Jackpot

Maxime Lovino Thomas Ibanez Vincent Tournier

29 janvier 2017

1 Introduction

Nous avons réalisé un petit programme de slot machine en utilisant l'architecture multi-threadée suivante :
- Un thread par "roue" de la slot machine
- Un thread pour la gestion de l'affichage sur le terminal
- Un thread pour la gestion du jeu et des signaux qui vont nous servir à jouer

Le thread main servira à lancer tous les autres threads, et nous avons également des constantes globales contenues dans le fichier `const.h`

Vous pouvez trouver en dernière page de ce rapport un flowchart du déroulement du programme.

2 Logique du programme - Etats

Nous définissons dans `const.h` une énumération des différents états dans lequel peut se trouver notre programme, il s'agit des états suivants :

- **QUITTING** : Etat dans lequel se trouve le programme après réception du `SIGQUIT`, tous les threads se termiment et nous quittons le programme, il s'agit donc de notre condition d'arrêt
- **RUNNING** : Etat où les roues sont en train de tourner (au moins une d'entre elles)
- **DONE** : Toutes les roues ont fini de tourner, on affiche le résultat et on attend avant d'afficher le message pour relancer une partie
- **WAITING** : Nous sommes en attente d'une pièce pour commencer la partie

3 Main

Dans le thread main nous allons initialiser les différentes variables qui seront partagées entre les threads, entre autre le tableau où seront stockées les valeurs des différentes roues, ainsi que la variable de condition et la mutex qui serviront à lancer les roues.

Nous allons ensuite créer les threads avec leur arguments, puis les join avant de terminer le programme.

Dans le thread main, nous avons déjà bloqué tous les signaux, de ce fait il n'y aura pas besoin de les bloquer dans les threads enfants de main, c'est-à-dire tous les autres threads de notre programme.

4 Handler

Le thread "Handler", ou contrôleur, va gérer les signaux reçus en utilisant `sigwait()` et mettre à jour l'état du jeu, calculer le gain, le montant de la caisse, etc.

La variable `currentWheel` spécifie quelle est la roue à arrêter lors du prochain `SIGINT`. Nous arrêtons également la roue courante à la réception d'un `SIGALRM`, nous nous servons de `alarm()` pour lancer une alarm de 3 secondes au lancement des roues ainsi qu'après l'arrêt de la roue précedente. Etant donné que le lancement d'une alarm stoppe l'alarm précedente, cela nous permet de faire tourner chaque roue pendant maximum 3 secondes après la précédente, sans empêcher l'utilisateur de l'arrêter avant via un `CTRL+C`. A l'arrêt de toutes les roues, nous utilisons `alarm(0)` pour annuler une éventuelle alarm en cours.

5 Display

Au niveau de l'affichage, nous avons une boucle qui va se terminer lors du passage à l'état `QUITTING` et qui affiche les différentes vues du programme en fonction de l'état courant. A la sortie de la boucle, nous affichons la vue avec le message "Come again soon". Le dernier gain, l'état de la caisse et le type de gain sont uniquements affichés ici, ils sont calculés et stockés par le thread Handler.

6 Wheels

Au niveau des roues, nous avons N booléens pour les N roues de la machines qui sont partagés entre les threads de roues et le handler. Nous avons également une variable de condition (et sa mutex) pour la synchronisation en attente passive.

Si on veut lancer une roue, il suffit donc de mettre son booléen à vraie et de signaler (broadcast dans ce cas) la variable de condition. C'est ce que nous faisons à l'insertion d'une pièce, nous mettons tous les booléens à TRUE et nous broadcastons la variable de condition.

Pour arrêter une roue, nous mettons son booléen à FALSE et la roue va se mettre en attente passive en attendant sur la variable de condition directement après.

