Rapport de projet de LP25

- ALEXANDRE CUNEY TC 5

- Jean DELEPINE TC 3

- Amber GUYENOT-COSIO TC5

- Nikola MARCHAL TC 3

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.),
- et 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

Avant LP25, nos compétences en C s'appuyaient sur les cours/TD/TP : Amber avait un niveau plutôt avancé, mais anticipait des difficultés sur l'intégration système (interaction avec Linux, permissions, API bas niveau) et sur les interfaces textuelles — TUIs qu'elle ne connaissait pas au départ et qu'elle a ensuite apprises. Jean, davantage orienté C "standard", identifiait l'UI terminal et la communication réseau comme points sensibles ; il a appris ncurses et les TUIs, ainsi que la mise en réseau par SSH. Nikola, fort des TP, s'attendait à des défis de concurrence et de synchronisation, et a découvert et pratiqué le multi-threading pour y répondre. Alex, partant d'une base C des TD/TP, anticipait la gestion des options et de la configuration ; il a appris à gérer les arguments et les fichiers de configuration (parsing, validation et traitement des erreurs). Enfin, nous avons tous appris à précompiler et compiler directement avec gcc (alors qu'auparavant nous passions surtout par notre IDE).

Répartition des tâches : planification

Au démarrage du projet, la répartition des tâches était plutôt floue. Nous avons d'abord défini l'ordre des étapes et celles pouvant être menées en parallèle. Il avait été convenu que chacun avancerait à son rythme et passerait à la tâche suivante une fois la sienne terminée. Amber s'est

occupée de l’initialisation du projet (Makefile, structure et organisation des fichiers), puis a entamé la récupération des processus. En parallèle, Alex a travaillé sur la gestion des options, Nikola à entmé les manager ainsi que du multithreading, tandis que Jean a commencé l’interface utilisateur avec ncurses.

Répartition des tâches : réalisation

L’organisation initiale a été respectée et le démarrage s’est déroulé comme prévu. Au début, Amber s’occupait principalement de la gestion de GitHub (suivi des branches, merges et intégrations). En fin de projet, ce rôle a été davantage assuré par Jean. Parallèlement, Jean a implémenté la partie réseau avec Nikola, tandis qu’Alex s’est concentré sur le parsing des fichiers de configuration, et Amber a finalisé l’interface utilisateur.

Difficultés rencontrées

Compréhension des objectifs

Deux difficultés d’interprétation des consignes ont marqué le projet. D’abord, la consigne de dry-run nous paraissait incomplète : nous l’avions comprise comme un simple lancement sans UI validé s’il ne plante pas, alors qu’elle implique un diagnostic complet du programme ; nous avons affiné notre compréhension via recherches et exemples de pratiques, et cela aurait pu être anticipé par une définition fonctionnelle plus précise dès le cadrage. Ensuite, la consigne de configuration réseau laissait ouverte la priorité entre fichier de configuration et arguments (voire leur cumul au risque de conflits) ; nous avons tranché en donnant la priorité au fichier sur les arguments après analyse des scénarios d’usage, décision qui aurait pu être formalisée plus tôt en documentant la règle de précédence et en l’intégrant aux tests. Enfin, la consigne sur le transport réseau n’était pas claire quant à l’utilisation de SSH, de Telnet ou des deux ; nous avons complété notre compréhension en évaluant les contraintes et en standardisant d’abord autour de SSH, en laissant Telnet en option, ce qui aurait pu être acté plus tôt via un choix de transport formalisé dans un document de design et des tests exploratoires.

Réalisation des objectifs

Gestion des arguments réseau: les symptômes étaient une intégration difficile entre le parsing et le reste du projet, avec des interfaces peu pratiques; nous avons résolu en passant `mng_arg*` en tableau de pointeurs et en propageant tous les arguments dans `ntk_init` pour un point d’entrée unique; cela était partiellement anticipable avec une réflexion initiale sur le flux de données; une spécification précoce des structures d’arguments et une API claire auraient facilité la résolution. Validation des paramètres: les symptômes incluaient la validation d’entrées volontairement problématiques (inversions, paramètres de connexion vérifiés en mode local) qui pouvaient stopper le programme; nous avons restructuré le validator pour ne vérifier qu’un paramètre s’il est effectivement utilisé et ignorer la connexion en usage local; c’était anticipable; des tests ciblés de validation négative et une distinction nette des modes (local/distant) auraient accéléré la correction.

