

# Projet de réseau

## 1 Mode d'emploi

Tout d'abord, notre application nécessite l'installation de la bibliothèque [Ncurses](#) qui gère l'interface console. Il suffit ensuite de compiler grâce au Makefile.

L'exécutable du serveur se nomme "serveur" et accepte les paramètres suivants :

- -n nom de l'hôte (par default 'localhost')
- -a adresse ip de l'hôte (par default '127.0.0.1')
- -p numero de port qu'utilisera le serveur d'envoi (par default '13321')
- -ps numero de port secondaire qu'utilisera le serveur de réception (par default '13322')
- -h ou - -help affichage de l'aide

L'exécutable du client se nomme "client" et accepte les paramètres suivants :

- -n nom de l'hôte (par default 'localhost')
- -N nom de l'hôte du serveur (par default 'localhost')
- -a adresse ip de l'hôte (par default '127.0.0.1')
- -A adresse ip du serveur (par default '127.0.0.1')
- -P numero de port qu'utilise le serveur d'envoi (par default '13321')
- -S numero de port qu'utilise le serveur de réception (par default '13322')
- -h affichage de l'aide

Par conséquent, pour lancer l'application en local, il suffit d'appeler les exécutables du serveur et du client sans paramètres. Voici un exemple de ligne de commande pour lancer l'application en réseau :

```
$ serveur -a 192.168.1.131
```

```
$ client -a 192.168.1.96 -A 192.168.1.13
```

Une fois le client lancé, la demande de contrôle de la caméra s'effectue en pressant la touche 'c'. On peut relâcher le contrôle prématurément en appuyant sur 'q'. La direction est donnée à la caméra grâce aux touches directionnelles.

Les exécutables se quittent en envoyant le signal SIGINT (ctrl-c).

## 2 Architecture de l'application

Commençons par examiner l'architecture du serveur. Notre serveur est divisé en deux processus, les deux étant multi-thread. Ainsi, les deux tâches distinctes que sont l'envoi de la grille et la réception des commandes pour le contrôle sont prises en charge par deux processus différents (père/fils). Le fait que chacun des processus soit multi-thread sur permet d'effectuer leurs nombreuses tâches de façon parallèle.



### 3 Protocoles d'échange

#### 3.1 Echanges entre le serveur d'envoi d'images et la partie affichage du client

Le protocole est très simple puisqu'il est à sens unique : on procède simplement à une création de sockets et une demande de connexion côté client (primitive connect). Du côté serveur, il y a acceptation de la connexion (primitive accept), création d'un thread secondaire et envoi de données. Les données en question (grille) sont envoyées par la primitive send sous forme d'un tableau de caractères.

À la déconnexion, le client ferme sa socket ce qui induit un retour d'erreur pour le thread du serveur d'envoi. Ce thread se termine donc proprement. La figure 2 (p.3) est une schématisation possible de ces échanges.

