Ecole Publique d'Ingénieurs en 3 ans

Rapport projet sytstèmes d'exploitation

A LICENCE TO KILL

le 22 decembre 2022,
version 1.0

Youssef AGHZERE
Mohamed CHERGUI
Ayoub ED-DAHMANY

Colin HARTVICK
Mohamed El-Mokhtar SIDI-ABDALLAH

www.ensicaen.fr

TABLE DES MATIÈRES

INTRODUCTION

Ce projet s'inscrit dans le cadre de l'application des différentes notions vues en cours de systèmes d'exploitation. L'objectif est de réaliser une simulation de ville dans laquelle il y a, d'un côté un réseau d'espions malveillants essaie de voler des informations appartenant à des entreprises sensibles de la ville, et de l'autre côté. Un officier contre-espionnage qui, à l'aide d'un réseau de surveillance placé dans l'hôtel de la ville, tente de suivre les malveillants afin de trouver la boîte aux lettres du réseau d'espionnage tout en évitant de rencontrer l'un des espions.

Comme demandé par l'énoncé, on a utilisé *gitlab* comme moyen pour contrôler les versions et bien sûr pour tester au fur et à mesure le travail fait.

Le projet a été programmé en langage C, et inclus l'application de plusieurs fonctionnalités et les bibliothèques de ce langage de programmation et surtout les sémaphores, les signaux, les tubes et la mémoire partagée.

CHOIX DE CONCEPTIONS – DIFFICULTES RENCONTREES

Dans ce chapitre, nous allons détailler les choix de conceptions et les difficultés rencontrées pour chaque programme que nous avons fait.

1. Ville, état des personnages

La création de la ville et des personnages est faite via le programme *spy_simulation*. Il a été laborieux de créer ce programme de par la complexité du projet. Au fil des itérations, le programme est devenu de plus en plus précis et en adéquation avec le sujet. Ce programme est le premier à être lancé. Il crée la mémoire partagée avec les fonctions données par *posix_semaphore*. Il crée ensuite une carte similaire à la configuration donnée dans le sujet. Ensuite, il affecte la *mailbox* à un bâtiment résidentiel puis crée tous les personnages (espions, officier de contre-espionnage, citoyens) en leur donnant un lieu d'habitation et de travail pour les citoyens. Enfin il crée des messages pour les entreprises en fonction de leur importance et du nombre de travailleurs. Une fois fait il fallait préparer le programme à recevoir des signaux à la fin de chaque tour par le programme *timer*, pour cela on utilise le fonction *sigaction*.

Ce programme contient aussi le réseau maillé de surveillance. Pour cela, à chaque tour, le réseau regarde si un personnage est à côté d'une entreprise sensible. Il stocke dans un tableau, pour chaque personnage, le nombre de tour consécutif passé à côté d'une entreprise sensible. Si ce nombre de tour atteint 6, il crée un tube nommé *counterintelligence_officer* avec la fonction *mkfifo* puis écrit dans ce tube l'identifiant du personnage à cibler ainsi que sa position actuelle (ligne et colonne). De plus, il envoie un signal *SIGALARM* au programme gérant l'officier de contre-espionnage : *counterintelligence*_officer. Cependant l'envoit du signal causant un bug, nous avons décidé de le mettre en commentaire car nous n'avons eu le temps de régler le problème.

Nous avons décidé que pour chaque type de personnage (espion, citoyen…) il y aurait une structure associée car chaque type de personnages a des fonctions tellement différentes qu'il aurait été difficile de les regrouper en une structure. C'est la même chose pour les différents types de bâtiments qui ont une structure associée à un type. De plus la mailbox a une structure associée, bien qu'elle ne soit pas un bâtiment, afin de contenir les messages des espions. Le réseau de surveillance possède aussi une structure contenant le tableau cité précédemment. Enfin dans la mémoire partagée, il y a aussi le nombre de tours réalisés, qui est mis à jour à chaque tour.

2. Citoyens

Les citoyens sont associés au programme *citizen_manager*. Pour les citoyens, à chaque tour, un nombre de threads correspondant au nombre de citoyen est créé. Chaque thread est associé à un citoyen et accède à la mémoire partagée afin d'effectuer sa routine. Les threads sont synchronisés. La mémoire partagée contient une liste de citoyen. Chaque citoyen à une structure associée qui contient ses informations comme sa localisation, son lieu de travail, son lieu d'habitation et son type de travail. Cette structure contient aussi son identifiant de personnage.

Au début de la journée on décide si le citoyen va aller au supermarché après avoir travaillé ou nom. Ensuite, en fonction de l'heure, on fait bouger ou nom le citoyen vers son lieu de travail, d'habitation ou un supermarché. Pour cela on regarde pour chaque case il peut se déplacer, si cette case rapproche le citoyen de sa destination. Le choix a été de fait que chaque citoyen puisse se déplacer sur chaque case qu'elle contienne n'importe quel bâtiment. Nous avons fait ce choix car quand nous avons essayé de limiter la possibilité de déplacement à des cases sans bâtiment, plusieurs citoyens étaient bloqués dans des boucles et n'atteignaient pas leur destination. Comme nous n'avons pas eu le temps d'implémenter une solution permettant de limiter les déplacements à des cases vides, nous avons fait ce choix. Ensuite, on choisit au hasard parmi les cases le rapprochant. On fait cela jusqu'à atteindre la destination.

