Listă proiecte

1 Regulament

- Proiectul trebuie încheiat până la sfarsitul semestrului
- Proiectul valorează maxim 30 de puncte
- Toți membrii echipei primesc același punctaj
- Fiecare membru trebuie să își prezinte contribuția (afectând nota finală corespunzător)

2 Proiecte

1. **UserFS** (1–2 studenți) – Scrieti un program (script) shell care reprezinta fiecare utilizator activ din sistem printr-un director in care se gaseste un fisier *procs* care listeaza procesele curente ale utilizatorului. Toate aceste directoare care contin informatii despre un utilizator activ sunt grupate intr-un acelasi director, numit director radacina. Periodic, sa zicem la fiecare 30s, informatia despre utilizatorii activi se actualizeaza. Pentru utilizatorii care dispar din sistem in timp, directorul corespunzator va afisa un fisier *procs* gol si un fisier suplimentar *lastlogin* care afiseaza data ultimei sesiuni a utilizatorului in sistem.

2. **MyLast** (1–2 studenți) – Scrieți un program (script) shell care emuleaza comportamentul comenzilor `last/lastb` folosind in acest sens continutul fisierului `/var/log/auth.log[1-4]` (Atentie: nu `/var/log/wtmp` sau `/var/log/btmp` care sunt fisiere binare). Implementarea trebuie sa suporte minimal comportamentul comenzilor `last/lastb` indus de folosirea flag-urilor `-n, -p, -s, -t`. Puteti implementa doua comenzi diferite, la fel ca in cazul `last/lastb`, sau puteti implementa o singura comanda care sa emuleze comportamentul celor doua.
   Obs: Comanda `less` stie sa decomprime fisiere de tip gzip, adica puteti rula comanda `less /var/log/auth.log.3.gz` si sa obtineti rezultatul anticipat, dar alte comenzi, de pilda `grep` sau `more`, nu stiu.

3. **HTMLPrettyPrinter** (1–2 studenți) – Scrieti un program (script) shell care primeste ca input un fisier HTML si il formateaza a.i. sa reflecte vizual structura ierarhica si nivelul de imbricare al resurselor folosite prin indentarea corespunzatoare a tag-urilor si strigurilor.

4. **LazyWGET** (1–2 studenți) – Scrieti un program (script) shell `lwget` care foloseste comanda `wget` NERECURSIV (adica nu puteti folosi comanda cu flag-ul *-r*) pentru a descarca pe calculatorul local documente HTML de pe internet. `lwget` descarca un SINGUR fisier HTML de pe internet cu ajutorul comenzii `wget` la un moment dat, ii parseaza continutul si descarca apoi recursiv resursele HTML din document, emuland comportamentul comenzii `wget -r`.

Diferenta fata de `wget -r` este ca descarcarea recursiva a resurselor referite in documentul HTML se face *lenes/lazy*, in sensul ca toate tag-urile HTML care definesc resurse ce trebuie descarcate sunt definite ca *promisiuni/promises* care nu sunt evaluate decat atunci cand comanda `lget` *forteaza* aceste promisiuni printr-un nou apel.
Concret, fiecare comanda `lwget <URL>` *forteaza* evaluarea nivelului curent de recursivitate in arborerele de documente: primul apel descarca fiserul HTML furnizat ca parametru in linie de comanda (radacina arborelui de documente), al doilea apel parseaza acest fisier si descarca local documentele referite de fisierul HTML, al treilea apel parseaza toate documentele HTML descarcate in etapa anterioara si descarca documentele HTML referite de catre acestea, samd. Dupa fiecare etapa, puteti folosi comenzi uzuale de tip `ls` sau `tree` pentru a verifica comportamentul comenzii `lwget`.

5. **FIFOServer** (2–3 studenți) – Scrieti un program client-server folosind bash care foloseste fisiere speciale de tip FIFO drept canale de comunicatie intre clienti si server. Serverul foloseste un fisier FIFO cu nume cunoscut (*well-known FIFO*), configurabil intr-un fisier de configurare. Serverul primeste in fisierul FIFO cereri de la clienti pentru pagini de manual referitoare la diferite comenzi. Formatul cererilor clientilor este urmatorul:

   BEGIN-REQ [client-pid: command-name] END-REQ

   unde *pid* este ID-ul de proces al clientului iar *command-name* este numele comenzii shell pentru care clientul cere informatiile corespunzatoare din pagina de manual (i.e., executia comenzii *man command-name*). Serverul citeste cererile din FIFO si raspunde clientilor in fisiere FIFO personalizate pentru fiecare client. Aceste fisiere FIFO personalizate au un nume standard al caror nume contine PID-ul clientului (eg. */tmp/server-reply-XXXX* unde XXXX reprezinta valoarea PID-ului client). Raspunsul concret este continutul paginilor de manual pentru *command-name*. Dupa citirea paginilor de manual si afisarea continutului lor pe ecran, clientul sterge fisierul FIFO personal.

6. **XML** (2–3 studenți) – Scrieți un program (script) shell care să parseze fișiere XML. Acesta trebuie să poată citi și scrie date din, respectiv, în format XML.

