TIC-DVC3 / Bomberman / Client - Serveur

Modalités

Dépôt	https://rendu-git.etna-alternance.net/module-6426/activity-35936/group-760845
Arborescence du dépôt	./client_serveur/
Fichiers requis	*Makefile*, code source
Durée	1 run
Correction	Correction à distance
Langage autorisé	Langage C
Bibliothèques autorisées	libC, SDL, SDLTTF, SDLImage
Environnement	• Debian Buster • Version du standard C : c17
Flags de compilation	`-Wall -Wextra -Werror`

Objectifs

- Utiliser le C dans un paradigme de programmation réseau.
- Utiliser des sockets pour établir la communication client/serveur.
- Développer un serveur en C

Consignes

Ce sujet est découpé en plusieurs parties pour faciliter votre progression.

Comme précisé dans les modalités, vous ne devez rendre qu'un fichier *Makefile* et votre code source.

Votre Makefile doit compiler votre programme et générer un exécutable nommé `bomberman`. Votre Makefile doit comporter les règles suivantes : `all`, `clean`, `fclean` et `re`.

Une fois ce sujet réalisé, vous aurez en votre possession une base solide afin de réaliser votre bomberman.

Vous devez faire une demande de validation seulement lorsque l'intégralité de ce sujet est réalisée.

Mon premier serveur

Vous devez réaliser un serveur qui répond aux critères suivants :

- il prend en paramètre un numéro de port et écoute sur ce dernier,
- il doit écouter sur toutes les interfaces,
- il doit afficher les messages reçus sur la sortie standard,
- il doit renvoyer "OK" au client une fois que le message est reçu et affiché.

Les échanges doivent correspondre aux schémas suivants : client -> serveur -> client.

Mon premier client

Vous devez maintenant réaliser un client qui se connecte au serveur précédemment réalisé et lui envoie des messages.

Le client doit prendre en paramètre :

- l'IP du serveur distant,
- le port de connexion du serveur distant.

Une fois lancé, votre programme doit lire l'entrée standard et envoyer les messages écrits dans cette dernière au serveur.

Pour tester votre client en local vous pouvez aussi utiliser `netcat` comme suit :

```
netcat -l -p ;
```

Si votre client fonctionne correctement, vous devriez voir les messages apparaître sur la sortie standard de votre serveur ou de la commande `netcat`.

Plusieurs clients

Votre serveur fonctionne désormais avec un client. Il faut maintenant penser plus grand.

Vous devez faire évoluer votre serveur afin qu'il puisse :

- accepter plusieurs clients (4 maximum),
- répondre au client qui lui envoie un message,
- traiter les messages client qu'importe le client qui envoie le message en premier,
- gérer les déconnexions des clients,
- accepter des clients à la volée :
  - Le serveur doit pouvoir traiter les messages des clients connectés même si le nombre de clients connectés est inférieur au maximum (moins de 4).
  - votre serveur doit pouvoir accepter un nouveau client même si il a déjà commencé à échanger avec les autres,
  - si un client se déconnecte, un nouveau doit pouvoir prendre sa place.

Aide

Voir le *man* de `select()`.

Client et SDL

Pour résumer : vous disposez d'une application utilisant la SDL qui affiche une fenêtre et gère les contrôles du clavier, d'un client minimaliste et d'un serveur capable d'héberger plusieurs clients.

Vous devez désormais fusionner votre client et votre application graphique.

Votre programme doit désormais proposer un menu graphique après l'initialisation de votre fenêtre permettant à l'utilisateur de saisir l'adresse IP et le port du serveur.

Suite à cela, votre client doit envoyer un message au serveur pour les évènements clavier listés :

- flèche du haut : envoyer "up",
- flèche du bas : envoyer "down",
- flèche de droite : envoyer "right",
- flèche de gauche : envoyer "left",
- touche action : envoyer "bomb".

Serveur et SDL

Comme spécifié dans le sujet général, l'objectif final est d'avoir un seul exécutable qui sert de client et de serveur.

Votre programme doit donc inclure votre serveur.

Pour cela, ajouter un menu principal avec les choix suivants : "Se connecter à une partie" et "Héberger une partie",

Si l'utilisateur choisit de se connecter à une partie, le comportement mis en place précédemment est utilisé.
Si l'utilisateur choisit d'héberger une partie, le programme affiche un champ pour saisir le port. À la validation de celui-ci, le serveur se lance.