Documentatie Automountershell

Instrumente și tehnici de bază în informatică - 2025
Autori - Ababei Raul-Costin si Iosub Dragos-Casian

CUPRINS

1. Descrierea problemei și contextul

În administrarea sistemelor de operare de tip UNIX/Linux, gestionarea dispozitivelor de stocare (partiții, unități externe, imagini de disc) reprezintă un punct critic de interacțiune între utilizator și kernel. Deși sistemul de fișiere virtual (VFS - Virtual File System) oferă o abstracție unificată (totul este un fișier), conectarea efectivă a dispozitivelor fizice la arborele de directoare ridică probleme semnificative de eficiență și administrare.

Proiectul AMSH (Automounter Shell) a fost conceput pentru a remedia deficiențele modelului tradițional de interacțiune cu sistemul de fișiere, adresând următoarele probleme fundamentale:

1.1. Fricțiunea în Fluxul de Lucru (Workflow Friction)

În mod convențional, accesarea datelor de pe un dispozitiv care nu este montat impune o întrerupere a fluxului de lucru al utilizatorului. Procesul este reactiv și manual:

- Utilizatorul încearcă să acceseze o cale (ex: cd /mnt/data).
- Sistemul returnează o eroare sau afișează un director gol, deoarece dispozitivul nu este atașat.
- Utilizatorul trebuie să oprească activitatea curentă.
- Trebuie identificate drepturile necesare și executată comanda privilegiată mount.
- Abia apoi se poate relua operațiunea inițială.

1.2. Persistența Nejustificată a Resurselor („Stale Mounts”)

O problemă majoră în administrarea sistemelor este tendința resurselor de a rămâne alocate mult timp după ce necesitatea lor a încetat. Utilizatorii montează frecvent dispozitive pentru o sarcină scurtă (ex: copierea unui fișier), dar omit să execute operațiunea inversă (umount).

Această neglijență generează multiple riscuri și ineficiențe:

- Riscuri de Securitate: Un dispozitiv montat rămâne accesibil oricărui proces sau utilizator cu drepturi suficiente. Dacă dispozitivul conține date sensibile (ex: „Secret Disk” menționat în configurare), menținerea sa activă crește suprafața de atac.
- Integritatea Datelor: Dispozitivele externe (USB stick-uri) sunt adesea deconectate fizic brusc. Dacă sistemul de fișiere nu a fost demontat logic (umount), buffer-ele de scriere nu sunt golite pe disc, ducând la coruperea datelor.
- Blocarea Resurselor: Un punct de montare activ poate preveni alte operațiuni administrative asupra dispozitivului sau directorului respectiv.

1.3. Lipsa de Transparență în Execuția Comenzilor

Shell-urile standard (Bash, Zsh) nu au cunoștință despre intenția utilizatorului în raport cu punctele de montare. Ele execută orbește comenzile.

- Scenariul cd: Comanda cd într-un shell obișnuit verifică doar existența directorului. Nu verifică dacă acel director *ar trebui* să fie punctul de intrare pentru un alt sistem de fișiere.
- Comenzi externe: Utilitare precum ls, cp sau editoarele de text eșuează dacă calea furnizată ca argument traversează un *mountpoint* inactiv. Shell-ul nu intervine pentru a pregăti mediul necesar execuției comenzii.

Această limitare forțează utilizatorul să micro-gestioneze starea sistemului, în loc să se concentreze pe sarcinile de nivel înalt. AMSH trebuie să acționeze ca un strat intermediar inteligent, interceptând aceste comenzi și asigurând „Just-in-Time Mounting” – montarea exact în momentul necesității.

2. Specificația soluției

Soluția **AMSH (Automounter Shell)** a fost concepută ca un middleware inteligent între utilizator și kernel, având scopul de a fluidiza interacțiunea cu dispozitivele de stocare. Prin preluarea automată a responsabilităților administrative, shell-ul elimină barierele procedurale cauzate de gestionarea manuală a punctelor de montare, permițând utilizatorului să se concentreze exclusiv pe accesarea datelor.

2.1. Automatizarea Accesului („Just-in-Time Mounting”)

Soluția elimină necesitatea comenzilor manuale de pregătire a mediului. Sistemul propune o schimbare de paradigmă de la modelul *Mount-Access-Unmount* la modelul simplificat *Access-Only*.

