PROGRAMMATION CONCURRENTE

Jackpot

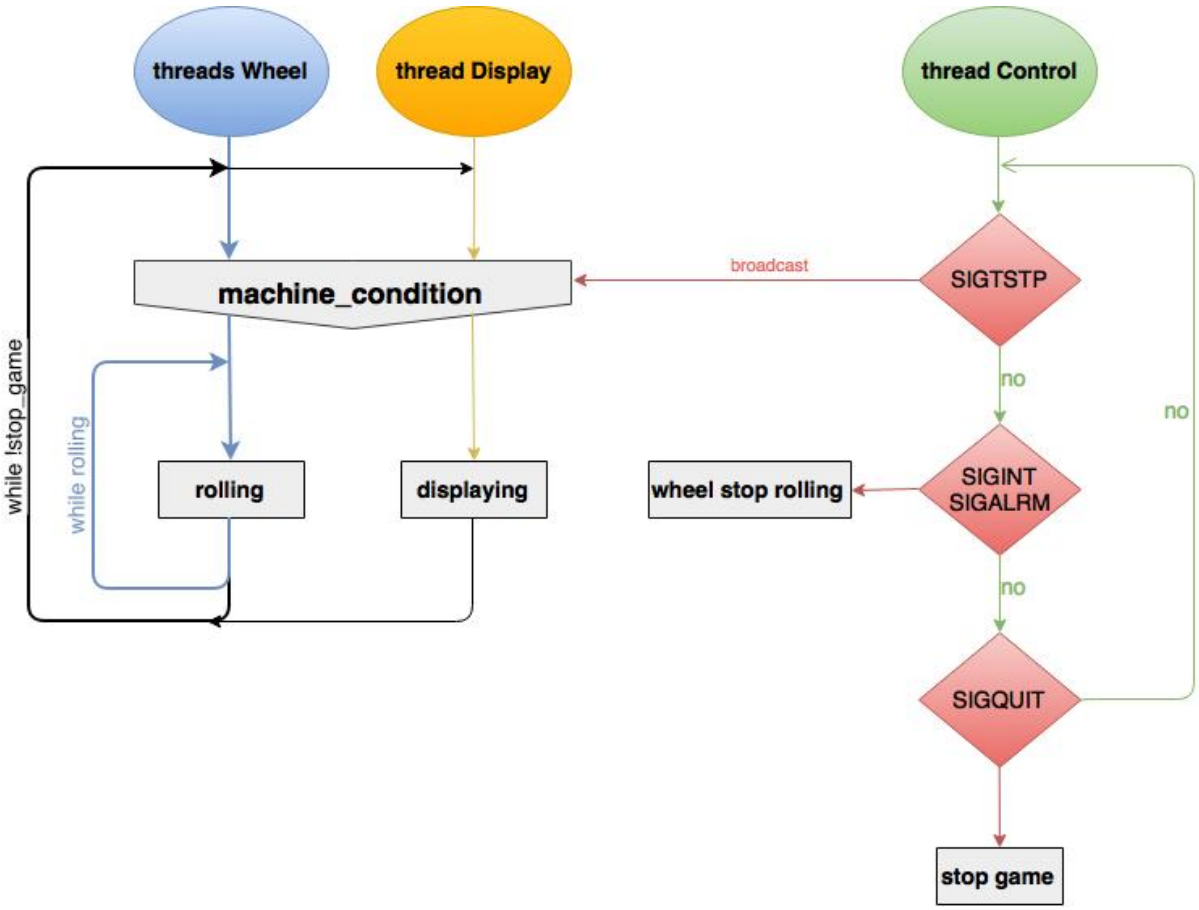
Raed Abdennadher – Orphée Antoniadis – Steven Liatti

Table des matières

Introduction

Ce programme modélise une machine à sous de type « Jackpot », telles que celles que l'on peut trouver dans les casinos et autres salles de jeux. Pour interagir avec la machine, nous avons fait usage de signaux sysytème.

