

Client socket corrigé



```
Canvas client
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>

int main() {
    int sd;
    struct sockaddr_in serveraddr;
    char request[256];

    //creation du socket
    sd=socket(AF_INET, SOCK_STREAM,0);
    if(sd<0) printf("erreur de creation du socket");
    //preparation de l'adresse de serveur
    serveraddr.sin_family=AF_INET;
    serveraddr.sin_port=1500;
    serveraddr.sin_addr.s_addr=INADDR_ANY;
    //demande de connexion
    if(connect(sd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) 0)
        printf("erreur de connexion");
    //lecture et envoie du message
    printf("entrer a message");
    scanf("%s", request);
    send(sd, request, strlen(request), 0);
    //fermeture du socket
    close(sd);
}
```



Socket Client

```
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>

int main() {
    int sd;
    struct sockaddr_in serveraddr;
    char request[256];

    // Creation du socket
    sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd < 0) {
        perror("Erreur de creation du socket");
```

```
return 1;
}
```

```
// Preparation de l'adresse du serveur
```

Voici une version corrigée et améliorée de votre code pour un client socket. Des ajustements ont été faits pour améliorer la gestion des erreurs et éviter des problèmes potentiels tels que les dépassements de tampon.

```
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(1500);
serveraddr.sin_addr.s_addr = INADDR_ANY;
```



```
// Demande de connexion
```

```
if (connect(sd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
```

```
    perror("Erreur de connexion");
```

```
    pourquoi ça arrive
```

```
    return 1;
```

```
}
```



```
// Lecture et envoi du message
```

```
printf("Entrez un message: ");
```

J'ai corrigé et amélioré votre code pour plusieurs raisons :

```
scanf("%255s", request); // Limite la lecture pour éviter un dépassement de tampon
```

```
if (send(sd, request, strlen(request), 0) < 0) {
```

```
    perror("Erreur lors de l'envoi du message");
```

```
    close(sd);
```

- **Problème :** Vous utilisez `if(connect(...) 0)` au lieu de `if (connect(...) < 0)` pour vérifier si la connexion échoue. Cela peut entraîner un comportement inattendu.

- **Solution :** Utilisez `connect` correctement avec une vérification explicite.

```
close(sd);
```

- **Problème :** Si une erreur survient dans la création du socket ou dans la connexion, votre programme continue.

- **Solution :** Ajout de `perror` pour afficher des messages d'erreur précis et arrêter le programme en cas d'échec.

2. Prévention des dépassements de tampon :

- **Problème :** Vous utilisez `scanf("%s", request)` sans limiter la longueur du texte saisi par l'utilisateur, ce qui peut provoquer un dépassement de tampon si l'utilisateur entre plus de 255 caractères.
- **Solution :** Remplacement par `scanf("%255s", request)`, qui limite la taille des données saisies.

3. Initialisation de la structure `sockaddr_in` :

- **Problème :** Vous ne nettoyez pas la structure `serveraddr` avant de l'utiliser. Cela peut entraîner un comportement imprévisible si des valeurs résiduelles sont présentes.
- **Solution :** Utilisation de `memset` pour initialiser correctement la structure.

4. Bonne pratique pour les constantes :

- **Problème :** Vous n'utilisez pas `htons` pour convertir le numéro de port dans l'ordre correct pour le réseau.
- **Solution :** Utilisation de `htons(1500)` pour garantir que le numéro de port est interprété correctement.

5. Fermeture propre du programme en cas d'échec :

- **Problème** : En cas d'erreur après la création du socket, le socket n'est pas fermé proprement.
- **Solution** : Ajout de `close(sd)` pour garantir la libération des ressources.

Ces corrections visent à rendre votre programme plus robuste, sécurisé et conforme aux bonnes pratiques de programmation réseau.



```
Canvas serveur
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>

int main() {
    int sd, sb, sl, sa, size;
    char message[256];
    int cssd[10];
    struct sockaddr_in serveraddr;

    //creation du socket
    sd=socket(AF_INET, SOCK_STREAM,0);
    if (sd<0) printf("erreur de creation du socket");

    //preparation de l'adresse
    serveraddr.sin_family=AF_INET;
    serveraddr.sin_port=1500;
    serveraddr.sin_addr.s_addr=INADDR_ANY;

    //attachement du socket a l'adresse
    sb=bind(sd, (struct sockaddr *)&serveraddr, sizeof(serveraddr));
    if(sb<0) printf("erreur d'attachement d'adresse");;
    //ecoute des connexion
    sl=listen(sd,1);
    if (sl<0) printf("erreur d'ecoute");

    //accepter une demande de connexion
    sa = accept(sd, 0, 0);

    //reception de message et son affichage avec taille
    while ((size =recv(sa, message, sizeof(message),0))>0)
    {
        message[size] = '\0';
        printf("Message Recu: %s(%d)\n", message,size);
    }
    //fermeture de la connexion
    close(sa);
    //fermeture du socket
```

```
close(sd);
}
```