7. **SLCheck** (1–2 studenți) – Scrieti un program (script) shell care primeste ca parametru un director (eventual chiar radacina sistemului de fisiere) si verifica daca exista *broken links*, link-uri simbolice care refera fisiere inexistente. Programul functioneaza recursiv, verificand intreaga ierarhie de fisiere si directoare accesibila pornind din directorul furnizat ca parametru al programului. Atunci cand intalneste un link simbolic catre un director, comportamentul implicit al programului este sa nu dereferentieze link-ul, adica sa nu porneasca recursiv analiza directorului referentiat. Acest comportament se poate schimba daca programul este apelat cu flag-ul *–follow-symlinks*, caz in care analiza recursiva continua cu directorul referit in link-ul simbolic.
   Indicatie Cel mai simplu mod de a genera *broken links* este sa folositi un RAM disk (sistem de fisiere tip `tmpfs` in Linux) pe care sa copiati o structura de directoare, de pilda `/etc`. La dezinstalarea RAM discului (operatie de `umount`), datele dispar automat si eventualele link-uri simbolice create catre RAM disc devin broken.
   N.B. Parcurgerea in adancime a ierarhiei de directoare trebuie implementata direct in program, nefiind permisa folosirea unor comenzi shell care ajuta in acest task, cum ar fi comanda `find`.

8. **PackageMonitor** (2–3 studenți) – Scrieti un program (script) shell care monitorizeaza pachetele software existente pe un sistem Linux folosind informatia din `/var/log/dpkg.log`. Programul urmareste in principal operatiile de *install* si *remove*. Programul are doua componente principale: un front-end utilizator si monitorul propriu-zis. Monitorul este lansat periodic din `cron` si parseaza informatia din `/var/log/dpkg.log` pe care o organizeaza intr-un director de lucru in asa fel incat sa fie permisa indexarea dupa numele pachetului software, respectiv dupa data. Fiecare pachet software are propriul subdirector in directorul de lucru in care se gaseste istoria operatiilor de *install/remove* ale pachetului. Front-end-ul analizeaza informatiile stranse de monitor in directorul de lucru si ofera utilizatorului urmatoarele functionalitati:
   - lista pachetelor software curent instalate in sistem si data ultimei instalari
   - lista pachetelor software care au fost instalate candva in sistem si au fost inlaturate intre timp precum si data ultimei inlaturari
   - istoria operatiilor efectuate in timp asupra unui anumit pachet software
   - lista pachetelor software instalate/inlaturate intr-o perioada data de timp

   Analiza monitorului se poate face pe o perioada de timp flexibila, furnizata ca parametru in linia de comanda.

9. **AutomounterShell** (2–3 studenți) – Scrieti un program (script) shell `amsh` care implementeaza un shell. Programul afiseaza un prompt (e.g., *amsh*>), citeste in bucla comenzi externe introduse de la tastatura si le executa folosind comanda `sh -c`. Pentru a naviga prin sistemul de fisiere, `amsh` trebuie sa implementeze comanda interna `cd`, care are un comportament diferit de comanda interna similara din shell. Mai exact, `cd` e implementata in `amsh` ca o functie care parseaza calea primita si verifica daca aceasta cale nu traverseaza cumva un mountpoint definit intr-un fisier de configurare similar cu `/etc/fstab`. Daca da, si daca mountpoint-ul nu este instalat in sistemul de fisiere, `cd` executa mai intai operatia de `mount` cu sistemul respectiv de fisiere si abia apoi schimba corespunzator directorul. Daca mountpoint-ul era deja instalat, `cd` se margineste la comportamentul uzual schimband corespunzator directorul. In mod similar trebuie parsata orice cale furnizata ca parametru unei comenzi externe care traverseaza un astfel de mountpoint. In cazul in care mountpoint-ul nu este instalat, el trebuie intai inclus in sistemul de fisiere cu comanda `mount` si abia apoi se executa comanda shell care necesita mountpoint-ul.
   In rezumat, orice comanda *atinge* un astfel de mountpoint va genera o operatie de `mount` corespunzatoare daca sistemul de fisiere respectiv nu era deja instalat.
   Odata instalat in sistemul de fisiere, un astfel de mountpoint are o durata de viata limitata, care poate fi setata in fisierul de configurare, sa zicem 5 minute. Dupa aceasta perioada, `amsh` verifica daca exista procese care folosesc mountpointul si, in caz negativ, dezinstaleaza mountpoint-ul apeland comanda `umount`. In caz contrar, lasa mountpointul instalat.
   Obs: daca shell-ul insusi foloseste un astfel de mountpoint, simpla operatie de schimbare a directorului in afara subarborelui cu radacina in mountpoint trebuie sa genereze dezinstalarea lui la sfarsitul urmatoarei perioade de 5 min, cu conditia ca nici un alt program sa nu fi folosit mountpoint-ul intre timp.

10. **File&DiskUsageMonitor** (1–2 studenți) – Comanda `script` permite inregistrarea unei sesiuni de lucru in shell intr-un fisier numit implicit `typescript`. Scrieti un program (script) shell care foloseste una sau mai multe inregistrari `typescript` pentru a analiza si monitoriza evolutia in timp a structurii de fisiere si directoare de interes pentru utilizator, precum si a spatiului utilizat pe disc. Concret, programul poate analiza output-ul unor comenzi de tip `ls`

`-l` la momente diferite de timp (i.e., intre diferite fisiere `typescript`, sau intre un anumit fisier `typescript` si momentul curent) si sa genereze un raport cu privire la fisierele/directoarele aparute/disparute in timp. Diferenta intre rezultatele diferitelor comenzi `ls` se poate face cu comanda `diff`, dar programul trebuie sa afiseze aceste diferente intr-o forma accesibila utilizatorului neprofesionist. Similar, programul trebuie sa faca o analiza a utilizarii spatiului de disc asa cum e raportata de comanda `df`.