquerons les problèmes majeurs qui sont regroupés en trois sous-parties : Difficultés de conception, difficultés liées à l'envoi de données via la socket et difficultés diverses . De plus, nous illustrerons les solutions apportées aux difficultés.

3.1 Difficultés de conception

Le nombre de threads à implémenter

Dans un premier temps nous nous sommes focalisés sur la conception de l'application. Une des difficultés majeures était donc de choisir le nombre de threads.

Une des possibilités que nous avions envisagées était la création de trois threads : Un thread serveur, un thread contrôleur et un thread employé. Ainsi, un thread serveur serait lancé par le thread principal (main()) à chaque connexion d'un client. Une deuxième possibilité était l'implémentation d'un seul thread principal (main) et d'un seul thread client. En fonction de l'identifiant envoyé par le client, les actions seraient effectuées dans ce seul thread. Finalement nous avons décidé d'implémenter un thread principal (le serveur), un thread pour le contrôleur et un thread par client connecté. Ces deux derniers sont lancés en fonction de l'identifiant reçu. Par exemple, le processus contrôleur envoie automatiquement l'identifiant "controleur" dès sa connexion et le thread correspondant sera lancé. Cette solution nous a garanti d'implémenter un serveur concurrent et de pouvoir gérer la synchronisation et l'accès concurrent aux données.

3.2 Difficultés liées à l'envoi de données via socket

3.2.1 Envoyer de longues chaînes de caractères

Comme il est indiqué dans le sujet, il est nécessaire d'éviter la lecture et l'écriture de longs messages sur les sockets. Dans un premier temps nous arrivions seulement à envoyer des blocs de 981 caractères.

La solution finale consiste à envoyer dans un premier temps le nombre de caractères du bloc à envoyer au serveur. Par la suite, la boucle "while" s'exécute jusqu'à ce que le nombre de caractères reçus est égal au nombre de caractère envoyés. L'application est maintenant capable de gérer et de manipuler de très longues chaînes de caractères.

3.2.2 Envoyer tableau malloc

Il n'est pas possible d'envoyer un tableau malloc via une socket à un autre processus car le malloc est un pointeur et donc lié à un seul espace d'adressage.

Le serveur va ainsi copier le tableau malloc (dans lequel sont stockés les identifiants des employés ayant rédigé un rapport) dans une structure contenant un tableau de chaînes de caractères. De plus, ce tableau contient le nombre d'employés ayant enregistrés un rapport ce qui va faciliter l'affichage du tableau au niveau du processus receveur (dans notre cas le contrôleur).

3.2.3 Envoyer fichier pdf

En local, nous arrivions sans problèmes à envoyer des fichiers pdf, voir des fichier de taille assez grande (par exemple un fichier musique de 10 Mo) sans aucun problème. Par contre, lors des essais sur un réseau local (sur les machines de l'UM2 au bâtiment 5), le processus receveur recevait seulement 1.9 Ko par envoi.

Nous avons pu résoudre cette difficulté en appliquant le même principe que pour l'envoi de longues chaînes de caractères : Nous envoyons d'abord la taille en bytes du fichier à envoyer et par la suite nous bouclons sur cette donnée jusqu'à ce que toutes les données soient reçues.

3.3 Difficultés diverses

3.3.1 Multiclient

Après une première implémentation nous avons procédé à plusieurs tests. Ainsi, nous avons remarqué que notre implémentation ne gérait pas plusieurs clients simultanément. Les chaînes de caractères envoyés par un client étaient stockées dans le rapport de l'employé qui s'était connecté en dernier. De plus il nous arrivait qu'un des employés restait bloqués ou que les blocs qu'il avait saisies n'étaient enregistrées null part.

Au niveau du serveur nous avons donc codé une structure qui comprend les informations suivantes : la socket à laquelle le client est connecté, l'identifiant de l'employé et le nom du thread. Par la suite nous avons implémenté un tableau de cette structure. A chaque connexion d'un client, une nouvelle structure est insérée dans ce tableau. Lors de la création d'un thread, la structure correspondant au client est passée en argument du thread. Ainsi chaque thread connaîtra à tout moment son nom et l'identifiant du client auquel il est attribué.