- Navigare Fluidă: Atunci când utilizatorul dorește să intre într-un director asociat unui dispozitiv extern (prin comanda de navigare), shell-ul detectează intenția și pregătește resursa instantaneu. Dacă dispozitivul nu este accesibil, acesta este montat automat înainte ca utilizatorul să ajungă efectiv în locație.
- Execuție Condiționată: Soluția extinde acest comportament și către aplicațiile sau comenzile externe. Dacă un utilizator încearcă să deschidă un fișier de pe un disc nemontat (folosind un editor text sau un player media), shell-ul interceptează cererea, montează discul necesar și abia apoi lansează aplicația solicitată. Astfel, erorile de tip „File not found” cauzate de dispozitive nemontate sunt eliminate.

2.2. Gestiunea Dinamică a Resurselor (TTL & Cleanup)

Pentru a optimiza consumul de resurse și a îmbunătăți securitatea, soluția implementează un mecanism de „Auto-Vindecare” a stării sistemului. Dispozitivele nu sunt lăsate conectate pe termen nelimitat, ci au o durată de viață (Time To Live - TTL) dinamică.

- Monitorizarea Inactivității: Sistemul urmărește constant când a fost utilizat ultima oară un dispozitiv. Odată ce perioada de inactivitate definită (de exemplu, 5 minute) a expirat, procesul de retragere a resursei este inițiat automat.
- Protecție Inteligentă la Demontare: Soluția distinge între „timp expirat” și „dispozitiv liber”. Chiar dacă timpul a expirat, sistemul nu va deconecta forțat un dispozitiv dacă acesta este încă utilizat de o aplicație sau de un proces de fundal. Demontarea se produce doar atunci când este 100% sigură, prevenind astfel pierderile de date sau coruperea sistemului de fișiere.

2.3. Transparență și Feedback Vizual

Deși automatizarea se produce în fundal, soluția propusă menține utilizatorul informat pentru a evita confuzia.

- Notificări în Timp Real: Sistemul trimite alerte vizuale non-intruzive către mediul desktop al utilizatorului de fiecare dată când o resursă este montată sau demontată automat.
- Raportare de Stare: Utilizatorul are acces oricând la o vizualizare clară a stării sistemului, putând vedea ce dispozitive sunt active și cât timp mai au până la expirarea sesiunii, permițând un control granular dacă situația o impune.

În esență, soluția AMSH transformă administrarea dispozitivelor dintr-o sarcină activă, repetitivă, într-un proces pasiv, gestionat autonom de către shell.

3. Designul și arhitectura aplicației

Sistemul AMSH (Automounter Shell) este proiectat pe baza unei arhitecturi modulare, ierarhizate, care separă responsabilitățile de interacțiune cu utilizatorul, logica de business (managementul montării) și prezentarea vizuală. Design-ul urmează principiul Separation of Concerns (SoC), asigurând extensibilitatea și mentenanța facilă a codului.

Din punct de vedere structural, aplicația funcționează ca un *wrapper* peste shell-ul nativ al sistemului de operare, orchestrând accesul la resurse printr-o serie de module interconectate.

3.1. Structura Modulară (Component Diagram)

Sistemul este divizat în patru componente logice majore, fiecare având un rol bine definit în ecosistemul aplicației:

A. Nucleul de Execuție (Core Orchestrator)

Această componentă este punctul central de intrare în aplicație, reprezentată de fișierul amsh.sh. Ea funcționează ca un Controller în arhitectura aplicației.

- Responsabilitate: Gestionează ciclul de viață al aplicației (bucla infinită de tip *REPL - Read-Eval-Print Loop*).
- Design: Este proiectat să captureze input-ul utilizatorului, să îl parseze și să decidă ruta de execuție: fie către o comandă internă (built-in), fie către sistemul de operare pentru comenzi externe.
- Integrare: Importă și coordonează funcționalitățile din modulele auxiliare (source ./mount_manager.sh, source ./style.sh).

B. Managerul de Resurse (Business Logic Layer)

Logica complexă a sistemului este izolată în modulul mount_manager.sh. Acesta acționează ca un strat de servicii.

- Responsabilitate: Gestionează starea efectivă a dispozitivelor de stocare. Nu interacționează direct cu utilizatorul, ci răspunde solicitărilor venite de la Nucleul de Execuție.
- Design: Encapsulează regulile de validare a căilor, verificarea existenței dispozitivelor, logica de expirare (TTL) și interacțiunea cu kernel-ul pentru operațiunile de mount și umount. Tot aici se află logica de scanare a hardware-ului nou (lsblk).

