Nordine El Hassouni
Frank Petitdemande

Master 1 AIGLE
27/12/2012

Master 1 Informatique - Réseaux et Communications - Projet
Enregistrement de rapports d'activité dans une entreprise.

Fonctionnalités développés

Avant de coder on à défini pour chaque applications les fonctionnalités suivantes.

Employer (client)

→ Se connecté au serveur.
→ Écrire rapport.
→ récupéré rapport.
→ Déconnexion

Serveur

→ Recevoir liste rapport.
→ Traiter demande rédaction rapport.
→ Sauvegarder rapport.
→ Mettre sous format PDF.
→ Envoyer rapport.

Contrôleur

→ Se connecté au serveur.
→ Envoyer liste d'employé.
→ Vérifier présence rapport.
→ Déconnexion.

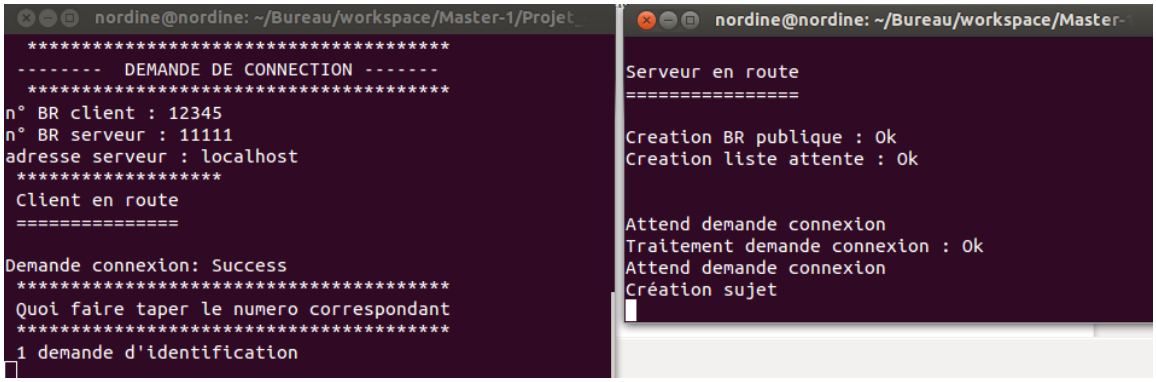
Mode d'emplois

Pour illustrer le mode d'emplois nous avons décidé de mettre des « screenshot » accompagné de commentaires.

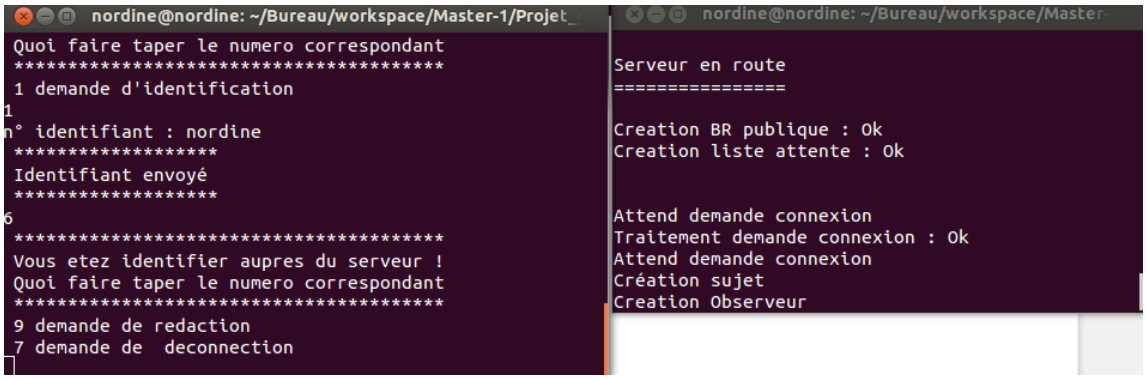
L'utilisation est la même pour le contrôleur et le client (employé) seul les taches sont différentes. Les « screenshot » en démonstration sont celle du client, mais le mode d'utilisation est similaire.

L'utilisation est très simple, le serveur est en route et en attende, pour traiter les demandes de connexion.

