Rapport de projet de LP25

MICHEL Matis, OLIVEIRA COSTA Raphaël, HIETEL François, BELKADI Iéto (TC3)

Automne 2025

Contexte du projet

Le projet consiste à concevoir un **programme de gestion de processus pour systèmes Linux**, capable de fonctionner aussi bien **en local que sur des hôtes distants**. L'objectif est de fournir une interface interactive, inspirée de l'outil `htop`, permettant :

- D'afficher dynamiquement la liste des processus actifs sur une ou plusieurs machines,
- De visualiser des informations détaillées (PID, utilisateur, consommation CPU/mémoire, temps d'exécution, etc.),
- D'interagir directement avec les processus (mise en pause, arrêt, reprise, redémarrage). L'outil reposera sur des mécanismes standards de communication et d'administration à distance (SSH, etc.), compatibilité avec les principales distributions Linux.

État des lieux des compétences du groupe

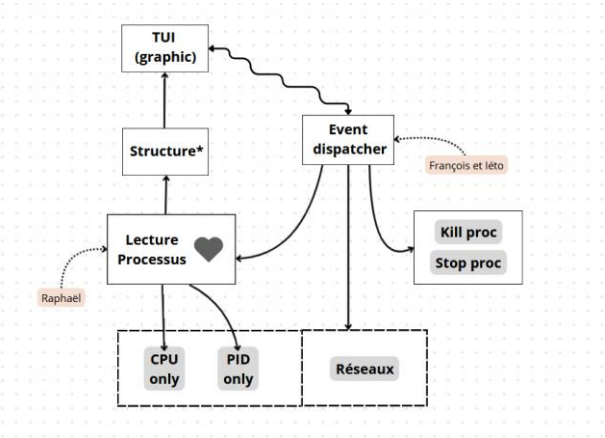
Matis également des connaissances en C/C++ accumulée à travers des projets personnels sur Arduino et la réalisation de moteur graphique et audio.

François, Raphaël et Iéto disposaient uniquement de bases en langage C acquises en IF2, IF3 et lors des cours, travaux dirigés et travaux pratiques précédents.

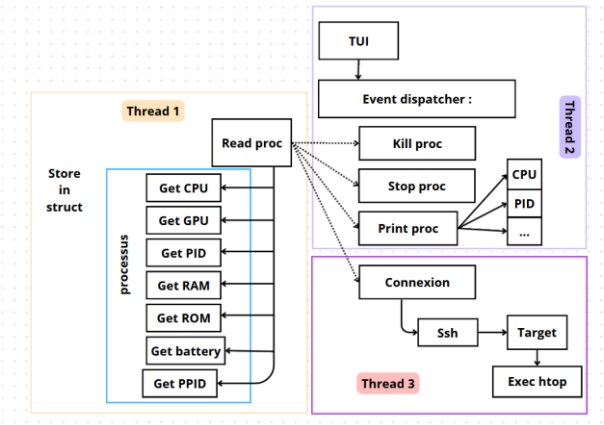
L'ensemble des connaissances que nous possédions au commencement couvraient principalement la syntaxe du langage, la gestion et l'optimisation de la mémoire, l'utilisation des structures, ainsi que des notions élémentaires de programmation système.