Dry-run: le symptôme principal était une description floue de la fonctionnalité; nous avons d'abord repris le code local sans lancer la boucle UI, puis créé une version distante, complexe car dépendant à la fois de la gestion des processus et du réseau; c'était peu anticipable; une définition fonctionnelle plus précise et un découpage en sous-modes avec mocks réseau auraient facilité l'implémentation.
Architecture réseau et transport: le symptôme était d'avancer « à l'aveugle » lors de la conception des arguments et de `network_init`, sans vision claire de la persistance de connexion; la résolution a été d'abandonner la connexion persistante au profit d'outils externes (SSH/Telnet) et de standardiser les opérations; cela aurait pu être anticipé par davantage de points d'alignement en équipe; un design document précoce et des spikes techniques sur les transports auraient réduit l'itération.
Interactions distantes (SSH/Telnet): les symptômes étaient la difficulté à interagir de manière fiable avec des processus distants (SSH OK mais Telnet non abouti); nous avons stabilisé l'implémentation autour de SSH et défini un périmètre minimal d'actions, en reportant Telnet; c'était en partie anticipable vu la complexité protocolaire; l'appui sur des bibliothèques éprouvées, un environnement de test distant et une gestion d'erreurs normalisée auraient facilité la réalisation.
Formatage ncurses: les symptômes étaient des retours à la ligne involontaires, des décalages de tabulations et des colonnes variant avec la taille de fenêtre; nous avons corrigé via des largeurs calculées dynamiquement, l'abandon des tabs au profit d'alignements fixes, et une gestion explicite des sauts de ligne; cela était partiellement anticipable; des gabarits d'UI, des tests sur tailles de fenêtre variées et l'usage de `pad`/mesures de largeur dès le début auraient évité une partie des problèmes.

Proposition d'une amélioration

Pour améliorer le projet, nous pourrions ajouter une détection fine de l'utilisation réseau par processus (débit entrant/sortant, connexions et sockets), ainsi que l'attribution des processus aux utilisateurs et aux machines distantes (via UID, hôte/IP). Nous proposerions aussi la possibilité d'interagir à distance avec les processus (kill, restart, terminate) de manière sécurisée, avec confirmations et journalisation. Par ailleurs, un mode de connexion réseau par Telnet (notamment vers le PC actuel) pourrait être pris en charge, tandis que la connexion SSH est déjà fonctionnelle. Enfin, une série de tests (unitaires, intégration et end-to-end) renforcerait la qualité : couverture des métriques réseau, du mapping utilisateur/machine, des commandes distantes, des permissions et des principaux cas d'erreur.

Analyse des résultats

- Objectifs vs livrables: l'application atteint l'affichage dynamique des processus en local avec UI ncurses, CPU/Mémoire fiables et interactions de base; côté distant, SSH est opérationnel pour la collecte et un périmètre minimal d'actions, Telnet reste optionnel/inachevé; le dry-run existe (local puis distant), mais sa sémantique a été clarifiée en cours de route.
- Points forts: montée en compétence notable (ncurses/TUI, SSH, multi-threading, parsing config/args), structuration progressive (validator conditionnel, API d'initialisation unifiée via `ntk_init`), corrections UI (alignements, largeur dynamique) et gouvernance Git stabilisée en fin de projet.

•Écarts et limites: métriques réseau paraissent globales et répliquées par processus (pas de débit par PID), Telnet non finalisé, définition initiale floue du dry-run, priorité config/args décidée tard; couverture de tests insuffisante pour verrouiller régression et cas d'erreur.
•Risques: ambiguïtés de sémantique (réseau par PID, horodatage), couplage transport si Telnet évolue, erreurs silencieuses côté validation en modes mixtes (local/distant) sans tests négatifs systématiques.
•Recommandations: implémenter réseau par PID (mapping sockets → PID, deltas), expliciter horodatage (start/elapsed), finaliser abstraction de transport SSH/Telnet, prioriser une suite de tests (unitaires/intégration/E2E) et documenter les règles de précédence config/args et le scope du dry-run.

Retour d'expérience

Plan vs Réalisation

•Objectif: htop-like local/distant avec UI, métriques et interactions; Réalisation: UI ncurses et collecte locale solides, SSH fonctionnel pour distant (Telnet en option), dry-run clarifié en cours de route, gouvernance Git stabilisée en fin de projet.

Divergences Majeures

•Définition floue initiale: répartition des tâches et sémantique du dry-run.
•Transport réseau: hésitation SSH vs Telnet (puis standardisation SSH, Telnet en option).
•Réseau par processus: métriques globales d'abord répliquées par PID, pas de débit PID fiable au début.

