# Jeu Streams - Analyse de l’application

## Infos administratives

Numéro de groupe: 18

Etudiant 1 : Dubois Josué

Etudiant 2 : Rousseaux Diego

Etudiant 3 : Drion Antoine

## Communication

**Messages via les sockets :**

- Les types de messages sont `INSCRIPTION\_REQUEST`, `INSCRIPTION\_OK`, `TILE`, `PLAYED`, `CANCEL\_GAME`, `END\_GAME`, `SCORE`, `RANKING`.

**Messages via les pipes :**

- Les types de messages sont `PIPE\_END\_GAME`, `PIPE\_SCORE`, `PIPE\_TILE`, `PIPE\_PLAYED`.

**Scénarios d'exécution :**

1. Inscription des joueurs :

- Le serveur écoute les demandes d'inscription des joueurs via des sockets.

- Lorsqu'un joueur envoie une demande d'inscription (`INSCRIPTION\_REQUEST`), le serveur accepte ou refuse l'inscription et envoie un message correspondant (`INSCRIPTION\_OK` ou `CANCEL\_GAME`) via le socket.

2. Déroulement du jeu :

- Une fois le nombre requis de joueurs inscrits, le serveur commence la partie.

- Le serveur envoie des tuiles (`TILE`) aux clients via les pipes.

- Les clients envoient leurs coups (`PLAYED`) au serveur via les sockets.

- Le serveur envoie des messages (`PIPE\_PLAYED`) aux enfants via les pipes pour indiquer la fin du tour.

- Ce processus se répète jusqu'à la fin de la partie.

3. Fin de partie et classement :

- Après la fin du jeu, le serveur envoie un message de fin de partie (`END\_GAME`) aux clients via les sockets.

- Les clients envoient leur score (`SCORE`) au serveur via les sockets.

- Le serveur envoie le score des joueurs aux processus enfants via les pipes (`PIPE\_SCORE`).

- Le serveur classe les joueurs en fonction de leur score et enregistre le classement dans une mémoire partagée.

4. Arrêt du serveur :

- L'arrêt du serveur peut être déclenché par un signal SIGINT (`CTRL+C`).

- Le serveur ferme les sockets et les pipes, libère les ressources et termine son exécution.

## Découpe de l’application

### Modules

**client.c :**

Permets de gérer la partie client du programme

1. void endProgram(int code) :
   * Description : Cette fonction prend un code de sortie en argument et termine le programme en fermant d'abord le socket réseau (sockfd) puis en appelant exit() avec le code spécifié.
2. void placeTile(int tile, int index) :
   * Description : Cette fonction prend en paramètre une tuile (tile) et un index, puis place la tuile dans le tableau stream à l'index spécifié.
3. void printStream(int remaining) :
   * Description : Cette fonction prend en paramètre le nombre de tuiles restantes et affiche le tableau stream avec les indices correspondants.
4. int calculateScore(int arr[], int size) :
   * Description : Cette fonction prend en paramètres un tableau d'entiers représentant le flux de tuiles jouées et sa taille, puis calcule et retourne le score total en fonction des valeurs définies dans scoreValues.
5. int main(int argc, char const \*argv[]) :
   * Description : Cette fonction est la fonction principale du programme. Elle gère l'inscription au serveur de jeu, la réception des tuiles, le placement des tuiles, le calcul du score et l'affichage du classement final.

**Game.c :**

Logique de jeux

1. void sendTile(Player \*players, int size, int tile) :
   * Description : Cette fonction envoie une tuile spécifiée à tous les joueurs en écrivant la tuile dans le tube parentToChild de chaque joueur.
2. void waitForPlayed(Player \*players, int size) :
   * Description : Cette fonction attend que tous les joueurs jouent leur tour en lisant les messages des tubes childToParent de chaque joueur. Elle affiche également le nom du joueur qui a joué.
3. void endGame(Player \*players, int size) :
   * Description : Cette fonction envoie un message de fin de partie à tous les joueurs en écrivant dans le tube parentToChild de chaque joueur.
4. void waitForScore(Player \*players, int size, PlayerIpc \*playerIpcs) :
   * Description : Cette fonction attend que tous les joueurs envoient leur score final en lisant les messages des tubes childToParent de chaque joueur. Elle stocke ensuite les informations de chaque joueur dans un tableau de structures PlayerIpc et affiche leur nom et leur score.