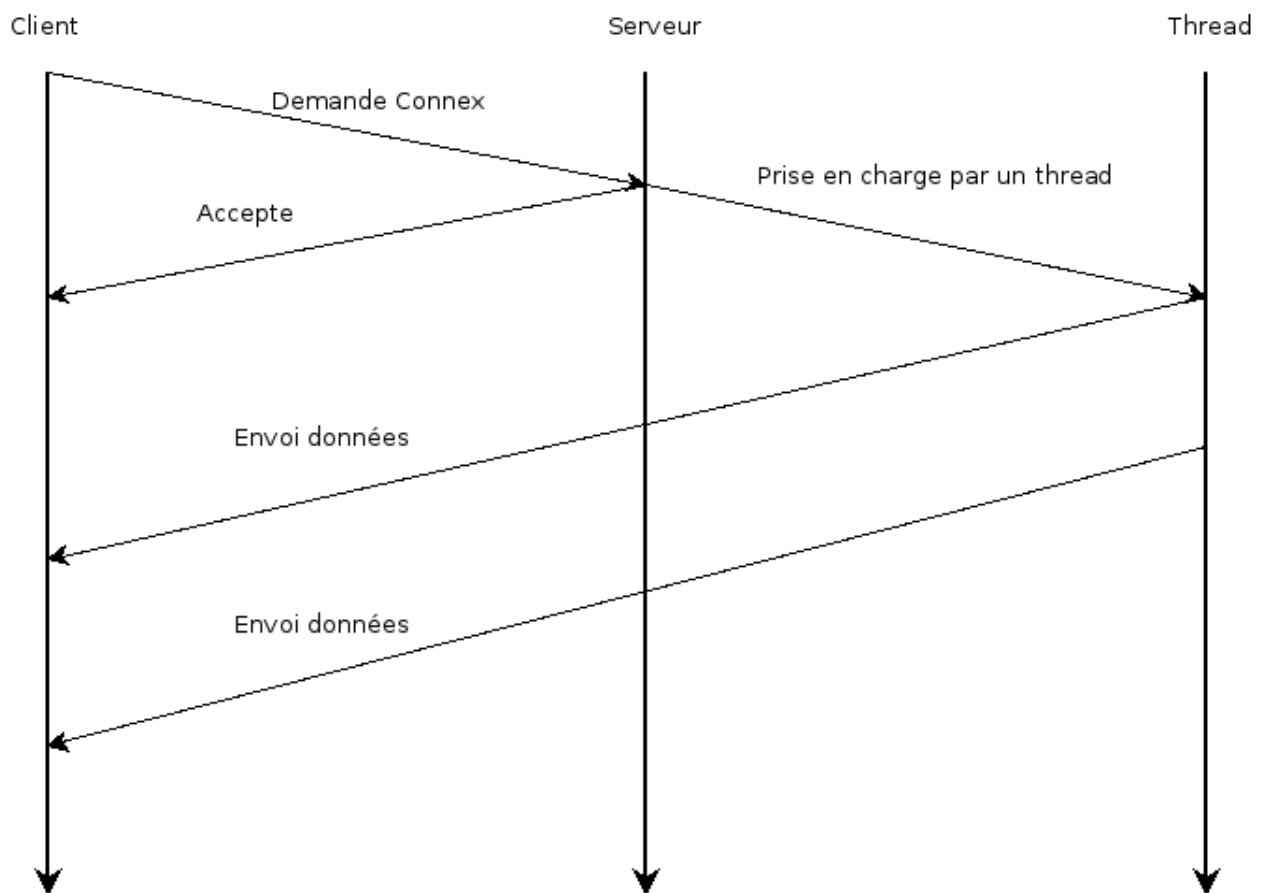


FIGURE 2 – Protocole d'échange entre le serveur d'émission d'images et le client

#### 3.2 Echanges entre le serveur de contrôle de la caméra et le client

Le protocole est un petit peu plus compliqué, en effet une fois la connexion au serveur de contrôle de caméra demandée (primitive connect), celle ci est automatiquement

acceptée si le nombre de clients déjà connectés est inférieur à une constante (20 en l'occurrence). Les sockets correspondant aux clients sont sauvegardées dans une file d'attente qui permet de savoir en temps réel le nombre de clients en attente du contrôle de la caméra<sup>1</sup>.

Une fois le contrôle libre, le serveur prend la première socket de la file et crée un thread qui prend en charge le client. Pour garder le client en attente, une fois la connexion effectuée celui ci attends un message de la part du serveur lui servant d'*accusé de prise en charge*. Cet accusé correspond au temps maximum accordé au client pour le contrôle.

Les échanges ayant lieu ensuite sont à sens unique du client détenant le contrôle vers le thread de gestion des clients du serveur de contrôle. Le client envoie le caractère 'q' pour quitter ou un caractère correspondant à la direction qu'il désire que la caméra prenne. La libération du contrôle de la caméra est représentée par une mise à zéro de la variable globale `cam_moving` côté serveur. Le client quant à lui est mis au courant de la fin de sa session par fermeture de sa socket du côté serveur.

La figure 3 (p. 4) est une schématisation possible de l'échange.

