Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

Rapport

PROJET SYSTEME D'EXPLOITATION

le 10 avril 2017,
version 1.1

Clément JANTET
Calliste Ravix
Guillain Le Goff
Maxime Michel

www.ensicaen.fr

TABLE DES MATIERES

TABLE DES FIGURES

1. Introduction

Le projet "Simulation d'Espionnage" est une application visant à recréer l'activité d'un réseau d'espionnage dans une ville. Réalisé dans le cadre d'un cours sur les systèmes d'exploitation, ce travail met l'accent sur l'utilisation du langage de programmation C et explore divers aspects des systèmes d'exploitation UNIX. L'objectif est de démontrer une maîtrise pratique des communications inter-processus, de la gestion des threads, des signaux, et de la mémoire partagée, et d'autres concepts essentiels des systèmes d'exploitation. La simulation intègre divers personnages, tels que des espions, un officier de contre-espionnage, et des citoyens, qui interagissent dans une multitude de situations. Elle englobe plusieurs dimensions, incluant la collecte et le transfert d'informations, des opérations de surveillance, ainsi que les dynamiques d'interaction entre les différents acteurs.

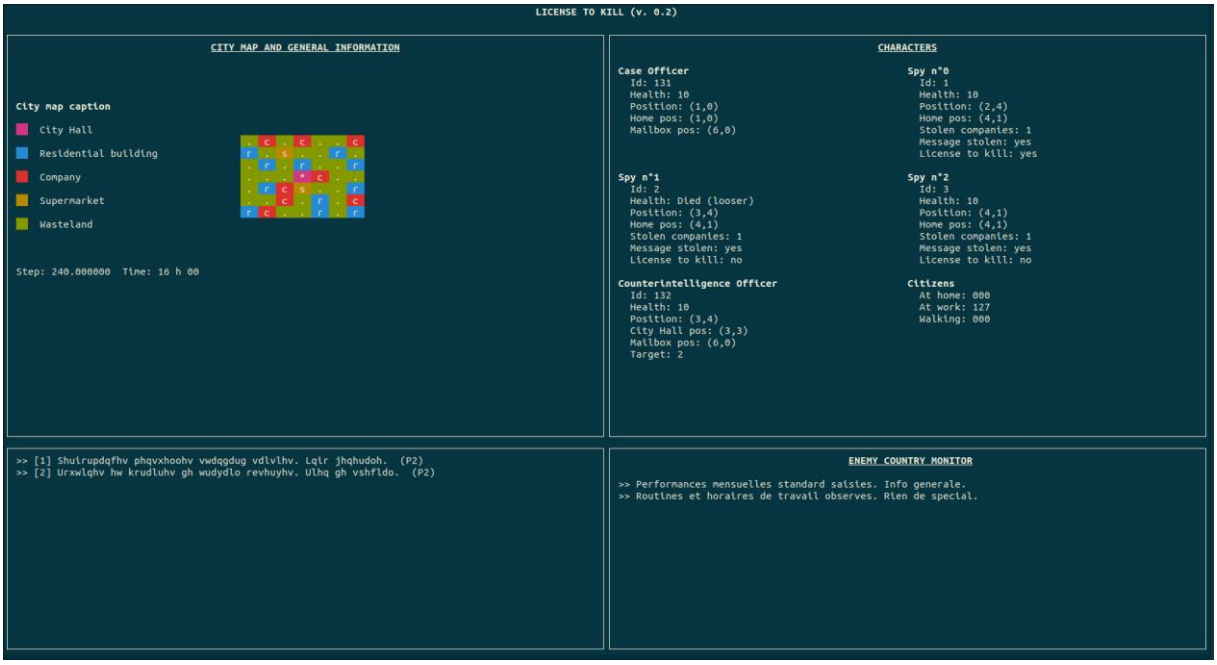

Figure 1 : Execution de notre projet

2. Outils utilisés

2.1. CLion

CLion a été choisi comme éditeur de code principal pour sa simplicité d'utilisation. Sa capacité à gérer des projets complexes et ses outils d'analyse de code ont considérablement facilité le processus de développement.

2.2. Git

Git a permis un suivi efficace des modifications, facilitant la collaboration et la révision du code. L'un des principaux avantages de Git a été sa capacité à gérer les branches de

développement parallèles, permettant à l'équipe de travailler sur différentes fonctionnalités simultanément sans perturber le flux de travail principal.

2.3. ChatGPT

ChatGPT a servi d'outil complémentaire pour la recherche rapide d'informations (liés aux consignes ou à la matière) et le débogage occasionnel. De plus, l'utilisation de ChatGPT pour le débogage et l'apprentissage de concepts a enrichi la compréhension de la matière et a aidé à surmonter certains des défis techniques du projet.