C. Stratul de Prezentare (UI/UX Layer)

Interfața cu utilizatorul este decuplată de logică prin modulul style.sh.

- Responsabilitate: Asigură consistența vizuală a aplicației (TUI - Text User Interface).
- Design: Conține definițiile pentru paleta de culori, formatarea tabelelor (pentru istoric sau status), bannerele ASCII și elementele de branding. Acest design permite modificarea aspectului aplicației fără a risca alterarea funcționalității de bază.

D. Configurație și monitorizarea activității (Data & State)

Sistemul utilizează o abordare hibridă pentru stocarea datelor, combinând configurarea statică cu starea dinamică.

- Configurația Statică (amsh.conf): Servește drept bază de date persistentă a sistemului, mapând dispozitivele fizice la punctele de montare și atributele lor (sistem de fișiere, TTL). Este un fișier text structurat, ușor de editat manual sau programatic.
- Starea Efemeră (TTL Tracking): Pentru a monitoriza activitatea, sistemul utilizează fișiere santinelă (marker files) stocate într-un director temporar (/tmp/amsh_ttl). Design-ul acestui mecanism permite persistența stării de "ultim acces" independent de memoria procesului shell.

E. Inițializarea mediului de testare

Pentru a asigura un mediu de rulare consistent, arhitectura include un modul de inițializare (setup_demo.sh). Deși nu face parte din runtime-ul aplicației, acesta este esențial pentru design-ul sistemului, deoarece generează infrastructura necesară (imagini de disc virtuale, formatarea partițiilor) pe care se bazează logica din amsh.conf.

4. Implementare

Implementarea se axează pe execuția securizată a comenzilor interne, pe gestionarea punctelor de montare/demontare, pe eficiența algoritmului de monitorizare a inactivității prin parametrii TTL (Time To Live) și pe utilizarea simbiotică a utilitarelor native Linux pentru a asigura un management robust al resurselor. Prin acest demers, soluția dezvoltată transformă regulile administrative rigide în procese autonome, garantând integritatea datelor și o experiență de utilizare fluidă, fără intervenții manuale repetitive.

4.1. Implementarea Comenzilor Interne (Built-in)

Deoarece AMSH rulează într-o buclă infinită (while true) care interceptează input-ul utilizatorului, comenzile interne sunt implementate ca funcții Bash apelate direct din această buclă, înainte ca interpretorul să încerce executarea lor ca programe externe.

Comanda cd (Change Directory)

Spre deosebire de comanda standard, implementarea cd în AMSH include un pas de pre-procesare critic:

1. Normalizarea Căii: Argumentul primit este convertit într-o cale absolută folosind realpath -m. Aceasta permite compararea corectă cu intrările din fișierul de configurare, indiferent dacă utilizatorul folosește căi relative (ex: ../usb) sau absolute.
2. Verificare și Montare: Se apelează funcția smart_mount cu calea absolută. Dacă directorul țintă este un *mountpoint* definit dar nemontat, scriptul îl montează automat înainte de a schimba directorul de lucru.

3. Schimbarea Contextului: Doar după succesul montării se execută builtin cd pentru a muta procesul shell în noul director.

Comanda scan

Această comandă automatizează popularea fișierului de configurare. Implementarea se bazează pe parsarea ieșirii comenzii lsblk:

- Filtrează dispozitivele care sunt partiții (TYPE="part") și au un sistem de fișiere valid, dar nu sunt deja montate de sistem.
- Exclude dispozitivele loop și cele care există deja în amsh.conf.
- Pentru dispozitivele valide, creează automat un punct de montare în /tmp/ și adaugă o intrare în configurare cu un TTL implicit de 5 minute.

Comanda history

Implementează un istoric persistent între sesiuni, salvând comenzile în fișierul ~/.amsh_history.

- La fiecare comandă validă introdusă, aceasta este adăugată (append) în fișier.
- Opțiunea history -c trunchiază fișierul (> "$HISTORY_FILE") pentru a șterge istoricul, oferind feedback vizual imediat.

Comanda status

Această comandă oferă o imagine de ansamblu în timp real asupra stării sistemului de stocare gestionat. Implementarea, localizată în funcția show_mount_status din mount_manager.sh, interoghează starea fiecărei intrări din fișierul de configurare.