Schéma bloc du programme



Méthodologie

Notre programme est divisé en 3 parties :

- La partie de contrôle qui attend sur les signaux (*control.c*).
- La deuxième partie gérant les threads roues (*wheel.c*).
- Enfin, la partie pour l'affichage (*display.c*).

Partie 1 – Contrôles

Cette partie gère tous les signaux. Quatre signaux sont "capturés" : SIGTSTP, SIGINT, SIGALRM et SIGQUIT.

Le signal SIGTSTP va appeler la fonction *insert_coin* qui va lancer la machine, une nouvelle partie et faire tourner toutes les roues. On utilise pour ça la variable de condition et on fait un broadcast. Une alarme est également lancée pour arrêter la première roue au bout du temps demandé (3 secondes) si le joueur n'a pas lui-même arrêté la roue.

Les signaux SIGINT et SIGALRM appellent la fonction *stop_wheel* qui arrête la roue courante et reset l'alarme. Elle lance une nouvelle alarme pour la roue suivante. Si toutes les roues ont été arrêtées, on reset l'alarme et stoppe la machine. C'est aussi ici que l'on fait le sleep à la fin de la partie.

Le signal SIGQUIT appelle la fonction *exit_game*, qui relance les roues (avec un broadcast) et la machine uniquement pour débloquer les threads d'affichage et des roues et les terminer proprement.

Partie 2 – Roues

Chaque roue est modélisée par un thread *wheel*. Son fonctionnement est le suivant : tant que le jeu n'est pas arrêté (*machine->stop_game*), les roues attendent que le jeu les démarre (*wheel->rolling*) puis font défiler leur *value* (de 0 à 9) et "dorment" le temps dédié à chaque roue. On utilise la variable de condition et le booléen pour attendre sur le moment où les roues doivent se mettre à tourner.

Partie 3 – Affichage

Tous le contenu qui sera afficher à l'écran est géré par le thread display_thred. Ce dernier va tourner tant que le jeu n'est pas arrêté par la réception du signal SIGQUIT (c'est via le test sur la variable booléenne stop_game de la structure machine_t).

Le thread d'affichage fonctionne de la manère suivante :

S'il s'agit de la première partie, alors le message « Insert a coin to start the game... » est affiché. Sinon, le thread va être en attente des 5 secondes de la fin de la partie précédente et puis afficher le message.

Se mettre en attente (passive) grâce à la variable de condition cond de la structure machine_t tant que la partie n'est pas lancée par la réception du signal SIGTSTP.

Après le lancement de la partie, le thread commence à afficher la valeur de chaque roue (tant que la partie est en cours) tout en respectant les fréquences d'affichage de chacune. Pour ce faire, on a implémenté la fonction adapt_frequency qui permet de mettre le thread en sleep avec la bonne valeur.

Remarque : Ici on a utilisé « printf("\e[l;cH"); » (l pour ligne et c pour colonne) pour positionner le curseur de la console au bon endroit.

Une fois que la partie se termine, la fonction won_coins_compute va vérifier les valeurs des roues, calculer le nombre de pièces gangés et l'afficher, et afficher le nombre de pièces qui restent dans la machine.

Se mettre à nouveau en attente (passive) des 5 secondes de fin de partie.

Se libérer de l'attente et afficher à nouveau le message « Insert a coin to start the game... »

Répartition du travail

Steven s'est principalement occupé des threads *wheels* et en partie du thread *control*.

Orphée s'est principalement occupé du thread *control* et de l'initialisation de la machine et des roues.

Raed s'est principalement occupé du thread *display*.

Pour le reste, et surtout pour l'implémentation et la gestion des mécanismes de synchronisation, nous avons collaboré soit par combinaisons de binômes selon les disponibilités de chacun, soit tous ensemble, essentiellement en mettant nos idées sur papier puis en vérifiant la bonne exécution avec le compilateur.