3. Architecture du projet

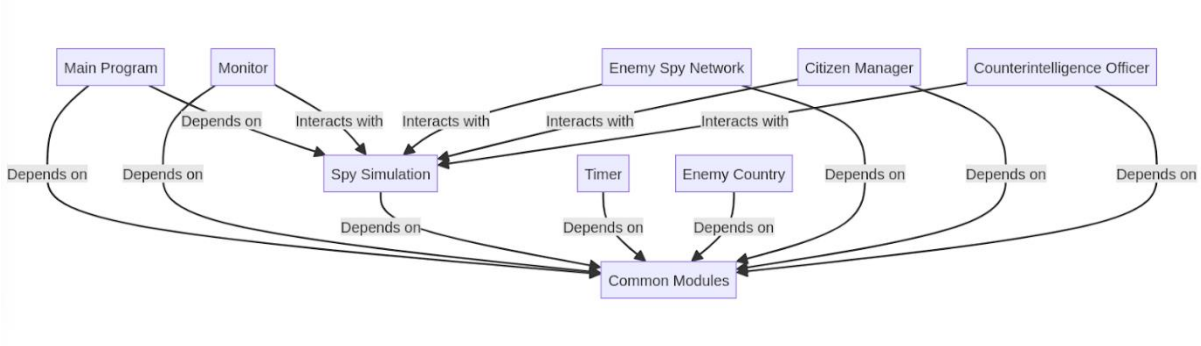

Figure 2 : Graphe d'architecture de notre projet

Le projet de simulation d'espionnage est conçu comme une application multiprocessus. L'architecture repose sur la création de plusieurs processus et threads indépendants, chacun gérant des aspects spécifiques de la simulation. Cette approche permet une séparation claire des responsabilités et une meilleure gestion des ressources système.

3.1. Modularité :

Le projet est divisé en modules distincts, chacun encapsulant une fonctionnalité spécifique. Cette structure modulaire facilite la maintenance, le test, et l'extension du code.

Par exemple, les modules `enemy_spy_network`, `counterintelligence_officer`, et `citizen_manager` gèrent les différents types d'acteurs dans la simulation, tandis que `spy_simulation` orchestre la logique globale de la simulation.

3.2. Processus et Threads :

L'utilisation de processus séparés pour différents composants de la simulation assure une isolation et une sécurité accrues, chaque processus ayant son propre espace mémoire.

Les threads sont utilisés au sein de chaque processus pour gérer les actions simultanées de multiples entités, comme les déplacements des personnages. Cela permet une utilisation efficace des ressources du processeur.

3.3. Communication et Synchronisation :

La communication inter-processus est réalisée à l'aide de files de messages, et mémoire partagée, permettant un échange d'informations efficace entre les processus.

Des mécanismes de synchronisation comme les sémaphores sont utilisés pour éviter les conflits d'accès aux ressources partagées.

4. Description des Composants Principaux

4.1. Main Program (bin/main_program)

Le composant principal de l'application, `Main Program`, sert de point d'entrée et coordonne les interactions entre les différents modules du système. Il initie la simulation en lançant les processus nécessaires et en établissant les mécanismes de communication et de synchronisation.

- Fonctions : Coordonne l'exécution globale, initialise les ressources partagées, et lance les processus de simulation.
- Dépendances : Dépend directement de `Spy Simulation` et des `Common Modules` pour son fonctionnement.

4.2. Monitor (bin/monitor)

Le module `Monitor` est l'interface utilisateur de la simulation, fournissant un affichage en temps réel des activités et des événements qui se déroulent dans la simulation.

- Fonctions : Affiche l'état actuel de la simulation, y compris les positions des personnages et les événements clés.
- Interactions : Interagit étroitement avec Spy Simulation pour obtenir des informations en temps réel et dépend des `Common Modules`.

4.3. Spy Simulation (bin/spy_simulation)

`Spy Simulation` est au cœur de la logique de la simulation, gérant les règles, les états, et les interactions entre les différents personnages et éléments de la simulation.

- Fonctions : Gère la logique de la simulation, traite les actions des personnages, et maintient l'état de la grille de la ville.

- Dépendances : Utilise les `Common Modules` pour des fonctionnalités telles que la journalisation et la gestion de la mémoire.

4.4. Timer (bin/timer)

Le `Timer` gère le flux temporel de la simulation, contrôlant la progression des tours et synchronisant les actions dans le temps.

- Fonctions : Indique le début et la fin des tours, synchronisant les actions des différents modules.
- Dépendances : S'appuie sur les `Common Modules` pour des opérations auxiliaires.

4.5. Enemy Country (bin/enemy_country)

Ce module simule le pays ennemi dans le contexte de l'espionnage, traitant les informations recueillies par les agents d'espionnage.