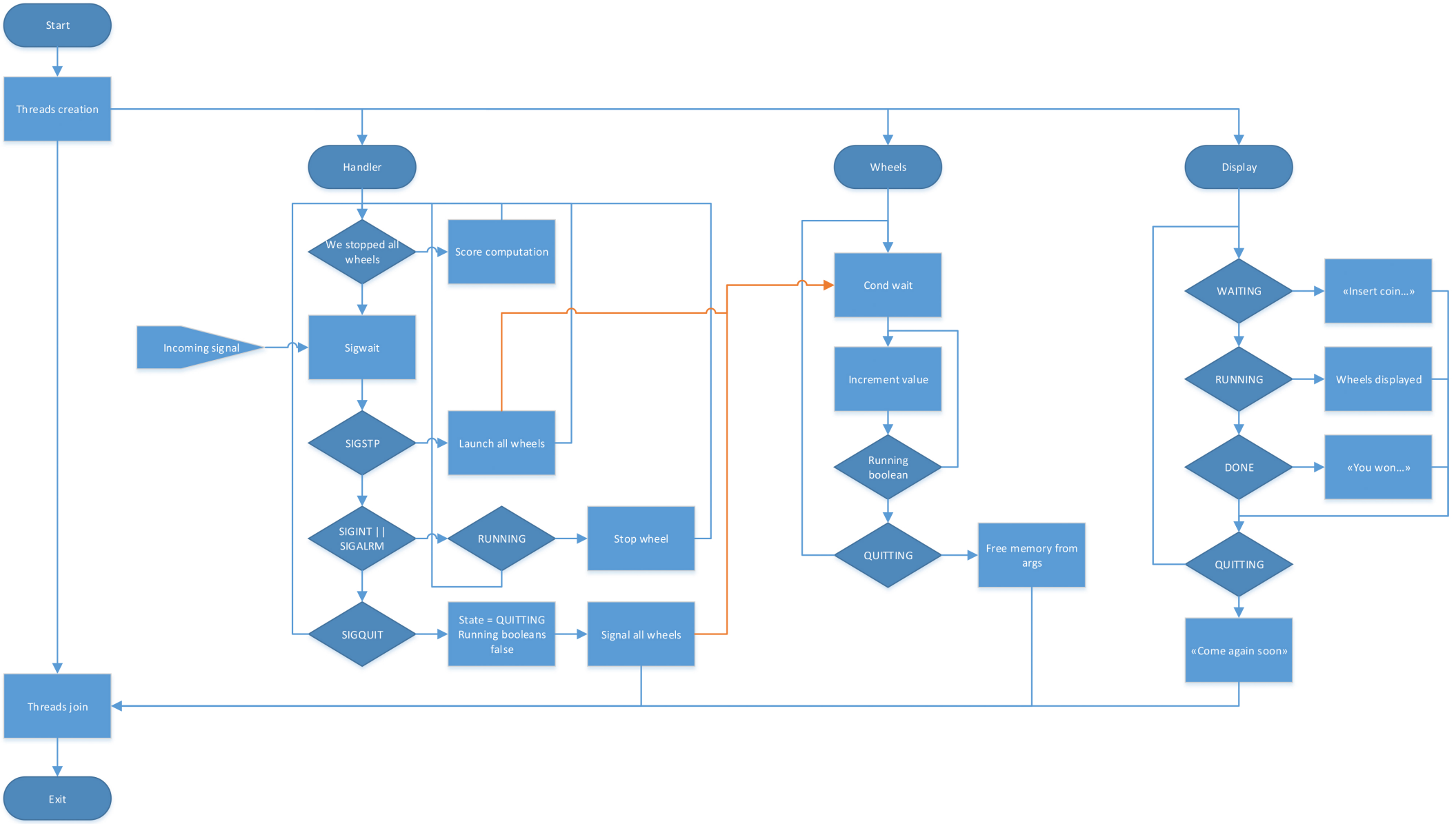
Lors du passage à l'état QUITTING, nous mettons tous les booléens à FALSE et nous broadcastons la variable de condition pour être sûrs que toutes les roues (qui tournaient ou pas) finissent leur thread proprement et puissent être join. Nous libérons la mémoire de leur arguments également à la sortie de leur thread car il s'agissant des seules structures allouées dynamiquement.

7 Bugs

Nous n'avons pas rencontré de bugs dans ce programme.

8 Conclusion

En conclusion, nous avons beaucoup apprécié travailler sur ce projet, qui était fortement intéressant et nous a bien aidés à comprendre l'utilité des signaux et des variables de condition. De façon générale, ces TPs étaient toujours fortement instructifs et un plaisir à réaliser, particulièrement parce qu'ils nous permettaient de travailler en groupe également.


Start
Threads creation
Handler
Wheels
Display
We stopped all wheels
Score computation
Incoming signal
Sigwait
SIGSTP
Launch all wheels
SIGINT || SIGALRM
RUNNING
Stop wheel
SIGQUIT
State = QUITTING
Running booleans false
Signal all wheels
Cond wait
Increment value
Running boolean
QUITTING
Free memory from args
WAITING
«Insert coin...»
RUNNING
Wheels displayed
DONE
«You won...»
QUITTING
«Come again soon»
Threads join
Exit