Laborator 3 – Sisteme de Operare

Ce reprezinta kernel-ul sistemului de operare ?

Kernel-ul este cea mai importanta si sensibila componenta dintr-un sistem de operare. El reprezinta componenta software fundamentala ce manageriaza toate resursele calculatorului. Printre rolurile sale se numara: managementul memoriei, lucrul cu servicile de retea, managementul proceselor, accesul la perifericele calculatorului etc.

Programatorii de aplicatii nu trebuie sa isi scrie ei singuri proceduri prin care sa lucreze cu dispozitivile de I/O sau dispozitivile de retea. In schimb, kernel-ul abstractizeaza aceste operatii si expune programatorului o interfata. Astfel, programtorii se pot concentra pe logica si functionalitatea aplicatiei.

Un **API (** *Application Programming Interface* **)** reprezinta o interfata prin care se pot construi aplicatii mai complexe pornind de la ceva deja scris. API-ul este strans legat de paradigma limbajului in care ne scriem programele. De exemplu, in limbajele procedurale precum C, Fortan, avem la dispozitie proceduri de I/O, manipulare de stringuri etc (printf() , strlen()), in limbajele orientate obiect precum Java sau C++ avem la dispozitie clase pentru astfel de operatii (iostream, string , System).

La inceputurile sale, sistemul de operare UNIX exista in mai multe versiuni ce functionau pe calculatoare cu arhitecturi diferite, versiuni ce erau departe de a fi complet compatibile. Din acest motiv a fost nevoie de o standardizare a modului in care utilizatorii interactionau cu sistemul. Acest standard avea sa poarte numele de **POSIX** (*Portable Operating System Interface*), ultimul 'X' apare deoarce majoritatea variantelor de UNIX se termina in 'X'.

Redactat de catre IEEE (*Institute of Electrical and Electronic Engineers*), **POSIX** documenteaza un API pentru un set de servicii ce ar trebuie puse la dispozitie unui program de catre un sistem de operare ce se conformeaza la acest standard.

Standardul la care se conformeaza sistemele de operare din familia Windows se numeste **WinAPI** (Win32 pentru platformele pe 32-bit sau Win64 pentru platformele de 64-bit).

Pe parcursul acestui semestru noi o sa lucram numai cu API-ul POSIX. Acest API este focusat pe limbajul de programare C.

Pe parcursul executiei sale, un proces se poate afla in doua moduri:

- **user mode**, atunci cand executa instructiuni utilizator precum sortarea unui vector, numararea caracterelor dintr-un string etc.
- **kernel mode,** atunci cand executa instructiuni privilegiate precum scrierea pe disk, citirea de pe un socket de retea, afisare pe ecran etc.

Un proces intra in kernel mode atunci cand executa un **apel sistem**. Din punctul de vedere al programatorului, un apel sistem seamana cu un apel normal de functie C. In spate, mult mai multe lucruri se intampla. Vezi :

https://www.youtube.com/watch?v=FkIWDAtVIUM

Din acest motiv, functile din C se pot imparti in doua categori:

- **functii de biblioteca**, ce nu provoaca intrarea procesului in kernel mode. Un exemplu ar fi aproape toate functile de manipulare de stringuri din header-ul “string.h”, precum strlen(), strchr(), strcat() etc.
- **functii de apel sistem,** ce reprezinta un wrapper peste apelurile sistem din sistemul de operare.

Biblioteca C standard este o biblioteca portabila deoarece interfata de programare este aceeasi independent de arhitectura hardware si sistemul de operare. Adica functia “printf()” poate sa fie folosita atat de catre un cod sursa C peste Linux cat si peste Windows.

Gestionarea erorilor

Constituie o grava eroare de programare faptul de a nu testa daca un apel de functie s-a soldat cu esec. Deobicei, pentru functile care returneaza **int,** un cod de retur negativ semnaleaza faptul ca functia s-a soldat cu esec, pentru functile care returneaza pointer de orice tip, codul de retur **NULL** reprezinta un esec. In cazul in care dorim sa aflam mai multe informatii despre cauza insuccesului unui asemenea apel putem utiliza variabila globala:

```
extern int errno;
```

Declaratia acestei variabile se gaseste in fisierul header “errno.h”. In urma fiecarui apel sistem ce a esuat lui **errno** ii este asignata o valoare intreaga corespunzatoare codului de esec. Aceasta ramane asignata pana la urmatorul apel sistem ce se termina cu esec sau pana se face o asignare noua manual.

In header-ul “errno.h” se gasesc macrouri ce definesc fiecare cod de eroare. (EACCES, EAGAIN, etc).