Répartition des tâches : planification

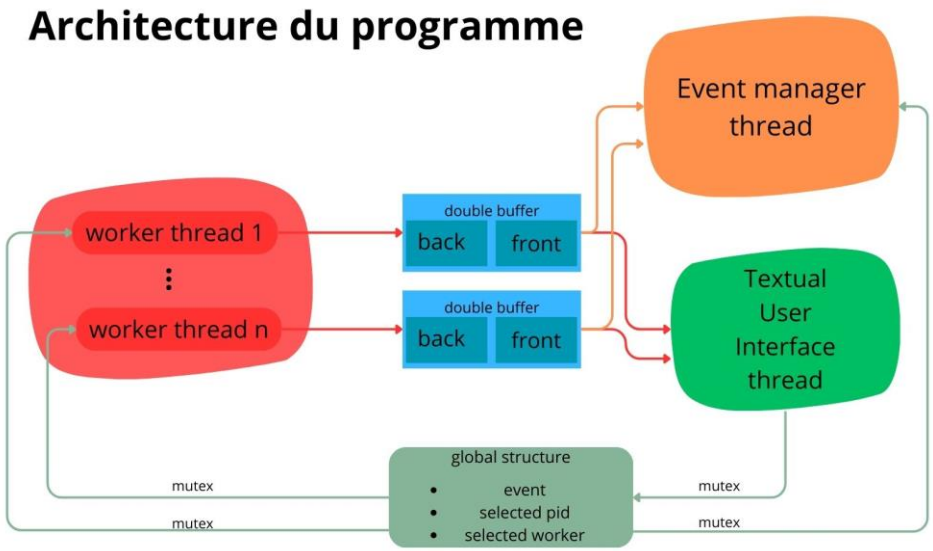
Ces schémas ont été réalisés au tout début du projet, permettant de poser les premières idées et de visualiser de manière globale l'architecture envisagée.



Après avoir analysé nous avons élaboré



l'architecture suivante :



Nous avons décidé de répartir les tâches comme suit :

• **Matis** : Réalisation de la TUI, gestion globale des threads, réalisation des structures de données nécessaires (tableaux dynamiques, double buffer)
• **Raphaël** : Récupération des données relatives aux processus locaux / distant et parsing des fichiers.
• **Iéto** : Parsing du fichier .config, vérification des droits 600 et gestion des options passées au programme avec getopt.
• **François** : Etablissement la connexion SSH, récupération des données distantes et gestion de la vie des processus locaux / distant (kill / terminate / stop / continue / restart)

Répartition des tâches : réalisation

La charge de travail a été répartie entre les différents membres de l'équipe en fonction des modules du projet.

François a mis en place l'infrastructure de communication via les sessions SSH, garantissant la sécurisation des échanges avec les hôtes distants.
Également, il a développé la logique de contrôle des processus en implémentant les fonctions de gestion des signaux système :

- Côté Distant (SSH) : Il a conçu le module **ssh_connexion.h** pour gérer les sessions sécurisées et les commandes à distance (ssh_kill_processus, ssh_stop_processus, ssh_cont_processus, ssh_term_processus, ssh_restart_processus,).

- Côté Local : Il a implémenté le module **processus_signal.h** pour manipulation directe pour la machine hôte en utilisant les signaux système Linux (proc_kill, proc_term, proc_stop, proc_cont, proc_restart)

Raphaël s'est chargé de la lecture et de l'analyse des informations des processus système. Il a développé un module permettant de récupérer pour chaque processus à partir de son fichier :

PID	*/proc/*
Nom, état du processus, PPID, CPU (utime, stime, start_time)	*/proc/<pid>/stat*
Exécutable et arguments de lancement	*/proc/<pid>/cmdline*
Variables d'environnement	*/proc/<pid>/environ*
Nom de l'utilisateur	*/proc/stat*

Toutes ces données sont centralisées dans une structure, facilitant leur exploitation par les autres modules du projet, notamment pour l'affichage et le monitoring des processus.

Iéto a développé le module de gestion des options et de la configuration, définissant l'interface entre l'utilisateur et l'application. Les options sont stockées dans une structure grâce à l'analyse de la ligne de commande avec getopt_long, ce qui permet leur transmission fluide aux autres composants du projet.
Elle a également conçu un parseur pour le fichier .config en extrayant les informations essentielles (adresse, utilisateur, mot de passe, type de connexion). Enfin, les permissions du fichier de configuration sont strictement contrôlées (chmod 600), la cohérence des options est vérifiée, et les valeurs manquantes sont complétées automatiquement, notamment les ports par défaut (22 pour SSH, 23 pour Telnet).