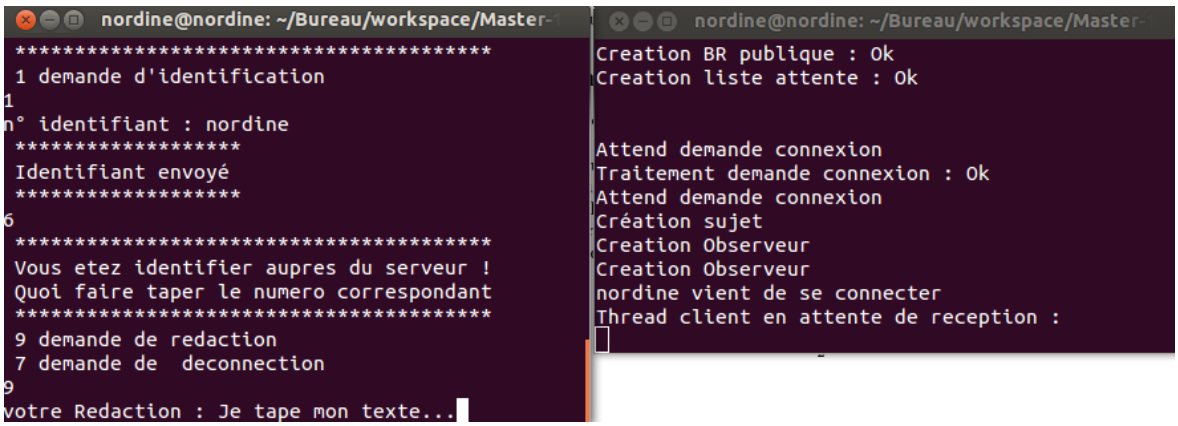
Le client va faire une demande de connexion, après avoir était connecter au serveur l'option « demande d'identification » va lui être proposé.



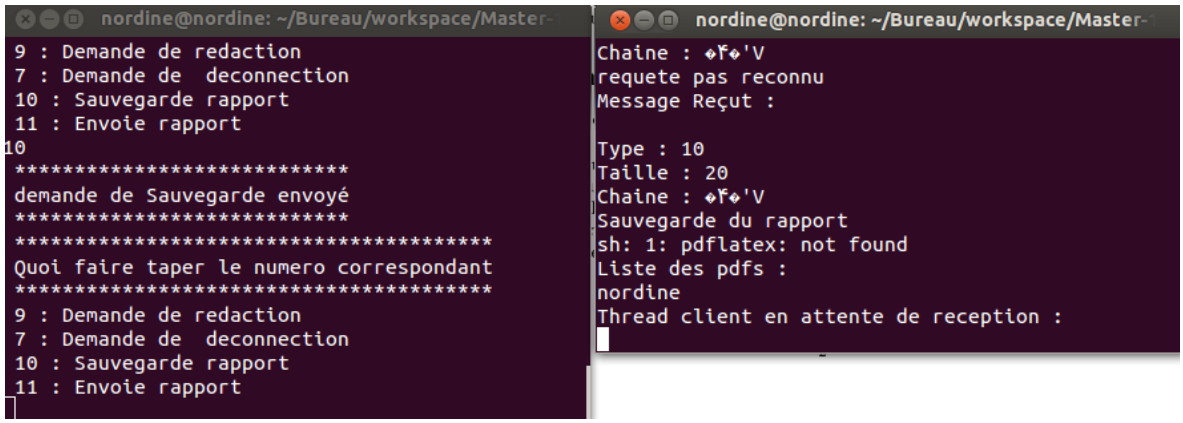
Après identification, accès à d'autres options sous certaine condition. L'identification s 'effectue avec succès seulement si l'employé est sur la liste des employés qui doivent saisir leur rapport.



Rédaction du rapport, pour sortir de cette option il faut taper « fini »



Après la rédaction du rapport les options suivantes sont à disposition.
→ 11 pour recevoir le rapport sous format PDF, après sauvegarde.