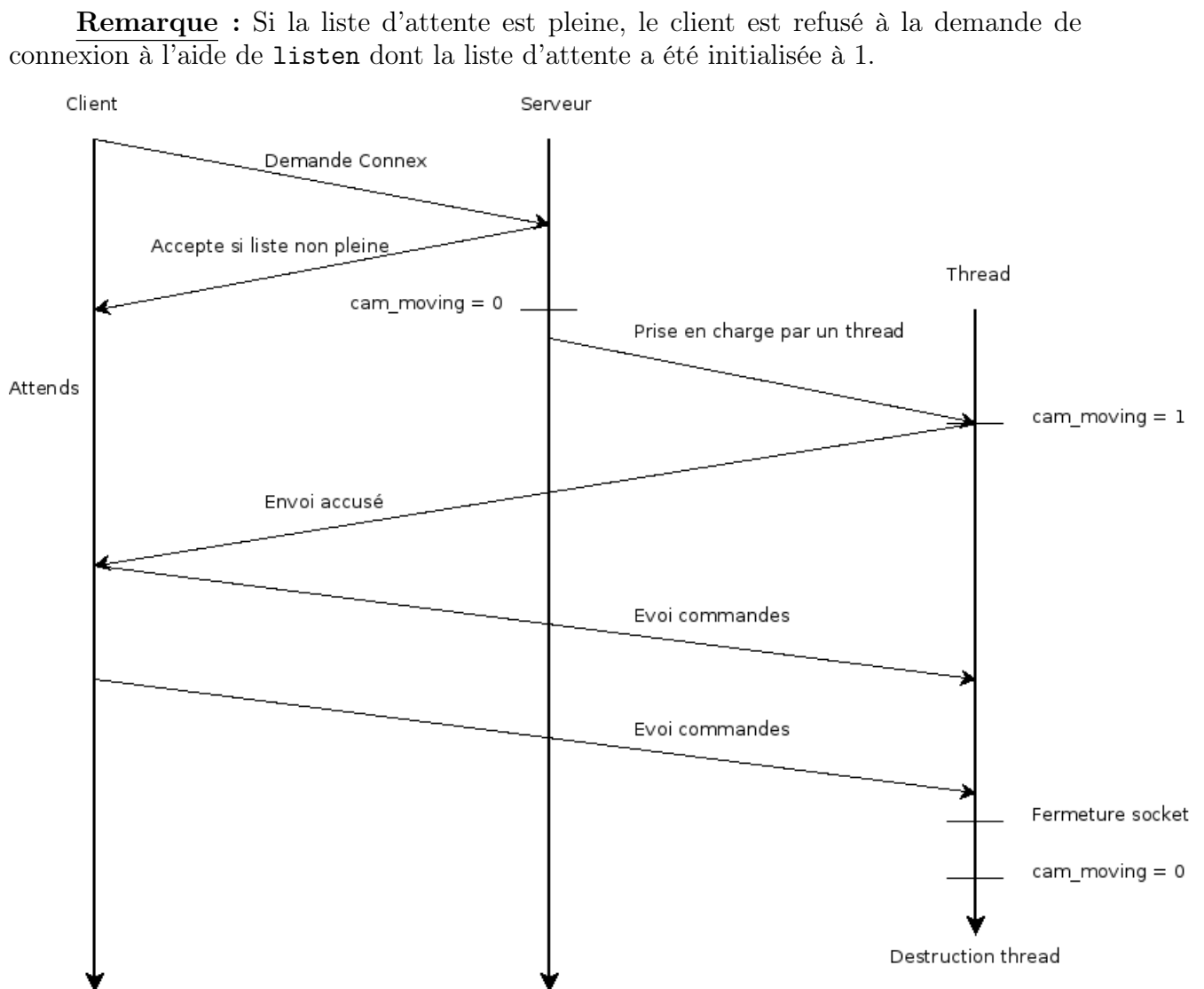


FIGURE 3 – Protocole d'échange entre le serveur de controle de la caméra et le client

1. Ce nombre permet de calculer le temps maximum de contrôle de la caméra pour le client en cours d'utilisation

### 3.3 Echanges entre les deux composantes du serveur

Les échanges entre les deux parties du serveurs sont très limités, ils se résument à un segment de mémoire partagée contenant la grille et à l'échange de signaux en cas d'interruption du serveur. Par soucis de rapidité (surtout avec un grand nombre de client), les accès à la grille ne sont pas protégés puisque le thread de gestion de client du serveur de contrôle est le seul à y accéder en écriture, le sémaphore est donc superflu.

## 4 Schémas algorithmiques

Vous trouverez les schémas algorithmiques des différentes composantes de notre application ci-dessous : figures 4 à 9.