Socket Server

```
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <arpa/inet.h>

int main() {
    int sd, sb, sl, sa, size;
    char message[256];
    struct sockaddr_in serveraddr;

    // Creation du socket
    sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd < 0) {
        perror("Erreur de creation du socket");
        return 1;
    }
}
```

// Preparation de l'adresse

Voici une version corrigée et améliorée de votre code serveur :

```
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(1500);
serveraddr.sin_addr.s_addr = INADDR_ANY;
```

Changements apportés :

1. Gestion des erreurs :

- o Attachement du socket a l'adresse
- o Utilisation de `perror` pour afficher des messages d'erreur explicites.
- o Fermeture du socket en cas d'erreur critique pour éviter les fuites de ressources.

2. Initialisation de la structure :

- o Ajout de `memset` pour initialiser correctement la structure `sockaddr_in`.

3. Validation des données reçues :

- o Ajout de `message[size] = '\0'` pour s'assurer que la chaîne est correctement terminée.

4. Bonne pratique pour les constantes :

- o Conversion correcte du numéro de port avec `htons`.

5. Messages informatifs :

- o Ajout de messages pour indiquer le statut du serveur.

```
printf("Serveur en attente de connexions...");
```

Ce code est plus robuste et conforme aux bonnes pratiques pour les applications réseau.

```
// Accepter une demande de connexion
```

```
sa = accept(sd, NULL, NULL);
```

```
if (sa < 0) {
```

```
    perror("Erreur lors de l'acceptation de connexion");
```

donner moi dans un tableau les différents version de windows server 2012 r2 et leur

caractéristiques

```
}
```



```
printf("Connexion acceptee.");
```

```
// Reception de message et affichage avec taille
```

```
while ((size = recv(sa, message, sizeof(message) - 1, 0)) > 0) {
```

Voici un tableau qui résume les différentes éditions de Windows Server 2012 R2 ainsi que leurs caractéristiques principales :

| Edition | Usage | Limite de licences d'accès client (CAL) | Caractéristiques principales |
|--|---|---|--|
| <pre> message[size] = '\0'; printf("Message reçu: %s (%d octets)\n", message, size); if (size < 0) { perror("Erreur de reception"); } // Fermeture de la connexion close(sd); </pre> Windows Server 2012 R2 Standard | Pour les environnements physiques ou virtualisés limités. | Requiert des CALs | <ul style="list-style-type: none"> - Jusqu'à 2 machines virtuelles (VM). - Inclut toutes les fonctionnalités de base. - Idéal pour petites entreprises. |
| <pre> return 0; } </pre> Windows Server 2012 R2 Datacenter | Pour les environnements fortement virtualisés ou cloud. | Requiert des CALs | <ul style="list-style-type: none"> - Nombre illimité de machines virtuelles. - Toutes les fonctionnalités de Standard. - Idéal pour grandes entreprises. |
| Windows Server 2012 R2 Essentials | Pour les petites entreprises jusqu'à 25 utilisateurs et 50 périphériques. | Inclus, pas besoin d'acheter des CALs | <ul style="list-style-type: none"> - Pas de limite de rôles serveur. - Ne prend pas en charge Hyper-V. - Simplifié pour les petites entreprises. |
| Windows Server 2012 R2 Foundation | Pour les très petites entreprises (15 utilisateurs maximum). | Inclus, pas besoin d'acheter des CALs | <ul style="list-style-type: none"> - Ne prend pas en charge la virtualisation. - Limité à 15 utilisateurs. - Moins de rôles serveur pris en charge. |

Comparaisons :

1. **Standard vs Datacenter** : La principale différence est le nombre de machines virtuelles prises en charge (2 pour Standard, illimité pour Datacenter).
2. **Essentials** : Idéal pour les petites entreprises sans besoin de virtualisation complexe ou gestion avancée des rôles.
3. **Foundation** : Très limité en fonctionnalités et utilisateurs, uniquement pour les besoins les plus simples.