Enfin, **Matis** a développé l'interface utilisateur en mode terminal en utilisant ncurses, pour que l'application ressemble à htop.
Affichage dynamique :

- Affichage des processus dans un tableau (PID, CPU, RAM), avec défilement et temps d'exécution lisible (JJ:HH:MM:SS).

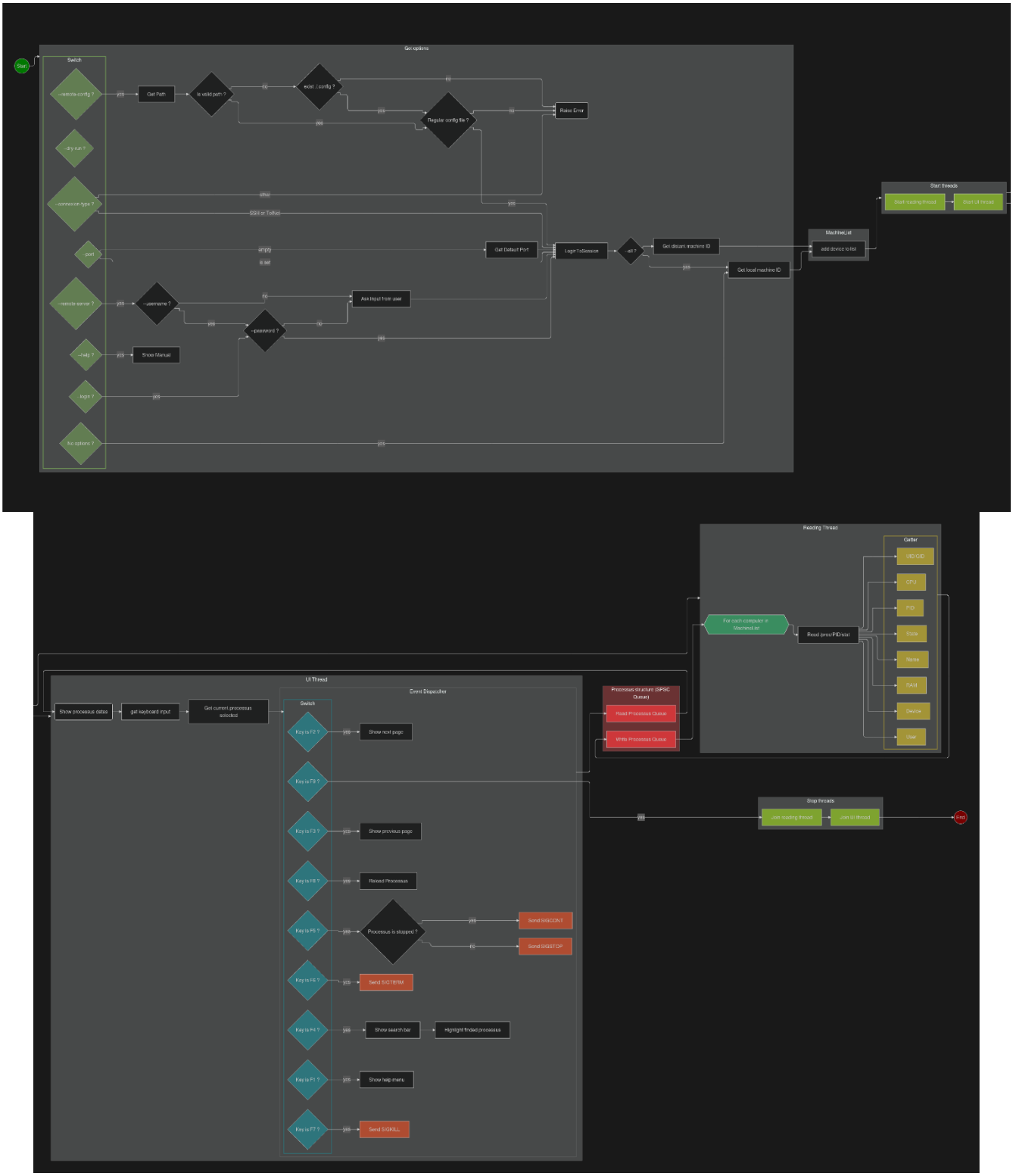
Interactions clavier :

- Capture des touches (flèches, F1-F9).
- Navigation dans les onglets et sélection des processus.
- Tri et filtrage des processus pour les retrouver facilement.

