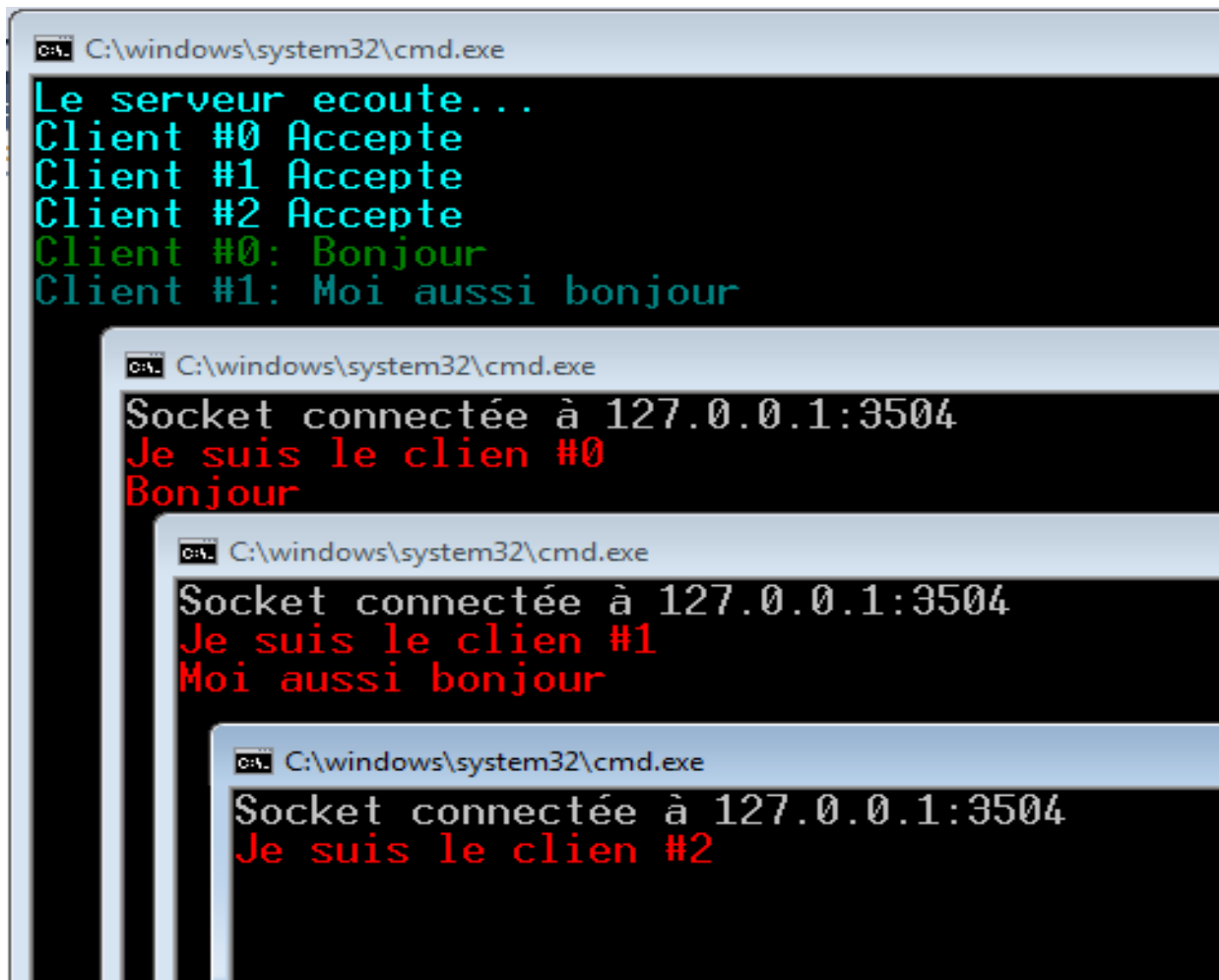


## Travail 4

Dans ce travail vous devez réaliser un système de chat multi-clients unidirectionnel, le chat doit se faire dans un seul sens, du client vers le serveur. Le serveur doit gérer les sessions des clients par des Threads. Vous devez également implémenter un mécanisme de synchronisation au niveau de votre serveur basé sur les *sémaphores* pour limiter le nombre de clients qui peuvent chatter avec le serveur. Ce mécanisme ne doit pas empêcher un client de se connecter à un serveur, mais plutôt de chatter avec lui. En d'autres termes, un client peut se connecter à un serveur, mais doit attendre son tour pour pouvoir envoyer des messages. Comme vous pouvez le voir, dans l'exemple suivant, le mécanisme de synchronisation autorise uniquement les **deux** clients Client 0 et Client 1 à écrire, client 2 est en attente, il ne peut pas écrire, mais il est connecté comme mêmes.



```
C:\windows\system32\cmd.exe
Le serveur ecoute...
Client #0 Accepte
Client #1 Accepte
Client #2 Accepte
Client #0: Bonjour
Client #1: Moi aussi bonjour

C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #0
Bonjour

C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #1
Moi aussi bonjour

C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #2
```

Dès qu'un client se connecte au serveur, le serveur doit lui envoyer un message pour lui donner son numéro (dans l'exemple ci-dessus, le message est : **Je suis le client # ?**). Le client qui n'a pas l'autorisation de chatter avec le serveur, il ne pourra pas également écrire sur sa propre console à lui.