Si vous avez des besoins spécifiques, je peux vous aider à identifier l'édition qui convient le mieux à votre cas.



les versions standards



Windows Server 2012 R2 Standard est conçu pour des environnements à faible virtualisation ou des environnements physiques. Voici ses caractéristiques principales :

| Caractéristique | Détails |
|--------------------------------------|---|
| Public cible | Moyennes et grandes entreprises avec des besoins modérés en virtualisation. |
| Virtualisation | - Supporte jusqu'à 2 machines virtuelles (VM) par licence. - Rôles physiques ou hybrides. |
| Fonctionnalités principales | - Inclut toutes les fonctionnalités des rôles serveur (Active Directory, DNS, DHCP, etc.). |
| Licences d'accès client (CAL) | Requiert des CALs pour chaque utilisateur ou appareil accédant au serveur. |
| Nombre maximal de processeurs | Supporte jusqu'à 2 sockets physiques (processeurs physiques). |
| Mémoire maximale | Jusqu'à 4 To de RAM . |
| Prix/licensing | Par serveur, avec licence CAL séparée pour les utilisateurs ou périphériques. |
| Hyper-V | Intégré avec des fonctionnalités de virtualisation de base. |
| Utilisation recommandée | - Environnements modestes nécessitant quelques VM. - Serveurs physiques avec rôles multiples. |

Cas d'utilisation

- Petite entreprise qui souhaite héberger des applications ou des bases de données.
- Organisation ayant besoin de quelques machines virtuelles pour des tests ou un déploiement limité.
- Infrastructure avec une faible densité de serveurs.

Si vous avez un environnement où une virtualisation illimitée est requise, l'édition Datacenter serait préférable.



les version core ect



Windows Server 2012 R2 propose deux options principales pour l'installation, indépendamment de l'édition choisie (Standard, Datacenter, Essentials, ou Foundation) : **Server Core** et **Server with a GUI** (Graphical User Interface). Voici un tableau détaillant les différences et caractéristiques de ces versions :

| Version | Description | Avantages | Inconvénients |
|--------------------------|--|---|---|
| Server Core | Une version minimale de Windows Server sans interface graphique. | <ul style="list-style-type: none"> - Consomme moins de ressources système. - Réduit la surface d'attaque pour la sécurité. - Moins de mises à jour requises. - Idéal pour les serveurs distants ou spécialisés (DNS, DHCP, Active Directory). | <ul style="list-style-type: none"> - Pas d'interface graphique pour la configuration. - Configuration uniquement via PowerShell ou en ligne de commande. - Courbe d'apprentissage plus élevée. |
| Server with a GUI | La version complète avec interface graphique classique (Desktop Experience). | <ul style="list-style-type: none"> - Facile à utiliser et à configurer via des outils graphiques. - Convient aux | <ul style="list-style-type: none"> - Consomme plus de ressources système. - Plus de mises à jour nécessaires. - Plus grande surface |

| Version | Description | Avantages | Inconvénients |
|---------------------------------|--|---|--|
| | | administrateurs habitués aux interfaces Windows. | d'attaque pour les menaces de sécurité. |
| Minimal Server Interface | Une version intermédiaire où certaines fonctionnalités graphiques sont disponibles, mais sans le bureau complet. | <ul style="list-style-type: none"> - Réduit la consommation de ressources par rapport à la version GUI complète. - Permet une transition progressive vers Core. | <ul style="list-style-type: none"> - Moins complet qu'un environnement GUI. - Moins léger que Server Core. - Peut être plus complexe à gérer. |

Comparaison des fonctionnalités

| Fonctionnalité | Server Core | Server with a GUI | Minimal Server Interface |
|----------------------------|--|--|--|
| Interface graphique | Non | Oui | Partiellement |
| Consommation de ressources | Faible | Élevée | Modérée |
| Outils de gestion | PowerShell, ligne de commande | Outils graphiques, PowerShell | PowerShell, outils graphiques partiels |
| Idéal pour | Serveurs distants, DNS, DHCP, Active Directory | Applications nécessitant une interface utilisateur | Transition vers Server Core |

Transition entre les versions

Windows Server 2012 R2 permet de passer de **Server Core** à **Server with a GUI** (et inversement) après l'installation en utilisant les outils de gestion tels que **PowerShell** pour ajouter ou supprimer les fonctionnalités liées à l'interface graphique.

Commande PowerShell :

- Pour ajouter l'interface GUI à Server Core : `