3. Réseau d'espionnage

Le réseau d'espionnage n'a pas été terminé. Mais beaucoup de choses ont été faites. Tout d'abord, le programme est basé sur le concept du multithreading. En effet, tous les espions arrivent à accomplir leurs tâches quotidiennes (partir au supermarché, repérer une entreprise, voler une entreprise, partir au mailbox ...) de manière simultanée. Pour ce faire, le programme a été écrit en se basant sur 3 fonctions principales à savoir *night_routine* qui gère les tâches à faire dans la période entre 17h00 jusqu'à 8h00, *day_routine* pour la période entre 8h00 et 17h00, et enfin *spie_routine* qui est le cœur de chacun des 3 espions. Par contre, on n'a pas codé le cas où un membre du réseau d'espionnage se rencontre avec le l'officier contre-espionnage.

La plupart des conditions ont été respecté :

-Un espion ne peut voler la même entreprise qu'une seule fois, et il vole une seule fois par nuit dans des horaires générées selon la probabilité de chaque cas.

-Les espions n'accèdent jamais simultanément au *mailbox*. Ceci a été réalisé à l'aide de la fonction *usleep* appelée au début de la fonction *night_routine* (0us pour espion0, 4000us pour espion1 et 8000 pour espion2). On assume que synchroniser les threads avec des sémaphores aurait été meilleur mais on a rencontré des soucis pour implémenter ce choix, vu qu'il y a déjà le sémaphore des processus.

Cependant, des conditions comme attendre 3 jours avant de voler une entreprise qui a été voler par un autre espion ne pas été respecté par souci de délais.

Le programme *enemy_spy_network* crache des fois au cours de la simulation. Mais ceci est rare et lorsqu'il arrive c'est souvent au millième tour. Cette erreur est du genre *Error [P()],* a un lien avec le sémaphore mais on n'a pas pû le débugger.

4. Officier de contre-espionnage

L'officier de contre-espionnage n'a pas été terminé. Pour l'instant il suit une logique simple. A chaque tour s'il n'a pas de cible, il ne fait rien. Quand il reçoit un signal du réseau de surveillance, il a une cible. Il ouvre donc le tube nommé crée par le réseau de surveillance *counterintelligence_officer* et lit les informations inscrites (identifiant du personnage, position). Désormais, à chaque tour, il se déplace jusqu'à la cible.

5. Enemy country

Le programme d'enemy country pour l'affichage des messages chiffré en provenance de réseau d'espionnage n'a pas été terminé, mais le mécanisme de réception a été fait ; un programme qui permet d'afficher les messages depuis une file de messages avec un priorité

qui est supposé déjà rempli par le réseau d'espionnage (partie non fait), et puis l'affichage continue sur l'interface des messages lus depuis la file.

6. Timer

C'est un programme qui gère la durée de simulation : le début, la fin et la durée de chaque étape entre le début et la fin. Le fonctionnement de *timer* est fait avec une communication par signaux entre les deux processus *timer* et *spy_simulation ;* après chaque quantum du temps de durée entre 0.1 et 1 seconds le premier processus envoie un signal SIGALRM à travers la primitive *kill()* au deuxième processus, le processus *spy_simulation* incrémente une variable *count*, qui présente le numéro de tour, dans la mémoire partagée. Tous les processus calculent l'heure actuelle en utilisant cette variable.

7. Moniteur

Pour faire fonctionner le moniteur, il a suffi de le faire ouvrir la mémoire partagée afin d'accéder aux informations de la simulation et de récupérer dans chaque fonction du moniteur les bonnes valeurs dans la mémoire.

8. Autres

Nous n'avons pas développé la fin de la simulation car elle intervient en cas de confrontation entre l'officier de contre-espionnage et un espion et nous n'avons pas implémenté ces confrontations. La seule fin possible pour notre simulation est la fin après 2016 tours. Nous avons créé un fichier *function.c* qui contient des fonctions utilisées par plusieurs processus, comme le calcul de la distance de Manhattan entre deux cellules ou le calcul de l'heure actuelle en fonction du tour actuel.

CONCLUSION

Nous n'avons pas réussi à terminer complètement la simulation. Nous avons eu beaucoup de problèmes les derniers jours avec des bugs que nous avons remarqué pendant les dernières heures et qui ont fait que nous avons dû enlever certaines fonctionnalités.

Cependant, nous avons quand même développés plusieurs programmes qui fonctionnent ensemble et qui respectent le sujet. La création de la ville ainsi que le programme timer fonctionne très bien. Les citoyens ont un bon comportement mais leur chemin n'est pas parfait. Les espions ont une routine, tout comme l'officier de contre espionage mais ils ne se recontrent pas. Le réseau de surveillance détecte les anomalies mais ne previent pas l'officier de contre espionage.

Nous avons aussi implémenté plusieurs notions que nous avons vu en cours de systèmes d'exploitation comme les threads, les signaux, les tubes et le clonage de processus ou les files de messages.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Normandie Université

Cti
Commission des titres d'ingénieur

cdefi
Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes
EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES