Licence 2 - Info 4B

Semestre 4 - Année 2021

Projet Principes des Systèmes d'Exploitation • à rendre la semaine du 6 Avril

Cadre général du projet

Le projet à pour but d'appliquer les principes des systèmes d'exploitation afin de réaliser l'une des 3 applications décrites dans la suite du document : 1) un moteur de recherche spécialisé, 2) un jeu des années 80 étendu avec un mode multi-joueurs et un mode réseau, 3) le jeu de Core War au cours duquel des programmes s'affrontent pour conquérir la mémoire.

Contraintes pour la réalisation

Le projet est à réaliser par groupe de deux étudiants. La programmation se fera en utilisant la plate-forme Java. Il n'est pas obligatoire de développer une interface graphique. Votre programme devra être conçu de façon modulaire (packages, interfaces, classes respectant une encapsulation stricte, classes utilitaires, constantes, fichiers de configuration, etc.) et il devra refléter la conception en couches d'un système d'exploitation (les couches et leurs fonctionnalités sont à détailler dans le rapport). Les structures de données que vous utiliserez ne sont pas nécessairement dynamiques (structure à base de références), vous pouvez utiliser des tableaux cependant les collections proposées par Java seront d'une grande utilité. L'emploi des membres de classe déclarés *static* devra être justifié. Il est également très fortement conseillé de respecter les conventions d'écriture de programmes Java qui sont des règles pour faciliter la compréhension et la maintenance du code (`https://www.jmdoudoux.fr/java/dej/chap-normes-dev.htm`). La lisibilité du code sera évaluée.

Vous devez faire une présentation de votre projet avant les vacances de printemps (semaine du 6 avril 2021). Ce jour là vous remettrez un rapport papier. Pour ce document, les consignes **strictes** de présentation sont : au moins 15 pages, police 12pts maximum, marges 2cm maximum. Le rapport doit comporter au moins les éléments suivants :

- **une analyse fonctionnelle** du sujet, précisant, entre autre, toutes les règles de fonctionnement de votre application et un découpage du problème en sous-problèmes (pour aboutir aux grandes lignes de l'architecture logicielle), c'est-à-dire les classes les plus importantes et leurs liens ;
- **une description des structures de données** envisagées et retenues ;
- **la spécification des classes principales** et des méthodes essentielles ;
- **l'architecture logicielle détaillée** de votre application, c'est-à-dire, dans le cadre du module Info4B, une conception en couches fonctionnelles à l'image de ce qui est réalisé dans l'architecture d'un système d'exploitation. Pour chaque couche vous spécifierez les classes qui implémentent les services de la couche fonctionnelle ;
- une description des **algorithmes principaux** ;
- **un jeu de tests** montrant que votre programme fonctionne.

Il est conseillé de rédiger le rapport en utilisant LaTeX car il permet d'inclure facilement des extraits de code source avec une mise en évidence des éléments syntaxiques. On trouve des applications permettant de rédiger collaborativement des documents LaTeX (`https://fr.sharelatex.com/`, `https://fr.overleaf.com/` ou sur le serveur `iemgit` accessible par VPN en demandant au préalable l'activation d'un compte). Pour le partage de code entre binôme l'outil git et son interface Web GitHub sont recommandés. Dans le document, il est conseillé d'utiliser des schémas pour illustrer vos propos. Des diagrammes UML comme le diagramme de classes ou de séquences peuvent être ajoutés à votre rapport.

Une archive[1] (tgz, jar ou zip exclusivement) de l'ensemble de code source et de l'exécutable de votre programme devra être produite à la date du jour de la démonstration, puis envoyée par mail, ou déposée dans vos espace personnel (`private_html` de votre répertoire).

1. Les autres formats d'archive ne sont pas autorisés

L'évaluation se fait sur la base du rapport papier, de la qualité technique de la réalisation, du déroulement de la démonstration. Lors de la démonstration vous devrez avoir préparé un scénario et des jeux de tests afin de présenter les fonctionnalités de votre application (vous disposerez de 10 minutes).