Intégration avec le projet :

- L'UI utilise les données des processus fournies.
- Elle permet d'exécuter les actions de contrôle (Kill, Stop…) mises en place.

Les schémas présentés correspondent aux premières pistes de réflexion et de recherche ayant guidé la conception initiale de l'architecture de l'application.



Dans le cadre du projet, nous avons choisi de mettre en place une architecture multi-thread afin de séparer les différentes tâches de l'application et d'améliorer sa réactivité. Cette organisation permet notamment d'exécuter en parallèle la gestion de l'interface utilisateur, la lecture des processus locaux et la communication avec les hôtes distants, sans qu'une opération lente ne bloque l'ensemble du programme.

Pour assurer une communication efficace et sûre entre les threads, nous avons utilisé un mécanisme de double tampon (double buffering). Ce principe repose sur l'utilisation de deux zones de données alternées : pendant qu'un thread écrit les informations dans un tampon, un autre peut lire les données du second. Cette approche limite les conflits d'accès aux données, réduit les problèmes de synchronisation et garantit la cohérence des informations échangées. Elle contribue ainsi à un fonctionnement plus fluide et stable de l'application, en particulier dans un contexte multi-thread.

Nous avions initialement organisé l'application autour de trois threads principaux afin de séparer clairement les responsabilités et d'améliorer la lisibilité de l'architecture : un thread dédié à l'interface utilisateur, un thread chargé de la gestion des processus locaux et un thread consacré à la communication avec les hôtes distants. Au cours du développement, cette architecture a évolué afin de mieux répondre aux besoins du projet. Nous avons ainsi adopté un modèle à N+2 threads, composé d'un thread principal dédié à l'interface utilisateur, de N threads de travail (chacun associé à une machine) et d'un thread de gestion.

Nous avons utilisé le standard C23 afin d'améliorer la robustesse et la qualité du code. Ce standard apporte plusieurs ajouts modernes qui permettent de limiter certaines erreurs courantes en langage C et de rendre le code plus lisible. L'utilisation de nullptr permet notamment d'éviter des confusions liées aux pointeurs nuls, tandis que constexpr et auto facilitent l'écriture de code plus sûr et plus expressif.
De plus, les attributs [[nodiscard]] et [[maybe_unused]] permettent d'améliorer la fiabilité du programme en signalant plus clairement les erreurs potentielles, comme une valeur de retour ignorée ou une variable volontairement inutilisée.
Afin de gérer efficacement des collections de taille variable, nous avons développé notre propre template de tableau dynamique, inspiré du conteneur std::vector du langage C++. Cette structure de données permet d'ajouter, de supprimer et d'accéder aux éléments de manière flexible, sans connaître à l'avance la taille des tableaux à manipuler, ce qui est particulièrement adapté à la gestion des listes de processus ou de serveurs distants.

Nous avons utilisé des outils d'intelligence artificielle au cours du projet LP25 (ChatGPT, gemini et Github Copilot). Ces outils ont été employés comme aide au développement, sans se substituer à la compréhension ni aux choix techniques réalisés par les membres du groupe.
L'IA a notamment été utilisée pour :

- Générer des squelettes et scénarios de tests unitaires, à partir des spécifications fonctionnelles des modules (lecture des processus, gestion des signaux, parsing des options, etc.) ;
- Identifier plus rapidement l'origine de certains comportements inattendus et d'assister à la résolution de problèmes

- Proposer des pistes d'optimisation fine, cela a permis de clarifier les portions complexes du programme tout en respectant l'architecture initiale.