Notons que le serveur de contrôle réinitialise la grille après 50 mouvements de caméra effectués.

## 5 Difficultés et solutions

Nous avons été confrontés à quelques problèmes lors de la mise en œuvre de notre application. Les principaux sont détaillés ci-dessous.

Lorsqu'un thread secondaire utilise les appels systèmes `recv` ou `send` en mode connecté alors que la socket a été fermée de l'autre côté, un signal `SIGPIPE` est émis. Ce signal doit être intercepté par le thread principal sous peine de terminer l'application.

L'interface console `Ncurses` supporte normalement le redimensionnement de la fenêtre de terminal. Cependant, nous n'avons pas trouvé le temps nécessaire pour prendre en charge le redimensionnement de la fenêtre dans laquelle est lancé le client.

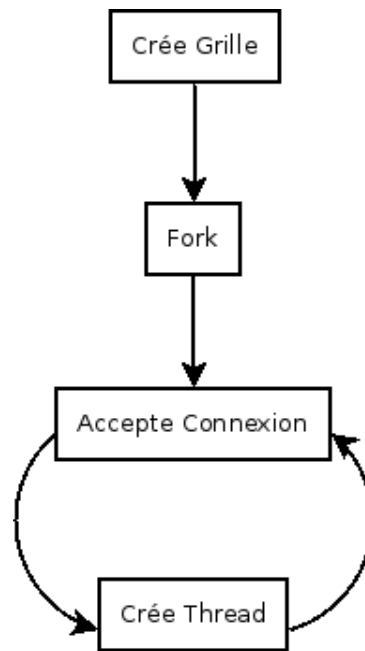


FIGURE 4 – Thread principal du serveur d’envoi d’images

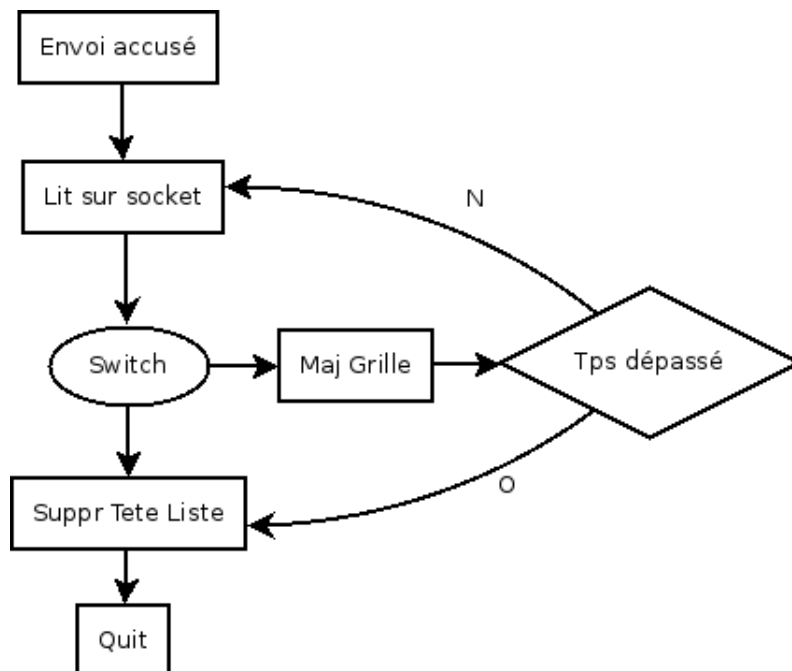


FIGURE 5 – Thread subsidiaire de gestion d’un client du serveur d’envoi d’images

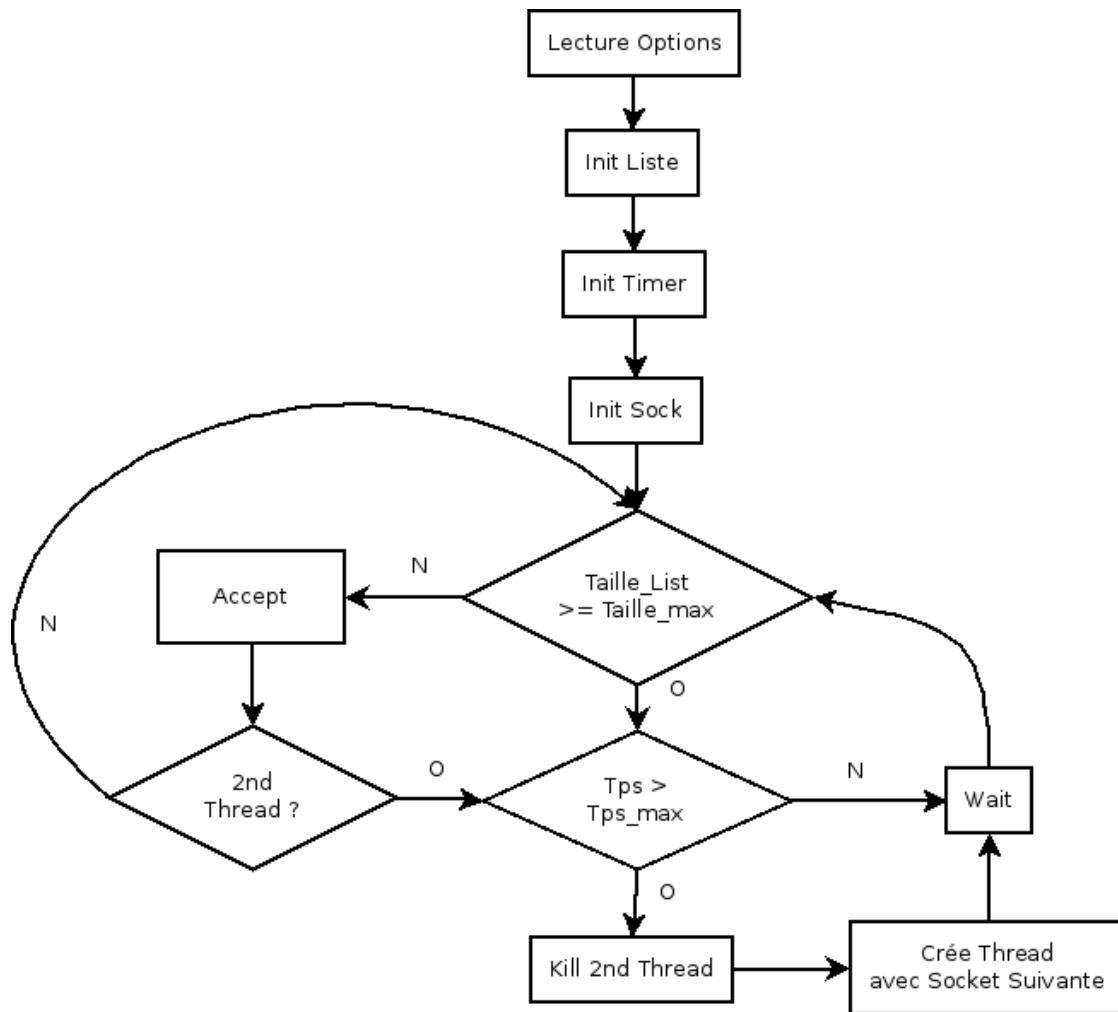


