Listă proiecte sisteme de operare

1 Regulament

- Implementarea trebuie urcată pe moodle de fiecare student până la data de 14.01.2019
- Proiectul valorează maxim 20 de puncte
- Toți membrii echipei primesc același punctaj
- Fiecare membru trebuie să își prezinte contribuția (afectând nota finală corespunzător)

2 Proiecte

1. **Shell** (2–3 studenți) – Scrieți un shell care să implementeze elementele de bază găsite în orice linie de comandă (ex. istoric comenzi, pipe, expresii logice, suspendarea unui program).
2. **ListPIDs** (1–2 studenți) – Implementați o funcție sistem care să ofere date (PID, stare, statistici legate de execuție etc.) despre toți descendenții unui proces dat (în ordine DFS). Scrieți programe în userland care să afișeze arborii rezultați folosind noua funcție de sistem.
3. **EventSync** (1–2 studenți) – Scrieți un set de funcții de sistem care să permită sincronizarea mai multor procese legate de apariția unui evenmient. Procesele își opresc execuția până un terț semnalează apariția evenimentului dorit. Funcții necesare `eventopen`, `eventclose`, `eventwait`, `eventsignal`.
4. **MutexPolicy** (3–4 studenți) – Scrieți un daemon în userland care să decidă politica de acces la mutecșii creați cu ajutorul unui set nou de funcții de sistem implementat de dumneavoastră: `mtxopen`, `mtxclose`, `mtxlock`, `mtxunlock`, `mtxlist` și `mtxgrant`. Ultimele două funcții sunt folosite de daemon pentru a stabili care proces obține acces la mutex. Mutecșii sunt vizibili de orice proces din sistem, dar fiecare proces păstrează o listă de mutecși folosiți (precum descriptorii de fișiere).
5. **UserSched** (2–3 studenți) – Implementați un user weighted round-robin scheduler: algoritmul parcurge lista de utilizatori cu procese gata de execuție în ordine round-robin. O dată ales un utilizator, algoritmul alege un proces aparținând acestuia și îl lasă să se execute un timp finit. Utilizatorii au o pondere asociată care dictează timpul de execuție permis.

6. **SchedSim** (2–3 studenți) – Scrieți un simulator de procese care să fie folosit pentru a evalua performanța algoritmilor de scheduling (cu prioritate și în timp real) oferind utilizatorilor indicatori standard precum utilizarea procesorului, timpul de așteptare etc.

7. **Alloc** (2 studenți) – Creați o bibliotecă care să ofere operații de alocare, realocare și eliberare a memoriei, dedesubt având grijă să păstreze eficient blocurile de date. Funcțiile implementate trebuie să poată fii apelate concomitent de mai multe procese din sistem. Scrieți programe de test care să demonstreze punctual corectitudinea diferitelor aspecte de implementare.

8. **UserFS** (2–3 studenți) – Scrieți un pseduo-sistem de fișiere care să afișeze utilizatorii activi din sistem și procesele asociate. Când este montat, în rădăcină se vor găsi directoare corespunzătoare fiecărui utilizator activ. În fiecare director se va găsi un fișier `procs` ce conține lista aferentă de procese active.

9. **ProcFS** (2–3 studenți) – Scrieți un pseduo-sistem de fișiere care să urmeze structura arborescentă a proceselor unui sistem. Când este montat, directoarele și subdirectoarele corespund PID-urilor și sunt numite ca atare. Fiecare director conține un singur fișier `stats` cu date statistice legate de proces.

10. **ContainerFS** (3 studenți) – Scrieți un sistem de fișiere minimal în userland folosind FUSE. Implementarea poate urmării un arhivator clasic de fișiere precum zip sau tar. Când sistemul este montat, rădăcina va conține directoarele și fișierele aferente arhivei.

11. **LockCheck** (3 studenți) – Scrieți un algoritm care să primească un fișier în care sunt folosite diferite mecanisme de sincronizare (ex. mutex, semafor) și la ieșire să prezinte un raport în care să arate dacă și unde există posibilitatea apariției unui fenomen de *d*eadlock sau *s*tarvation

12. **Lockers** (2–3 studenți) – Implementați propria bibliotecă care să ofere obiecte de sincronizare: mutecși, semafoare, rwlocks fără a folosi pe cele existente în biblioteci precum *p*thread. Pentru implementare puteți folosi doar primitive atomice precum *c*ompare-and-swap.

13. **Monitor** (2–3 studenți) – Implementați un obiect de sincronizare tip monitor. Folosiți variabile condiționale pentru a permite mai multor procese să coexiste în interiorul monitorului la un moment dat.

14. **ContribOS** (1 student) – Orice contribuție tehnică adusă (acum sau în trecut) unui sistem de operare open-source.