- Fonctions : Réceptionne et traite les informations envoyées par les espions, simulant la réaction du pays ennemi.
- Dépendances : Dépend des `Common Modules` pour des fonctions telles que la gestion des messages.

4.6. Enemy Spy Network (bin/enemy_spy_network)

Gère les activités des espions ennemis, y compris la collecte de renseignements et les interactions avec d'autres personnages.

- Fonctions : Dirige les mouvements et actions des espions ennemis dans la simulation.
- Interactions : Travaille en conjonction avec `Spy Simulation` pour la logique des espions et utilise les `Common Modules`.

4.7. Citizen Manager (bin/citizen_manager)

Ce module contrôle les citoyens ordinaires dans la simulation, simulant leurs routines quotidiennes et interactions.

- Fonctions : Gère le comportement et les mouvements des citoyens au sein de la simulation.
- Interactions : Interagit avec `Spy Simulation` pour l'intégration des citoyens dans l'environnement global et s'appuie sur les `Common Modules`.

4.8. Counterintelligence Officer (bin/counterintelligence_officer)

Simule les actions de l'officier de contre-espionnage, y compris la surveillance et la neutralisation des menaces.

- Fonctions : Responsable de la détection et de la réaction aux activités d'espionnage dans la simulation.
- Interactions : Collaborer avec `Spy Simulation` pour la surveillance et la réponse aux espions, en utilisant également les `Common Modules`.

4.9. Common Modules (obj/common/*.o)

Les Common Modules incluent des fonctionnalités partagées comme la journalisation (`logger`), la gestion de la mémoire (`memory`), et divers outils (`tools`). Ces modules fournissent des services essentiels utilisés par d'autres composants du projet.

- Fonctions : Offrent des services de base tels que la journalisation des événements, la gestion de la mémoire et des utilitaires généraux.
- Utilisation : Sont intégrés dans presque tous les autres modules pour fournir des fonctionnalités transversales nécessaires à leur opération.

5. Mécanismes de synchronisation et de communication

5.1. Files de Messages :

Les files de messages offrent un moyen flexible de communiquer entre processus. Elles sont utilisées pour transmettre des données structurées, comme les messages codés envoyés par les espions à leur pays d'origine. La file de messages permet une communication asynchrone et sécurisée entre les agents et leur réseau.

5.2. Mémoire Partagée :

La mémoire partagée est un élément crucial de notre simulation. Elle permet à différents processus de partager et d'accéder à des données communes sans dupliquer ces informations dans chaque processus. Par exemple, la représentation de la grille de la ville et l'état actuel de chaque cellule (ou zone) sont stockés dans la mémoire partagée, accessible par tous les processus de la simulation.

5.3. Signaux :

Les signaux sont utilisés pour des notifications simples et rapides entre processus. Dans notre simulation, des signaux tel que `SIGALRM` mais aussi les signaux `SIGUSR1` et `SIGUSR2` sont employés pour simuler des interactions entre personnages, comme des attaques ou

des interceptions. Ces signaux déclenchent des gestionnaires de signaux spécifiques qui modifient l'état des personnages en conséquence.

6. Défis et solutions

6.1. Gestion de la Concurrence :

Dans un environnement multiprocessus et multithread, l'un des défis majeurs est la gestion de la concurrence. Assurer que les processus et les threads interagissent sans causer de conditions de course ou de blocages (deadlocks) était une tâche complexe.

Les sémaphores ont aidé à gérer les limites sur le nombre de threads pouvant accéder à une ressource commune simultanément.

6.2. Gestion du lancement des programmes :

Notre projet étant multiprocessus, il fallait bien synchroniser le lancement de tous les processus. En effet la création des processus se faisant dans un ordre aléatoire, il arrivait que la création de notre mémoire partagée et de nos sémaphores se fasse après son ouverture.

Pour gérer ce problème nous nous sommes assurer que spy_simulation soit bien appelé en premier en ajoutant un sleep().

7. Conclusion

Ce projet a été une source d'apprentissage précieuse dans le domaine de la résolution de problèmes complexes et du développement de logiciels d'envergure. Cependant, il est important de souligner que certaines règles du projet n'ont pas pu être entièrement implémentées. Nous avons consacré une part significative de notre temps au débogage, ce qui a eu un impact sur notre planification initiale et a limité notre capacité à couvrir tous les aspects prévus du projet. Cette expérience souligne l'importance d'une conception soignée et d'une compréhension approfondie des principes des systèmes d'exploitation pour le développement de logiciels efficaces. Elle nous enseigne également la valeur d'une bonne gestion du temps dans les projets de grande envergure, et constitue une base solide pour nos futures entreprises, où nous viserons à surmonter ces défis.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04