- Verificarea Stării: Iterează prin amsh.conf și utilizează mountpoint -q pentru a determina dacă un dispozitiv este montat sau nu.
- Calculul Dinamic al TTL: Pentru dispozitivele montate, scriptul citește timestamp-ul din fișierul martor asociat (/tmp/amsh_ttl/...). Calculează diferența dintre timpul curent (date +%s) și ultimul acces, afișând timpul rămas până la demontare. Dacă timpul este critic (sub 60 de secunde), acesta este evidențiat vizual cu roșu.
- Feedback Vizual: Utilizează coduri de culoare ANSI (Verde/Roșu) definite în variabile (ex: ${GREEN}, ${RED}) pentru a indica rapid utilizatorului starea activă sau inactivă a fiecărui mountpoint.

Comanda locate

Funcționează ca un motor de căutare intern, specializat pe dispozitivele gestionate de AMSH. Spre deosebire de comanda standard locate din Linux care interoghează o bază de date indexată, implementarea din AMSH efectuează o căutare "live".

- Căutare Contextuală: Comanda parcurge lista de mountpoint-uri din configurație. Căutarea se efectuează *doar* în directoarele care sunt efectiv montate în momentul execuției (mountpoint -q), evitând erorile de I/O pe dispozitive deconectate.
- Utilizarea find: În spatele scenei, este apelat utilitarul find "$mpoint" -type f -iname "*$search_term*". Argumentul -iname asigură o căutare insensibilă la majuscule/minuscule, iar erorile standard (stderr) sunt redirecționate la /dev/null pentru a menține ieșirea curată.

Comanda usage

Oferă statistici despre gradul de ocupare a discurilor, fiind un wrapper personalizat peste utilitarul de sistem df (disk free). Implementarea se regăsește în funcția show_disk_usage.

- Extragerea Datelor: Pentru fiecare mountpoint activ, execută df -h. Ieșirea este procesată imediat folosind awk pentru a extrage doar coloanele relevante: Dimensiune, Spațiu Utilizat, Spațiu Disponibil și Procentaj de Ocupare.
- Formatare Tabelară: Datele sunt aliniate folosind printf pentru a genera un tabel lizibil în terminal. Pentru dispozitivele nemontate, scriptul afișează placeholder-ul - în loc de erori, indicând că datele nu sunt accesibile momentan.

Comanda help

Are rolul de a asista utilizatorul în navigarea funcționalităților shell-ului. Este implementată în modulul de stilizare (style.sh) prin funcția print_help_menu.

- Randare Statică: Afișează un meniu predefinit folosind instrucțiuni echo și printf, care listează toate comenzile disponibile, sintaxa acestora și o scurtă descriere a acțiunii.
- Consistență Vizuală: Utilizează aceleași elemente de design (borduri ASCII, culori Cyan/Yellow) ca și restul interfeței TUI pentru a menține unitatea estetică a aplicației.

Comanda exit

Această comandă nu doar termină execuția shell-ului, ci declanșează o procedură critică de curățare („Cleanup routine”) pentru a asigura integritatea sistemului la ieșire.

- Curățare Proactivă: Înainte de a opri procesul, apelează funcția cleanup_all_mounts. Aceasta iterează prin toate dispozitivele configurate și forțează demontarea (sudo umount) celor active, prevenind situația de "stale mounts" (dispozitive uitate montate) după închiderea aplicației.
- Ștergerea Urmelor: Înlătură fișierele de tracking TTL din /tmp/amsh_ttl/ corespunzătoare dispozitivelor demontate.

- Încheierea Buclei: După finalizarea operațiunilor de mentenanță și afișarea unui mesaj de rămas bun (print_exit_message), comanda break întrerupe bucla infinită while true din amsh.sh, permițând scriptului să se încheie natural.

4.2. Logica de Mount/Umount și Tracking TTL

Gestionarea ciclului de viață al dispozitivelor de stocare este realizată în modulul mount_manager.sh. Arhitectura prioritizează integritatea datelor, asigurând că operațiunile de montare sunt executate „la cerere”, iar demontarea se realizează doar în condiții de siguranță, validată de starea sistemului.

Mecanismul de Montare și Demontare Securizată

Logica de bază a sistemului AMSH este guvernată de principiul *Safety First*. Operațiunile asupra sistemului de fișiere sunt executate prin două funcții critice:

1. Montarea Activă (smart_mount): Această funcție reprezintă punctul de intrare pentru accesul la resurse. Înainte de a executa comanda de sistem mount, funcția validează existența dispozitivului și a punctului de montare. Dacă validările trec, se execută montarea cu parametrii specifici (-t fstype) preluați din configurație, asigurând accesul imediat la date.