Voici un exemple de prompt donné à une IA

« Analyse l'intégralité de mon code dans le dépot lp25-a25 et aide moi à détecter et résoudre :

- Les répétitions de codes qui peuvent etre évités
- Des fonctions "mortes" non utilisés, définie et non implémentées
- Des fuites de mémoires
- Des dead lock
- Des threads non safe
- Des pointeurs mal protégés
- Des optimisations mineures qui peuvent etre apportées pour améliorer la stabilité du code
- Liste les fichiers et les lignes exactes nécessitant des corrections et propose une solution pour chacun des problèmes que tu rencontreras tout en gardant la cohérence du code et en respectant l'architecture logicielle »

En complément, la qualité mémoire de l'application a été vérifiée à l'aide de l'outil Valgrind, permettant de détecter d'éventuelles fuites mémoire, accès invalides ou utilisations incorrectes de la mémoire dynamique. Cette étape a contribué à renforcer la robustesse et la fiabilité du programme, en particulier dans un contexte multi-thread et de gestion de structures dynamiques.
Voici la sortie que nous a donné valgrind

```
==40346== LEAK SUMMARY:
==40346==    definitely lost: 0 bytes in 0 blocks
==40346==    indirectly lost: 0 bytes in 0 blocks
==40346==      possibly lost: 201 bytes in 3 blocks
==40346==    still reachable: 652,885 bytes in 488 blocks
==40346==         suppressed: 0 bytes in 0 blocks
==40346==
==40346== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Note : les octets « possibly lost » sont issus de la libc, la libssh et de nucurses en grande majorité.

Difficultés rencontrées

Compréhension des objectifs

L'instruction concernant la récupération des données en local ne précisait pas si l'utilisation de commandes externes comme ps était autorisée ou s'il fallait privilégier le parsing manuel du système de fichiers /proc. Pour lever cette ambiguïté, nous avons considéré que l'objectif pédagogique consistait à comprendre le fonctionnement interne du système et nous avons donc choisi d'extraire moi-même les informations depuis /proc.

La demande de réaliser une interface de "type htop" manquait de précision sur le niveau de complexité attendu, notamment concernant les fonctionnalités interactives. Dans le doute, nous avons choisi d'inclure le tri et la recherche avancée pour m'assurer que l'outil soit conforme à l'expérience utilisateur suggérée par la référence à htop.

Réalisation des objectifs

Au début du projet, nous n’avons pas appliqué systématiquement les conventions de codage et les règles d’écriture des fonctions, malgré l’existence d’une convention prévue. Chaque membre a développé ses modules selon ses propres habitudes, ce qui a entraîné des incohérences dans le style du code. À la fin du projet, ces différences rendaient la lecture du code plus difficiles, et compliquaient la vérification de la cohérence des fonctionnalités.

Résolution du problème

Pour corriger cette situation, nous avons procédé à une vérification complète du code en fin de projet. Nous avons harmonisé les noms de fonctions, les types de retour et l’utilisation des codes d’erreur, afin que tous les modules respectent la même convention. Cette étape a permis de rendre le code plus lisible, plus cohérent et plus facile à maintenir.

Anticipation possible

Cette difficulté aurait pu être évitée si les conventions avaient été appliquées dès le début du projet et si des revues de code régulières avaient été organisées pour vérifier leur respect.

Facteurs facilitant la résolution

La mise en place d’un guide de style clair dès le départ et l’utilisation d’outils d’analyse statique du code auraient grandement facilité le maintien de la cohérence pendant le développement,réduisant ainsi le travail de vérification finale.

Une des difficultés était de réussir à se mettre d'accord sur ce qui était déjà fait et sur ce qu'il restait à accomplir pour éviter de coder inutilement ou de coder deux fois la même chose.
Même si les tâches étaient réparties, il n'y avait pas de consignes sur la manière de suivre l'avancement réel de chacun. Sans une communication constante, on risquait de passer du temps sur des fonctionnalités déjà développées par un autre ou de coder des fonctions qui ne servaient finalement à rien.
