```cpp
#include "AdminServer.h"

void _admin_login(ClientSocket sock);
void _actions(ClientSocket sock);

// Serveur assurant la gestion des commandes exécutées par les administrateurs
void *admin_server(void* arg)
{
    IniParser properties("admin_server.ini");

    int port = atoi(properties.get_value("port").c_str());
    int n_clients = atoi(properties.get_value("n_clients").c_str());

    with_server_socket(port, n_clients, _admin_login);
    return NULL;
}

// Gère la connexion d'un administrateur
void _admin_login(ClientSocket sock)
{
    admin_protocol packet;
    sock.receive<char>((char *) &packet.type);
    printf("Nouvel administrateur\n");

    if (packet.type == admin_protocol::LOGIN) {
        IniParser admins("admins.ini");

        sock.receive_string(packet.content.login.user);
        sock.receive_string(packet.content.login.password);

        const char* pass = admins.get_value(
                string(packet.content.login.user)
        ).c_str();

        if (strcmp(packet.content.login.password, pass) == 0) {
            // Mot de passe correct
            printf(
                "Administrateur connecte en tant que %s\n",
                packet.content.login.user
            );
            packet.type = admin_protocol::ACK;
            sock.send<char>((char *) &packet);
            return _actions(sock);
        } else {
            packet.type = admin_protocol::FAIL;
            sock.send<char>((char *) &packet);
        }
    }
}

// Gère les actions de l'administrateur
void _actions(ClientSocket sock)
{
    admin_protocol packet;
    sock.receive<char>((char *) &packet.type);

    switch (packet.type) {
    case admin_protocol::LCLIENTS:
        // Envoie une liste d'entiers. Le premier est le nombre de clients,
        // et le second les terminaux occupés
        printf("Envoi de la liste des clients a l'administrateur\n");
        sprintf(
            packet.content.list_clients.n_clients, "%ld",
            connected_clients.size()
        );
        sock.send_string(packet.content.list_clients.n_clients);

        for (list<int>::iterator it(connected_clients.begin());
```

```
            it != connected_clients.end(); it++) {
            sprintf(
                packet.content.list_clients.client, "%d",
                *it
            );
            sock.send_string(packet.content.list_clients.client);
        }
        return _actions(sock);
        break;
    case admin_protocol::PAUSE:
        printf("L'administrateur met en pause le serveur pour 20 secondes\n");
        pthread_mutex_lock(&mutex_pause);
        sleep(20);
        pthread_mutex_unlock(&mutex_pause);
        printf("Fin de la pause\n");

        return _actions(sock);
        break;
    case admin_protocol::STOP: {
        sock.receive_string(packet.content.stop);
        int sleep_time = atoi(packet.content.stop);
        printf(
            "L'administrateur arrete le serveur dans %d secondes\n", sleep_time
        );
        sleep(sleep_time);
        printf("Arret du serveur\n");
        exit(EXIT_SUCCESS);
        break;
    }
    default:
        break;
    }
}
```