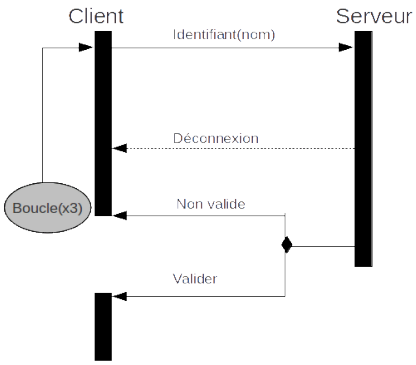


Protocole d’échange

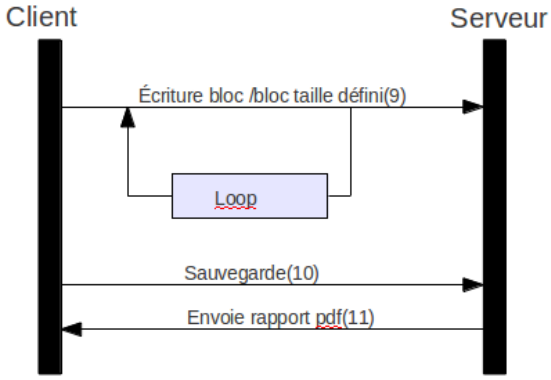
Pour que les clients et le serveur puissent dialoguer correctement, on a organisé tous les échanges d'informations. Pour cela, on a mis en place un protocole qui détermine le contenu des messages échangés entre un client et le serveur.

On a mis en place un protocole pour la connexion/déconnexion des clients au serveur, les requêtes des clients, l'envoi du contenu du rapport et les réponses du serveur.

Protocole de connexion



Protocole de dialogue

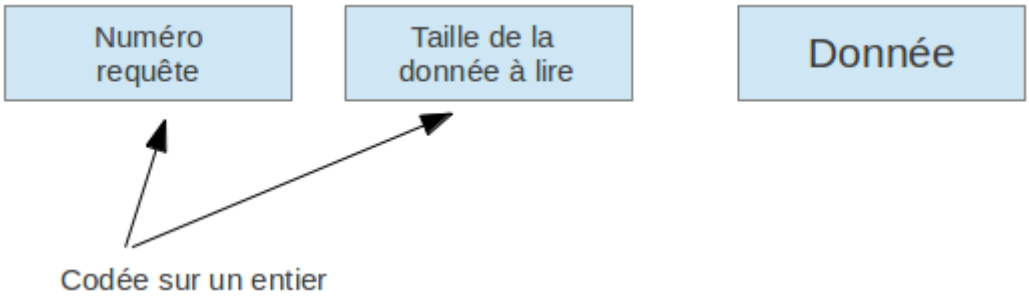


schémas 3

Les requêtes sont déterminé par 3 donnée : le numéro, la taille et la donnée

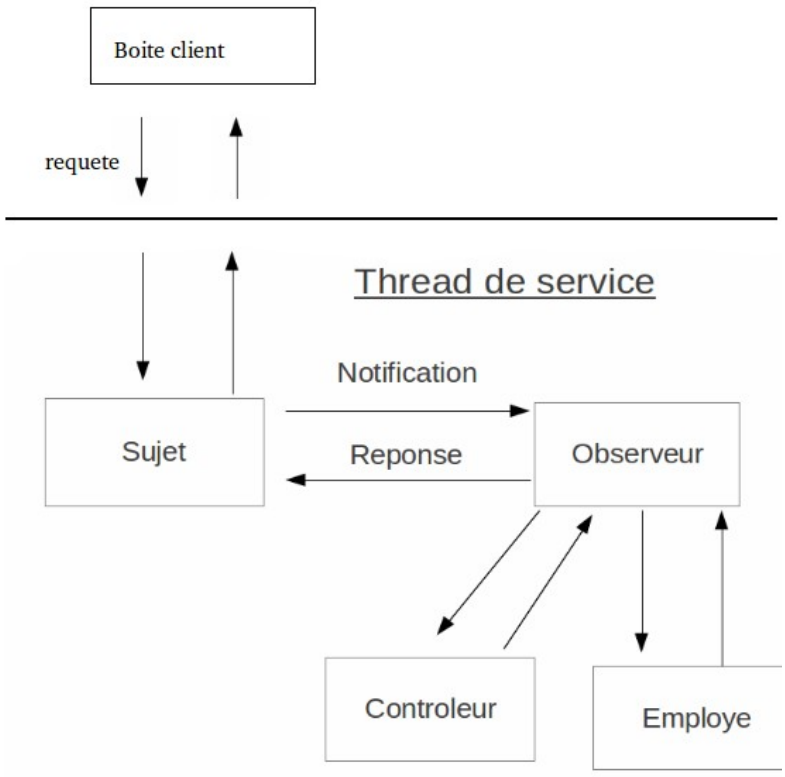
Le numéro de la requête :
Permet à celui qui est en attente de réception de connaître la demande.

Exemple :
L'employé envoi le numéro 1 le serveur comprend que c'est une demande de connexion, et va lui répondre par le type 5 ou 6.
cf schéma 1.