On a dû se parler beaucoup plus souvent pour faire le point sur l'état du projet. On a compris que le projet n'avance efficacement que si tout le monde partage la même vision du travail restant, ce qui permet de ne pas gaspiller d'énergie. Utiliser un tableau de suivi partagé dès le début aurait permis de savoir exactement ce que chacun était en train de faire. Cela nous aurait évité de coder des parties de manière isolée sans être sûrs qu'elles étaient encore nécessaires.

Une autre difficulté a été la gestion du traitement des erreurs. Certaines fonctions retournaient des valeurs différentes selon les modules, ce qui entraînait des comportements incohérents et rendait difficile l'identification de l'origine réelle d'un dysfonctionnement.
Pour résoudre ce problème, nous avons mis en place une gestion centralisée des erreurs en définissant des codes d'erreur communs à l'ensemble du projet. Chaque fonction importante retourne désormais un code d'erreur clair et standardisé. Cette approche a permis d'uniformiser le traitement des erreurs dans tous les modules et d'améliorer la lisibilité globale du code.

Proposition d'une amélioration

Actuellement, la sécurité repose uniquement sur les permissions du fichier de configuration (chmod 600). Une amélioration importante serait de chiffrer les mots de passe directement dans le fichier ".config". Cela permettrait de ne plus stocker d'identifiants en clair sur le disque, garantissant ainsi que les données restent protégées même si le fichier est lu par un tiers.

Analyse des résultats

L'analyse des résultats confirme que les objectifs techniques ont été atteints, notamment grâce à la stabilité de l'architecture multi-threadée en C23. L'implémentation du double tampon a permis de maintenir une interface fluide, garantissant une navigation sans scintillement ni blocage, même lors des phases intensives de calcul. L'outil remplit sa fonction première en offrant une gestion dynamique des processus, conforme à l'expérience utilisateur attendue d'un utilitaire de type htop.

Sur le plan des performances, le programme affiche une gestion rigoureuse des ressources, validée par l'absence de fuites mémoire dans les rapports Valgrind. On note toutefois une latence de 5 secondes lors de la récupération des données via SSH, un délai inhérent aux contraintes réseau qui n'impacte pas la réactivité de l'interface grâce à notre modèle asynchrone. En conclusion, la structure logicielle actuelle assure une base robuste, dont la sécurité pourrait être encore renforcée par le chiffrement des données de configuration en complément des permissions d'accès actuelles.

Retour d'expérience

Cette analyse critique identifie les blocages rencontrés et les enseignements essentiels à notre futur métier d'ingénieur. Si nous devions refaire ce projet, nous privilégierions une standardisation immédiate du code et un suivi rigoureux de l'avancement. L'absence initiale de conventions uniformes a dégradé la lisibilité et compliqué l'intégration finale, allongeant les délais pour harmoniser les modules. L'application d'un guide de style et de revues de code dès le lancement aurait garanti une meilleure cohérence technique.

Le manque de coordination sur l'avancement des tâches a également généré des risques de redondance. Cette difficulté, liée à l'absence d'un outil de suivi partagé, a nécessité une communication verbale intensive pour réaligner la vision du groupe. Un tableau de bord commun aurait facilité la gestion de l'architecture multi-threadée. Par ailleurs, l'hétérogénéité du traitement des erreurs a été résolue par une gestion centralisée, améliorant ainsi la robustesse globale de l'application.

Conclusion

Le projet LP25 nous a permis de passer de la théorie du langage C à une application système concrète et complexe. Nous avons appris à manipuler les fichiers virtuels de Linux, à sécuriser des échanges réseau via libssh et à concevoir une interface interactive. Au-delà du code, c'est la gestion de la communication interne au groupe qui a été notre plus grand apprentissage : passer d'un travail isolé à une intégration cohérente est le véritable défi d'un projet d'ingénierie.