Projet L3 Système et Réseaux 2021 : The Mind

Introducion

Dans ce compte rendu nous allons voir les différentes manières mises en place pour créer le jeu The Mind.

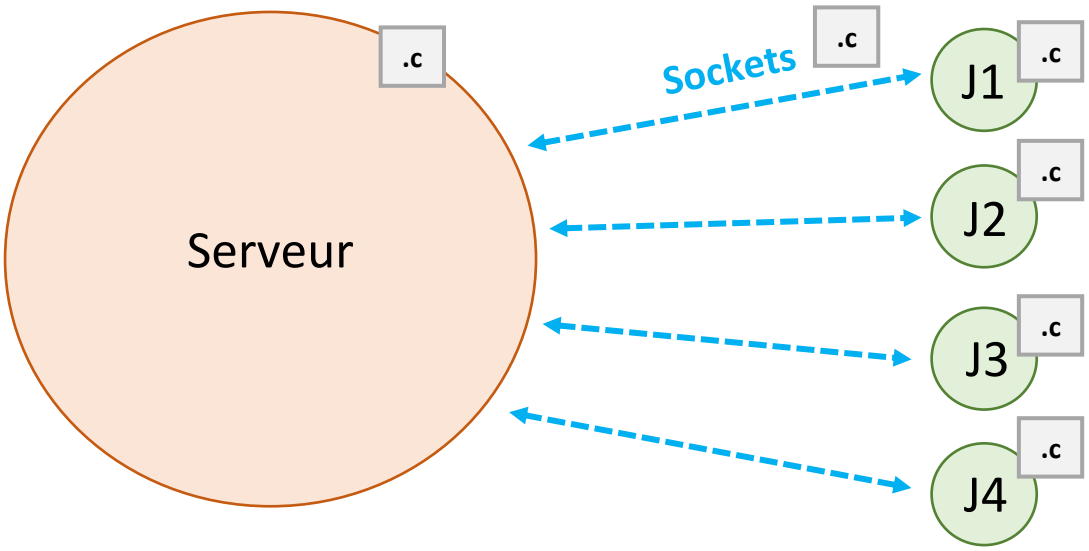
The Mind est un jeu simple. Au premier tour de jeu 4 joueurs reçoivent un numéro allant de 1 à 100. Ils doivent ensuite les poser par ordre croissant sans avoir connaissances des autres numéros. Au tour suivant chaque joueur reçoit 2 numéros, et ainsi de suite.

Nous allons donc voir en premier temps l'architecture créée pour simuler ce jeu, puis les différentes méthodes de communication inter-processus choisies.

L'architecture du jeu

Afin de créer ce jeu j'ai mis en place une architecture simple :

- Un serveur faisant office de gestionnaire de jeu. C'est lui qui reçoit les connexions des clients, qui met en place le jeu et qui s'assure son bon fonctionnement.
- Des clients faisant office de joueurs. Ils se connectent au serveur et peuvent à tout moment envoyer leur(s) numéro(s).



Architecture du jeu

Le serveur possède ainsi plusieurs attributs, notamment un tableau des descripteurs des clients, un tableau de leurs PIDS, et toutes les variables nécessaires au fonctionnement du jeu. Il a également 2 états principaux, le premier étant l'état de *connexion* où il attend que tous les joueurs se connectent ou que l'un d'eux lance la partie. Le deuxième est l'état *jouer* qui, comme son nom l'indique, est l'état en cours lors du déroulement du jeu.

Les joueurs quant à eux possèdent le descripteur du serveur, son PID et d'autres variables pour le jeu.

Projet L3 Système et Réseaux 2021 : The Mind

Les méthodes de communication

Dans le pré-rapport j'avais fait état de certaines méthodes concernant les moyens de communication qui seraient mises en place. Je pensais utiliser des simple kill et trap en shell pour réalisation la séquence de *connexion* des clients au serveur. Cependant je n'ai pas pu effectuer cela ne trouvant pas de moyen de récupérer les PIDS des processus émettant un signal.

Pour les communications dans l'état *jouer* il était question d'utiliser des mkfifo. J'ai également changé d'idée, les mkfifo étant assez limité du fait que chaque read et write soient bloquant pour un processus tant que personne n'est là pour écouter/écrire lorsqu'on qu'on y écrit/écoute.

La méthode retenue a donc été les sockets connectés :

- C'est la méthode la plus couramment utilisée en matière de communication
- Ils permettent de facilement faire passer différents types d'informations

L'état *connexion* et *jouer* se passent donc uniquement en C à l'aide des sockets. Grâce à cela on obtient un programme simple à comprendre et à utiliser.

Lors de l'état *connexion* on utilise cependant un peu de shell. Un client connecté peut demander le lancement du jeu même si le nombre de joueurs maximal n'est pas atteint. Ainsi le serveur lancement des bots pour prendre les places restantes. Ce lancement est réalisé en shell à l'aide d'un simple *./bot*.

On retrouve également dans l'état *jouer* des signaux C. En effet lorsque chaque joueur peut envoyer son nombre les processus sont bloqués sur un *scanf()*. J'ai donc mis au point un système de signal pour permettre de quand même afficher les mises à jour du serveur, comme le fait qu'un autre joueur joue son numéro avant nous.

Pour régler les problèmes d'ordonnancement j'ai profité de la propriété bloquante de la méthode *read()* qui peut également, à l'aide de flags, être non bloquante. Souvent dans le code le serveur ou un client après un *write()* demande un « accusé de réception » avant de continuer toute action. Cela permet de s'assurer que les actions sont effectuées dans le bon ordre. Par exemple lors de la distribution de plusieurs numéros à un même client le serveur doit distribuer le premier numéro, puis attendre que le client lui confirme sa réception avant de faire de même pour le suivant.

Enfin au sujet des robots il avait été dit dans le pré-rapport que j'envisageais de leurs faire utiliser une formule de probabilité prenant en compte les dernier chiffre joué, ainsi que le temps et d'autres critères. J'ai finalement opté pour un fonctionnement plus simple : ils n'utilisent pas une formule de probabilité mais prennent simplement en compte le dernier nombre joué et le temps qui passe.

Chaque seconde ils incrémentent une variable temps de 1. Dès que cette variable est égale à leur nombre minimal ils l'envoient au serveur. Si un joueur joue un nombre alors la variable temps prend automatique cette valeur et continue son incrémentation.

Concernant le nombre de tours de jeu il a été fixé de manière totalement arbitraire à 3, ainsi chaque joueur recevra au maximum 3 chiffres.

Projet L3 Système et Réseaux 2021 : The Mind

Conclusion

Au cours de ce compte rendu nous avons vu la réalisation mise en place pour créer le jeu The Mind à l'aide de trois processus, *serveur*, *client* et *bot*.

L'architecture mise en place est très simple : Un serveur reçoit la connexion de plusieurs clients et s'occupe du bon déroulement du jeu.

Le serveur et les clients communiquent à l'aide de sockets. Au cours de la phase de connexion un client peut lancer la partie ce qui a pour effet de remplir les places restantes par des bots (*./bot*). Au cours du jeu chaque client ou bot peut envoyer son nombre le plus petit lorsqu'il le désire. Les autres joueurs sont alors prévenus à l'aide d'un signal. La partie a été fixée à 3 tours de jeu maximum.

Les problèmes d'ordonnancement entre les processus ont été réglés par la demande d'« accusé de réception » grâce à la méthode *read()* bloquante.