Blocages — Analyse

•Arguments réseau
  •Impact: intégration pénible, frictions entre parsing et modules; délais de colle.
  •Cause: API d'init non unifiée.
  •Résolution: passage à un tableau de pointeurs `mng_arg*` et propagation via `ntk_init`.
  •Mieux faire: définir l'API d'initialisation tôt; schéma de flux d'arguments.
•Validation des paramètres
  •Impact: arrêts injustifiés en mode local; perte de temps debug.
  •Cause: validation non contextuelle (local/distant) et cas négatifs peu testés.
  •Résolution: validator conditionnel (vérifier seulement les paramètres utilisés).
  •Mieux faire: tests négatifs dès le début; matrice des modes.
•Dry-run
  •Impact: aller-retour conception/implémentation; ambiguïtés de périmètre.
  •Cause: consigne incomplète; attentes non formalisées.
  •Résolution: version locale sans boucle UI, puis version distante plus riche.
  •Mieux faire: fiche de définition (objectifs, entrées/sorties, critères d'acceptation).
•Architecture réseau / transport
  •Impact: itérations "à l'aveugle"; temps perdu sur connexions persistantes.
  •Cause: choix de transport non figé; manque de spike technique.
  •Résolution: abandon persistance; outillage externe SSH/Telnet; opérations standardisées.
  •Mieux faire: design doc court, spike comparatif SSH/Telnet + décision écrite.
•Interactions distantes (SSH/Telnet)
  •Impact: Telnet non abouti; périmètre d'actions limité.
  •Cause: complexité protocolaire; environnement de test distant tardif.
  •Résolution: stabilisation autour de SSH; Telnet reporté.

- Mieux faire: s’appuyer sur lib éprouvée; bac à sable distant + jeux d’erreurs.
- Formatage ncurses
  - Impact: affichage instable (sauts de ligne, colonnes décalées).
  - Cause: tabs, largeurs non dynamiques, gestion fenêtre insuffisante.
  - Résolution: largeurs calculées, abandon des tabs, gestion explicite des sauts.
  - Mieux faire: gabarits d’UI, tests multi-tailles fenêtre, usage de pads dès le départ.

Causes Racines

- Spécifications partielles (dry-run, priorité config vs args, transport SSH/Telnet).
- Architecture décidée en marchant (init/arguments, réseau par PID).
- Tests insuffisants au début (peu de négatifs, peu d’intégration).

Actions Correctives

- Cadrage express (1–2 pages): objectifs, règles de précédence (config > args), choix transport (SSH principal, Telnet option), critères d’acceptation dry-run.
- API First: figer `manager`/`network`/init en signatures et structures partagées avant coder.
- Spikes courts: prototype transport, réseau par PID (mapping sockets → PID, deltas), ncurses gabarit.
- Tests minimaux précoces: négatifs validation, intégration local/distant, snapshot UI.
- Gouvernance: rôles Git et rythmes de merge définis dès S1; points d’alignement hebdo.

Enseignements Clés

- Clarifier tôt les consignes ambiguës évite des semaines d’itération.
- Un point d’entrée unique pour la config et les args simplifie tout le pipeline.
- Les TUIs demandent des gabarits et des métriques stables dès le début.
- Le transport réseau est un choix d’architecture, pas un détail d’implémentation.
- Un petit set de tests négatifs et d’intégration apporte un levier énorme de qualité.

Conclusion

En conclusion, le projet a atteint l’essentiel de ses objectifs: un htop-like en C avec une UI ncurses fluide, l’affichage fiable des processus (CPU/Mémoire) et un premier support distant via SSH, appuyés par une structuration plus nette (API d’initialisation unifiée, validation contextuelle des paramètres) et une vraie montée en compétence de l’équipe (TUI, réseau, multi-threading, parsing config/args, outillage gcc). Les écarts majeurs — sémantique du dry-run, choix de transport SSH/Telnet, mesures réseau par PID, mise en forme ncurses — ont été identifiés, traités pour l’essentiel ou cadrés pour la suite. Le produit est aujourd’hui une base stable et maintenable, prête à évoluer. Les priorités suivantes sont claires: ajouter des métriques réseau par processus, finaliser l’abstraction de transport (Telnet optionnel), renforcer les interactions distantes et doter le projet d’une suite de tests/CI. Sur le plan méthodo, nous retenons l’intérêt d’un cadrage court en amont (règles de précédence config/args, définition du dry-run), d’un “API-first” sur les modules clés et de tests négatifs précoces: autant de leviers qui accéléreront la prochaine itération et amélioreront encore la qualité.