Au niveau du serveur chaque client doit être géré avec une couleur différente. Dès qu'un client se déconnecte, le serveur doit attribuer son numéro au prochain client qui se connecte. Dans l'image ci-dessous le client 1 s'est déconnecté. Le nouveau client qui s'est connecté à eu le numéro 1. Le client 2, qui était en attente, a été autorisé à chatter, après qu'une place vient de se libérer. Maintenant nous avons les clients 0 et 2 qui ont l'autorisation d'écrire, le client 1 ne l'a pas.

```
Le serveur ecoute...
Client #0 Accepte
Client #1 Accepte
Client #2 Accepte
Client #0: Bonjour
Client #1: Moi aussi bonjour
Client #1 Deconnecte
Client #2: Je peux ecrire maintenant
Client #1 Accepte
```

```
C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #0
Bonjour
```

```
C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #2
Je peux ecrire maintenant
```

```
C:\windows\system32\cmd.exe
Socket connectée à 127.0.0.1:3504
Je suis le clien #1
```

Il faut faire la différence entre un client connecté et un client autorisé à chatter. Tous les clients peuvent se connecter sans aucune contrainte, mais juste un nombre limité qui seront autorisés à chatter. Ce nombre est égal à 2 dans notre précédent exemple. La gestion du nombre de clients autorisés à chatter doit se faire avec les sémaphores. Pour la gestion du nombre de clients connecté, vous pouvez utiliser un vecteur comme mentionné dans le code de départ proposé à la fin de ce document. Toutes les exigences du TP2 et TP3 seront maintenues (l'envoi des gros fichiers, le cryptage ...). Vu que le chat est uni directionnel, du client vers le serveur, vous devez identifier les messages des différents clients par leur numéro et une couleur. Dans l'exemple ci-dessus, tous les messages du client0 sont de couleur verte et précédée par le texte : "Client #0: " pour les fichiers, vous devez inclure le nom du client dans le nom du fichier afin d'identifier la

source d'envoi. Si, par exemple, le client 0 envoie le fichier "test.txt", dans le serveur vous allez le nommer "Client0-test.txt"

Le serveur doit être développé en C++ et le client en C#.

Ce travail compte pour 12.5% de la note finale. Il est à remettre au plus tard le 13 décembre. Aucune remise n'est autorisée après cette date.

# Code de départ pour votre serveur

```
#include <winsock2.h>
...
struct MesClients
{
    int id;
    SOCKET socket;
};
const int MAX_CLIENTS = 5;
std::vector<MesClients> clients(MAX_CLIENTS);
int main();
DWORD WINAPI ThreadClient(void* pParam)
{
    MesClients* new_client = reinterpret_cast<MesClients*>(pParam);
    std::string msg;
    while (...)
    {
        if (new_client->socket != 0)
        {
            int iResult = recv(...);

            if (...)
            {
                // afficher le message du client
                ...
            }
            else
            {
                //afficher que le client est déconnecté
                closesocket(new_client->socket);
                closesocket(clients[new_client->id].socket);
                clients[new_client->id].socket = INVALID_SOCKET;
                break;
            }
        }
    }

    return 0;
}
```

```

int main()
{
    ...
    int NumClient;
    SOCKET server_socket = INVALID_SOCKET;
    HANDLE my_thread[MAX_CLIENTS];
    //Création, bind de la socket
    ...

    listen(server_socket, 2);

    //Initialiser le vecteur clients
    for (int i = 0; i < MAX_CLIENTS; i++)
    {
        clients[i] = !!
    }

    while (...)
    {

        SOCKET nouveau = INVALID_SOCKET;
        nouveau = accept(server_socket, NULL, NULL);
        if (nouveau == INVALID_SOCKET) continue;
        ...

        //Donner un numéro au client (NumClient) et l'insérer dans le vecteur
        //clients
        ...

        if (!!)// si le serveur n'est pas plein
        {
            //Afficher que le client #? Est accepté
            //Et envoyer un message au client pour lui donner son numéro.

            //Créer un Thread pour le client
            my_thread[NumClient]=CreateThread(NULL,
            0,ThreadClient,&clients[NumClient],0, 0);
        }
        else
        {
            //Afficher serveur plein et envoyer un message au client pour
            // l'informer
        }
    } //fin du while

    //fermer la socket
    closesocket(server_socket);

    //fermer les clients
    for (int i = 0; i < MAX_CLIENTS; i++)
    {
        CloseHandle(my_thread[i]);
        closesocket(clients[i].socket);
    }
    WSACleanup();

    return 0;
}

```