Safety Lock (Prevenirea Erorilor EBUSY): Cea mai complexă parte a logicii de demontare este prevenirea întreruperii proceselor active. Chiar dacă perioada de utilizare a expirat, AMSH refuză să demonteze un dispozitiv aflat în uz. Scriptul utilizează utilitarul fuser (File User) pentru a interoga kernel-ul cu privire la procesele care dețin descriptori de fișiere deschiși pe punctul de montare vizat.

2. Comanda umount este executată condiționat, doar dacă fuser returnează un cod de ieșire care indică lipsa activității. Această barieră logică este esențială pentru a preveni coruperea datelor sau blocarea sistemului.

Monitorizarea Inactivității (TTL Tracking)

Decizia de a încerca demontarea (care va fi supusă verificării de siguranță de mai sus) este luată pe baza unui algoritm de tracking al timpului, care funcționează astfel:

- Fișiere Martor (Timestamp Files): Sistemul nu menține timere în memorie, ci utilizează sistemul de fișiere pentru persistență. Un director temporar dedicat, /tmp/amsh_ttl, stochează fișiere goale care acționează ca indicatori de timp. La fiecare accesare validă (prin smart_mount), timpul de acces al fișierului martor corespunzător este actualizat folosind comanda touch. Numele fișierului este derivat din calea punctului de montare (ex: /mnt/usb devine _mnt_usb) pentru unicitate.
- Algoritmul de Verificare (check_and_umount_expired): Această rutină rulează ciclic în bucla principală a shell-ului și evaluează "vârsta" fiecărei conexiuni:

1. Citește momentul ultimei accesări a fișierului martor folosind stat -c %Y.
2. Compară această valoare cu timpul curent al sistemului (date +%s).
3. Calculează delta (diferența) în minute. Dacă (Timp_Curent - Ultimul_Acces) > TTL_Configurat, se inițiază procedura de demontare securizată descrisă anterior.

4.3. Integrarea Tehnologiilor și Utilitarelor Linux

AMSH nu reinventează mecanismele de sistem, ci le orchestrează folosind utilitare standard Linux.

fuser (File User)

Este utilizat în modul "silent" (-s) pentru a detecta blocajele asupra sistemelor de fișiere. Este piesa centrală a mecanismului de siguranță, garantând că AMSH nu demontează niciodată un dispozitiv aflat în uz.

notify-send (Desktop Notifications)

Pentru a trimite notificări din scriptul care rulează cu privilegii root (via sudo) către sesiunea grafică a utilizatorului obișnuit, implementarea folosește o tehnică de injectare a variabilelor de mediu:

- Identifică utilizatorul real folosind variabila ${SUDO_USER}.
- Setează DBUS_SESSION_BUS_ADDRESS pentru a comunica cu magistrala de mesaje a sesiunii utilizatorului (ex: /run/user/$UID/bus).
- Aceasta permite afișarea notificărilor vizuale (bule de tip "Toast") în GNOME/KDE când se montează/demontează un disc.

lsblk (List Block Devices)

Este motorul comenzii scan. Este rulat cu parametrii -rn -o PATH ,FSTYPE ,TYPE, MOUNTPOINT pentru a obține o ieșire "raw" (fără formatare tabelară), ușor de parsat programatic. Aceasta permite identificarea precisă a tipului de dispozitiv și a sistemului de fișiere detectat (ex: vfat, ext4).

trap (Signal Trapping)

Scriptul implementează prinderea semnalului SIGINT (generat de combinația de taste Ctrl+C). Funcția handle_sigint este definită pentru a preveni închiderea accidentală a shell-ului AMSH, asigurând continuitatea sesiunii de lucru.

5. Scenarii de testare

Pentru validarea soluției AMSH, a fost concepută o metodologie de testare menită să simuleze condiții reale de utilizare a unui sistem de operare, dar într-un mediu controlat și reproductibil. Testele au vizat verificarea corectitudinii mecanismelor de automatizare (mount/umount), a preciziei algoritmului TTL și a robusteții în situații de conflict de resurse.

5.1. Mediul de Experimentare

Deoarece testarea cu dispozitive fizice (stick-uri USB, HDD-uri externe) este dificil de automatizat și replicat identic, mediul de testare a fost virtualizat folosind scriptul dedicat setup_demo.sh.