Architecture de l'application

Dès qu'une connexion est accepté par la serveur, l'application crée un thread de service basé sur le patron de conception observateur.
Le sujet, lit les messages reçut et envoie un notification aux observateurs. Suivant que le client s'est identifié en contrôleur ou employé l'observateur est différent, donc le traitement des messages.



Schémas algorithmiques

Les applications se décomposent d'une fonction principale d'exécution le main(), et de plusieurs fonctions secondaire pour l'exécution des taches voulus. Voici quelque fonction intéressante simplifier que l'on a décider de mettre dans ce rapport.

Application Client(employé) → Fonction de réception du rapport au format pdf :

```
int EnvoiRapport(int brclient)
{
 int typeR;
 nt tailleMsg;
 char chaine[20];
 int descBrCli = brclient;

 tailleMsg = sizeof(chaine);
 typeR = 11;
send(descBrCli, &typeR, sizeof(int), 0);
send(descBrCli, &tailleMsg, sizeof(int), 0);
send(descBrCli, chaine, sizeof(chaine), 0);
cout<<" ********** "<<endl;
cout<<" demande envoyé "<<endl;
cout<<" ********** "<<endl;

//Recupere rapport
cout<<recv(descBrCli, &typeR, sizeof(int), 0)<<endl;
cout<<"Type recut :"<<typeR<<endl;
 cout<<recv(descBrCli, &tailleMsg, sizeof(int), 0)<<endl;
cout<<"Taille : "<<tailleMsg<<endl;

 ofstream flux("rapport.pdf");
 char car;

while(tailleMsg > 0
{
        cout<<"attend reception"<<endl;
        if (recv(descBrCli, &car, sizeof(char), 0) != 0) perror("rcv");
        flux<<car;
        tailleMsg--;
        cout<<tailleMsg<<endl;
 }

flux.close();
cout<<"fichier recu "<<endl;
return 1;
}
```

Serveur : Traitement des requêtes

```
1  void SubjectClient::run(){
2  //Tant que le client est connecté,
3    on se met en attente de reception sur la Boite réseau//
4  while(connecter){
5          try{
6                  msgR = Message(descBr);
7                  Notify();//averti observeur attaché, d'une reception du msg
8          }catch(int  &descr){
9          //Capture les erreurs de reception sur la BR
10         perror("recv");
11         break;      }
12         }
13 }
```

Ci-dessous le détail du contructeur de l'objet Message.

Listing 1: Code constructeur Message(int descBr)

```
1  Message::Message(int descBr){
2    //Recupère type de la requête
3          int reception = recv(descBr, &type, sizeof(int), 0);
4          if (reception <= 0) throw descBr;
5
6    //Lit taille msg à lire
7          reception = recv(descBr, &taille, sizeof(int), 0);
8          if (reception <= 0 || taille > 500) throw descBr;//Limite la taille
9
10   //Recupère msg de la requête
11         if (taille > 0){
12         chaine =(char *) malloc(taille);
13         reception = recv(descBr, chaine, taille, 0);
14         if (reception <= 0) throw descBr;
15         }else{
16         chaine = NULL;
17         }
18 }
```

Client Accès concurrents

La listes des employés, “lEmploye” devant rédiger un rapport, et la liste des rapports rédiger, “lPdf” sont des données en accès concurent (lecture/ecriture).

La liste des employés devant rédiger un rapport est modifié par le controleur et est consulté par un employé qui souhaite s'avoir si il doit rédiger un rapport ou non. Ainsi pour maintenir un état cohérent de cette donnée, nous utilisons un verrou pour empêcher les employés d'acceder à la donnée, quand le controleur la met à jour.

Pour les opérations de lecture/ecriture sur les données concurrents nous avons implémenté une section critique.

```
int SubjectClient::ajoutEmploye(){

    pthread_mutex_lock(&mes_verrou->VlEmploye);

        //Tests si l'employe n'est pas deja dans la liste
        for (int i(0); i< mes_verrou->lEmploye.size();i++){
    if ( mes_verrou->lEmploye[i] == (string)msgR.chaine) return -1;
    }
    mes_verrou->lEmploye.push_back((string)msgR.chaine);
    cout<<nom<<" a ajouté \""<<msgR.chaine<<"\" dans la liste des employes"<<

    pthread_mutex_unlock(&mes_verrou->VlEmploye);


        return 0;
}
```