Rapport Final
Projet de Programmation UNIX

El Hadj Ibrahima Dit Dieudonné Traoré **&** Samou Silas Dao

Info4108
Prof: Eric Hervet
Hiver 2024

Résumé du projet : L'objectif est de jouer au jeu "Puissance-4" (Connect-4) contre l'ordinateur. Le projet fait intervenir les notions de duplications de processus (fork()) et de communications entre processus grâce à des tubes (pipe(), read(), write()).
Le projet doit être codé en C/C++ sous Linux.

I. Représentation & mise en place

`char tableau[6][7]`

Le plateau de jeu est représenté par un tableau de caractère bidimensionnel de 6 par 7. Ceci permet une simplicité au niveau de la mise à jour de l'état du jeu suite à un coup. Le caractère 'X' (88 en ascii) représente le joueur, le 'O' (79 en ascii) représente l'ordinateur, l'espace (32 en ascii) représente une case libre.

Le tableau `int T[7]` permet de connaître combien d'éléments les 7 colonnes du `tableau[][]` contiennent. De ce fait, le joueur ou l'ordinateur n'aura qu'à choisir un chiffre entre 0-6 (ou 1-7 pour le joueur) pour jouer à cette position.

L'affichage est assuré par `void afficher(char tableau[][7],int ligne)` (implémenté par Samou), elle affiche chaque élément de `tableau` séparé `||`.

Le premier `fork()` défini le fils #1, qui représentera l'ordinateur; le deuxième `fork()` défini le fils #2, le joueur. Ces deux fils du Père sont de la même génération.

II. Communication

Pour la communication entre les processus fils, nous avons décidé d'utiliser 2 tubes : `q[2]` pour la communication du premier fils vers le second et `p[2]` pour celle du second fils vers le premier fils. Le père n'intervenant qu'à la fin du jeu, c'est-à-dire après que les fils aient terminé leur travail et fait `exit`, nous avons simplement utilisé 2 `wait()` pour faire en sorte que le père attende la fin du jeu.

Lorsque le programme commence, le joueur (fils #2) demande à l'utilisateur d'entrer la position à laquelle il veut jouer (col 1-7), puis vérifie la validité de la position. Il met ensuite `tableau[][]` à jour changeant à la position indiqué l'espace par 'X', le tableau `T[]` est aussi mise à jour. Il vérifie si le joueur à gagner, ou s'il y a égalité (tableau plein) grâce aux fonctions `bool gagner(char tableau[][7], int ligne,int codeAscii)` et `bool pleine(int tableau[],int taille)`. Il écrit ensuite `write(p[1],&n,2)` dans le tube `P[1]` la position où le joueur à jouer afin d'envoyer cette information au fils #1 (l'ordinateur), puis attend une réponse : `read(q[0],&n,2).`

Pendant tout ce temps, fils #1 (l'ordinateur) était en attente de lecture `read(p[0],&n,2)` de l'information de fils #2. Maintenant qu'il à cette information, il met à jour son propre `tableau[][]` avec la position du joueur. Si `n` n'est pas 300 ou 200 (voir Gestion de fin), alors la partie continue. Il calcule ensuite le meilleur coup à jouer en se basant sur les positions actuelles.

Il vérifie d'abord s'il peut gagner:

Si oui, il joue à cette position.

Si non, il vérifie si le joueur peut gagner:

Si oui, il le bloque.

Sinon il met en place la "la stratégie 7"(voir IA).

Si il ne peut pas, il joue au hasard (Note: on pourrait faire en sorte qu'il ne joue jamais au hasard)

Il met son tableau à jour avec cette position ainsi que `t[]`, et vérifie s'il à gagner ou si le `tableau[][]` est plein. On affiche ensuite le tableau (dans le terminal)(l'affichage peut également être faite en fils #2). Enfin, l'ordinateur (fils #1) écrit dans `write(q[1],&n,2)` la position à laquelle il vient de jouer.

Fils #2, le joueur, qui attendait (`read(q[0],&n,2)`) reçoit cette information, vérifie que la partie n'est pas terminée, mets à jour `tableau[][]` et `t[]`. Puis cette boucle en fils #1 et fils #2 recommence jusqu'à la fin de la partie.

La communication entre les fils a été implémentée par El Hadj et la gestion des tableaux par Samou.