- Infrastructură: Testele au fost rulate pe o distribuție standard de Linux (ex: Ubuntu/Debian), necesitând privilegii de superutilizator (sudo) pentru manipularea sistemelor de fișiere.
- Virtualizarea Dispozitivelor: Au fost generate trei imagini de disc virtuale (loopback devices) pentru a simula hardware diferit:
    1. Simulare USB Stick: Fișier imagine de 20MB formatat cu sistemul de fișiere vfat (FAT32), tipic pentru memorii flash.
    2. Simulare HDD Work: Imagine formatată ext4, simulând o partiție de date persistentă.
    3. Simulare Disc Securizat: Imagine ext4 cu un TTL foarte scurt (2 minute), pentru testarea rapidă a expirării.
- Configurație Inițială: Sistemul a fost preîncărcat cu un fișier amsh.conf generat automat, care mapează aceste imagini virtuale către directoare din /tmp/ (ex: /tmp/amsh_usb), izolând astfel testele de restul sistemului de operare.

5.2. Scenarii de Testare Rulate

Scenariul A: Navigarea Transparentă (Comanda cd)

- Obiectiv: Verificarea montării automate la schimbarea directorului.
- Acțiune: Din promptul AMSH, s-a executat comanda cd /tmp/amsh_usb (în timp ce dispozitivul era demontat).
- Rezultat Obținut:
    - Sistemul a detectat intersecția cu un mountpoint configurat.
    - A afișat mesajul [SYSTEM] Montare automată: /tmp/amsh_usb.
    - A trimis o notificare desktop vizuală.
    - Promptul s-a schimbat, indicând noua cale, iar conținutul directorului a devenit accesibil instantaneu.

Scenariul B: Acces prin Comenzi Externe

- Obiectiv: Verificarea interceptării argumentelor pentru utilitare de sistem.
- Acțiune: S-a rulat comanda cat /tmp/amsh_secret/Parole.txt fără a monta manual discul "secret".

- Rezultat Obținut:
    - Shell-ul a analizat argumentele, a identificat calea /tmp/amsh_secret ca fiind inactivă.
    - A executat montarea în fundal ("Just-in-Time").
    - Comanda cat a afișat conținutul fișierului imediat după montare, fără a returna eroare de tip "File not found".

Scenariul C: Auto-Demontarea (TTL Expiration)

- Obiectiv: Validarea mecanismului de curățare a resurselor inactive.
- Acțiune: S-a accesat discul "Secret Disk" (TTL = 2 minute), apoi s-a revenit în directorul home (cd ~) și s-a așteptat 3 minute fără activitate.
- Rezultat Obținut:
    - La verificarea periodică din bucla principală, funcția check_and_umount_expired a detectat depășirea timpului limită.
    - S-a validat că resursa este liberă (folosind fuser).
    - Sistemul a executat umount și a trimis notificarea "TTL expirat pentru ...". Comanda status a confirmat ulterior starea "DEMONTAT".

Scenariul D: Protecția la Utilizare Activă (Safety Lock)

- Obiectiv: Testarea comportamentului când timpul expiră, dar resursa este ocupată.
- Acțiune:
    - S-a montat discul și s-a deschis un fișier de pe acesta într-un editor de text (ex: nano), menținându-l deschis peste limita de timp TTL.
    - Alternativ, s-a rămas cu shell-ul în interiorul directorului montat.
- Rezultat Obținut:
    - Deși timpul a expirat, AMSH a detectat prin fuser -s că resursa este blocată de un proces.
    - Demontarea a fost amânată. Sistemul a reîncercat verificarea la ciclurile următoare, executând umount doar după închiderea editorului de text sau părăsirea directorului.

Scenariul E: Detectarea Dinamică (scan)

- Obiectiv: Integrarea unui dispozitiv nou, neconfigurat.
- Acțiune: S-a atașat o nouă imagine de disc (loop device) neconfigurată și s-a rulat comanda scan.
- Rezultat Obținut:
    - Comanda lsblk a identificat noua partiție.
    - Scriptul a generat automat o intrare în amsh.conf și a creat punctul de montare necesar.
    - Dispozitivul a devenit imediat disponibil pentru comenzile cd și status.