**Ipc.c :**

Gestion de la mémoire partagée

1. void createIpc() :
   * Description : Cette fonction crée et initialise les IPC nécessaires. Elle crée une mémoire partagée en utilisant la fonction sshmget() avec les clés spécifiées (RAKING\_SHM\_KEY), une taille donnée (SHARED\_MEMORY\_SIZE), et des permissions spécifiées (IPC\_CREAT | IPC\_EXCL | PERM). Elle crée également un sémaphore en utilisant la fonction sem\_create() avec les mêmes clés, une valeur initiale de 1, et les mêmes permissions.
2. void detachIpc() :
   * Description : Cette fonction détache et détruit les IPC. Elle récupère l'identifiant de la mémoire partagée et du sémaphore en utilisant sshmget() et sem\_get() respectivement, puis les détruit en utilisant sshmdelete() et sem\_delete().
3. PlayerIpc \*getSharedMemory() :
   * Description : Cette fonction récupère et retourne un pointeur vers la mémoire partagée. Elle utilise sshmget() pour obtenir l'identifiant de la mémoire partagée, puis attache cette mémoire partagée en utilisant sshmat(). Enfin, elle retourne un pointeur vers la mémoire partagée de type PlayerIpc.

**Admin1.c :**

Permets de supprimer les IPC en cas de crash involontaire de l’app

1. void checkUsage(int argc, char \*argv[]) :
   * Description : Cette fonction vérifie si le programme est utilisé correctement en vérifiant le nombre d'arguments de la ligne de commande (argc) et en comparant le premier argument (argv[1]) avec les options valides. Elle affiche un message d'erreur si l'utilisation est incorrecte.
2. int main(int argc, char \*argv[]) :
   * Description : Fonction principale du programme. Elle prend en charge la logique principale du programme en fonction de l'option passée en ligne de commande. Si l'option est -c, elle crée des IPC. Si l'option est -d, elle détruit les IPC. Si l'option est -s, elle détruit le sémaphore uniquement.

**Message.h :**

Module pour définir les messages qui sera envoyé par les pipes et sockets

1. Code : Énumération définissant différents codes de message utilisés pour identifier le type de message échangé entre les processus. Les valeurs possibles sont les suivantes :
   * INSCRIPTION\_REQUEST : Requête d'inscription d'un joueur.
   * INSCRIPTION\_OK : Confirmation d'inscription réussie.
   * TILE : Tuile à jouer.
   * PLAYED : Confirmation de la tuile jouée.
   * CANCEL\_GAME : Annulation de la partie.
   * END\_GAME : Fin de la partie.
   * PIPE\_END\_GAME : Fin de la partie via un tube (pipe).
   * PIPE\_SCORE : Score envoyé via un tube.
   * SCORE : Score final du joueur.
   * RANKING : Classement des joueurs.
   * PIPE\_TILE : Tuile envoyée via un tube.
   * PIPE\_PLAYED : Confirmation de la tuile jouée via un tube.
2. StructMessage : Structure de message contenant les champs suivants :
   * code : Code du message représentant le type de message.
   * text : Texte associé au message, pouvant être utilisé pour transmettre des informations supplémentaires.
   * value : Valeur associée au message, utilisée par exemple pour représenter une tuile à jouer ou un score.
3. MAX\_CHAR : Constante définissant la taille maximale d'une chaîne de caractères dans la structure de message.

**Network.c :**

Permets de gérer le socket

1. int initSocketClient(char \*serverIP, int serverPort) :
   * Description : Cette fonction initialise un socket côté client. Elle crée un socket en appelant la fonction ssocket(), puis se connecte au serveur spécifié par l'adresse IP et le port passés en paramètres en utilisant sconnect(). Enfin, elle retourne le descripteur de fichier du socket créé.
2. int initSocketServer(int port) :
   * Description : Cette fonction initialise un socket côté serveur. Elle crée un socket en appelant la fonction ssocket(), puis configure l'option SO\_REUSEADDR en utilisant setsockopt() pour éviter l'erreur "Address Already in Use". Ensuite, elle lie le socket au port spécifié en utilisant sbind(). Enfin, elle met le socket en mode écoute en utilisant slisten() avec une taille de backlog spécifiée par la constante BACKLOG, et retourne le descripteur de fichier du socket créé.

### Types Utilisateurs

1. Player :

- Description : Représente un joueur participant au jeu. Contient des informations telles que le pseudo du joueur, le descripteur de fichier du socket associé au joueur et les descripteurs de fichier des tubes (pipes) pour la communication avec le processus enfant.

A screen shot of a computer code

Description automatically generated

2. PlayerIpc :

- Description : Représente un joueur pour la communication via IPC (mémoire partagée). Contient le pseudo du joueur et son score.

A screen shot of a computer code

Description automatically generated

## Makefile

Votre fichier Makefile doit être rédigé (il n’est pas nécessaire de le copier dans ce document).