Functia **perror()** se foloseste pentru a afisa descrierea erorii. Ea utilizeaza valoarea lui **errno** si dictionarul :

```
extern char **sys_errlist
```

ce este de lungime **extern int sys_errno.**

Exercitiu

Scrieti un program care tipareste pe ecran toate mesajele de eroare ce se pot obtine cu functia "perror", precedate de valoarea corespunzatoare a variabilei "errno".

!!! IMPORTANT !!!

Atunci cand facem un apel sistem, trebuie tratate erorile ce pot aparea.

Gestionarea sistemului de fisiere

Fisierul reprezinta un concept fundamental in lumea sistemelor de operare. El este modul prin care sistemul isi organizeaza informatia pe care o foloseste. Pentru a avea o intelegere buna despre fisiere si sistemul de fisiere este **FOARTE IMPORTANT** sa cititi capitolul 4 (*Sisteme de fisiere*) din cartea *Introducere in sisteme de operare* ce se afla in bibliografia data in laboratorul 1.

Fisierul este identificat de utilizator prin nume si cale, dar sistemul il recunoaste dupa numarul sau de **i-nod.** I-nod-ul este o structura de date ce este folosita sa reprezinte un obiect unic din sistemul de fisiere.

In Linux exista mai multe tipuri de fisiere:

- fisiere de tip **regular,** sunt orice fisiere ce reprezinta o insiruire simpla de octeti (fisiere text, fisiere executabile, fisiere word etc).
- fisiere de tip **director**, sunt cele care memoreaza legatura dintre numerele de i-nod si numele atribuite fisierelor.
- fisiere de tip **pipe**, sunt fisiere prin care se face o comunicare unidirectionala si functioneaza pe principiul FIFO (*First In First Out*)
- fisiere corespunzatoare perifericelor fizice care pot sa fie de tip **bloc** sau de tip **caracter.** Acestea se afla in mare parte in directorul **/dev,** unde sistemul isi abstractizeaza toate perifericele.
- fisiere de tip **socket** sunt folosite in comunicare prin retea.
- fisiere de tip **legatura simbolica,** zona sa de date reprezinta o referinta catre un alt i-nod din sistem.

Pana acum, am folosit functile de biblioteca **printf(), scanf(), fopen()** pentru lucrul cu fisiere avand o abstractizare asupra modului cum se lucreaza concret cu fisierele la nivel de apel sistem. Pe parcursul acestui semestru, o sa folosim apelurile sistem **POSIX** pentru manipularea lor.

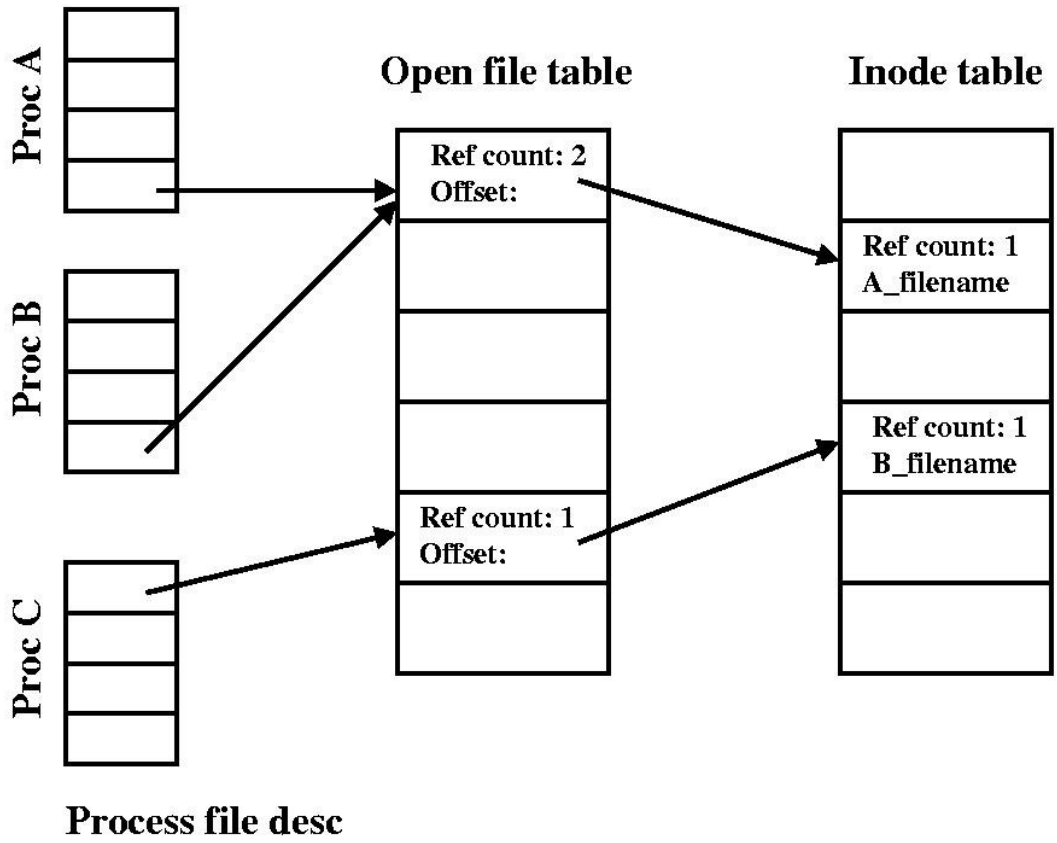
Intern, gestionarea fisierelor se face prin intermediul a trei tabele:

1. **Tabela de descriptori**. Fiecare proces are o tabela de descriptori. Ea este un vector de structuri, indexat de la 0 la cel mai mare descriptor posibil. Daca procesul a asociat unui fisier un descriptor **i** (numar natural), intrarea **i** din aceasta tabela e completata cu informatii specifice, printre care un pointer la o intrare in **tabela de fisiere deschise**. De regula primii trei descriptori sunt asignati automat dupa cum urmeaza:
    - 0 = intrarea standard;
    - 1 = iesirea standard;
    - 2 = iesirea standard pentru erori;

   ei pot fi desemnati si prin urmatoarele constante simbolice, definite in "unistd.h": **STDIN_FILENO** (=0), **STDOUT_FILENO** (=1), **STDERR_FILENO** (=2).

2. **Tabela de fisiere deschise.** Aceasta tabela este partajata de catre toate procesele. Orice intrare din tabela de descriptori a oricarui proces memoreaza adresa unei intrari din aceasta tabela. La deschiderea unui fisier de catre un proces, se creeaza o noua intrare in tabela de descriptori a procesului si o noua intrare in tabele de fisiere deschise. Aceasta contine un pointer catre o intrare din tabele de i-noduri din memorie.

3. **Tabela de i-noduri.** La deschiderea unui fisier, in caz ca datele despre i-nodul corespunzator nu se gasesc deja in aceasta tabela, se creeaza o noua intrare.



Mai sus avem o reprezentare a celor 3 tabele si a legaturilor dintre acestea !

Operatii asupra fisierelor

Deschiderea unui fisier ---> int open(char* nume_fisier,int mod_deschidere,mode_t drepturi).

Intoarce descriptorul corespunzator intrarii din tabel pentru fisierul **nume_fisier,** deschis cu modul **mod_deschidere** si cu masca de drepturi **drepturi**. In caz de succes o sa intoarca cel mai mic descriptor liber, in caz de insucces intoarce -1 si seteaza **errno.**

Headere necesare : “sys/types.h”, “sys/stat.h”, “fcntl.h”

Documentatie : http://linux.die.net/man/2/open

Inchiderea unui fisier ---> int close(int descriptor)

Inchide descriptorul de fisier deschis in prealabil. Functia o sa intoarca 0 in caz de succes si -1 in caz de esec si seteaza **errno.**

Headere necesare: “unistd.h”

Documentatie: http://linux.die.net/man/2/close

Citirea din fisier ---> ssize_t read(int descriptor, void* pointer_destinatie, size_t nr_oct)

Citeste din fisierul indicat de **descriptor** un numar de **nr_oct** si ii pune in pointer-ul **pointer_destinatie**, acesta trebuie sa fie alocat in prealabil. In caz de succes, intoarce numarul de octeti cititi, care poate sa fie mai mic decat **nr_oct**, iar in caz de esec -1 si seteaza **errno.**

Headere necesare: “unistd.h”

Documentatie: http://linux.die.net/man/2/read

Scrierea in fisier ---> ssize_t read(int descriptor, void* pointer_sursa, size_t nr_oct)

Scrie in fisierul indicat de **descriptor** un numar de **nr_oct** si ii pune in pointer-ul **pointer_sursa**, acesta trebuie sa fie alocat in prealabil. In caz de succes, intoarce numarul de octeti scrisi, iar in caz de esec -1 si seteaza **errno.**

Headere necesare: “unistd.h”

Documentatie: http://linux.die.net/man/2/write

Exercitii

1.Scrieti un program "copy" care se va lansa sub forma:
copy f1 + ... + fn f
(unde f1, ..., fn, f sunt fisiere) si are ca efect crearea lui f continand concatenarea lui f1, ..., fn; daca n=1 se copiaza f1 in f.

2. Scrieti un program care compara din punct de vedere lexicografic continutul a doua fisiere (privite ca sir de caractere). Numele fisierelor sunt date ca argument in linia de comanda.