RAPPORT DE PROJET : WILDWATER

Informatique 3

PréIng 2 - Mi3

Date de rendu : 21/12/25

Lina Porrinas

Oumeyma Adjaimi

Nabil Touat

1. Composition du groupe et répartition des tâches

Le projet WildWater a été réalisé en groupe avec une répartition claire des rôles afin d'assurer une progression efficace et cohérente.

- Oumeyma était en charge du développement en langage C, incluant la conception des structures de données (arbres AVL) et l'implémentation des algorithmes principaux de calcul des fuites et des statistiques.
- Nabil s'est occupé du développement du script Shell servant d'interface utilisateur, de la gestion des arguments et de l'intégration de Gnuplot pour l'automatisation des graphiques.
- Lina a assuré l'intégration globale du projet, la rédaction de la documentation (README et rapport PDF), l'implémentation des fonctions de base en C, la gestion du Makefile ainsi que la validation des tests fonctionnels.

2. Planning de réalisation

Le projet s'est déroulé sur plusieurs semaines selon les étapes suivantes :

- Semaine 1 : Analyse du sujet, répartition des rôles et conception des structures de données.
- Semaine 2 : Développement des modules principaux en C et mise en place du script Shell.
- Semaine 3 : Intégration des différents modules, tests de validation et rédaction de la documentation finale.

3. État du projet, limitations et difficultés

Compilation et qualité du code :

Le projet compile parfaitement via le Makefile. Aucun avertissement n'est généré lors de la compilation.

Fonctionnalités et bonus :

Les fonctionnalités principales sont opérationnelles (lecture fichier, AVL, détection fuites, statistiques .dat). Concernant les bonus, le groupe a privilégié la robustesse du cœur du programme.

Difficultés rencontrées :

Durant le développement, nous avons fait face à deux points techniques particuliers :

Algorithme de calcul des fuites : La compréhension de la partie leak etait assez compliquée au depart.

Messages "Broken pipe" du script Shell : Lors de l'exécution du script d'automatisation, des messages système sort: Broken pipe peuvent apparaître. Nous avons identifié l'origine : cela se produit lorsque la commande head arrête de lire la sortie de la commande sort avant que celle-ci ait fini de trier tout le fichier. Ce comportement est normal dans un pipe Unix et n'affecte pas les résultats générés.

Validation et preuves d’exécution

La validation du projet repose sur une compilation réussie, l'exécution du script et la génération des fichiers.

Cette capture montre le nettoyage (make clean), la compilation (make) et l'exécution du script (./shell.sh).

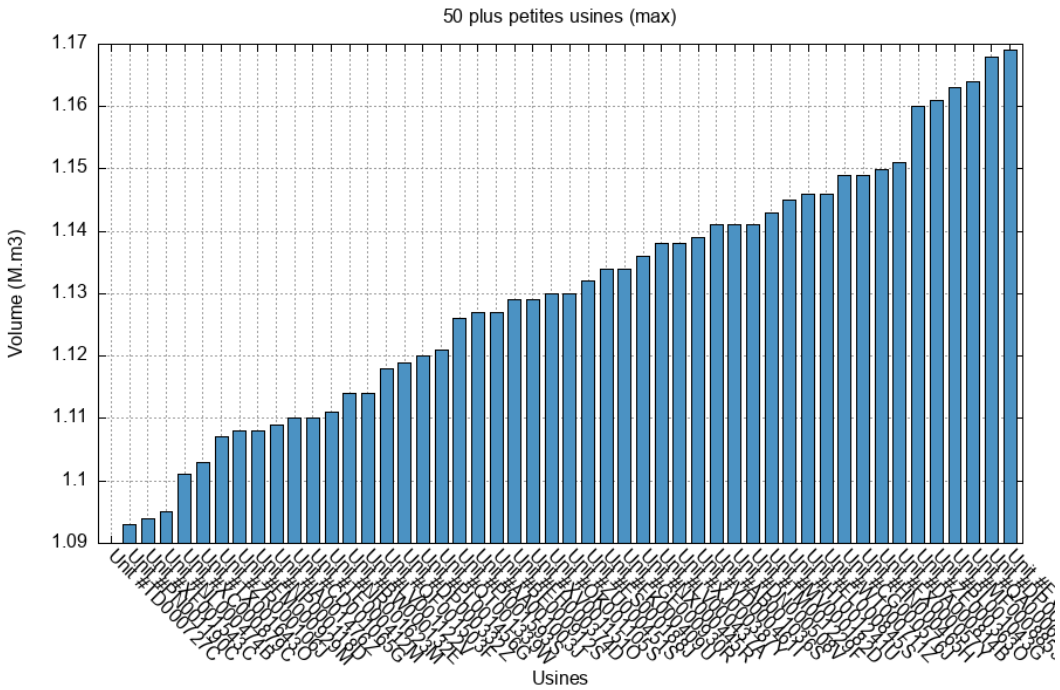
```
lina@Host-002 projetinfo2 % make clean
rm -f *.o
rm -f wildwater
rm -f *.png
rm -f *_e.dat
lina@Host-002 projetinfo2 % make
gcc -c main.c -o main.o
gcc -c leak.c -o leak.o
gcc -c histo.c -o histo.o
gcc -c fonction_base.c -o fonction_base.o
gcc main.o leak.o histo.o fonction_base.o -o wildwater
chmod +x shell.sh
lina@Host-002 projetinfo2 % ./shell.sh c-wildwater_v3.dat histo all
Histogramme genere : histo_all_e.dat
sort: Broken pipe
sort: Broken pipe
sort: Broken pipe
sort: Broken pipe
sort: Broken pipe
sort: Broken pipe
Histogrammes cumulés générés :
La durée totale est : 2000 mili-seconde
lina@Host-002 projetinfo2 %
```

Cette capture prouve que les calculs sont corrects.

leaks_e.dat

```
1    Unit #GV001308I;0.073104M.m3
2
```

Cette capture montre que les histogrammes sont bien générés.



4. Conclusion

Ce travail de groupe nous a permis de mieux maîtriser les aspects pointus du langage C, comme la gestion dynamique de la mémoire et l'organisation en modules. L'implémentation de l'algorithme récursif a représenté un vrai défi technique, mais elle fonctionne aujourd'hui parfaitement.