FIGURE 6 – Thread principal du serveur de contrôle de la caméra

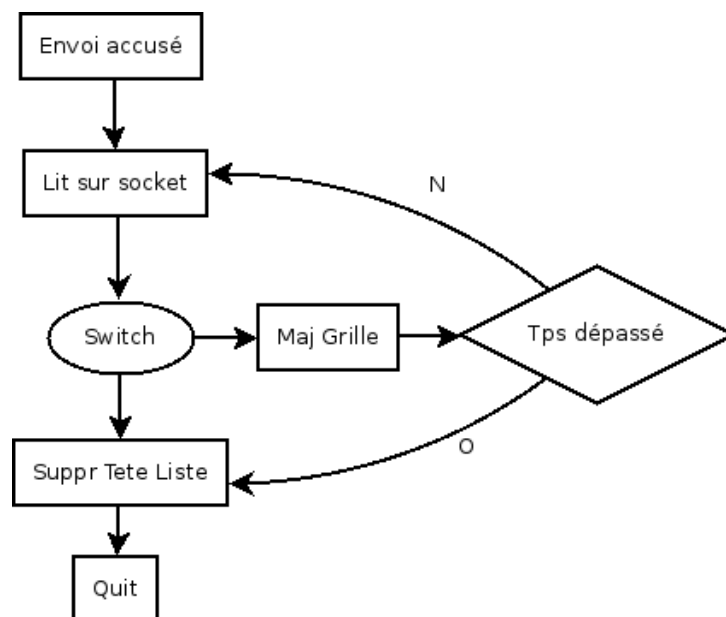


FIGURE 7 – Thread secondaire du serveur de contrôle de la caméra



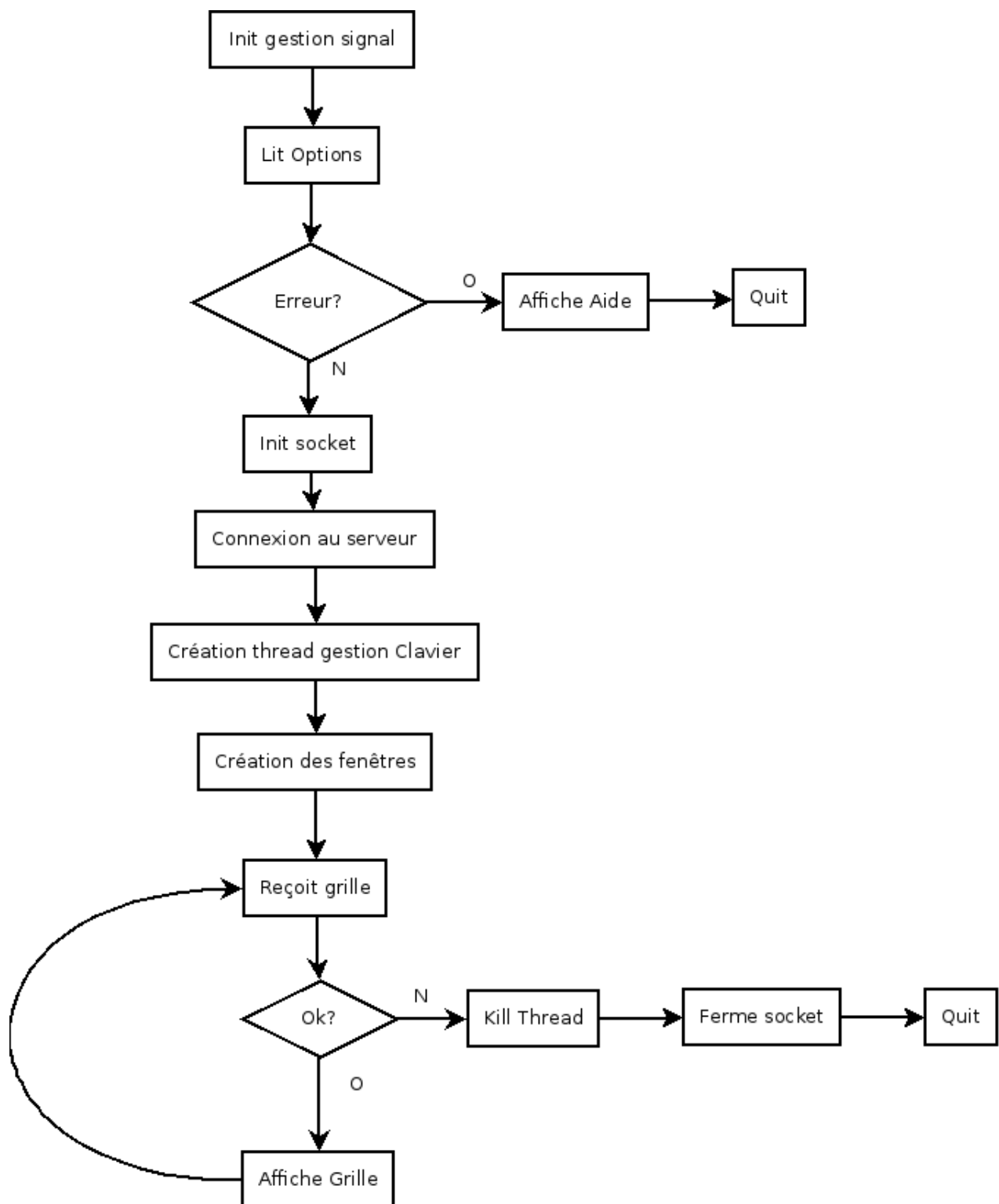


FIGURE 8 – Thread principal du client

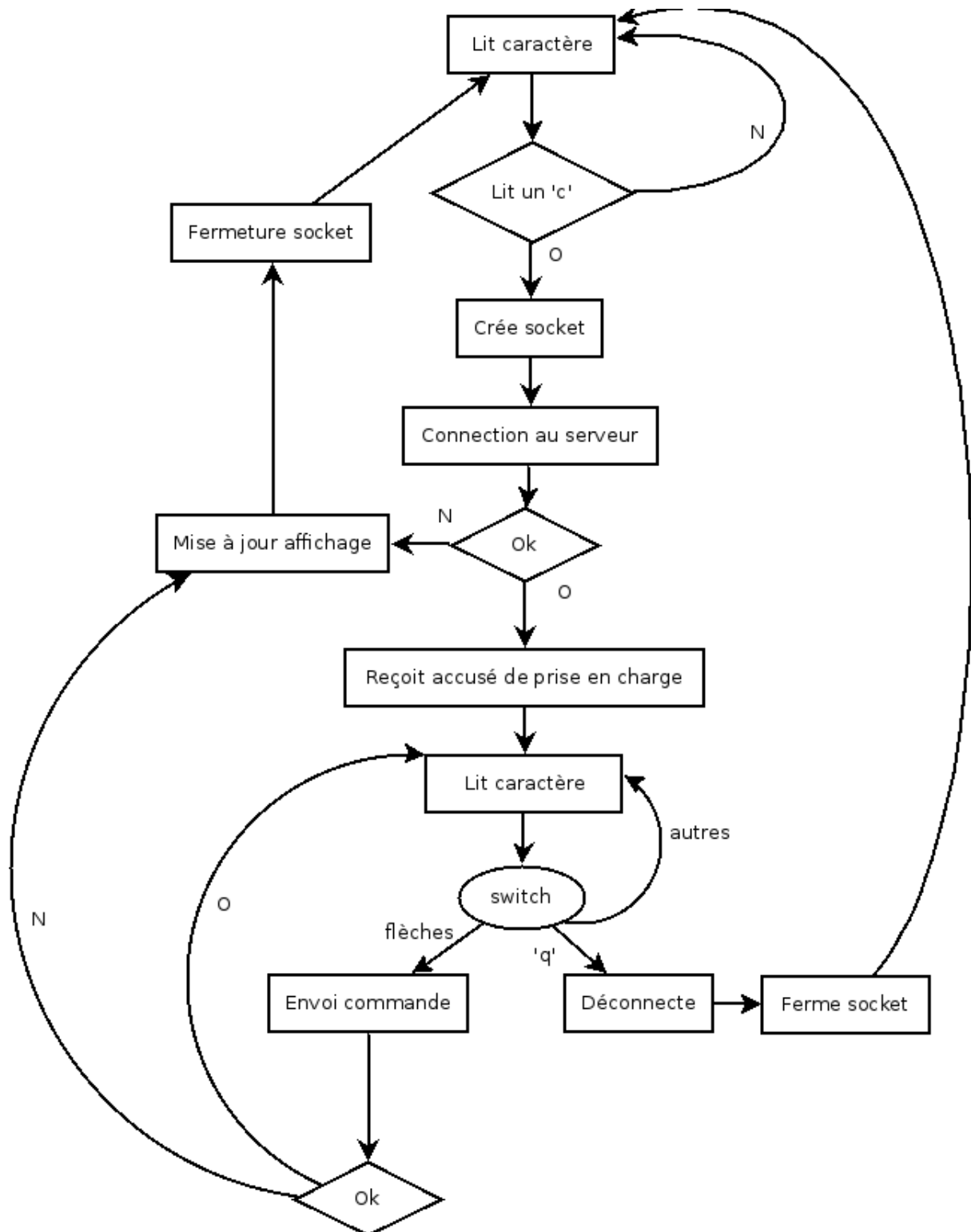


FIGURE 9 – Thread de contrôle de caméra du client