Le descriptif des projets est volontairement synthétique. Vous devez/pouvez déposer des questions dans le forum de Plubel prévu à cet effet. Il faut également exploiter les références données.

Sujet 1 : Moteur de recherche spécialisé

L'objectif est de développer l'infrastructure d'un moteur de recherche et d'implanter un algorithme très populaire (HITS) qui permet de déterminer et de classer les pages faisant autorité (*authorities*) et les pages qui concentrent les liens vers les autorités (*hubs*).

Un moteur de recherche est généralement composé de plusieurs éléments :

— Un module *crawler* qui explore le Web en suivant les liens contenus dans les pages. Plusieurs machines ou threads peuvent effectuer cette tâche. Les URL des pages à télécharger sont contenues dans une liste d'attente qui est remplie par les différentes instances du crawler qui analysent rapidement les pages téléchargées pour y découvrir de nouvelles URLs.

— Un entrepôt de pages Web qui contient les fichiers HTML des pages qui seront ensuite traités par un mécanisme d'indexation.

— Un module d'indexation qui doit être multi-thread et/ou multi-machines et qui extrait, pour chaque page, la source et la cible d'une URL spécifiée par `HREF` et Target d'une balise `A` dans le code HTML. Pour une même page HTML on obtient donc une série de triplets (URL de la page, mots cliquables, URL de la cible). Les triplets seront stockés et accessibles au moyen d'un ou de plusieurs index (par exemple des hashtables). Les triplets seront également rendus persistant dans un fichier au moyen du mécanisme de sérialisation.

— Un ensemble d'algorithmes qui peuvent être utilisés par une module d'interrogation pour classer les pages Web et répondre à une demande utilisateur. Dans notre cas il s'agit principalement de l'algorithme HITS.

Votre réalisation devra faire apparaître clairement ces différents éléments. Pour l'algorithme HITS, vous devrez implanter une version itérative (la version utilisant valeurs propres et vecteurs propres n'est pas demandée).

La programmation réseau ne peut pas se limiter à la capture des pages HTML (partie *crawler*), votre système doit pouvoir fonctionner en équilibrant le charge sur plusieurs machines, par exemple la partie indexation et l'algorithme HITS devront être exécutés sur une autre machine qui viendra consulter l'entrepôt des pages Web.

Vous devrez fournir des jeux de données (par exemple un fichier contenant plus de 500 000 triplets), les résultats produits et un scénario d'exécution afin de montrer la pertinence des résultats par rapport à une recherche d'un domaine précis, par exemple, sur des cours ou des notions d'informatique.

En option : utiliser les mots du texte de la page, ou les méta-tags pour appliquer l'algorithme sur un sous-graphe.

Remarques : Pour décoder (*parsing*) le contenu des pages Web, c'est-à-dire l'HTML vous pouvez vous aider de libraires externes comme : JSOUP https://jsoup.org/. Le lien suivant également peut vous aider : http://www.fobec.com/java/996/charger-extraire-liens-une-page-html.html.

Références sur l'algorithme HITS :

— https://fr.wikipedia.org/wiki/Algorithme_HITS (suivre le lien externe)

— https://en.wikipedia.org/wiki/HITS_algorithm (page en anglais de Wikipedia assez détaillée, contenant des exemples d'algorithmes)

— https://www.youtube.com/watch?v=zydSN8C1Et4 (avec le principe de l'algorithme expliqué pas à pas)