III. Gestion de fin

Le jeu se termine lorsque l'un des processus fils gagne ou lorsque le tableau est plein. Lorsqu'un processus fils joue et se rend compte que le tableau est plein, il écrit dans le tube le code `300` puis se termine avec le code de retour `3`. L'autre fils reçoit du tube le code `300`, et se termine à son tour avec le code de retour `3`.

Lorsque la partie est remporté par le fils #1, l'ordinateur, il écrit dans le tube le code `200` puis se termine avec le code de retour `2`. Le fils #1 reçoit le code `200` puis se termine avec le code de retour `2`. Si c'est le contraire, et que c'est le fils #1, le joueur qui gagne le code de communication est `100` et le code de retour est `1`.

Le code de communication entre les fils est arbitraire, nous l'avons choisie car les fils communiquent déjà entre eux avec les chiffres de 0-7, nous voulions des codes unique et simple.

Lorsque les processus se terminent, le père qui était en attente à cause de 2 `wait()`, reçoit le code de retour des deux fils qui est normalement le même. Dû au décalage de 8 bits vers la gauche, le code de retour est comme multiplié par 256D (1 000 000B). Le processus père compare donc le code avec 256D et 512D pour savoir si c'est le joueur ou l'ordinateur qui a gagné, si non c'est une égalité (le code est 768D). La gestion de fin à été implémentée par nous deux et celle des scores à été implémentée par El Hadj.

IV. Pseudo IA

Nous avons implémenté quelques fonctions pour permettre à l'ordinateur de jouer plus intelligemment, short choisirMeilleurCoup(char tableau[][7]) va retourner le meilleur coup à jouer :

```
bool coupGagnant(char tableau[][7], int codeAscii, short& colonne)
```

Cette fonction va vérifier pour s'il ya 3 pions (code Ascii) successif et calculer la position du 4e pion. Mais il ne jouera pas à cette position si la position juste en bas du 4e pion est vide.

		X				
	X	**O**	**X**			
X	**O**	**X**	**O**	**O**		
1	2	3	4	5	6	7

Exemple: Ici l'ordinateur ne jouera pas à la position 4, car cela donnerait la victoire au joueur

		X	**X**			
	X	**O**	**X**			
X	**O**	**X**	**O**	**O**		
1	2	3	4	5	6	7

Exemple: Ici l'ordinateur jouera à la position 4

La fonction sera appelée avec le code ascii 79, pour vérifier d'abord si l'on peut gagner. Puis elle sera appelée de nouveau, avec cette fois le code ascii 80 pour bloquer le joueur s'il est sur le point de gagner.

```
bool coupGagnantSpecial(char tableau[][7], int codeAscii, short& colonne)
```

Est une fonction qui fait suite à `coupGagnant()`, cette dernière déterminera si 3 points sont alignés, ici notre fonction détermine si 2 pions sont alignés avec un autre dans une autre case.

			X			
		X	X			
		O	X			
X	O	X	O	O		
1	2	3	4	5	6	7

Exemple: notre fonction bloquera le joueur en jouant à 2

Cette fonction prend également cette situation en compte :

O	O		X	X		
1	2	3	4	5	6	7

	O					
O	O		X	X	X	
1	2	3	4	5	6	7

Exemple: si on ne fait rien, le joueur pourrait jouer
à la position 6 et gagnerait ensuite le jeu au prochain coup. Mais la fonction prend ce cas en compte et joue exprès à la position 3 avant que le joueur n'aligne ces 3 points. Cette fonction est appelée avec le codeAscii 79 pour l'ordinateur et 88 pour le joueur.

```
bool strategie7(char tableau[][7],short& colonne)
```

C'est une stratégie d'attaque qui consiste à former un 7, s'il y parvient, il gagnera à tous les coups.

	O	O	O			
	X	O	X			
X	O	X	O	O	X	X
1	2	3	4	5	6	7

Exemple: la stratégie est en place, on gagne à tous les coups sur la colonne 5

Ces fonctions ont étés implémentés par Samou.

V. Conclusion

La communication par tube permet de large possibilité, il serait possible d'utiliser un seul tube en jouant peut être avec des `sleep()`. Il sera également intéressant d'ajouter un mécanisme d'attaque pour rendre le joueur ordinateur encore plus intelligent. Ce fut un passionnant projet.