5.3. Concluzii Experimentale

Testele efectuate au demonstrat că AMSH îndeplinește cu succes cerințele de automatizare și siguranță. Latența introdusă de procesul de montare este neglijabilă pentru utilizator (sub 1 secundă pentru imaginile testate), iar mecanismul de *safety lock* a prevenit eficient erorile de sistem în 100% din cazurile de conflict simulate. Sistemul s-a dovedit stabil, gestionând corect tranzițiile de stare ale dispozitivelor fără a necesita intervenția manuală a operatorului.

6. Concluzii

Proiectul AMSH (Automounter Shell) a reuşit să atingă obiectivele stabilite în tema de proiectare, livrând un sistem funcțional care automatizează complet interacțiunea utilizatorului cu dispozitivele de stocare externe. Prin implementarea unei arhitecturi modulare în Bash, s-a demonstrat că gestionarea resurselor de sistem poate fi abstractizată transparent, eliminând barierele operaționale dintre utilizator și datele sale.

6.1. Rezultate Obținute

Principala realizare a proiectului este crearea unui mediu de execuție robust, capabil să intercepteze intenția utilizatorului și să acționeze proactiv.

- Eficiență Operațională: Soluția a eliminat necesitatea intervenției manuale pentru operațiunile de mount și umount. Testele au confirmat că timpul necesar accesării unui dispozitiv nou a fost redus la simpla tastare a unei căi de director, restul procesului fiind gestionat invizibil de shell.
- Optimizarea Resurselor: Prin implementarea algoritmului TTL (Time To Live), s-a rezolvat problema "dispozitivelor uitate". Sistemul a demonstrat capacitatea de a elibera resursele inactive, prevenind încărcarea inutilă a sistemului de operare, dar menținând în același timp integritatea datelor prin mecanismul de protecție la utilizare (safety lock).
- Integrare și Feedback: Spre deosebire de scripturile de administrare silențioase, AMSH oferă o experiență interactivă completă, integrând notificări desktop (notify-send) și o interfață vizuală în terminal (TUI) pentru istoricul comenzilor și starea sistemului.

6.2. Concepte și Competențe Dobândite

Dezvoltarea AMSH a implicat aprofundarea unor concepte avansate de programare de sistem și administrare Linux:

- Manipularea Avansată a Shell-ului: Proiectul a necesitat utilizarea intensivă a procesării de text (via awk, grep), gestionarea semnalelor (trap SIGINT) și a structurilor de control complexe pentru a emula comportamentul unui shell real.
- Interacțiunea cu Kernel-ul: S-a înțeles în profunzime modul în care VFS (Virtual File System) gestionează punctele de montare și cum utilitare precum lsblk, mount sau fuser pot fi orchestrate pentru a manipula starea hardware-ului.
- Arhitectura Modulară: Separarea logicii de business (mount_manager.sh) de interfața cu utilizatorul (style.sh) și de bucla principală de execuție (amsh.sh) a evidențiat importanța design-ului software curat pentru mentenabilitatea codului.

6.3. Direcții de Dezvoltare Ulterioară

Deși versiunea curentă este stabilă și funcțională, arhitectura permite extinderi viitoare pentru a transforma AMSH într-un instrument de producție:

1. Suport pentru Autocompletare (Intelligent Tab-Completion)

În stadiul curent, AMSH procesează input-ul utilizatorului printr-o comandă standard read, ceea ce nu permite utilizarea tastei TAB pentru completarea automată a căilor sau a comenzilor. O evoluție majoră a UX (User Experience) ar fi integrarea legăturilor cu biblioteca readline sau implementarea unui parser de input mai avansat. Aceasta ar permite utilizatorului să autocompleteze rapid numele directoarelor din punctele de montare (chiar și a celor nemontate) și numele comenzilor interne, reducând timpul de tastare și riscul de erori sintactice.

2. Jurnalizare Auditabilă și Securizată (Secure Event Logging)

Deși AMSH oferă feedback imediat prin notificări și status, lipsește o persistență istorică detaliată a evenimentelor de sistem. O dezvoltare viitoare vizează implementarea unui registru de audit (Audit Log) dedicat, separat de istoricul comenzilor (.amsh_history).

Acest fișier ar înregistra cronologic, cu timestamp precis, fiecare operațiune de mount și umount reușită sau eșuată. Log-ul ar fi configurat cu permisiuni stricte (read-only pentru utilizator, append-only pentru sistem), oferind o trasabilitate clară a accesului la date sensibile și facilitând diagnosticarea problemelor de securitate.