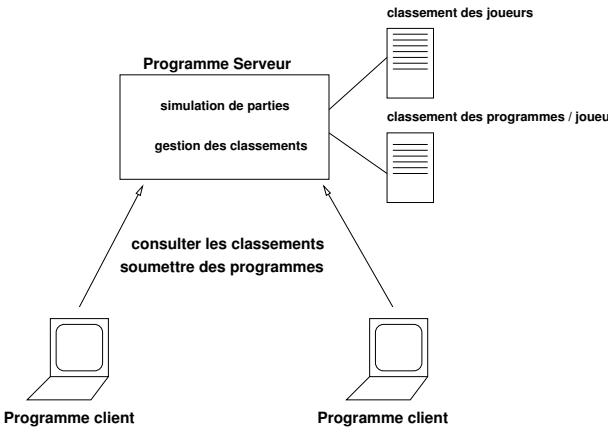

FIGURE 1 – Jeu de core war en réseau

Sujet 2 : Burger Time

BurgerTime est un jeu développé au début des années 1980. Un cuisinier contrôlé par le joueur évolue dans un environnement de plateformes reliées par des échelles, son but est de fabriquer des hamburgers en faisant tomber les différents constituants. Plusieurs entités hostiles tentent de le capturer (œufs, saucisses, etc.). Il peut fuir ou les immobiliser avec du poivre. L'étape se termine si les hamburgers sont prêts ou si le cuisinier a épuisé ses vies.

Vous avez libre choix d'adapter ou d'ajouter des règles en respectant l'esprit du jeu. Cependant vous devez réaliser les fonctionnalités suivantes :

- les adversaires du cuisinier doivent pourvoir être contrôlés soit par des joueurs ou soit par le programme. Dans ce dernier cas, plusieurs comportements doivent être implantés ;
- il doit être possible de jouer en réseau à plusieurs, éventuellement avec plusieurs cuisiniers en mode coopératif ;
- les scores doivent être enregistrés sur un serveur et la liste des meilleurs joueurs doit être maintenue a jour ;
- il doit être possible de définir des équipes en mode collaboratif et d'affronter d'autres équipes manipulant les monstres ;

L'état du jeu est représenté par trois structures : une structure statique, le décor contenant les plateformes et les échelles, deux structures dynamiques, les constituants des burgers, les entités (cuisinier et ses ennemis).

En option : les participants doivent pouvoir proposer eux mêmes des stratégies de déplacement des adversaires du cuisinier sous la forme de programmes Java (fichier .class) qui seront chargés dynamiquement sur le serveur. Il conviendra d'utiliser l'API *reflexion*, le chargement dynamique de classes. Le comportement des classes spécifiques (implantant la stratégie) sera conforme à une interface générale de déplacement.

Références :

- https://fr.wikipedia.org/wiki/Burgertime (suivre les liens)
- https://www.youtube.com/watch?v=5wEWftbwSm4

Sujet 3 : Core War

On se propose d'appliquer les concepts des systèmes d'exploitation afin de réaliser un simulateur *Mars* dont la description se trouve dans l'article de A. K. Dewdney cité dans les références et dont la traduction en français sera déposée sur la plateforme Plubel. Votre simulateur doit être capable d'exécuter simultanément plusieurs programmes écrits en *RedCode*. La taille mémoire de la machine est fixée à l'initialisation du jeu. Le programme doit déclarer un joueur gagnant et classer le ou les autres joueurs.

Les programmes *Mars* sont des fichiers texte que l'on charge en début de partie, qui sont interprétés lors de l'exécution (voir la description de la syntaxe dans les références).

En plus des éléments décrits dans les références, les fonctionnalités principales que vous avez à ajouter sont de deux types :

— par rapport à l'article original de Dewdney vous devez permettre la soumission de programmes par le réseau, c'est-à-dire développer une partie serveur et une partie client. Un même serveur peut héberger plusieurs parties se déroulant en même temps (figure 1). Une partie démarre lorsque chaque joueur participant a envoyé son programme ;

— par rapport à la gestion des joueurs et des programmes : à mesure des parties gagnées un joueur progresse dans le classement (stocké sur la machine qui héberge le programme serveur). On gardera aussi le *hit parade* des noms des programmes gagnants. Avant de lancer une partie, il est possible de consulter les classements.

Si vous souhaitez réaliser d'autres extensions du langage *Redcode*, votre interpréteur devra être capable de supporter deux syntaxes : celle des programmes en *RedCode Base* correspondant à description du langage dans l'article de A. Dewdney et la syntaxe du *RedCode Étendu* correspondant à vos ajouts. On doit pouvoir sélectionner l'une des syntaxes au lancement des programmes.

Références :

— Article original en anglais : https://www.corewars.org/sciam/

— http://www.koth.org/index.html

— https://fr.wikipedia.org/wiki/Core_War